第 5 章

Django

5.1 Django 简介

Django 是一个由 Python 编写的开放源代码的 Web 应用框架，用于快速开发可维护和可扩展的 Web 应用程序。

通过 Django，开发人员只需很少的代码，就可以轻松地完成一个正式网站所需要的大部分内容，并可以进一步开发出全功能的 Web 服务。

Django 是一个遵循 MVC 设计模式的框架，但由于其控制器中接收用户输入的部分是由框架自行处理的，所以 Django 更关注的是模型（Model）、模板（Template）和视图（Views），即 MTV 模式。

Django 的 MTV 模式本质上和 MVC 模式是一致的，均是为了各组件之间保持松耦合关系，只是在定义上有些许不同，其各部分的职责如下：

M 表示模型（Model），即数据存取层，主要用于处理与数据相关的所有事务，例如，如何存取、如何验证有效性、包含哪些行为及数据之间的关系等。

T 表示模板（Template），即表现层，主要用于处理与表现相关的业务，例如，如何在页面或其他类型文档中进行显示。

V 表示视图（View），即业务逻辑层，主要用于存取数据模型及调取恰当模板的相关逻辑，是数据模型与模板的桥梁。

5.2 安装 Django

由于 Django 属于 Python 的第三方库，所以需要进行安装，打开命令提示符窗口，输入命令 pip install django 即可。

5.3 第 1 个 Django 项目

创建 Django 项目可以通过两种方式，即使用 PyCharm（Professional）和使用命令提示符。

1. 使用 PyCharm（Professional）

1）创建项目

通过 PyCharm（Professional）可以快速地创建一个 Django 项目，具体步骤如下。

（1）打开 PyCharm(Professional)，选择 File→New Project，如图 5-1 所示。

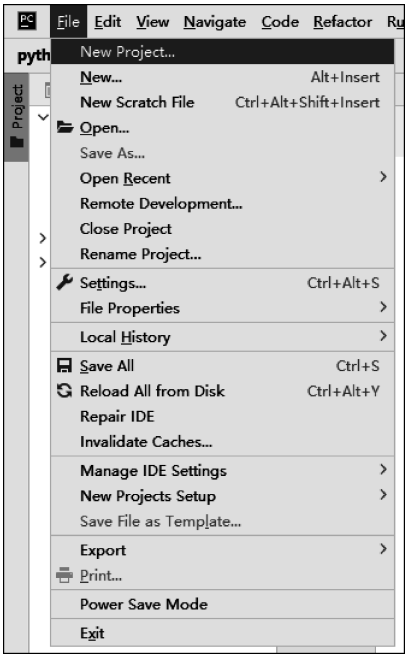


图 5-1　创建 Django 项目的第 1 步

（2）在当前界面的左侧栏中选择 Django，然后在右侧栏中单击 Previously configured interpreter，并在其中选择虚拟环境 django_env，最后单击 Create 按钮，即可创建一个名为 djangoProject 的项目，如图 5-2 所示。

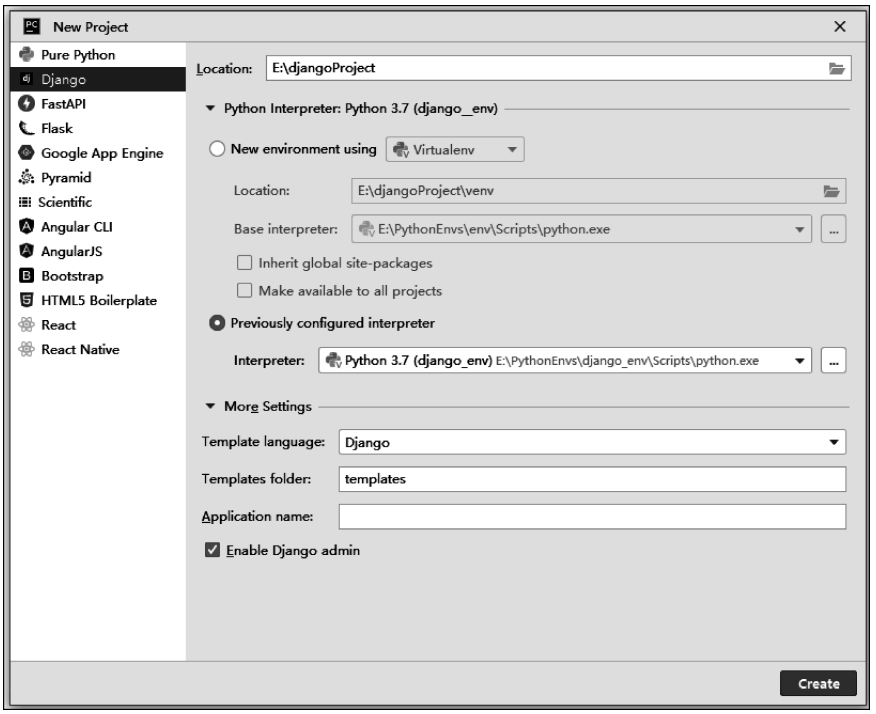


图 5-2　创建 Django 项目的第 2 步

此时，使用 PyCharm(Professional)创建的项目 djangoProject 的结构如图 5-3 所示。

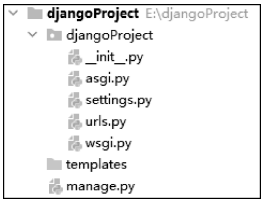


图 5-3 项目 djangoProject 的结构

其中，文件夹 djangoProject 表示项目中的应用，其与当前的项目名称相同；文件 manage.py 主要包含项目管理的子命令；文件夹 templates 主要用于存放模板文件；文件 asgi.py 为异步服务器网关接口的配置文件；文件 settings.py 为项目的配置文件；文件 wsgi.py 为 Web 服务网关的配置文件；文件 urls.py 为项目主路由的配置文件。

这里重点讲解项目和应用的关系。应用是 Django 项目的组成部分，一个应用代表项目中的一个模块，所有 URL 请求的响应都是由应用来处理的，例如豆瓣网中有图书、电影、音乐和同城等模块，而这些模块在 Django 项目中就是应用，图书、电影、音乐和同城等应用共同组成了豆瓣网这个项目，因此一个项目中可以包含多个应用，并且一个应用也可以在多个项目中使用。

运行上述项目，PyCharm 中的显示结果如图 5-4 所示。

```
E:\PythonEnvs\django_env\Scripts\python.exe E:/djangoProject/manage.py runserver 8000
Performing system checks...

System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for
 app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 19, 2024 - 22:57:37
Django version 2.1.15, using settings 'djangoProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

图 5-4 PyCharm 中的显示内容

此时，打开浏览器并在网址栏输入 http://127.0.0.1:8000，其显示内容如图 5-5 所示。

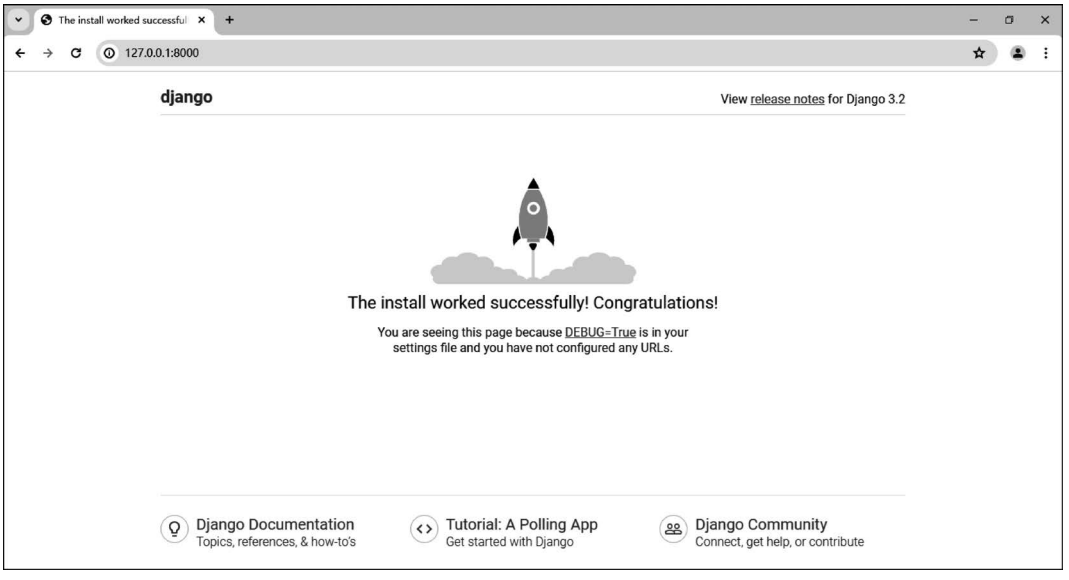


图 5-5 浏览器中的显示内容

2）修改端口号

通过单击 Edit Configurations，并在其显示的页面中修改 Port 所对应的值，即可完成端口号的修改，如图 5-6 所示。

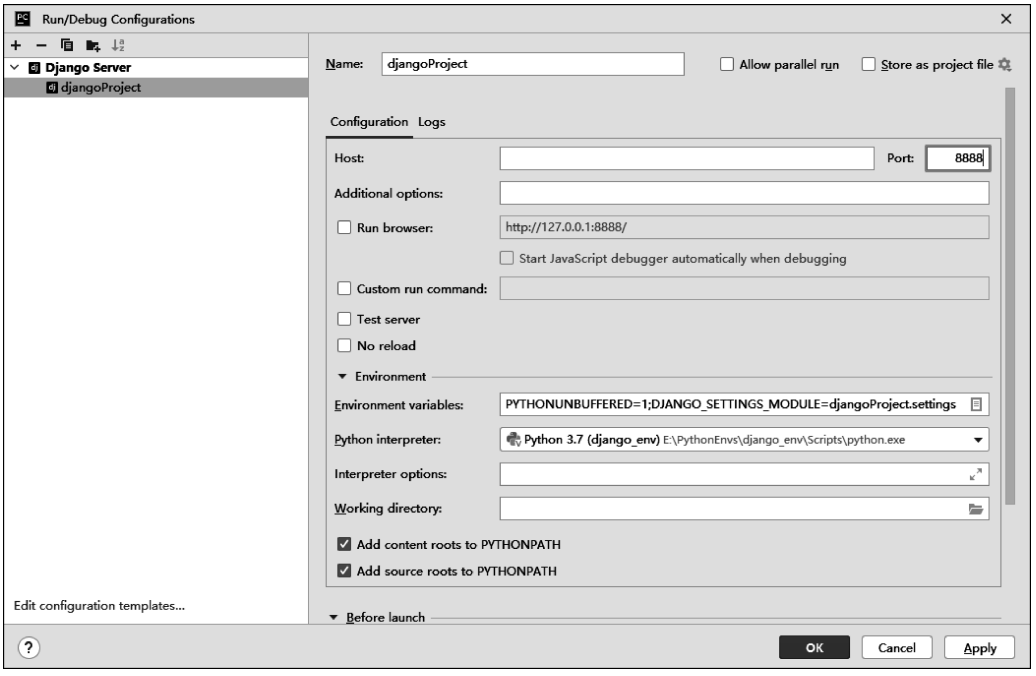


图 5-6　修改端口号

此时，打开浏览器并在网址栏输入 http://127.0.0.1:8888，其显示内容如图 5-7 所示。

图 5-7　浏览器中的显示内容

3）外部访问

通过单击 Edit Configurations，并在其显示的页面中将 Host 所对应的值修改为 0.0.0.0

(如图 5-8 所示),然后将配置文件 settings.py 中的参数 ALLOWED_HOSTS 的值修改为本机的 IP,即可使服务器被外部访问。

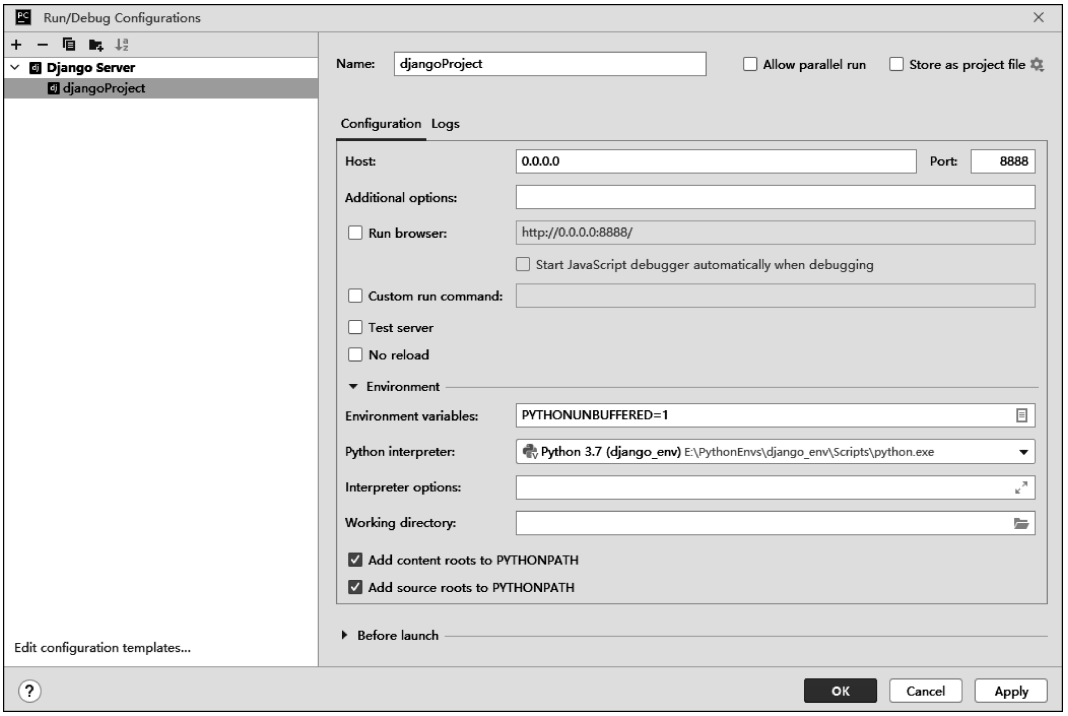


图 5-8 修改 Host

2. 使用命令提示符

1) 创建项目

打开命令提示符窗口,进入需要创建项目的目录中,激活虚拟环境,并输入命令 django-admin startproject djangoProject,创建名为 djangoProject 的 Django 项目,如图 5-9 所示。

```
(django_env) E:\>django-admin startproject djangoProject

(django_env) E:\>
```

图 5-9 创建项目

进入项目的根目录中,输入命令 python manage.py runserver 即可运行该项目,如图 5-10 所示。

```
(django_env) E:\djangoProject>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 17, 2024 - 15:38:56
Django version 3.2.25, using settings 'djangoProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

图 5-10 运行项目

此时,打开浏览器并在网址栏输入 http://127.0.0.1:8000,其显示内容如图 5-11

所示。

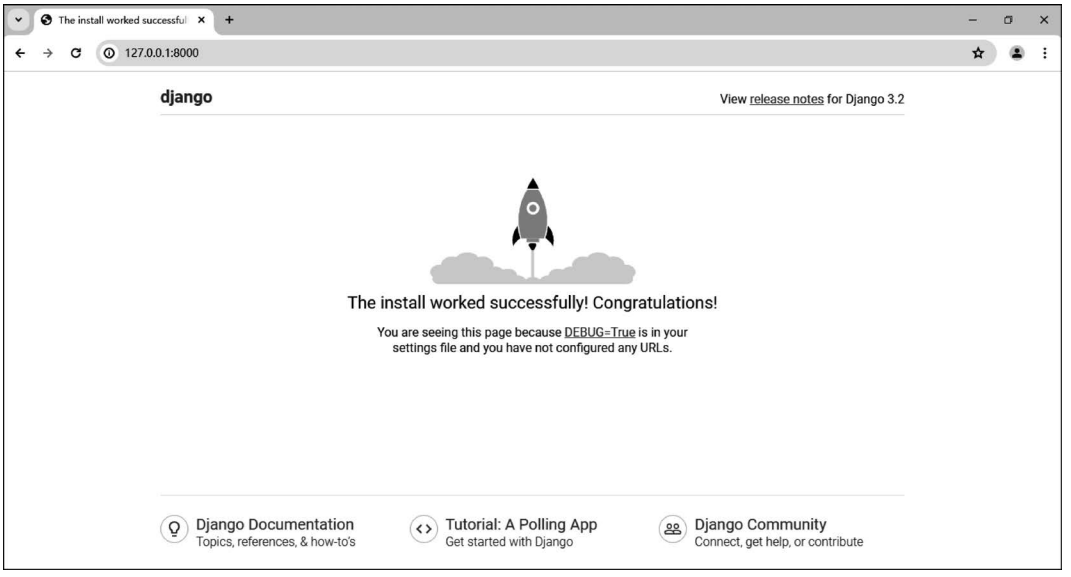


图 5-11　浏览器中的显示内容

2）修改端口号

打开命令提示符窗口，进入项目所在的目录中，激活虚拟环境，并输入命令“python manage.py runserver 端口号”，即可通过修改后的端口号运行 Django 项目，如图 5-12 所示。

```
(django_env) E:\djangoProject>python manage.py runserver 8888
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 19, 2024 - 17:49:36
Django version 3.2.25, using settings 'djangoProject.settings'
Starting development server at http://127.0.0.1:8888/
Quit the server with CTRL-BREAK.
```

图 5-12　修改端口号

此时，打开浏览器并在网址栏输入 http://127.0.0.1:8888，其显示内容如图 5-13 所示。

3）外部访问

首先，将保存在项目根目录下的配置文件 settings.py 中的参数 ALLOWED_HOSTS 的值修改为本机的 IP，然后打开命令提示符窗口，进入项目所在的目录中，激活虚拟环境，并输入命令“python manage.py runserver 0.0.0.0:端口号”，即可使服务器被外部访问，如图 5-14 所示。

此外，在 Django 中启用调试模式，需要将配置文件 settings.py 中的参数 DEBUG 的值设置为 True，并且需要确保参数 ALLOWED_HOSTS 的值包含本地开发主机。

需要注意的是，在生产环境中应将 DEBUG 设置为 False，否则将暴露关于当前应用和环境的敏感信息。

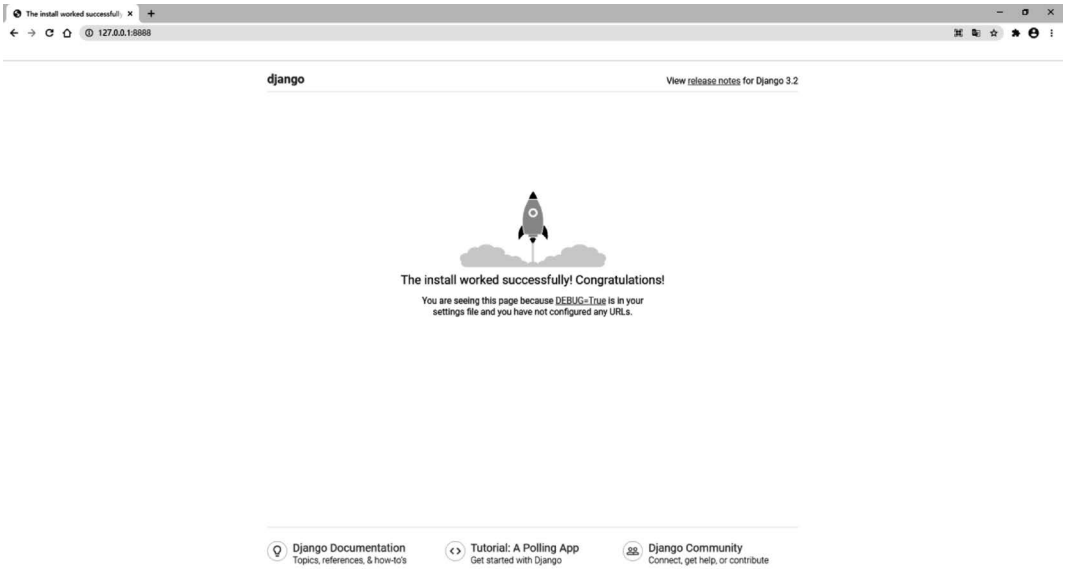


图 5-13 浏览器中的显示内容

```
(django_env) E:\djangoProject>python manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 23, 2024 - 16:49:23
Django version 3.2.25, using settings 'djangoProject.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CTRL-BREAK.
```

图 5-14 外部访问

5.4 路由

在 Django 中，用于处理请求 URL 和视图函数之间关系的程序称为路由。

5.4.1 视图函数

视图函数是一个简单的 Python 函数，其接受来自 Web 服务器的请求数据，并会将响应内容返给 Web 服务器。

一般情况下，视图函数均在应用中的文件 views.py 中进行编写。

视图函数中有两个重要的对象，即 HttpRequest 对象和 HttpResponse 对象。每个视图函数的第 1 个参数必须是 HttpRequest 对象，并且每个视图函数的返回结果都必须是一个 HttpResponse 对象。

下面通过一个示例演示一下如何创建视图函数。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-15 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-15　创建应用

(3) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.4\1\djangoProject\book\views.py
from django.http import HttpResponse
def book(request):
    return HttpResponse("这是书籍首页")
```

(4) 打开应用 djangoProject 中的文件 urls.py,编写 URL 映射(该部分内容将在后续章节为读者详细讲解),代码如下:

```
#资源包\Code\chapter5\5.4\1\djangoProject\djangoProject\urls.py
from django.urls import path
from django.http import HttpResponse
from book import views
def index(request):
    return HttpResponse('这是首页')
urlpatterns = [
    path('', index),
    path('book/', views.book)
]
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/book/,其显示内容如图 5-16 所示。

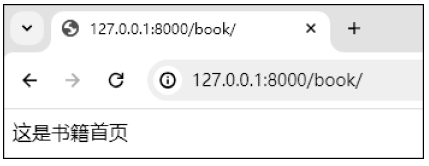


图 5-16　浏览器中的显示内容

5.4.2　URL 映射

在 Django 中,通过配置文件 settings.py 中的变量 ROOT_URLCONF 构建 URL 与视图函数的映射关系。

一般情况下,Django 项目中的路由通过 urls.py 文件进行配置,在该文件中定义了一个 urlpatterns 列表,而 URL 的定义就是在这个列表中完成的。

1. 普通路径

通过 django.urls 模块中的 path()函数定义普通的 URL 路径,其语法格式如下:

```
path(route, view, name, kwargs)
```

其中,参数 route 表示 URL 路径,参数 view 表示视图函数,参数 name 表示 URL 命名,用于反向获取 URL,参数 kwargs 用于传递给视图函数额外的关键字参数。

2. 正则路径

通过 django.urls 模块中的 re_path()函数定义匹配正则表达式的 URL 路径,其语法格

式如下：

```
re_path(route, view, name, kwargs)
```

其中，参数 route 表示 URL 路径，该值可以为正则表达式，参数 view 表示视图函数，参数 name 表示 URL 命名，用于反向获取 URL，参数 kwargs 用于传递给视图函数额外的关键字参数。

下面通过一个示例演示一下如何定义 URL 映射。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-17 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-17 创建应用

(3) 打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.4\2\djangoProject\book\views.py
from django.http import HttpResponse
def book(request):
    return HttpResponse("这是书籍首页")
def book_detail(request):
    return HttpResponse('这是书籍详细信息页面')
```

(4) 打开应用 djangoProject 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.4\2\djangoProject\djangoProject\urls.py
from django.urls import path, re_path
from django.http import HttpResponse
from book import views
def index(request):
    return HttpResponse('这是首页')
urlpatterns = [
    path('', index),
    path('book/', views.book),
    re_path(r'book_detail/[0-9]{4}/', views.book_detail)
]
```

(5) 运行上述项目，打开浏览器并在网址栏输入 http://127.0.0.1:8000/book_detail/8888/，其显示内容如图 5-18 所示。

图 5-18 浏览器中的显示内容

5.4.3 HttpRequest 对象

HttpRequest 对象是 Django 中用于封装 HTTP 请求信息的类，即当用户发起一个

HTTP 请求时，Django 会创建一个 HttpRequest 对象，并将其传递给相应的视图函数。

HttpRequest 对象包含了请求的各种信息，例如 HTTP 方法、URL、查询参数和表单数据等，并且在视图函数中，可以通过 HttpRequest 对象的属性来获取请求的详细信息，其常用属性和方法如表 5-1 所示。

表 5-1　HttpRequest 对象的常用属性和方法

属　　性	描　　述
method	获取 HTTP 请求方法
path	获取请求的 URL 路径
GET	获取 GET 请求的参数，返回一个 QueryDict 对象
POST	获取 POST 请求的参数，返回一个 QueryDict 对象
FILES	获取所有的上传文件，返回一个 QueryDict 对象
COOKIES	获取所有的 Cookie 数据
session	获取或设置 Session 数据
META	获取所有的 HTTP 头部信息
user	获取当前用户信息
方　　法	**描　　述**
get_full_path()	获取请求的完整路径
get_raw_uri()	获取完整的原始 URL，包括域名和查询字符串
get_host()	获取请求的原始主机名和端口
is_secure()	判断请求是否采用 HTTPS 协议
is_ajax()	判断请求是否采用 AJAX 发送

5.4.4　QueryDict 对象

在 Django 中，QueryDict 是一个类似字典的对象，用于处理 HTTP 请求中的 GET、POST 或 FILES 等数据，其常用的方法如表 5-2 所示。

表 5-2　QueryDict 对象的常用方法

方　　法	描　　述
get()	获取指定键所对应的值。如果键不存在，则默认返回 None。此外，可以通过该方法的第 2 个参数 default 来指定默认值
getlist()	获取指定键所对应的所有值。如果键不存在，则返回空列表

5.4.5　HttpResponse 对象

在 Django 中，HttpResponse 对象用于表示 HTTP 响应，包含响应状态码、头部信息和主体内容等，其常用属性和方法如表 5-3 所示。

表 5-3　HttpResponse 对象的常用属性和方法

属　　性	描　　述
content	响应内容
status_code	响应状态码
content_type	响应的内容类型

续表

方　法	描　述
set_cookie()	设置 Cookie
delete_cookie()	删除 Cookie
write()	用于将字符串数据写入响应中

5.4.6 JsonResponse 对象

在 Django 中，可以通过 JsonResponse 对象将字典直接转换为 JSON 响应。

5.4.7 重定向

重定向分为永久性重定向和暂时性重定向，在页面上体现的操作就是浏览器会从一个页面自动跳转到另一个页面，例如用户访问了一个需要权限的页面，但是该用户当前并没有登录，因此需要将该用户重定向至登录页面。

在 Django 中，可以通过 redirect()函数实现重定向，其语法格式如下：

```
redirect(to, permanent)
```

其中，参数 to 表示重定向的 URL，参数 permanent 表示重定向的状态，默认为暂时性重定向，当其值为 True 时，表示永久性重定向。

5.4.8 动态路由

当请求 URL 需要动态变化时，需要使用动态路由，即需要给请求 URL 传递参数，可以通过将请求 URL 路径中的一部分标记为< variable_name >的方式添加参数。

这里需要注意两点，一是当请求 URL 中包含参数时，视图函数中必须传递该参数；二是当视图函数中的参数为默认参数时，请求 URL 中可以不用传递参数。

下面通过一个示例演示一下如何定义动态路由。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-19 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-19　创建应用

(3) 打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.4\3\djangoProject\book\views.py
from django.http import HttpResponse
def book_info(request, book_info = "default"):
    return HttpResponse(f"书籍的信息为{book_info}")
def book_id(request, book_id):
    return HttpResponse(f"书籍的 id 为{book_id}")
```

(4) 打开应用 djangoProject 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.4\3\djangoProject\djangoProject\urls.py
from django.urls import path
from django.http import HttpResponse
from book import views
def index(request):
    return HttpResponse('这是首页')
urlpatterns = [
    path('', index),
    path('book_info/', views.book_info),
    path('book_id/<book_id>/', views.book_id),
]
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/book_id/20241001/,其显示内容如图 5-20 所示。

(6) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book_info/,其显示内容如图 5-21 所示。

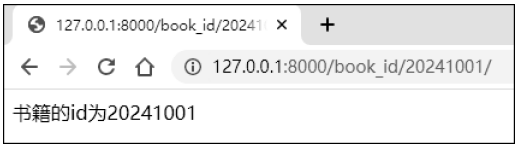


图 5-20 浏览器中的显示内容

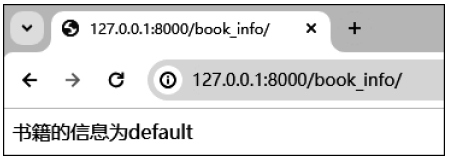


图 5-21 浏览器中的显示内容

5.4.9 动态构建请求 URL

当视图函数所对应的请求 URL 被大量修改时,如果通过动态构建请求 URL,则无须手动去更改其他地方已经使用该视图函数所对应的请求 URL。

可以通过 django.shortcuts 模块中的 reverse()函数动态地构建指定视图函数的请求 URL,其语法格式如下:

```
reverse(viewname, kwargs)
```

其中,参数 viewname 表示 URL 命名,参数 kwargs 表示请求 URL 中的参数。

下面通过一个示例,演示一下如何动态地构建请求 URL。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-22 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-22 创建应用

(3) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.4\4\djangoProject\book\views.py
from django.shortcuts import redirect, reverse
from django.http import HttpResponse
def index(request):
```

```
    username = request.GET.get('username')
    if username:
        return HttpResponse(f'这里是书籍详情的首页')
    else:
        return redirect(reverse('book_detail', kwargs={'book_id': '20241001'}))
def book_detail(request, book_id):
    return HttpResponse(f'书籍的 id 为{book_id}')
```

(4) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\4\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('', views.index, name='index'),
    path('detail/<book_id>/', views.book_detail, name='book_detail')
]
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/?username=oldxia,其显示内容如图 5-23 所示。

(6) 在浏览器中的网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-24 所示。

图 5-23 浏览器中的显示内容

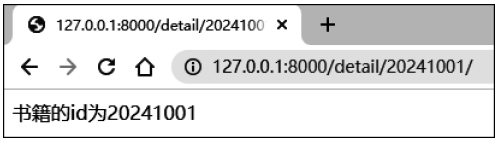


图 5-24 浏览器中的显示内容

5.4.10 路由分发

在 Django 中,可以通过 django.urls 模块中的 include()函数来实现路由分发,其通常用于将应用的路由组织在一起,或者将特定的路由模式分配给特定的应用。

include()函数具有以下 3 种常用的语法格式:

```
include(pattern, namespace)
```

其中,参数 pattern 表示应用中的模块,参数 namespace 表示实例命名空间。

```
include(pattern, app_namespace, namespace)
```

其中,参数 pattern 表示应用中的模块,参数 app_namespace 表示应用命名空间,参数 namespace 表示实例命名空间。

```
include(pattern_list)
```

其中,参数 pattern_list 表示 URL 映射组成的列表。

这里重点讲解 include()函数中的参数所表示的应用命名空间和实例命名空间。

1. 应用命名空间

由于在多个应用之间可能会存在同名的 URL,所以为了避免动态地构建请求 URL 时

产生混淆，可以通过应用命名空间进行区分。定义应用命名空间非常简单，只需在应用中的文件 urls.py 中定义变量 app_name，便可通过该变量指定当前应用的命名空间。

2. 实例命名空间

由于一个应用可以创建多个 URL 映射，所以就会产生一个问题，即在动态构建请求 URL 时，可能产生混淆。为了避免这个问题，可以使用实例命名空间。

除此之外，在使用实例命名空间时，必须指定应用命名空间，否则程序会报错。

下面通过一个示例演示一下如何使用路由分发。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp front、python manage.py startapp cms、python manage.py startapp movie 和 python manage.py startapp book，创建名为 front、cms、movie 和 book 的应用，如图 5-25 所示。

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp front
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp cms
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp movie
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-25 创建应用

(3) 分别在应用 front、cms 和 movie 中创建文件 urls.py，如图 5-26 所示。

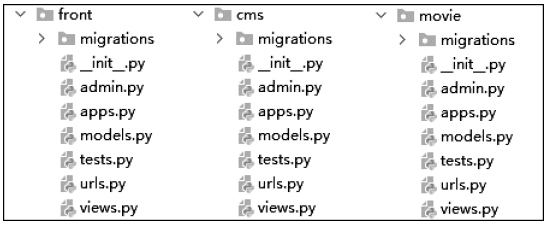


图 5-26 在应用中创建文件 urls.py

(4) 打开应用 front 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.4\5\djangoProject\front\views.py
from django.http import HttpResponse
from django.shortcuts import redirect, reverse
def index(request):
    username = request.GET.get('username')
    if username:
        return HttpResponse('这里是网站的首页')
    else:
        #使用应用命名空间
        return redirect(reverse('front:login'))
def login(request):
    return HttpResponse('这里是网站的登录页面')
```

(5) 打开应用 cms 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.4\5\djangoProject\cms\views.py
from django.http import HttpResponse
```

```
from django.shortcuts import redirect, reverse
def index(request):
    username = request.GET.get('username')
    if username:
        return HttpResponse('这里是 CMS 的首页')
    else:
        #获取当前实例命名空间
        current_namespace = request.resolver_match.namespace
        #使用实例命名空间
        return redirect(reverse(f'{current_namespace}:login'))
def login(request):
    return HttpResponse('这里是 CMS 的登录页面')
```

(6) 打开应用 movie 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.4\5\djangoProject\movie\views.py
from django.http import HttpResponse
from django.shortcuts import redirect, reverse
def index(request):
    username = request.GET.get('username')
    if username:
        return HttpResponse('这里是电影的首页')
    else:
        #使用应用命名空间
        return redirect(reverse('movie:login'))
def login(request):
    return HttpResponse('这里是电影的登录页面')
```

(7) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.4\5\djangoProject\book\views.py
from django.http import HttpResponse
def book(request):
    return HttpResponse("这里是书籍的首页")
def book_detail(request):
    return HttpResponse("这里是书籍详情的页面")
```

(8) 打开应用 front 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\5\djangoProject\front\urls.py
from django.urls import path
from front import views
#通过变量 app_name 设置应用命名空间
app_name = 'front'
urlpatterns = [
    path('', views.index, name = 'index'),
    path('login/', views.login, name = 'login')
]
```

(9) 打开应用 cms 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\5\djangoProject\cms\urls.py
from django.urls import path
from cms import views
```

```
#通过变量 app_name 设置应用命名空间
app_name = 'cms'
urlpatterns = [
    path('', views.index, name = 'index'),
    path('login/', views.login, name = 'login')
]
```

(10) 打开应用 movie 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\5\djangoProject\movie\urls.py
from django.urls import path
from movie import views
urlpatterns = [
    path('', views.index, name = 'index'),
    path('login/', views.login, name = 'login')
]
```

(11) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\5\djangoProject\djangoProject\urls.py
from django.urls import path, include
from book import views
urlpatterns = [
    path('', include('front.urls')),
      #在同一个应用下,具有两个实例,为了避免混淆,可以使用实例命名空间
     path('cms1/', include('cms.urls', namespace = 'cms1')),
    path('cms2/', include('cms.urls', namespace = 'cms2')),
    path('movie/', include(('movie.urls', 'movie'))),
    path('book/', include([
        path('', views.book),
        path('detail/', views.book_detail)
    ]))
]
```

(12) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-27 所示。

(13) 在浏览器中的网址栏输入 http://127.0.0.1:8000/?username=oldxia,其显示内容如图 5-28 所示。

图 5-27 浏览器中的显示内容(1)

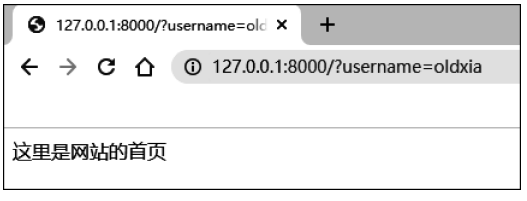


图 5-28 浏览器中的显示内容(2)

(14) 在浏览器中的网址栏输入 http://127.0.0.1:8000/cms1/login/,其显示内容如图 5-29 所示。

(15) 在浏览器中的网址栏输入 http://127.0.0.1:8000/cms2/?username=oldxia,其显示内容如图 5-30 所示。

图 5-29 浏览器中的显示内容(3)

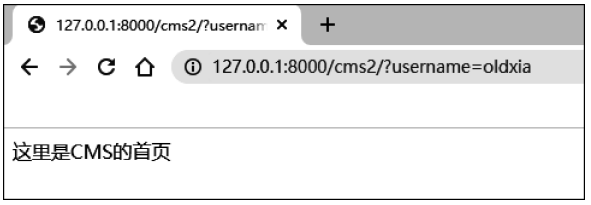


图 5-30 浏览器中的显示内容(4)

(16) 在浏览器中的网址栏输入 http://127.0.0.1:8000/movie/login/,其显示内容如图 5-31 所示。

(17) 在浏览器中的网址栏输入 http://127.0.0.1:8000/movie/?username=oldxia,其显示内容如图 5-32 所示。

图 5-31 浏览器中的显示内容(5)

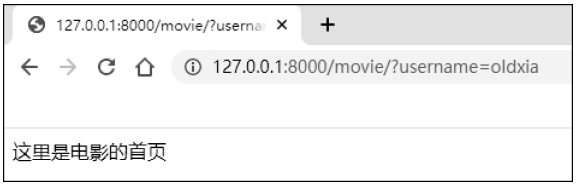


图 5-32 浏览器中的显示内容(6)

(18) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book/,其显示内容如图 5-33 所示。

(19) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book/detail/,其显示内容如图 5-34 所示。

图 5-33 浏览器中的显示内容(7)

图 5-34 浏览器中的显示内容(8)

5.4.11 路由转换器

路由转换器主要用于对请求 URL 的参数的类型进行限制,其可以分为内置转换器和自定义转换器。

1. 内置转换器

可以通过将请求 URL 路径中的一部分标记为< converter:variable_name >的方式添加内置转换器,其中,converter 表示限制的规则,包括 str(默认转换器,接受任何不包含斜杠的文本)、int(接受正整数)、slug(接受连字符、下画线、数字或字母)、path(类似 string,但可以接受包含斜杠的文本)和 uuid(接受 uuid 字符串)。

下面通过一个示例演示一下如何使用内置转换器。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-35 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-35 创建应用

(3) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.4\6\djangoProject\book\views.py
from django.http import HttpResponse
def book_int(request, test_int):
    return HttpResponse(f"书籍的 id 为{test_int}")
def book_str(request, test_str):
    return HttpResponse(f"书籍的 id 为{test_str}")
def book_slug(request, test_slug):
    return HttpResponse(f"书籍的 id 为{test_slug}")
def book_path(request, test_path):
    return HttpResponse(f"书籍的 id 为{test_path}")
def book_uuid(request, test_uuid):
    return HttpResponse(f"书籍的 id 为{test_uuid}")
```

(4) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\6\djangoProject\djangoProject\urls.py
from django.urls import path
from django.http import HttpResponse
from book import views
def index(request):
    return HttpResponse('这是首页')
urlpatterns = [
    path('', index),
    path('book_int/<int:test_int>/', views.book_int),
    path('book_str/<str:test_str>/', views.book_str),
    path('book_slug/<slug:test_slug>/', views.book_slug),
    path('book_path/<path:test_path>/', views.book_path),
    path('book_uuid/<uuid:test_uuid>/', views.book_uuid),
]
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/book_int/20241001/,其显示内容如图 5-36 所示。

(6) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book_str/oldxia/,其显示内容如图 5-37 所示。

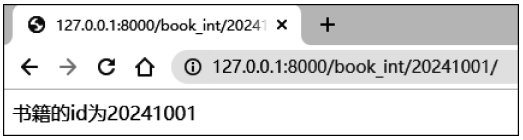


图 5-36 浏览器中的显示内容(1)

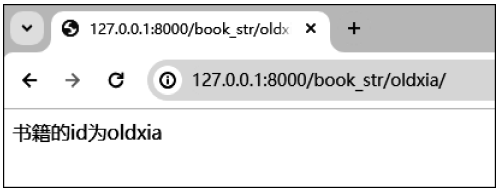


图 5-37 浏览器中的显示内容(2)

(7) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book_slug/oldxia-20241001_

2244999777/,其显示内容如图 5-38 所示。

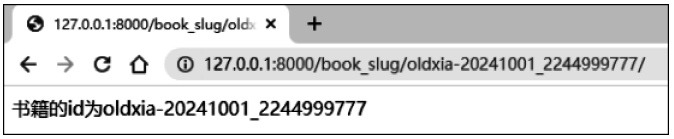


图 5-38 浏览器中的显示内容(3)

(8) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book_path/oldxia/13309861086/,其显示内容如图 5-39 所示。

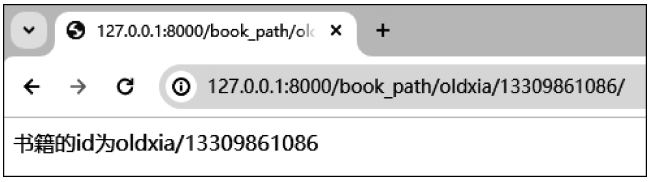


图 5-39 浏览器中的显示内容(4)

(9) 在浏览器中的网址栏输入 http://127.0.0.1:8000/book_uuid/123e4567-e89b-4123-b567-0987654321ab/,其显示内容如图 5-40 所示。

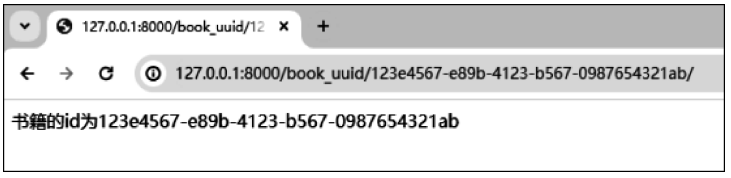


图 5-40 浏览器中的显示内容(5)

2. 自定义转换器

创建自定义转换器分为 3 步。

(1) 创建自定义转换器类,并在该类中实现相关属性和方法,如表 5-4 所示。

表 5-4 自定义转换器类中的属性和方法

属 性	描 述
regex	匹配请求 URL 参数的正则表达式
方 法	**描 述**
to_python()	当匹配到请求 URL 的参数后,该方法会将其返回值传递到该请求 URL 所对应的视图函数中
to_url()	当其他视图函数使 reverse()函数时,该方法会对传入的请求 URL 参数进行处理并返回

(2) 在自定义转换器类中,可以使用 django.urls 模块中的 register_converter()函数将该自定义转化器注册到 Django 中。

(3) 在项目同名的应用中的文件_init_.py 内导入该自定义转换器类。

下面通过一个示例演示一下如何自定义转换器。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令

python manage.py startapp book，创建名为 book 的应用，如图 5-41 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-41　创建应用

（3）在应用 book 中创建文件 urls.py 和 converters.py，如图 5-42 所示。

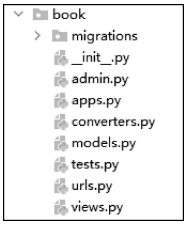


图 5-42　创建文件 urls.py 和 converters.py

（4）打开应用 book 中的文件 converters.py，用于自定义转换器类，代码如下：

```
#资源包\Code\chapter5\5.4\7\djangoProject\book\converters.py
from django.urls import register_converter
#自定义转换器类
class CategoryConverter(object):
    regex = r'\w+|(\w+\+\w+)+'
    def to_python(self, value):
        result = value.split('+')
        return result
    def to_url(self, value):
        if isinstance(value, list):
            result = '+'.join(value)
            return result
        else:
            raise RuntimeError('参数必须为【list】类型')
#将自定义转换器类注册到 Django 中
register_converter(CategoryConverter, 'cate')
```

（5）打开应用 djangoProject 中的文件__init__.py，代码如下：

```
#资源包\Code\chapter5\5.4\7\djangoProject\djangoProject\__init__.py
from book import converters
```

（6）打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.4\7\djangoProject\book\views.py
from django.http import HttpResponse
from django.shortcuts import reverse
def book_cate(request, categories):
    print(reverse('list', kwargs={'categories': categories}))
    return HttpResponse(f"该书籍的分类为{categories}")
```

（7）打开应用 book 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.4\7\djangoProject\book\urls.py
from django.urls import path
from book import views
```

```
urlpatterns = [
    path('cate/<cate:categories>/', views.book_cate, name = 'list'),
]
```

(8) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\7\djangoProject\djangoProject\urls.py
from django.urls import path, include
urlpatterns = [
    path('book/', include('book.urls'))
]
```

(9) 运行上述程序,打开浏览器并在网址栏输入 http://127.0.0.1:8000/book/cate/python+django+flask/,其显示内容如图 5-43 所示。

图 5-43 浏览器中的显示内容

此时,PyCharm 中的显示结果如图 5-44 所示。

```
E:\PythonEnvs\django_env\Scripts\python.exe E:/djangoProject/manage.py runserver 8000
Performing system checks...

System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for
 app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
August 15, 2024 - 22:38:48
Django version 2.1.15, using settings 'djangoProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[15/Aug/2024 22:38:57] "GET /book/cate/ptyhon+django+flask/ HTTP/1.1" 200 53
/book/cate/ptyhon+django+flask/
```

图 5-44 PyCharm 中的输出结果

5.4.12 限制请求方法

在 Django 中,可以通过 django.views.decorators.http 模块中的内置装饰器限制视图函数的请求方法,常用的装饰器包括 require_GET、require_POST 和 require_http_methods。

下面通过一个示例演示一下如何限制视图函数的请求方法。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-45 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-45 创建应用

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html 文件(模板技术将在后续章节为读者详细讲解),代码如下:

```
#资源包\Code\chapter5\5.4\8\djangoProject\templates\index.html
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <form action="{% url 'post' %}" method="post">
            <!-- CSRF 防御的相关知识点后续章节将为读者详细讲解 -->
            {% csrf_token %}
            <input type="submit" value="模拟 POST 提交">
        </form>
    </body>
</html>
```

(4) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.4\8\djangoProject\book\views.py
from django.http import HttpResponse
from django.shortcuts import render
from django.views.decorators.http import require_GET, require_http_methods
def index(request):
   return render(request, 'index.html')
#@require_GET 等价于@require_http_methods(['GET'])
@require_GET
def post_test(request):
   return HttpResponse("通过 POST 方法访问!")
```

(5) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.4\8\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
   path('', views.index),
   path('post_test/', views.post_test, name='post'),
]
```

(6) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-46 所示。

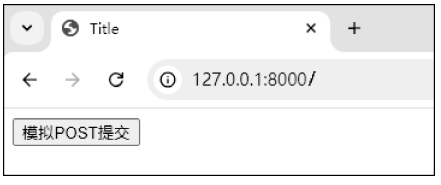


图 5-46 浏览器中的显示内容

(7) 单击“模拟 POST 提交”按钮,其显示内容如图 5-47 所示,即 HTTP 状态码 405,表示请求中指定的方法不被允许。

图 5-47 浏览器中的显示内容

5.5 模板

在 Django 中，模板是用于帮助开发者快速生成前端页面的工具，可以大幅降低代码的复杂度和维护成本。

Django 中内置了一款模板引擎，即 DTL(Django Template Language)。除此之外，Django 还支持第三方模板引擎，如 Jinja2 等，但由于 DTL 可以和 Django 达到无缝衔接而不会产生不兼容的情况，因此建议广大读者优先学习该模板引擎。

5.5.1 渲染模板

在 Django 中提供了两种常用的渲染方式，即 render_to_string()函数和 render()函数。

1. render_to_string()函数

通过 django.template.loader 模块中的 render_to_string()函数进行模板的渲染，其语法格式如下：

```
render_to_string(template_name, context)
```

其中，参数 template_name 表示待渲染模板的名称，参数 context 表示模板中的变量，并且必须为字典类型。

2. render()函数

通过 django.shortcuts 模块中的 render()函数进行模板的渲染，其语法格式如下：

```
render(request, template_name, context)
```

其中，参数 request 表示 HttpRequest 对象，参数 template_name 表示待渲染模板的名称，参数 context 表示模板中的变量，并且必须为字典类型。

下面通过一个示例演示一下如何渲染模板。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 在项目根目录下的文件夹 templates 中创建模板文件 render_to_string.html，代码如下：

```
#资源包\Code\chapter5\5.5\1\djangoProject\templates\render_to_string.html
<!DOCTYPE html>
```

```
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <style>
            h1 {
                color: red;
            }
        </style>
    </head>
    <body>
        <h1>这是使用 render_to_string()函数渲染的页面</h1>
    </body>
</html>
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 myrender.html,代码如下:

```
#资源包\Code\chapter5\5.5\1\djangoProject\templates\myrender.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <style>
            h1 {
                color: green;
            }
        </style>
    </head>
    <body>
        <h1>这是使用 render()函数渲染的页面</h1>
    </body>
</html>
```

(4) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\1\djangoProject\DjangoProject\urls.py
from django.contrib import admin
from django.urls import path
from django.template.loader import render_to_string
from django.shortcuts import render
from django.http import HttpResponse
def renderToString(request):
    html = render_to_string('render_to_string.html')
    return HttpResponse(html)
def myRender(request):
    return render(request, 'myrender.html')
urlpatterns = [
    path('admin/', admin.site.urls),
    path('rendertostring/', renderToString),
    path('myrender/', myRender),
]
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/rendertostring/,其显示内容如图 5-48 所示。

图 5-48 浏览器中的显示内容(1)

(6) 在浏览器中的网址栏输入 http://127.0.0.1:8000/myrender/,其显示内容如图 5-49 所示。

图 5-49 浏览器中的显示内容(2)

5.5.2 模板位置

在 Django 中,模板位置通过项目同名的应用中的配置文件 settings.py 的 TEMPLATES 配置中的 DIRS 进行设置,在默认情况下,模板文件会存放在项目根目录下的文件夹 templates 之中。

此外,通过将 TEMPLATES 配置中的 APP_DIRS 的值设置为 True,并将指定的应用添加到配置文件中的变量 INSTALLED_APPS 之中,即可完成应用注册操作,然后在指定的应用中创建文件夹 templates 并添加相应的模板文件,即可使模板引擎进入每个已注册的应用中查找模板。

模板查找的顺序,一是当 DIRS 中有模板路径,模板引擎会优先查找当前路径下的模板;二是在 APP_DIRS 的值为 True 时,模板引擎会开始查找已注册应用的模板。

下面通过一个示例,演示一下如何配置模板位置。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp front 和 python manage.py startapp cms,创建名为 front 和 cms 的应用,如图 5-50 所示。

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp front
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp cms
```

图 5-50 创建应用

(3) 分别在应用 front 和 cms 中创建文件夹 templates 和文件 urls.py,如图 5-51 所示。

(4) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

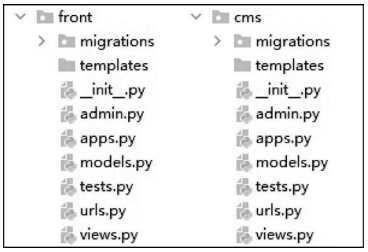


图 5-51　在应用中创建文件夹 templates 和文件 urls.py

```
#资源包\Code\chapter5\5.5\2\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <h1>这是网站的首页</h1>
    </body>
</html>
```

(5) 在应用 front 中的文件夹 templates 中创建模板文件 front_login.html,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\front\templates\front_login.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <h1>这是 Front 的登录页面</h1>
    </body>
</html>
```

(6) 在应用 cms 中的文件夹 templates 中创建模板文件 cms_login.html,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\cms\templates\cms_login.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <h1>这是 CMS 的登录页面</h1>
    </body>
</html>
```

(7) 打开应用 djangoProject 中的配置文件 settings.py,将应用 front 和 cms 添加到变量 INSTALLED_APPS 中,完成应用注册,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'front',
    'cms',
]
```

(8) 打开应用 front 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\front\views.py
from django.shortcuts import render
def login(request):
    return render(request, 'front_login.html')
```

(9) 打开应用 cms 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\cms\views.py
from django.shortcuts import render
def login(request):
    return render(request, 'cms_login.html')
```

(10) 打开应用 front 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\front\urls.py
from django.urls import path
from front import views
urlpatterns = [
    path('', views.login),
]
```

(11) 打开应用 cms 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\cms\urls.py
from django.urls import path
from cms import views
urlpatterns = [
    path('login/', views.login),
]
```

(12) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\2\djangoProject\djangoProject\urls.py
from django.urls import path, include
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
urlpatterns = [
    path('', index),
    path('login/', include('front.urls')),
    path('cms/', include('cms.urls'))
]
```

(13) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-52 所示。

(14) 在浏览器中的网址栏输入 http://127.0.0.1:8000/login/,其显示内容如图 5-53 所示。

图 5-52 浏览器中的显示内容(1)

图 5-53 浏览器中的显示内容(2)

(15) 在浏览器中的网址栏输入 http://127.0.0.1:8000/cms/login/,其显示内容如图 5-54 所示。

图 5-54 浏览器中的显示内容(3)

5.5.3 模板变量

在模板中可以通过标签{{ variable_name }}来获取 render()函数所传递的参数,并在渲染模板时将其解析成对应的值。

下面通过一个示例演示一下如何使用模板变量。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\3\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
class Course():
    def __init__(self, name, teacher):
        self.name = name
        self.teacher = teacher
def index(request):
    course = Course('Django', 'oldxia')
    context = {
        #字符串
        'username': 'oldxia',
        #字典
        'person': {
            'username': 'oldxia',
            'age': 38,
        },
```

```
        #类
        'course': course,
        #列表
        'language': ['Python', 'Java', 'PHP'],
    }
    return render(request, 'index.html', context = context)
urlpatterns = [
    path('', index),
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\3\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset = "UTF-8">
        <title>Title</title>
    </head>
    <body>
        <h1>{{ username }}</h1>
        <h1>{{ person.age }}</h1>
        <h1>{{ course.name }}</h1>
        <h1>{{ language.0 }}</h1>
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-55 所示。

图 5-55 浏览器中的显示内容

5.5.4 模板中的控制结构

Django 提供了多种控制结构,用于改变模板的渲染流程。

1. 选择结构

1) 单分支选择结构

通过标签{% if %}…{% endif %}来实现单分支选择结构。

2）双分支选择结构

通过标签{% if %}…{% else %}…{% endif %}来实现双分支选择结构。

3）多分支选择结构

通过标签{% if %}…{% elif %}…{% else %}…{% endif %}来实现多分支选择结构。

4）选择结构嵌套

对上述3种选择结构进行相互嵌套，即可用于表达更加复杂的选择结构。

下面通过一个示例演示一下如何使用选择结构。

（1）创建名为djangoProject的Django项目。

（2）打开应用djangoProject中的文件urls.py，代码如下：

```
#资源包\Code\chapter5\5.5\4\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
#单分支选择结构
def single_select(request):
    context = {
        'username': 'oldxia',
    }
    return render(request, 'single_select.html', context = context)
#双分支选择结构
def dual_select(request):
    context = {
        'username': 'xzd',
    }
    return render(request, 'dual_select.html', context = context)
#多分支选择结构
def multi_select(request):
    context = {
        'score': 95,
    }
    return render(request, 'multi_select.html', context = context)
#选择结构嵌套
def nested_select(request):
    context = {
        'score': 36,
    }
    return render(request, 'nested_select.html', context = context)
urlpatterns = [
    path('single_select/', single_select),
    path('dual_select/', dual_select),
    path('multi_select/', multi_select),
    path('nested_select/', nested_select),
]
```

（3）在项目根目录下的文件夹templates中创建模板文件single_select.html，代码如下：

```
#资源包\Code\chapter5\5.5\4\djangoProject\templates\single_select.html
<!DOCTYPE html>
<html lang = "en">
```

```
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% if username == 'oldxia' %}
            <h1>欢迎用户{{ username }}登录网站</h1>
        {% endif %}
    </body>
</html>
```

(4) 在项目根目录下的文件夹 templates 中创建模板文件 dual_select.html,代码如下:

```
#资源包\Code\chapter5\5.5\4\djangoProject\templates\dual_select.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% if username == 'oldxia' %}
            <h1>欢迎用户{{ username }}登录网站</h1>
        {% else %}
            <h1>用户{{ username }}不是本网站用户,无法登录!</h1>
        {% endif %}
    </body>
</html>
```

(5) 在项目根目录下的文件夹 templates 中创建模板文件 multi_select.html,代码如下:

```
#资源包\Code\chapter5\5.5\4\djangoProject\templates\multi_select.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% if score >= 90 %}
            <h1>您的分数{{ score }}分,成绩优秀</h1>
        {% elif score >= 70 %}
            <h1>您的分数{{ score }}分,成绩良好</h1>
        {% elif score >= 60 %}
            <h1>您的分数{{ score }}分,成绩及格</h1>
        {% else %}
            <h1>您的分数{{ score }}分,成绩不及格</h1>
        {% endif %}
    </body>
</html>
```

(6) 在项目根目录下的文件夹 templates 中创建模板文件 nested_select.html,代码如下:

```
#资源包\Code\chapter5\5.5\4\djangoProject\templates\nested_select.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% if score >= 60 %}
            <h1>您的分数{{ score }}分,考试及格</h1>
        {% else %}
            {% if score >= 45 %}
                <h1>您的分数{{ score }}分,考试不及格,但可以参加补考</h1>
            {% else %}
                <h1>您的分数{{ score }}分,考试不及格,并且不可以参加补考</h1>
            {% endif %}
        {% endif %}
    </body>
</html>
```

(7) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/single_select/,其显示内容如图 5-56 所示。

图 5-56 浏览器中的显示内容(1)

(8) 在浏览器中的网址栏输入 http://127.0.0.1:8000/dual_select/,其显示内容如图 5-57 所示。

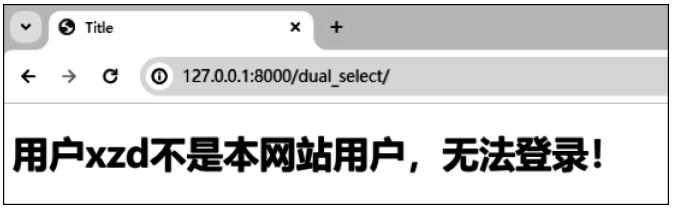


图 5-57 浏览器中的显示内容(2)

(9) 在浏览器中的网址栏输入 http://127.0.0.1:8000/multi_select/,其显示内容如图 5-58 所示。

(10) 在浏览器中的网址栏输入 http://127.0.0.1:8000/nested_select/,其显示内容如图 5-59 所示。

2. 循环结构

在模板中,可以通过标签{% for %}…{% endfor %}、标签{% for %}…{% else %}…{% endfor %}或{% for %}…{% empty %}…{% endfor %}来实现循环结构。

图 5-58 浏览器中的显示内容(3)

图 5-59 浏览器中的显示内容(4)

此外，在循环结构中还包含多个内置的循环变量，主要用于获取当前遍历的状态，具体如表 5-5 所示。

表 5-5 内置循环变量

内置循环变量	描 述
forloop.counter	当前迭代的索引，开始的索引从 1 开始
forloop.counter0	当前迭代的索引，开始的索引从 0 开始
forloop.revcounter	当前迭代的反向索引，最后一个索引从 1 开始
forloop.revcounter0	当前迭代的反向索引，最后一个索引从 0 开始
forloop.first	是否是第 1 次迭代，返回值为 True 或 False
loop.last	是否是最后一次迭代，返回值为 True 或 False
loop.length	序列的长度

下面通过一个示例演示一下如何使用循环结构。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.5\5\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    context = {
        'books': [
            {
                "name": "《Python 全栈开发——基础入门》",
                "author": "夏正东",
                "price": 79
            },
            {
                "name": "《Python 全栈开发——高阶编程》",
                "author": "夏正东",
                "price": 89
```

```
            },
            {
                "name": "《Python 全栈开发——数据分析》",
                "author": "夏正东",
                "price": 79
            },
            {
                "name": "《Python 全栈开发——Web 编程》",
                "author": "夏正东",
                "price": "待定"
            }
        ]
    }
    return render(request, 'index.html', context = context)
urlpatterns = [
    path('', index)
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\5\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset = "UTF-8">
        <title>Title</title>
    </head>
    <body>
        <table border = "1" cellspacing = "0">
            <tr>
                <th>序号 0</th>
                <th>序号 1</th>
                <th>书名</th>
                <th>作者</th>
                <th>价格</th>
            </tr>
            {% for book in books %}
                {% if forloop.first %}
                    <tr style = "background: yellow">
                {% elif forloop.last %}
                    <tr style = "background: greenyellow">
                {% else %}
                    <tr>
                {% endif %}
                    <td>{{ forloop.counter0 }}</td>
                    <td>{{ forloop.counter }}</td>
                    <td>{{ book.name }}</td>
                    <td>{{ book.author }}</td>
                    <td>{{ book.price }}</td>
                </tr>
            {% endfor %}
        </table>
        <ul>
            {% for comment in comments %}
                <li>comment</li>
```

```
            {% empty %}
                <li>没有任何评论!</li>
            {% endfor %}
        </ul>
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-60 所示。

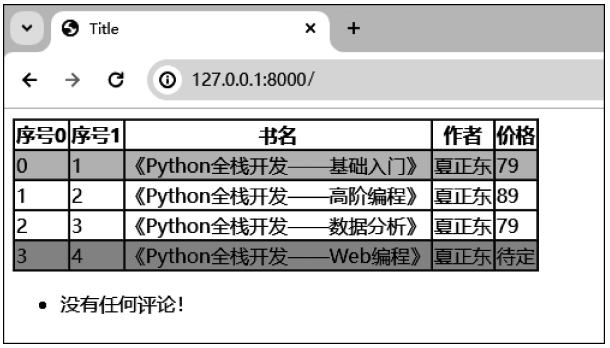


图 5-60 浏览器中的显示内容

5.5.5 模板注释

在模板中,可以通过标签{# … #}添加注释,需要注意的是模板注释不会出现在 HTML 文档之中。

下面通过一个示例演示一下如何使用模板注释。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\6\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
urlpatterns = [
    path('', index)
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\6\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
```

```
        {# 这是模板注释 #}
        <h1>模板注释</h1>
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-61 所示。

图 5-61 浏览器中的显示内容

5.5.6 常用标签

1. with 标签

标签{% with… %}…{% endwith %}用于定义作用域在该标签内的模板变量。

下面通过一个示例演示一下如何使用 with 标签。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
# 资源包\Code\chapter5\5.5\7\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
urlpatterns = [
    path('', index)
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
# 资源包\Code\chapter5\5.5\7\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% with website="http://www.oldxia.com" %}
            <p>老夏学院的网址:{{ website }}</p>
        {% endwith %}
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-62 所示。

图 5-62 浏览器中的显示内容

2. url 标签

标签{% url…%}用于动态构建请求 URL。

下面通过一个示例演示一下如何使用 url 标签。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\8\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
def login(request, username, course):
    if username:
        context = {
            'username': username,
            'course': course,
        }
        return render(request, 'login.html', context = context)
urlpatterns = [
    path('', index),
    path('login/<username>/<course>', login, name = 'login')
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\8\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset = "UTF-8">
        <title>Title</title>
    </head>
    <body>
        <h1><a href = "{% url 'login' username = 'oldxia' course = 'Django' %}">单击此处进行
登录</a></h1>
    </body>
</html>
```

(4) 在项目根目录下的文件夹 templates 中创建模板文件 login.html,代码如下:

```
#资源包\Code\chapter5\5.5\8\djangoProject\templates\login.html
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset = "UTF-8">
```

```
        <title>Title</title>
    </head>
    <body>
        <h1>欢迎{{ username }}登录本网站,您学习的课程内容是{{ course }}</h1>
    </body>
</html>
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-63 所示。

图 5-63 浏览器中的显示内容(1)

(6) 单击“单击此处进行登录”按钮,页面将跳转至动态构建的请求 URL,其显示内容如图 5-64 所示。

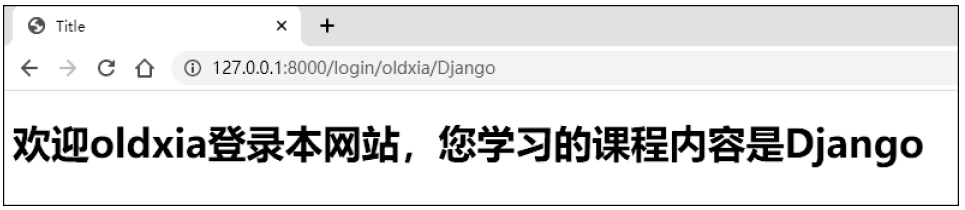


图 5-64 浏览器中的显示内容(2)

3. spaceless 标签

标签{% spaceless %}…{% endspaceless %}用于移除 HTML 中标签与标签之间的空白符。

下面通过一个示例演示一下如何使用 spaceless 标签。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\9\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
urlpatterns = [
    path('', index)
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\9\djangoProject\templates\index.html
<!DOCTYPE html>
```

```
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% spaceless %}
            <p>
                <a href="">http://www.oldxia.com</a>
            </p>
        {% endspaceless %}
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,然后在当前页面右击,选择“查看网页源代码”,可以看到 p 标签和 a 标签之间的空白符已经被去除,其显示内容如图 5-65 所示。

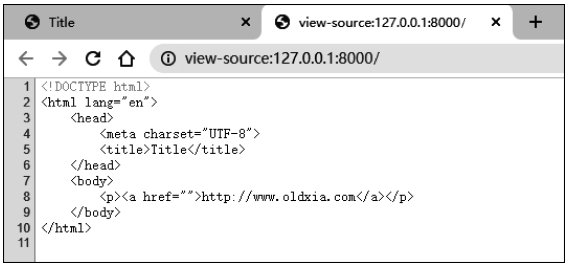


图 5-65 浏览器中的显示内容

4. autoescape 标签

标签{% autoescape on[off] %}…{% endautoescape %}用于开启或关闭 HTML 标签的自动转义功能。

下面通过一个示例演示一下如何使用 autoescape 标签。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\10\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    context = {
        'website': "<a href='http://www.oldxia.com'>老夏学院</a>"
    }
    return render(request, 'index.html', context=context)
urlpatterns = [
    path('', index)
]
```

(3) 在项目根目录中的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\10\djangoProject\templates\index.html
<!DOCTYPE html>
```

```
<html lang = "en">
    <head>
        <meta charset = "UTF - 8">
        <title> Title </title>
    </head>
    <body>
        { % autoescape off % }
            <h1>{{ website }}</h1>
        { % endautoescape % }
    </body>
</html>
```

图 5-66 浏览器中的显示内容

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-66 所示。

5. verbatim 标签

标签{% verbatim %}…{% endverbatim %}用于关闭 DTL 模板引擎的解析功能。

下面通过一个示例演示一下如何使用 verbatim 标签。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
# 资源包\Code\chapter5\5.5\11\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
   context = {
      'website': "<a href = 'http://www.oldxia.com'>老夏学院</a>"
   }
   return render(request, 'index.html', context = context)
urlpatterns = [
   path('', index)
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
# 资源包\Code\chapter5\5.5\11\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang = "en">
    <head>
        <meta charset = "UTF - 8">
        <title> Title </title>
    </head>
    <body>
        { % verbatim % }
            <h1>{{ website }}</h1>
        { % endverbatim % }
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容

如图 5-67 所示。

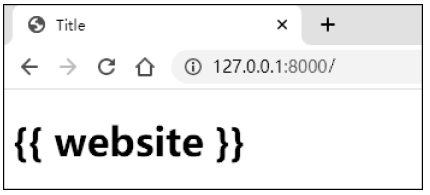


图 5-67 浏览器中的显示内容

6. include 标签

标签{% include… %}用于将一个模板引入另外一个模板中的指定位置。

下面通过一个示例演示一下如何使用 include 标签。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\12\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
urlpatterns = [
    path('', index)
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\12\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        {% include "header.html" %}
        <form action="">
            名称:<input type="text" name="user"><br><br>
            密码:<input type="password" name="password"><br><br>
            <input type="submit" name="submit" value="提交">
            <input type="reset" name="reset" value="重置">
        </form>
        {% include "footer.html" %}
    </body>
</html>
```

(4) 在项目根目录下的文件夹 templates 中创建模板文件 header.html,代码如下:

```
#资源包\Code\chapter5\5.5\12\djangoProject\templates\header.html
<h1 style="background: red">这是 header</h1>
```

(5) 在项目根目录下的文件夹 templates 中创建模板文件 footer.html,代码如下:

```
#资源包\Code\chapter5\5.5\12\djangoProject\templates\footer.html
<h1 style="background: green">这是 footer</h1>
```

(6) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-68 所示。

图 5-68　浏览器中的显示内容

5.5.7　模板中的过滤器

在模板中，过滤器相当于一个特殊的函数，主要用于修改和过滤变量的值。过滤器通过管道符号“|”实现。

1. 内置过滤器

在 DTL 中，内置了许多过滤器，如表 5-6 所示。

表 5-6　常用的内置过滤器

过　滤　器	描　　述
add	将传递的参数与原值相加
cut	将原值中所有指定的字符串移除
date	将日期按照指定的格式格式化为字符串
default	为原值设置默认值
default_if_none	当原值为 None 时，为其设置默认值
first	返回序列中的第 1 个元素
last	返回序列中的最后一个元素
floatformat	对浮点数进行格式化
join	对原值与传递的参数进行拼接
length	返回序列或字典的长度
lower	将原值中的字符转换为小写
upper	将原值中的字符转换为大写
random	随机返回序列中的值
safe	关闭字符串的自动转义
slice	对原值进行切片操作
striptags	去除原值中的 HTML 标签
truncatechars	按照指定长度对原值进行截断，并在末尾拼接 3 个点
truncatechars_html	按照指定长度对原值中的非 HTML 标签部分进行截断，并在末尾拼接 3 个点

下面通过一个示例演示一下如何使用内置过滤器。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\13\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
from datetime import datetime
def index(request):
    context = {
        'add_filter': 'django',
        'cut_filter': 'd-j-a-n-g-o',
        'date_filter': datetime.now(),
        'default_filter': '',
        'default_if_none_filter': None,
        'first_filter': ['Python', 'Java', 'Go'],
        'last_filter': ['Python', 'Java', 'Go'],
        'floatformat_filter': 100.656,
        'join_filter': ['Django', 'Flask'],
        'length_filter': ['Django', 'Flask'],
        'lower_filter': 'Hello Django',
        'upper_filter': 'hello django',
        'random_filter': [1, 2, 3, 4, 5, 6, 7],
        'safe_filter': "<a href='http://www.oldxia.com'>老夏学院</a>",
        'slice_filter': 'www.oldxia.com',
        'striptags_filter': '<h1>老夏学院</h1>',
        'truncatechars_filter': '<h1>Python-Django</h1>',
        'truncatechars_htmls_filter': '<h1>Python-Django</h1>',
    }
    return render(request, 'index.html', context=context)
urlpatterns = [
    path('', index),
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\13\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <!-- 注意:过滤器后的冒号前后不能有空格! -->
        <p>add: {{add_filter|add:'+Python'}}</p>
        <p>cut: {{cut_filter|cut:'-'}}</p>
        <p>date: {{date_filter|date:'Y-m-d h:i:s'}}</p>
        <p>default: {{default_filter|default:'原值为空值,使用默认值'}}</p>
        <p>default_if_none: {{default_if_none_filter|default:'原值为 None,使用默认值'}}</p>
        <p>first: {{first_filter|first}}</p>
        <p>last: {{last_filter|last}}</p>
        <p>floatformat: {{floatformat_filter|floatformat}}</p>
        <p>floatformat: {{floatformat_filter|floatformat:2}}</p>
        <p>join: {{join_filter|join:'-'}}</p>
        <p>length: {{length_filter|length}}</p>
        <p>lower: {{lower_filter|lower}}</p>
```

```
        <p>upper: {{upper_filter|upper}}</p>
        <p>random: {{random_filter|random}}</p>
        <p>safe: {{safe_filter|safe}}</p>
        <p>slice: {{slice_filter|slice:'4:10'}}</p>
        <p>striptags: {{striptags_filter|striptags}}</p>
        <p>truncatechars: {{truncatechars_filter|truncatechars:5}}</p>
        <p>truncatechars_html: {{truncatechars_htmls_filter|truncatechars_html:5}}</p>
    </body>
</html>
```

(4) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-69 所示。

图 5-69 浏览器中的显示内容

2. 自定义过滤器

在实际的开发过程中,内置过滤器往往无法满足一些特殊需求,因此 DTL 还支持自定义过滤器。

自定义过滤器的本质就是自定义一个函数,其创建步骤可以分为 4 步。

(1) 在应用中创建一个 Python 包 templatetags(该包名称不可以进行自定义)。

(2) 首先在 templatetags 包中创建自定义过滤器文件,并在该文件中编写自定义过滤器函数,然后使用 Library 对象的 filter()方法对自定义过滤器进行注册。

(3) 将当前应用添加到配置文件中的变量 INSTALLED_APPS 之中,完成应用注册操作。

(4) 在模板中使用标签{% load… %}加载自定义过滤器。

下面通过一个示例演示一下如何自定义过滤器。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-70 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-70 创建应用

(3) 在应用 book 中创建 Python 包 templatetags,如图 5-71 所示。

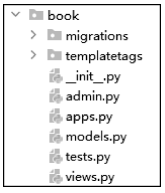


图 5-71 在应用中创建包 templatetags

(4) 在包 templatetags 中创建文件 my_filter.py,代码如下:

```
#资源包\Code\chapter5\5.5\14\djangoProject\book\templatetags\my_filter.py
from django import template
#自定义过滤器函数,该函数最多只能拥有两个参数
def add_filter(value, word = None):
    return value + ' + ' + word
#注册自定义过滤器
register = template.Library()
#filter(name, filter_func),其中,参数 name 表示自定义过滤器名称; 参数 filter_func 表示自定
#义过滤器函数
register.filter('add_filter', add_filter)
```

(5) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.5\14\djangoProject\book\views.py
from django.shortcuts import render
def index(request):
    context = {
        'course': 'Python'
    }
    return render(request, 'index.html', context = context)
```

(6) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\14\djangoProject\djangoProject\urls.py
```

```
from django.urls import path
from book import views
urlpatterns = [
    path('book/', views.index),
]
```

(7) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\14\djangoProject\templates\index.html
<!-- 加载自定义过滤器 -->
{% load my_filter %}
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        自定义过滤器 add_filter: {{ course|add_filter:'Django' }}
    </body>
</html>
```

(8) 打开应用 djangoProject 中的配置文件 settings.py,将应用 book 添加到变量 INSTALLED_APPS 中,完成应用注册,代码如下:

```
#资源包\Code\chapter5\5.5\14\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'book',
]
```

(9) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/book/,其显示内容如图 5-72 所示。

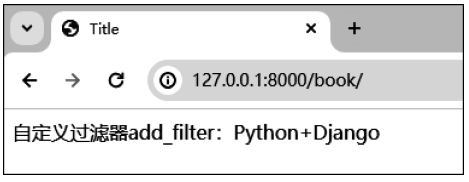


图 5-72 浏览器中的显示内容

5.5.8 模板继承

模板继承是 DTL 的重要特性之一。通过模板继承,可以将模板中重复出现的元素提取出来,并存放在一个已定义的父模板之中,进而达到避免重复编写代码的目的。

模板继承可以分为 3 步。

(1) 在父模板中使用标签{% block…%}…{% endblock %}保存 Web 页面中的常用元素。

(2) 在子模板中使用标签{% extends… %}继承父模板。

(3) 在子模板中使用标签{% block… %}…{% endblock %}将子模板中的内容插入父模板中,需要注意的是,子模板标签{% block…%}…{% endblock %}中的内容会覆盖父模板中的内容。

此外还需要注意两点,一是如果需要保留父模板标签{% block…%}…{% endblock %}中的内容,则需要在子模板中的标签{% block…%}…{% endblock %}内使用标签{{ block.super }};二是子模板中的标签{% block… %}…{% endblock %}之外的代码将不会被模板引擎渲染。

下面通过一个示例演示一下如何使用模板继承。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\15\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
def index(request):
   return render(request, 'index.html')
urlpatterns = [
   path('', index),
]
```

(3) 在项目根目录下的文件夹 templates 中创建模板文件 base.html,代码如下:

```
#资源包\Code\chapter5\5.5\15\djangoProject\templates\base.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}老夏学院{% endblock %}</title>
    </head>
    <body>
        {% block head %}
            <h3>网站顶部内容(父模板)</h3>
        {% endblock %}
        {% block main %}
            <h1>网站主体部分(父模板)</h1>
        {% endblock %}
        {% block footer %}
            <h3>网站底部内容(父模板)</h3>
        {% endblock %}
    </body>
</html>
```

(4) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\15\djangoProject\templates\index.html
{% extends "base.html" %}
{% block title %}
```

```
    老夏学院 - 首页
{% endblock %}
{% block main %}
    {{ block.super }}
    <h1 style = "background: greenyellow">网站主题内容(子模板 index)</h1>
{% endblock %}
<!-- 此部分内容不会显示 -->
老夏学院: http://www.oldxia.com
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-73 所示。

图 5-73 浏览器中的显示内容

5.5.9 加载静态文件

常用的静态文件主要包括 CSS 文件、JavaScript 脚本和图片等。

在 DTL 中,主要通过 static 标签进行静态文件的加载,其步骤可以分为 5 步。

(1) 确保项目同名的应用中的配置文件中的变量 INSTALLED_APPS 包含值 django.contrib.staticfiles。

(2) 确保项目同名的应用中的配置文件中的变量 STATIC_URL 的值不为空,其值为静态文件的访问路径。

(3) 在项目同名的应用中的配置文件中添加变量 STATICFILES_DIRS,其值为存放静态文件的路径,其可以使 DTL 优先在该路径下查找静态文件。

(4) 在项目根目录下,或者在应用所在的目录下创建文件夹 static,用于存放静态文件。

(5) 首先在模板中使用标签{% load… %}加载 static 标签,然后在需要加载静态文件的地方使用标签{% static… %}即可。

此外,如不想每次在模板中加载静态文件时都使用 load 标签,则需要在项目同名的应用中的配置文件中的变量 TEMPLATES 中的 OPTIONS 添加值'builtins':['django.templatetags.static'],即将 static 标签设置为内置标签。

下面通过一个示例演示一下如何加载静态文件。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp front 和 python manage.py startapp cms,创建名为 front 和 cms 的应

用，如图 5-74 所示。

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp front
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp cms
```

图 5-74 创建应用

（3）在项目根目录下创建文件夹 static，以及在应用 front 和 cms 中创建文件夹 templates 和 static，如图 5-75 所示。

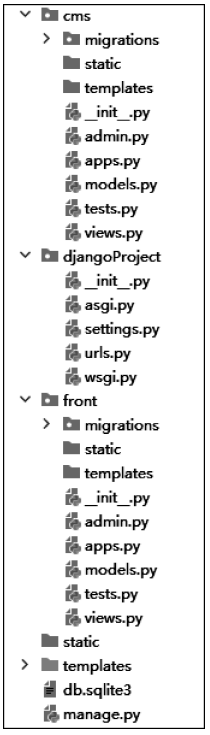


图 5-75 创建文件夹

（4）打开应用 djangoProject 中的配置文件 settings.py，完成以下 3 个操作，一是将应用 front 和 cms 添加到变量 INSTALLED_APPS 中；二是添加变量 STATICFILES_DIRS；三是给变量 TEMPLATES 中的 OPTIONS 添加值，代码如下：

```
#资源包\Code\chapter5\5.5\16\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'front',
    'cms',
]
TEMPLATES = [
```

```
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [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',
            ],
            'builtins':['django.templatetags.static']
        },
    },
]
STATICFILES_DIRS = [
     os.path.join(BASE_DIR,"static")
]
```

(5) 打开应用 front 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\front\views.py
from django.shortcuts import render
def front(request):
   return render(request, 'front_index.html')
```

(6) 打开应用 cms 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\cms\views.py
from django.shortcuts import render
def cms(request):
   return render(request, 'cms_index.html')
```

(7) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\djangoProject\urls.py
from django.urls import path
from django.shortcuts import render
from front import views as front_views
from cms import views as cms_views
def index(request):
   return render(request, 'index.html')
urlpatterns = [
   path('', index),
   path('front/', front_views.front),
   path('cms/', cms_views.cms),
]
```

(8) 在项目根目录下的文件夹 static 中创建文件 index.css,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\static\index.css
h1{
   font-size: 30px;
   font-weight: bold;
```

```
    color: red;
}
```

(9) 为了避免加载时产生混淆,在应用 front 的文件夹 static 中创建文件夹 front,并在其中创建文件 index.css,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\front\static\front\index.css
h1{
    font-size: 25px;
    font-weight: bold;
color: blue;
}
```

(10) 在应用 front 的文件夹 static 中的文件夹 front 内创建文件 index.js,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\front\static\front\index.js
alert("欢迎访问 front 页面!")
```

(11) 为了避免加载时产生混淆,在应用 cms 的文件夹 static 中创建文件夹 cms,并在其中创建文件 index.css,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\cms\static\cms\index.css
h1{
    font-size: 25px;
    font-weight: bold;
    color: green;
}
```

(12) 在应用 cms 的文件夹 static 中的文件夹 cms 内创建文件 index.js,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\cms\static\cms\index.js
alert("欢迎访问 cms 页面!")
```

(13) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <link rel="stylesheet" href="{% static 'index.css' %}">
    </head>
    <body>
        <h1>这是 index 页面</h1>
    </body>
</html>
```

(14) 在应用 front 的文件夹 templates 中创建模板文件 front_index.html,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\front\templates\front_index.html
<!DOCTYPE html>
<html lang="en">
```

```
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <link rel="stylesheet" href="{% static 'front/index.css' %}">
        <script src="{% static 'front/index.js' %}"></script>
    </head>
    <body>
        <h1>这是 front 页面</h1>
    </body>
</html>
```

(15) 在应用 cms 的文件夹 templates 中创建模板文件 cms_index.html,代码如下:

```
#资源包\Code\chapter5\5.5\16\djangoProject\cms\templates\cms_index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <link rel="stylesheet" href="{% static 'cms/index.css' %}">
        <script src="{% static 'cms/index.js' %}"></script>
    </head>
    <body>
        <h1>这是 cms 页面</h1>
    </body>
</html>
```

(16) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-76 所示。

图 5-76 浏览器中的显示内容(1)

(17) 在浏览器中的网址栏输入 http://127.0.0.1:8000/front/,其显示内容如图 5-77 所示。

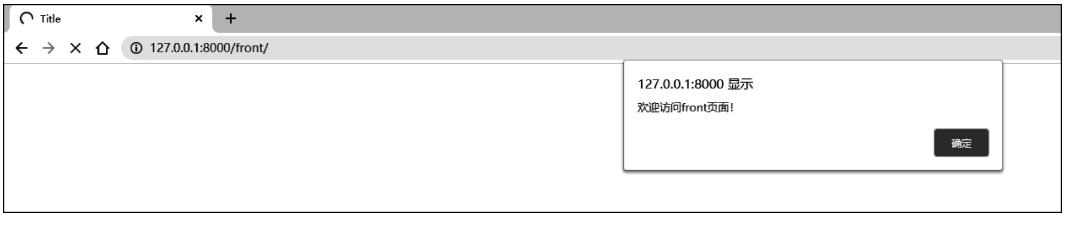


图 5-77 浏览器中的显示内容(2)

(18) 在浏览器中的网址栏输入 http://127.0.0.1:8000/cms/,其显示内容如图 5-78 所示。

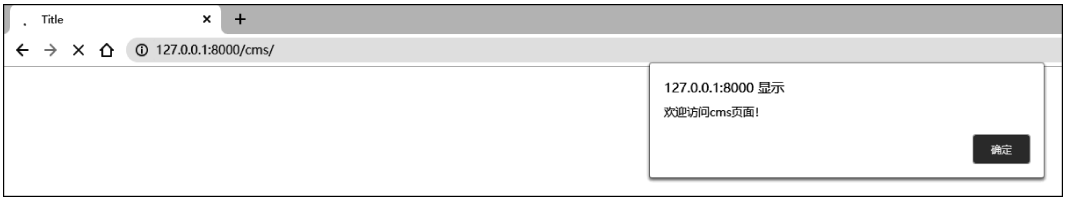


图 5-78 浏览器中的显示内容(3)

5.6 类视图

除视图函数之外,视图也可以基于类来实现,而类视图的好处就是支持继承,即首先可将共性的内容抽取出来放到父类中,然后在子类中完成各自的业务逻辑,并继承父类即可。

类视图是一个 Python 类,它继承自 django.views.generic 中的 View 类,并实现了 View 类中定义的方法。

类视图中具有 get()、post()、put()、delete()和 head()等方法,用于处理不同的 HTTP 请求,而当客户端发起的 HTTP 请求中使用了视图不支持的方法时,可以通过 http_method_not_allowed()方法进行处理。

此外,在配置类视图的路由时,不能直接传入该类的名称,而是需要使用类视图的 as_view()方法将类转换成可以为路由注册的视图函数。

下面通过一个示例演示一下如何使用类视图。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-79 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-79 创建应用

(3) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.6\1\djangoProject\templates\index.html
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <form action="" method="post">
            {% csrf_token %}
            <p>作者名称:<input type="text" name="book_author"></p>
            <input type="submit" value="提交">
        </form>
    </body>
</html>
```

(4) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.6\1\djangoProject\book\views.py
from django.http import HttpResponse
from django.shortcuts import render
from django.views.generic import View
class BookDetailView(View):
    def get(self, request):
        return render(request, "index.html")
    def post(self, request):
        book_author = request.POST.get('book_author')
        return HttpResponse(f'作者姓名：{book_author}')
class BookListView(View):
    def post(self, request):
        return HttpResponse('这是 POST 请求')
    def http_method_not_allowed(self, request):
        return HttpResponse('不支持非 POST 请求的请求方法!')
```

(5) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.6\1\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('bookdetail/', views.BookDetailView.as_view()),
    path('booklist/', views.BookListView.as_view()),
]
```

(6) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/bookdetail/,其显示内容如图 5-80 所示。

此时,输入作者名称“夏正东”,并单击“提交”按钮,其显示内容如图 5-81 所示。

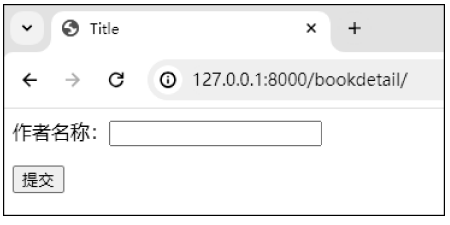


图 5-80 浏览器中的显示内容(1)

图 5-81 浏览器中的显示内容(2)

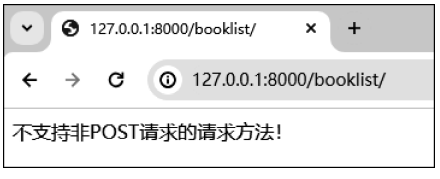


图 5-82 浏览器中的显示内容(3)

(7) 在浏览器中的网址栏输入 http://127.0.0.1:8000/booklist/,其显示内容如图 5-82 所示。

此外,在类视图中可以通过 django.utils.decorators 模块中的 method_decorator()方法将函数装饰器应用于类视图中的方法或类视图本身。

下面通过一个示例,演示一下如何给类视图添加装饰器。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-83 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-83 创建应用

(3) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.6\2\djangoProject\book\views.py
from django.views.generic import View
from django.shortcuts import redirect, reverse
from django.http import HttpResponse
from django.utils.decorators import method_decorator
def login_required(func):
    def wrapper(request):
        username = request.GET.get('username')
        if username:
            return func(request, username)
        else:
            return redirect(reverse('login'))
    return wrapper
#装饰类视图
@method_decorator(login_required, name = 'get')
class IndexView(View):
    #装饰类视图中的方法
    #@method_decorator(login_required)
    def get(self, request, username):
        return HttpResponse(f'这是登录之后的页面,欢迎用户{username}!')
def login(request):
    return HttpResponse('登录页面')
```

(4) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.6\2\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('', views.IndexView.as_view()),
    path('login/', views.login, name = 'login'),
]
```

(5) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/login/,其显示内容如图 5-84 所示。

(6) 在浏览器中的网址栏输入 http://127.0.0.1:8000/?username=oldxia,其显示内容如图 5-85 所示。

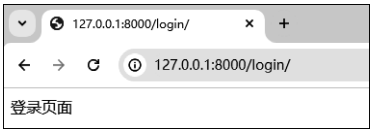


图 5-84 浏览器中的显示内容(1)

图 5-85 浏览器中的显示内容(2)

5.7 数据库

在 Django 中,可以通过 ORM 技术对数据库进行访问,其允许开发者使用 Python 对象来操作数据库,而不必直接编写 SQL 查询。

ORM技术将数据库中的表映射为一个Python类，将表中的一条数据映射为类的实例，将表的标签映射为类的属性，这使数据库的操作更容易、更直观。

5.7.1 定义数据模型

创建数据模型可以分为5步。

(1) 在项目同名的应用中的配置文件内，通过修改变量DATABASES的值，用于进行数据库的相关配置，以便达到连接数据库的目的。

(2) 在应用中的文件models.py中定义数据模型。

(3) 将定义数据模型的应用添加至项目同名的应用中的配置文件内的变量INSTALLED_APP。

(4) 打开命令提示符窗口，进入项目所在的目录中，激活虚拟环境，并执行命令python manage.py makemigrations，用于生成迁移脚本文件。

(5) 执行命令python manage.py migrate，用于将迁移的脚本文件映射到数据库中。

下面通过一个示例演示一下如何定义数据模型。

(1) 创建名为djangoProject的Django项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令python manage.py startapp book，创建名为book的应用，如图5-86所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图5-86 创建应用

(3) 打开应用djangoProject中的文件__init__.py，代码如下：

```
#资源包\Code\chapter5\5.7\1\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(4) 打开应用djangoProject中的配置文件settings.py，进行数据库的相关配置，用于连接数据库，代码如下：

```
#资源包\Code\chapter5\5.7\1\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接MySQL的用户名
        'USER': 'root',
        #连接MySQL的密码
        'PASSWORD': '12345678',
        #MySQL的主机地址
        'HOST': '127.0.0.1',
        #MySQL的端口号
        'PORT': '3306',
    }
}
```

(5) 打开应用 djangoProject 中的配置文件 settings.py,将应用 book 添加到变量 INSTALLED_APPS 中,用于完成应用注册,代码如下:

```
#资源包\Code\chapter5\5.7\1\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'book',
]
```

(6) 打开应用 book 中的文件 models.py,用于定义数据模型,代码如下:

```
#资源包\Code\chapter5\5.7\1\djangoProject\book\models.py
from django.db import models
class Book(models.Model):
    id = models.AutoField(primary_key = True)
    name = models.CharField(max_length = 20, null = False)
    author = models.CharField(max_length = 20, null = False)
    price = models.FloatField(null = False, default = 0)
```

(7) 打开命令提示符窗口,登入 MySQL。

(8) 在 MySQL 命令行窗口中输入 SQL 语句"create database django_db;",创建数据库 django_db,其结果如图 5-87 所示。

(9) 打开命令提示符,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py makemigrations,生成迁移脚本文件,如图 5-88 所示。

```
mysql> create database django_db;
Query OK, 1 row affected (0.00 sec)
```

图 5-87 创建数据库

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'book':
  book\migrations\0001_initial.py
    - Create model Book
```

图 5-88 生成迁移脚本文件

(10) 再次输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-89 所示。

(11) 在 MySQL 命令行窗口中输入 SQL 语句"show tables;",查询当前数据库中的数据表,其结果如图 5-90 所示。

```
Operations to perform:
  Apply all migrations: admin, auth, book, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying book.0001_initial... OK
  Applying sessions.0001_initial... OK
```

图 5-89 将迁移脚本文件映射到数据库中

```
mysql> show tables;
+----------------------------+
| Tables_in_django_db        |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| book_book                  |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
+----------------------------+
11 rows in set (0.00 sec)
```

图 5-90 查询数据表

其中，数据表 book_book 就是应用 book 中 Book 类映射到数据库中的数据表。

(12) 再次输入 SQL 语句“desc book_book;”，查询该数据表的表结构，其结果如图 5-91 所示。

```
mysql> desc book_book;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| name   | varchar(20) | NO   |     | NULL    |                |
| author | varchar(20) | NO   |     | NULL    |                |
| price  | double      | NO   |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

图 5-91　数据表 book_book 中的表结构

5.7.2　Manager 类和 QuerySet 类

在 Django 中，每个数据模型都有一个默认的 Manager 类(管理器类)的实例对象 objects，用于提供对数据库中相应表的操作接口，而 QuerySet 类则用于数据库的所有查询及更新交互，其允许开发人员以面向对象的方式操作数据库，并且其提供了丰富的方法和链式操作，使查询数据库变得非常方便和灵活。

此外，Manager 类本身不具有任何属性和方法，其方法均是通过 Python 动态添加的方式从 QuerySet 类中复制过来的，所以 Manager 类的绝大部分方法基于 QuerySet 类，并且一个 QuerySet 类可以包含一个或多个数据模型对象。

QuerySet 类的常用方法如表 5-7 所示。

表 5-7　QuerySet 类的常用方法

方　法	描　述
filter()	返回满足指定条件的数据
exclude()	返回不满足指定条件的数据
get()	返回满足指定条件的唯一数据
all()	返回数组模型中的所有数据
first()	返回第 1 条数据
last()	返回最后一条数据
distinct()	返回去重后的数据
annotate()	返回使用聚合函数进行操作后的数据，支持分组聚合
aggregate()	返回使用聚合函数进行操作后的数据
order_by()	返回按照指定字段进行排序的数据
reverse()	返回反向排序的数据
values()	返回给定字段的数据所组成的字典
values_list()	返回给定字段的数据所组成的元组
select_related()	根据外键关系(一对多或一对一)返回相关联数据模型中的数据
prefetch_related()	根据外键关系(多对多或多对一)返回相关联数据模型中的数据
defer()	返回除指定字段之外的所有数据
only()	返回指定字段所对应的数据
create()	创建一条数据
get_or_create()	创建或返回一条数据
bulk_create()	创建多条数据

续表

方　法	描　述
count()	返回数据的数量
exists()	判断指定条件的数据是否存在
update()	根据指定条件更新数据
delete()	根据指定条件删除数据

除此之外，如果想获取部分数据，则可以对 QuerySet 对象使用切片操作，即相当于在数据库层面使用 LIMIT 和 OFFSET 操作。

5.7.3 查询条件

查询条件作为 QuerySet 类中方法的参数，用于在 Django 框架中进行数据库查询时对结果进行过滤的一种机制，其允许开发者根据特定的条件来筛选出符合要求的数据。

查询条件可以分为 4 大类，即查询过滤器、聚合函数、F 表达式和 Q 表达式。

1. 查询过滤器

查询过滤器通过“字段＋__＋查询过滤器”的方式实现，其常用的查询过滤器如表 5-8 所示。

表 5-8 常用的查询过滤器

查询过滤器	描　述
exact	根据指定字段的值进行精确匹配查询
iexact	根据指定字段的值进行精确匹配查询(不区分大小写)
contains	筛选出根据指定字段包含指定值的数据
icontains	筛选出根据指定字段包含指定值的数据(不区分大小写)
in	筛选出指定字段的值在给定列表中的数据
gt	筛选出指定字段的值大于指定值的数据
gte	筛选出指定字段的值大于或等于指定值的数据
lt	筛选出指定字段的值小于指定值的数据
lte	筛选出指定字段的值小于或等于指定值的数据
startswith	筛选出指定字段的值以指定值开头的数据
endswith	筛选出指定字段的值以指定值结尾的数据
range	筛选出指定字段的值在给定范围内的数据
date	筛选出指定字段的值为指定日期的数据
time	筛选出指定字段的值为指定时间的数据
year	筛选出指定字段的值为指定年份的数据
week_day	筛选出指定字段的值为指定星期的数据
isnull	筛选出指定字段的值为空值的数据
regex	根据指定字段的值进行正则表达式匹配

2. 聚合函数

在 Django 中，聚合函数通过 aggregate()方法或 annotate()方法实现，其常用的聚合函数如表 5-9 所示。

表 5-9 常用的聚合函数

聚合函数	描述
Count()	统计指定字段的个数
Sum()	求和
Max()	求最大值
Min()	求最小值
Avg()	求平均值

3. F 表达式

F 表达式是一个强大的工具，允许在查询表达式中引用模型的字段，从而在数据库层面执行各种操作，而无须将数据加载到 Python 内存中。这不仅可以提高性能，还允许利用数据库的优化功能，尤其是在处理大量数据时，可以显著地提高性能并减少资源消耗。

F 表达式的常用应用场景：字段引用，即在查询表达式中直接引用模型的字段；算术运算，即使用 F()函数执行算术运算，如加、减、乘、除等；条件表达式，F()函数可用于创建复杂的条件逻辑；更新字段值，即在更新操作中，F()函数可以用于基于现有字段值的更新；数据库函数调用，F()函数可以与数据库特定的函数结合使用；字段间运算，F()函数可以用于字段间的运算。

4. Q 表达式

Q 表达式是 Django 框架中用于构建复杂查询表达式的一种功能，其可以灵活地构建查询条件，从而实现更复杂的查询需求。

Q 表达式的应用场景非常广泛，例如在需要筛选多个条件的数据时，或者需要构建复杂的查询逻辑时。

下面通过一个示例演示一下如何进行数据的增、删、改、查等操作。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-92 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-92 创建应用

(3) 打开应用 djangoProject 中的文件__init__.py，代码如下：

```
#资源包\Code\chapter5\5.7\2\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(4) 打开应用 djangoProject 中的配置文件 settings.py，进行数据库的相关配置，用于连接数据库，代码如下：

```
#资源包\Code\chapter5\5.7\2\djangoProject\djangoProject\settings.py
DATABASES = {
   'default': {
      #数据库引擎
      'ENGINE': 'django.db.backends.mysql',
      #数据库名称
```

```
        'NAME': 'django_db',
        #连接 MySQL 的用户名
        'USER': 'root',
        #连接 MySQL 的密码
        'PASSWORD': '12345678',
        #MySQL 的主机地址
        'HOST': '127.0.0.1',
        #MySQL 的端口号
        'PORT': '3306',
    }
}
```

(5) 打开应用 djangoProject 中的配置文件 settings.py,将应用 book 添加到变量 INSTALLED_APPS 中,用于完成应用注册,代码如下:

```
#资源包\Code\chapter5\5.7\2\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'book',
]
```

(6) 打开应用 book 中的文件 models.py,用于定义数据模型,代码如下:

```
#资源包\Code\chapter5\5.7\2\djangoProject\book\models.py
from django.db import models
class Book(models.Model):
    name = models.CharField(max_length = 100)
    author = models.CharField(max_length = 20, null = False)
    price = models.FloatField(null = False, default = 0)
    publishing_time = models.DateTimeField(null = False)
```

(7) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.7\2\djangoProject\book\views.py
from book.models import Book
from django.http import HttpResponse
from datetime import datetime
from django.db.models import Avg, Count, Sum, Max, Min, F, Q
from django.db import connection
def add_one_data(request):
    Book.objects.create(name = '《Python 全栈开发——Web 编程》', author = '夏正东', price = '999',
publishing_time = datetime(year = 2024, month = 12, day = 1))
    return HttpResponse('已成功添加一册图书!')
def add_more_data(request):
    Book.objects.bulk_create([Book(name = '《Python 全栈开发——基础入门》', author = '夏正东',
price = 79, publishing_time = datetime(year = 2022, month = 7, day = 1)),
                              Book(name = '《Python 全栈开发——高阶编程》', author = '夏正东',
price = 89, publishing_time = datetime(year = 2022, month = 8, day = 1)),
```

```
                                    Book(name = '《Python 全栈开发——数据分析》', author = '夏正东',
price = 79, publishing_time = datetime(year = 2023, month = 2, day = 1))])
    return HttpResponse('已成功添加多册图书!')
def filter_exact(request):
    #在精确匹配查询时,id__exact = 1 等价于 id = 1
    books = Book.objects.filter(id__exact = 1)
    book = Book.objects.get(id = 1)
    return HttpResponse(f'查询过滤器 exact 等价的 SQL 语句为{books.query}< br >查询结果为
{book.name}')
def filter_contains(request):
    books = Book.objects.filter(name__contains = "Python")
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 contains 等价的 SQL 语句为{books.query}< br >查询结果为
{books_lt}')
def filter_gt(request):
    books = Book.objects.filter(price__gt = 100)
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 gt 等价的 SQL 语句为{books.query}< br >查询结果为{books_
lt}')
def filter_endswith(request):
    books = Book.objects.filter(name__endswith = "编程》")
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 endwith 等价的 SQL 语句为{books.query}< br >查询结果为
{books_lt}')
def filter_range(request):
    books = Book.objects.filter(price__range = (50, 100))
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 range 等价的 SQL 语句为{books.query}< br >查询结果为
{books_lt}')
def filter_date(request):
    books = Book.objects.filter(publishing_time__date = datetime(year = 2023, month = 2, day =
1))
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 date 等价的 SQL 语句为{books.query}< br >查询结果为
{books_lt}')
def filter_isnull(request):
    books = Book.objects.filter(publishing_time__isnull = True)
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 isnull 等价的 SQL 语句为{books.query}< br >查询结果为
{books_lt}')
def filter_regex(request):
    books = Book.objects.filter(name__regex = r"^《")
```

```
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'查询过滤器 regex 等价的 SQL 语句为{books.query}<br>查询结果为
{books_lt}')
def aggregate_avg(request):
    books = Book.objects.aggregate(price_avg = Avg("price"))
    return HttpResponse(f'聚合函数 Avg 等价的 SQL 语句为{connection.queries[-1]}<br>查询结
果为{books}')
def aggregate_count(request):
    books = Book.objects.aggregate(book_count = Count("id"))
    return HttpResponse(f'聚合函数 Count 等价的 SQL 语句为{connection.queries[-1]}<br>查询
结果为{books}')
def aggregate_max_min(request):
    books = Book.objects.aggregate(price_max = Max("price"), price_min = Min("price"))
    return HttpResponse(f'聚合函数 Max 和 Min 等价的 SQL 语句为{connection.queries[-1]}<br>
查询结果为{books}')
def aggregate_sum(request):
    books = Book.objects.aggregate(price_sum = Sum("price"))
    return HttpResponse(f'聚合函数 Sum 等价的 SQL 语句为{connection.queries[-1]}<br>查询结
果为{books}')
def expression_f(request):
    Book.objects.update(price = F("price") + 5)
    books = Book.objects.all().order_by('price')
    books_lt = []
    for book in books:
        books_lt.append(book.price)
    return HttpResponse(f'F 表达式等价的 SQL 语句为{connection.queries[-2]}<br>查询结果为
{books_lt}')
def expression_q(request):
    books = Book.objects.filter(Q(price__lte = 90) | Q(name__contains = "Web"))
    books_lt = []
    for book in books:
        books_lt.append(book.name)
    return HttpResponse(f'Q 表达式等价的 SQL 语句为{connection.queries[-1]}<br>查询结果为
{books_lt}')
def del_data(request):
    Book.objects.filter(name = '《Python 全栈开发——基础入门》').delete()
    return HttpResponse('删除书籍成功!')
```

(8) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.7\2\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('add_more_data/', views.add_more_data),
    path('add_one_data/', views.add_one_data),
    path('filter_exact/', views.filter_exact),
    path('filter_contains/', views.filter_contains),
    path('filter_gt/', views.filter_gt),
    path('filter_endswith/', views.filter_endswith),
    path('filter_range/', views.filter_range),
    path('filter_date/', views.filter_date),
    path('filter_isnull/', views.filter_isnull),
```

```
    path('filter_regex/', views.filter_regex),
    path('aggregate_avg/', views.aggregate_avg),
    path('aggregate_count/', views.aggregate_count),
    path('aggregate_max_min/', views.aggregate_max_min),
    path('aggregate_sum/', views.aggregate_sum),
    path('expression_f/', views.expression_f),
    path('expression_q/', views.expression_q),
    path('del_data/', views.del_data),
]
```

(9) 打开命令提示符窗口,登入 MySQL。

(10) 在 MySQL 命令行窗口中输入 SQL 语句“create database django_db;”,创建数据库 django_db,其结果如图 5-93 所示。

(11) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py makemigrations,生成迁移脚本文件,如图 5-94 所示。

```
mysql> create database django_db;
Query OK, 1 row affected (0.00 sec)
```

图 5-93 创建数据库

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'book':
  book\migrations\0001_initial.py
    - Create model Book
```

图 5-94 生成迁移脚本文件

(12) 再次输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-95 所示。

```
Operations to perform:
  Apply all migrations: admin, auth, book, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying book.0001_initial... OK
  Applying sessions.0001_initial... OK
```

图 5-95 将迁移脚本文件映射到数据库中

(13) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/add_more_data/,其显示内容如图 5-96 所示。

(14) 在浏览器中的网址栏输入 http://127.0.0.1:8000/add_one_data/,其显示内容如图 5-97 所示。

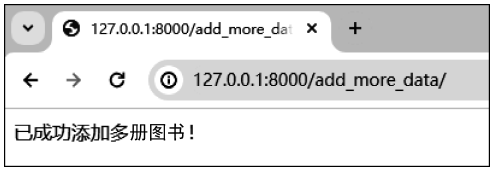


图 5-96 浏览器中的显示内容(1)

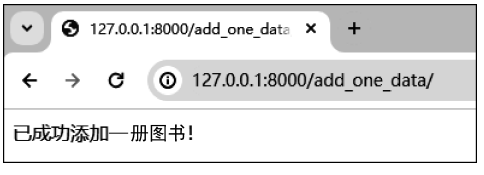


图 5-97 浏览器中的显示内容(2)

(15) 打开命令提示符窗口,登入 MySQL。

(16) 在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”,其结果如图 5-98 所示。

```
mysql> select * from book_book;
+----+-----------------------------------------+----------+-------+----------------------------+
| id | name                                    | author   | price | publishing_time            |
+----+-----------------------------------------+----------+-------+----------------------------+
|  1 | 《Python全栈开发——基础入门》            | 夏正东   |    79 | 2022-07-01 00:00:00.000000 |
|  2 | 《Python全栈开发——高阶编程》            | 夏正东   |    89 | 2022-08-01 00:00:00.000000 |
|  3 | 《Python全栈开发——数据分析》            | 夏正东   |    79 | 2023-02-01 00:00:00.000000 |
|  4 | 《Python全栈开发——Web编程》             | 夏正东   |   999 | 2024-12-01 00:00:00.000000 |
+----+-----------------------------------------+----------+-------+----------------------------+
4 rows in set (0.00 sec)
```

图 5-98 数据表 book_book 中的数据

(17) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_exact/,其显示内容如图 5-99 所示。

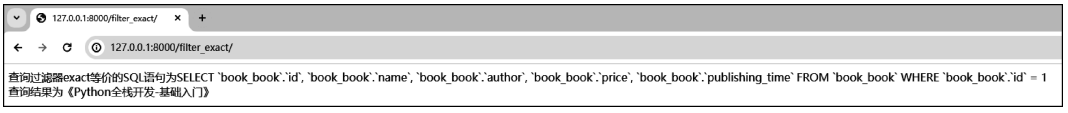


图 5-99 浏览器中的显示内容(1)

(18) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_contains/,其显示内容如图 5-100 所示。

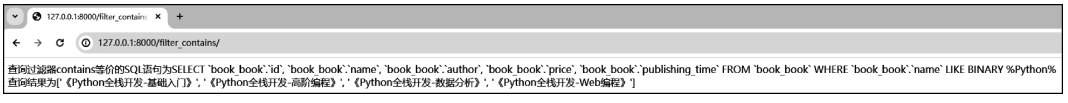


图 5-100 浏览器中的显示内容(2)

(19) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_gt/,其显示内容如图 5-101 所示。

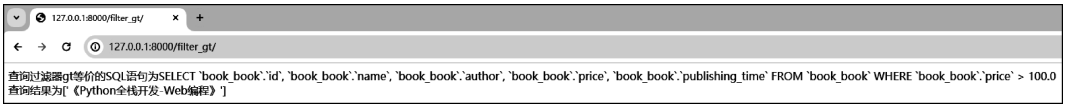


图 5-101 浏览器中的显示内容(3)

(20) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_endswith/,其显示内容如图 5-102 所示。

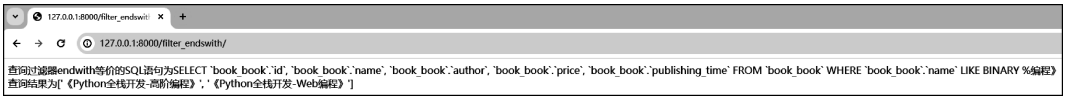


图 5-102 浏览器中的显示内容(4)

(21) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_range/,其显示内容如图 5-103 所示。

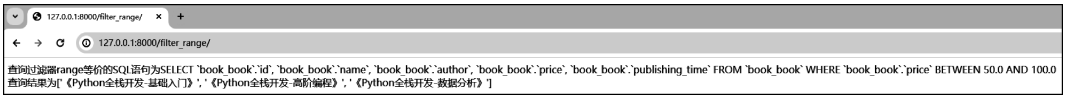


图 5-103 浏览器中的显示内容(5)

(22) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_date/,其显示内容如图 5-104 所示。

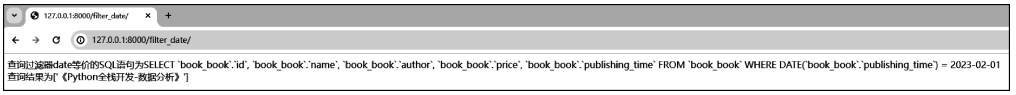


图 5-104 浏览器中的显示内容(6)

(23) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_isnull/,其显示内容如图 5-105 所示。

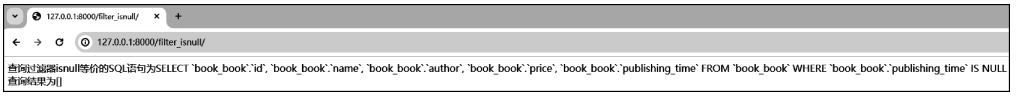


图 5-105 浏览器中的显示内容(7)

(24) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_regex/,其显示内容如图 5-106 所示。

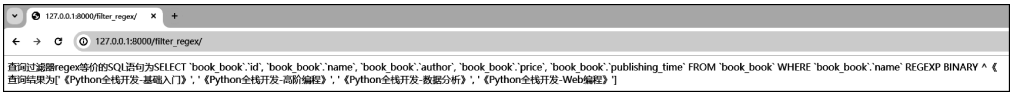


图 5-106 浏览器中的显示内容(8)

(25) 在浏览器中的网址栏输入 http://127.0.0.1:8000/aggregate_avg/,其显示内容如图 5-107 所示。

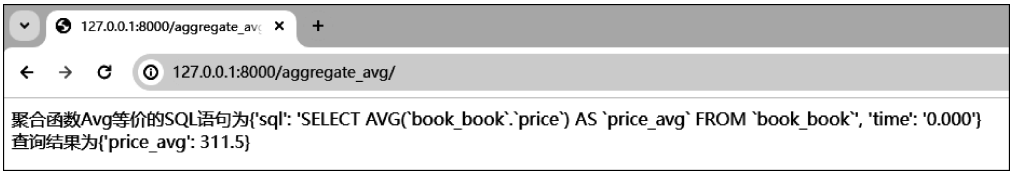


图 5-107 浏览器中的显示内容(9)

(26) 在浏览器中的网址栏输入 http://127.0.0.1:8000/aggregate_count/,其显示内容如图 5-108 所示。

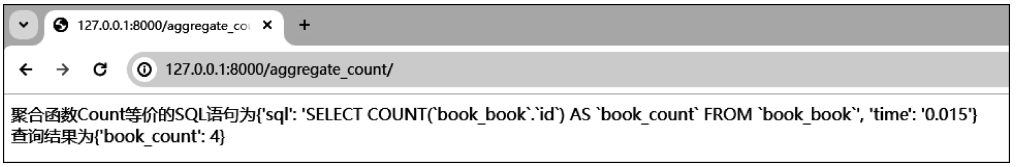


图 5-108 浏览器中的显示内容(10)

(27) 在浏览器中的网址栏输入 http://127.0.0.1:8000/aggregate_max_min/,其显示内容如图 5-109 所示。

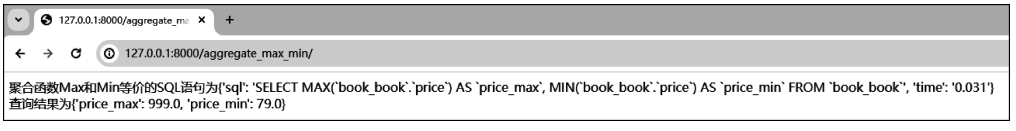


图 5-109 浏览器中的显示内容(11)

(28) 在浏览器中的网址栏输入 http://127.0.0.1:8000/aggregate_sum/,其显示内容如图 5-110 所示。

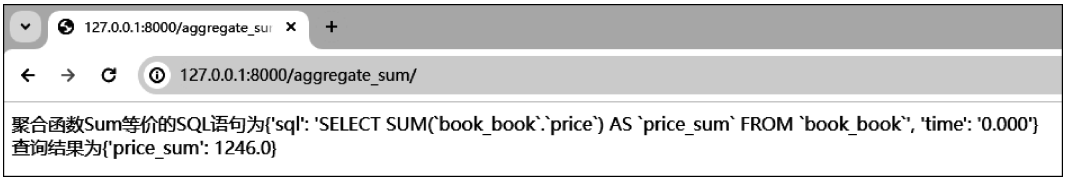


图 5-110 浏览器中的显示内容(12)

(29) 在浏览器中的网址栏输入 http://127.0.0.1:8000/expression_q/,其显示内容如图 5-111 所示。

图 5-111 浏览器中的显示内容(13)

(30) 在浏览器中的网址栏输入 http://127.0.0.1:8000/expression_f/,其显示内容如图 5-112 所示。

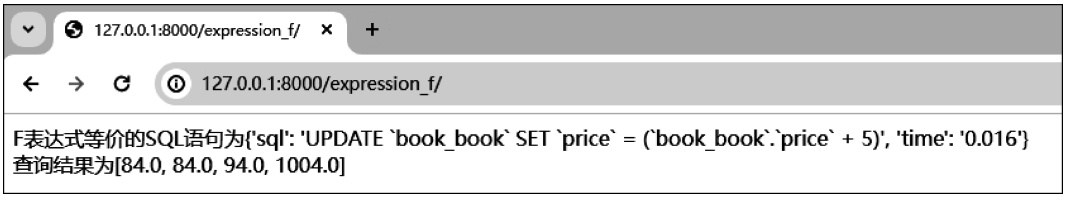


图 5-112 浏览器中的显示内容(14)

(31) 在浏览器中的网址栏输入 http://127.0.0.1:8000/del_data/,其显示内容如图 5-113 所示。

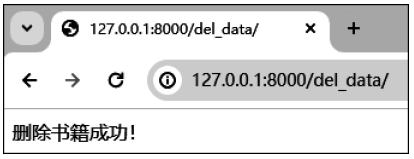
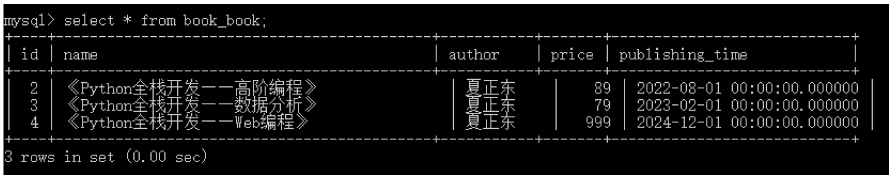


图 5-113 浏览器中的显示内容(15)

(32) 在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”,其结果如图 5-114 所示。



```
mysql> select * from book_book;
+----+-----------------------------------------+-----------+-------+---------------------+
| id | name                                    | author    | price | publishing_time     |
+----+-----------------------------------------+-----------+-------+---------------------+
|  2 | 《Python全栈开发——高阶编程》           | 夏正东    |    89 | 2022-08-01 00:00:00.000000 |
|  3 | 《Python全栈开发——数据分析》           | 夏正东    |    79 | 2023-02-01 00:00:00.000000 |
|  4 | 《Python全栈开发——Web编程》            | 夏正东    |   999 | 2024-12-01 00:00:00.000000 |
+----+-----------------------------------------+-----------+-------+---------------------+
3 rows in set (0.00 sec)
```

图 5-114 数据表 book_book 中的数据

5.7.4 常用字段

在 ORM 中，Python 类中的属性对应为数据表中的字段，而字段中的属性则可以通过各种字段类型进行创建，其常用的字段类型如表 5-10 所示。

表 5-10 常用的字段类型

字段类型	字段类型常用属性	描述
AutoField()	参数 null 用于设置在数据库中字段是否为空。 参数 blank 用于设置在表单验证时字段是否为空。 参数 db_column 用于设置字段是否为空。 参数 default 用于设置字段的默认值。 参数 primary_key 用于设置字段是否为主键。 参数 unique 用于设置字段的值是否唯一，即不允许有重复值	自动增长类型字段
BooleanField()		布尔类型字段
NullBooleanField()		允许为空的布尔类型字段
CharField()		字符串类型字段
DataField()		日期类型字段
DateTImeField()		日期和时间类型字段
TimeField()		时间类型字段
EmailField()		提供验证 Email 机制的字符串类型字段
ImageField()		图像类型字段
FileField()		文件类型字段
FloatField()		浮点数类型字段
IntegerField()		整数类型字段
BigIntegerField()		长整型字段
PositiveIntegerField()		正整型字段
SmallIntegerField()		短整型字段
PositiveSmallIntegerField()		正短整型字段
TextField()		文本类型字段
UUIDField()		提供验证 UUID 机制的字符串类型字段
URLField()		提供验证 URL 机制的字符串类型字段
ForeignKey		外键类型字段

5.7.5 Meta 类

每个数据模型类中都有一个子类 Meta，该类中封装了一些数据库信息，称为数据模型的元数据。

Django 会将 Meta 类中的元数据选项定义附加到数据模型中，常见的元数据定义有 db_table(数据表名称)、abstract(抽象类)和 ordering(字段排序)等，而 Meta 类作为内部类，它定义的元数据可以让 admin 管理后台更加友好，并且数据的可读性更高。

此外，由于 Meta 类定义的元数据相当于数据模型的配置信息，所以开发人员可以根据自己的需求选择性地进行添加，而当没有需求时也可以不定义 Meta 类，此时，Django 会应用默认的元数据。

5.7.6 外键

在 MySQL 中，外键可以让多表之间的关系更加紧密，而 Django 同样支持外键，其通过数据模型中的 ForeignKey 字段实现，其语法格式如下：

```
ForeignKey(to, on_delete, related_name, related_query_name)
```

其中，参数 to 表示关系另一侧的数据模型名称，参数 on_delete 用于设置外键约束，其包括以下 5 种，即 models.CASCADE(表示级联删除，即删除父表的某一条数据时，其子表中使用该外键的数据也会被删除)、models.PROTECT(表示保护模式，即删除父表的某一条数据时会抛出 ProtectedError 异常，阻止删除操作)、models.SET_NULL(表示设置为 NULL，即删除父表的某一条数据时，其子表中的外键会被设置为 NULL)、models.SET_DEFAULT(表示设置为默认值，即删除父表的某一条数据时，其子表中的外键会被设置为默认值)和 models.SET(表示设置为指定值，即删除父表的某一条数据时，其子表中的外键会被设置为自定义的外键字段的值)，参数 related_name 用于指定反向引用的名称；参数 related_query_name 用于在关联数据模型查询时，指定字段名称，默认使用的是数据模型的名称。

下面通过一个示例演示一下如何使用外键。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book 和 python manage.py startapp press，创建名为 book 和 press 的应用，如图 5-115 所示。

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp press
```

图 5-115 创建应用

(3) 打开应用 djangoProject 中的文件__init__.py，代码如下：

```
#资源包\Code\chapter5\5.7\3\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(4) 打开应用 djangoProject 中的配置文件 settings.py，进行数据库的相关配置，用于连接数据库，代码如下：

```
#资源包\Code\chapter5\5.7\3\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接 MySQL 的用户名
        'USER': 'root',
        #连接 MySQL 的密码
        'PASSWORD': '12345678',
        #MySQL 的主机地址
        'HOST': '127.0.0.1',
        #MySQL 的端口号
        'PORT': '3306',
    }
}
```

(5) 打开应用 djangoProject 中的配置文件 settings.py，将应用 book 和 press 添加到变

量 INSTALLED_APPS 中，用于完成应用注册，代码如下：

```
#资源包\Code\chapter5\5.7\3\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'book',
    'press',
]
```

（6）打开应用 book 中的文件 models.py，代码如下：

```
#资源包\Code\chapter5\5.7\3\djangoProject\book\models.py
from django.db import models
#父表
class Category(models.Model):
    cate = models.CharField(max_length=100)
#子表
class Book(models.Model):
    name = models.CharField(max_length=100)
    author = models.CharField(max_length=20, null=False)
    category = models.ForeignKey("Category", on_delete=models.CASCADE)
    #category = models.ForeignKey("Category", on_delete=models.PROTECT)
    #category = models.ForeignKey("Category", null=True, on_delete=models.SET_NULL)
    #category = models.ForeignKey("Category", null=True, on_delete=models.SET_DEFAULT,
default=Category.objects.get(cate='Python 编程'))
    #category = models.ForeignKey("Category", null=True, on_delete=models.SET(Category.
objects.get(cate='Django 编程')))
    press = models.ForeignKey('press.Press', on_delete=models.CASCADE, null=True)
```

（7）打开应用 press 中的文件 models.py，代码如下：

```
#资源包\Code\chapter5\5.7\3\djangoProject\press\models.py
from django.db import models
class Press(models.Model):
    press_name = models.CharField(max_length=100, null=True)
```

（8）打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.7\3\djangoProject\book\views.py
from book.models import Book, Category
from press.models import Press
from django.http import HttpResponse
def add_data(request):
    category = Category(cate='计算机编程')
    category.save()
    press = Press(press_name='清华大学出版社')
    press.save()
    book = Book(name='《Python 全栈开发——基础入门》', author='夏正东')
    book.category = category
    book.press = press
```

```
    book.save()
    return HttpResponse('新的图书已经添加成功!')
def del_category_data(request):
    category = Category.objects.get(cate='计算机编程')
    category.delete()
    return HttpResponse('数据删除成功!')
#关联模型查询
def filter_data(request):
    categories = Category.objects.filter(book__name__contains="Python")
    categories_lt = []
    for category in categories:
        categories_lt.append(category.cate)
    return HttpResponse(f'查询结果为{categories_lt}')
```

(9) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.7\3\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('add_data/', views.add_data),
    path('del_category_data/', views.del_category_data),
    path('filter_data/', views.filter_data),
]
```

(10) 打开命令提示符窗口,登入 MySQL。

(11) 在 MySQL 命令行窗口中输入 SQL 语句“create database django_db;”,创建数据库 django_db,其结果如图 5-116 所示。

(12) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py makemigrations,生成迁移脚本文件,如图 5-117 所示。

```
mysql> create database django_db;
Query OK, 1 row affected (0.00 sec)
```

图 5-116 创建数据库

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'press':
  press\migrations\0001_initial.py
    - Create model Press
Migrations for 'book':
  book\migrations\0001_initial.py
    - Create model Category
    - Create model Book
```

图 5-117 生成迁移脚本文件

(13) 再次输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-118 所示。

(14) 在 MySQL 命令行窗口中输入 SQL 语句“desc book_book;”,查询该数据表的表结构,其结果如图 5-119 所示。

(15) 再次输入 SQL 语句“desc book_category;”,查询该数据表的表结构,其结果如图 5-120 所示。

(16) 再次输入 SQL 语句“desc press_press;”,查询该数据表的表结构,其结果如图 5-121 所示。

(17) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/add_data/,其显示内容如图 5-122 所示。

```
Operations to perform:
  Apply all migrations: admin, auth, book, contenttypes, press, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying press.0001_initial... OK
  Applying book.0001_initial... OK
  Applying sessions.0001_initial... OK
```

图 5-118　将迁移脚本文件映射到数据库中

```
mysql> desc book_book;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| name        | varchar(100)| NO   |     | NULL    |                |
| author      | varchar(20) | NO   |     | NULL    |                |
| category_id | int(11)     | NO   | MUL | NULL    |                |
| press_id    | int(11)     | YES  | MUL | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

图 5-119　数据表 book_book 的表结构

```
mysql> desc book_category;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| cate  | varchar(100)| NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

图 5-120　数据表 book_category 的表结构

```
mysql> desc press_press;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| press_name | varchar(100)| YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

图 5-121　数据表 press_press 的表结构

(18) 在浏览器中的网址栏输入 http://127.0.0.1:8000/filter_data/,其显示内容如图 5-123 所示。

图 5-122　浏览器中的显示内容(1)

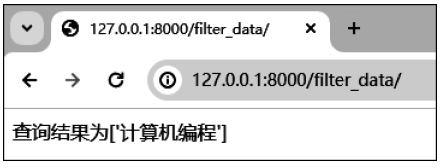


图 5-123　浏览器中的显示内容(2)

(19) 在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”,其结果如图 5-124 所示。

```
mysql> select * from book_book;
+----+----------------------------------------+----------+-------------+----------+
| id | name                                   | author   | category_id | press_id |
+----+----------------------------------------+----------+-------------+----------+
|  1 | 《Python全栈开发——基础入门》           | 夏正东   |           1 |        1 |
+----+----------------------------------------+----------+-------------+----------+
1 row in set (0.00 sec)
```

图 5-124 数据表 book_book 中的数据

(20) 输入 SQL 语句“select * from book_category;”,其结果如图 5-125 所示。

(21) 输入 SQL 语句“select * from press_press;”,其结果如图 5-126 所示。

```
mysql> select * from book_category;
+----+-----------------+
| id | cate            |
+----+-----------------+
|  1 | 计算机编程      |
+----+-----------------+
1 row in set (0.00 sec)
```

图 5-125 数据表 book_category 中的数据

```
mysql> select * from press_press;
+----+----------------------+
| id | press_name           |
+----+----------------------+
|  1 | 清华大学出版社       |
+----+----------------------+
1 row in set (0.00 sec)
```

图 5-126 数据表 press_press 中的数据

(22) 此时,在浏览器中的网址栏输入 http://127.0.0.1:8000/del_category_data/时,不同的外键约束会显示不同的运行的结果。

一是当外键约束为 models.CASCADE 时,在浏览器中的网址栏输入 http://127.0.0.1:8000/del_category_data/,其显示内容如图 5-127 所示。

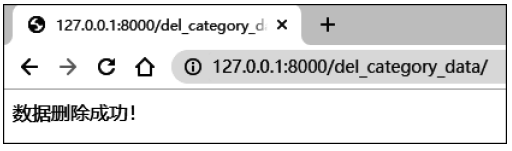


图 5-127 浏览器中的显示内容

此时,在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”,其结果如图 5-128 所示。

再次输入 SQL 语句“select * from book_category;”,其结果如图 5-129 所示。

```
mysql> select * from book_book;
Empty set (0.00 sec)
```

图 5-128 数据表 book_book 中的数据

```
mysql> select * from book_category;
Empty set (0.00 sec)
```

图 5-129 数据表 book_category 中的数据

二是当外键约束为 models.PROTECT 时,在浏览器中的网址栏输入 http://127.0.0.1:8000/del_category_data/,其显示内容如图 5-130 所示。

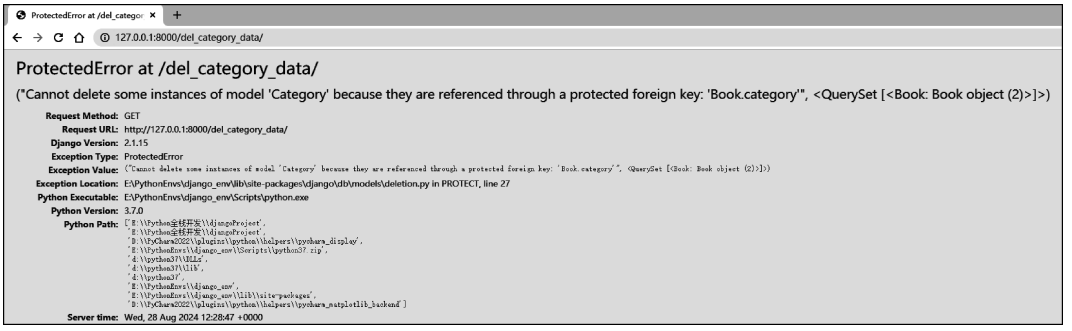


图 5-130 浏览器中的显示内容

此时，在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”，如图 5-131 所示。

```
mysql> select * from book_book;
+----+--------------------------------------+----------+-------------+----------+
| id | name                                 | author   | category_id | press_id |
+----+--------------------------------------+----------+-------------+----------+
|  2 | 《Python全栈开发-基础入门》          | 夏正东   |           2 |        2 |
+----+--------------------------------------+----------+-------------+----------+
1 row in set (0.00 sec)
```

图 5-131　数据表 book_book 中的数据

再次输入 SQL 语句“select * from book_category;”，如图 5-132 所示。

三是当外键约束为 models.SET_NULL 时，在浏览器中的网址栏输入 http://127.0.0.1:8000/del_category_data/，其显示内容如图 5-133 所示。

```
mysql> select * from book_category;
+----+-----------------+
| id | cate            |
+----+-----------------+
|  2 | 计算机编程      |
+----+-----------------+
1 row in set (0.00 sec)
```

图 5-132　数据表 book_category 中的数据

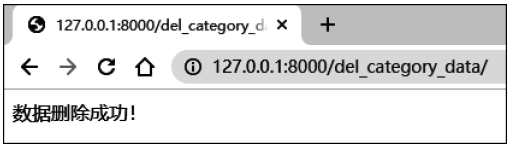


图 5-133　浏览器中的显示内容

此时，在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”，其结果如图 5-134 所示。

```
mysql> select * from book_book;
+----+--------------------------------------+----------+-------------+----------+
| id | name                                 | author   | category_id | press_id |
+----+--------------------------------------+----------+-------------+----------+
|  3 | 《Python全栈开发-基础入门》          | 夏正东   |        NULL |        3 |
+----+--------------------------------------+----------+-------------+----------+
1 row in set (0.00 sec)
```

图 5-134　数据表 book_book 中的数据

再次输入 SQL 语句“select * from book_category;”，其结果如图 5-135 所示。

四是当外键约束为 models.SET_DEFAULT 时，在 MySQL 命令行窗口中输入 SQL 语句“insert into book_category (cate) values ('Python 编程');”，添加新的数据，然后再次输入 SQL 语句“select * from book_category;”，查看数据表中的数据，其结果如图 5-136 所示。

```
mysql> select * from book_category;
Empty set (0.00 sec)
```

图 5-135　数据表 book_category 中的数据

```
mysql> select * from book_category;
+----+--------------+
| id | cate         |
+----+--------------+
|  4 | Python编程   |
+----+--------------+
1 row in set (0.00 sec)
```

图 5-136　数据表 book_category 中的数据

需要注意的是，必须先在数据表 book_category 中添加数据，然后在浏览器中的网址栏输入 http://127.0.0.1:8000/add_data/，添加其他数据；最后，在浏览器中的网址栏输入 http://127.0.0.1:8000/del_category_data/，其显示内容如图 5-137 所示。

此时，在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”，其结果如图 5-138 所示。

再次输入 SQL 语句“select * from book_category;”，其结果如图 5-139 所示。

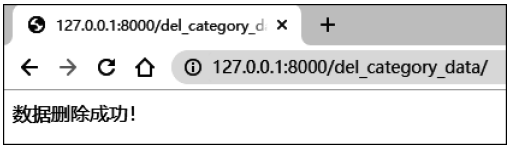


图 5-137 浏览器中的显示内容

```
mysql> select * from book_book;
+----+--------------------------------------+-----------+-------------+----------+
| id | name                                 | author    | category_id | press_id |
+----+--------------------------------------+-----------+-------------+----------+
|  4 | 《Python全栈开发-基础入门》          | 夏正东    |           4 |        4 |
+----+--------------------------------------+-----------+-------------+----------+
1 row in set (0.00 sec)
```

图 5-138 数据表 book_book 中的数据

五是当外键约束为 models.SET 时，在 MySQL 命令行窗口中输入 SQL 语句“insert into book_category (cate) values ('Django 编程');”，添加新的数据，然后再次输入 SQL 语句“select * from book_category;”，查看数据表中的数据，如图 5-140 所示。

```
mysql> select * from book_category;
+----+--------------+
| id | cate         |
+----+--------------+
|  4 | Python编程   |
+----+--------------+
1 row in set (0.00 sec)
```

图 5-139 数据表 book_category 中的数据

```
mysql> select * from book_category;
+----+--------------+
| id | cate         |
+----+--------------+
|  6 | Django编程   |
+----+--------------+
1 row in set (0.00 sec)
```

图 5-140 数据表 book_category 中的数据

需要注意的是，必须先在数据表 book_category 添加数据，然后在浏览器中的网址栏输入 http://127.0.0.1:8000/add_data/，添加其他数据；最后，在浏览器中的网址栏输入 http://127.0.0.1:8000/del_category_data/，其显示内容如图 5-141 所示。

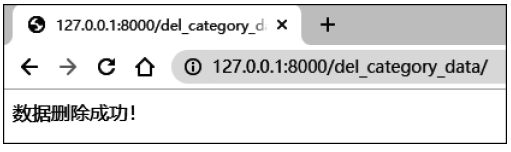


图 5-141 浏览器中的显示内容

此时，在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”，其结果如图 5-142 所示。

```
mysql> select * from book_book;
+----+--------------------------------------+-----------+-------------+----------+
| id | name                                 | author    | category_id | press_id |
+----+--------------------------------------+-----------+-------------+----------+
|  5 | 《Python全栈开发-基础入门》          | 夏正东    |           6 |        5 |
+----+--------------------------------------+-----------+-------------+----------+
1 row in set (0.00 sec)
```

图 5-142 数据表 book_book 中的数据

再次输入 SQL 语句“select * from book_category;”，其结果如图 5-143 所示。

```
mysql> select * from book_category;
+----+--------------+
| id | cate         |
+----+--------------+
|  6 | Django编程   |
+----+--------------+
1 row in set (0.00 sec)
```

图 5-143 数据表 book_category 中的数据

5.7.7 多表间关系

1. 一对多关系

一对多关系表示一个数据模型可以对应多个其他数据模型，而其他数据模型只能对应一个数据模型，例如，有两个数据模型 User 和 Book，即一位作者可以编写多本书籍，而一本书籍只能有一位作者。可以通过数据模型中的 ForeignKey 字段实现。

下面通过一个示例演示一下如何实现一对多关系。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-144 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-144 创建应用

(3) 打开应用 djangoProject 中的文件__init__.py，代码如下：

```
#资源包\Code\chapter5\5.7\4\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(4) 打开应用 djangoProject 中的配置文件 settings.py，进行数据库的相关配置，用于连接数据库，代码如下：

```
#资源包\Code\chapter5\5.7\4\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接 MySQL 的用户名
        'USER': 'root',
        #连接 MySQL 的密码
        'PASSWORD': '12345678',
        #MySQL 的主机地址
        'HOST': '127.0.0.1',
        #MySQL 的端口号
        'PORT': '3306',
    }
}
```

(5) 打开应用 djangoProject 中的配置文件 settings.py，将应用 book 添加到变量 INSTALLED_APPS 中，用于完成应用注册，代码如下：

```
#资源包\Code\chapter5\5.7\4\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'book',
]
```

(6) 打开应用 book 中的文件 models.py,代码如下:

```
#资源包\Code\chapter5\5.7\4\djangoProject\book\models.py
from django.db import models
class Author(models.Model):
    name = models.CharField(max_length=100)
class Book(models.Model):
    name = models.CharField(max_length=100)
    press = models.CharField(max_length=20, null=False)
    #反向引用的名称为 books
    author = models.ForeignKey("Author", on_delete=models.CASCADE, related_name='books')
```

(7) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.7\4\djangoProject\book\views.py
from book.models import Book, Author
from django.http import HttpResponse
def one_to_many_set(request):
    author = Author(name='夏正东')
    author.save()
    book1 = Book(name='《Python 全栈开发——基础入门》', press='清华大学出版社')
    book1.author = author
    book1.save()
    book2 = Book(name='《Python 全栈开发——高阶编程》', press='清华大学出版社')
    book2.author = author
    book2.save()
    return HttpResponse('新的图书已经添加成功!')
def one_to_many_get(request):
    author = Author.objects.first()
    #获取当前作者所编写的所有书籍,即反向引用
    books = author.books.all()
    book_lt = []
    for book in books:
        book_lt.append(book.name)
    return HttpResponse(book_lt)
```

(8) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.7\4\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('one_to_many_set/', views.one_to_many_set),
    path('one_to_many_get/', views.one_to_many_get),
]
```

(9) 打开命令提示符窗口,登入 MySQL。

(10) 在 MySQL 命令行窗口中输入 SQL 语句“create database django_db;”,创建数据

库 django_db，其结果如图 5-145 所示。

（11）打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py makemigrations，生成迁移脚本文件，如图 5-146 所示。

```
mysql> create database django_db;
Query OK, 1 row affected (0.00 sec)
```

图 5-145　创建数据库

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'book':
  book\migrations\0001_initial.py
    - Create model Author
    - Create model Book
```

图 5-146　生成迁移脚本文件

（12）再次输入命令 python manage.py migrate，将迁移脚本文件映射到数据库中，如图 5-147 所示。

（13）运行上述项目，打开浏览器并在网址栏输入 http://127.0.0.1:8000/one_to_many_set/，其显示内容如图 5-148 所示。

```
Operations to perform:
  Apply all migrations: admin, auth, book, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying book.0001_initial... OK
  Applying sessions.0001_initial... OK
```

图 5-147　迁移脚本文件映射到数据库中

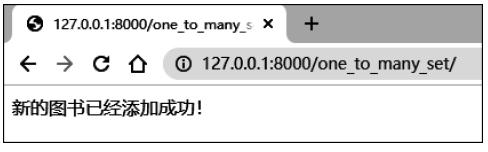


图 5-148　浏览器中的显示内容

（14）在 MySQL 命令行窗口中输入 SQL 语句“select * from book_book;”，其结果如图 5-149 所示。

```
mysql> select * from book_book;
+----+-----------------------------------------+---------------------+-----------+
| id | name                                    | press               | author_id |
+----+-----------------------------------------+---------------------+-----------+
|  1 | 《Python全栈开发——基础入门》            | 清华大学出版社      |         1 |
|  2 | 《Python全栈开发——高阶编程》            | 清华大学出版社      |         1 |
+----+-----------------------------------------+---------------------+-----------+
2 rows in set (0.00 sec)
```

图 5-149　数据表 book_book 中的数据

（15）再次输入 SQL 语句“select * from book_author;”，其结果如图 5-150 所示。

（16）在浏览器中的网址栏输入 http://127.0.0.1:8000/one_to_many_get/，其显示内容如图 5-151 所示。

```
mysql> select * from book_author;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | 夏正东    |
+----+-----------+
1 row in set (0.00 sec)
```

图 5-150　数据表 book_author 中的数据

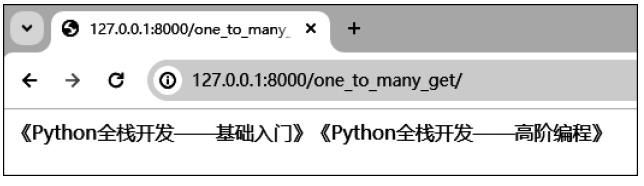


图 5-151　浏览器中的显示内容

2. 一对一关系

一对一关系表示两个数据模型之间存在一个唯一的对应关系，例如，有两个数据模型

User 和 UserExtend，即一个用户只能有一个该用户的扩展，而扩展也只能对应一个用户。可以通过数据模型中的 OneToOneField 字段实现。

下面通过一个示例，演示一下如何实现一对一关系。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp user，创建名为 user 的应用，如图 5-152 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp user
```

图 5-152 创建应用

(3) 打开应用 djangoProject 中的文件__init__.py，代码如下：

```
#资源包\Code\chapter5\5.7\5\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(4) 打开应用 djangoProject 中的配置文件 settings.py，进行数据库的相关配置，用于连接数据库，代码如下：

```
#资源包\Code\chapter5\5.7\5\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接 MySQL 的用户名
        'USER': 'root',
        #连接 MySQL 的密码
        'PASSWORD': '12345678',
        #MySQL 的主机地址
        'HOST': '127.0.0.1',
        #MySQL 的端口号
        'PORT': '3306',
    }
}
```

(5) 再次打开应用 djangoProject 中的配置文件 settings.py，将应用 user 添加到变量 INSTALLED_APPS 中，用于完成应用注册，代码如下：

```
#资源包\Code\chapter5\5.7\5\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'user',
]
```

(6) 打开应用 user 中的文件 models.py，代码如下：

```
#资源包\Code\chapter5\5.7\5\djangoProject\user\models.py
from django.db import models
class User(models.Model):
    username = models.CharField(max_length=100)
class UserExtend(models.Model):
    job = models.CharField(max_length=100)
    user = models.OneToOneField("User", on_delete=models.CASCADE, related_name=
'userextend')
```

(7) 打开应用 user 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.7\5\djangoProject\user\views.py
from user.models import User, UserExtend
from django.http import HttpResponse
def one_to_one_set(request):
    user = User(username='夏正东')
    user.save()
    userextend = UserExtend(job='老师')
    userextend.user = user
    userextend.save()
    return HttpResponse('新的用户信息已经添加成功!')
def one_to_one_get(request):
    user = User.objects.first()
    #获取当前用户所对应的扩展信息,即反向引用
    userextend = user.userextend
    return HttpResponse(f'当前用户的职业是:{userextend.job}')
```

(8) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.7\5\djangoProject\djangoProject\urls.py
from django.urls import path
from user import views
urlpatterns = [
    path('one_to_one_set/', views.one_to_one_set),
    path('one_to_one_get/', views.one_to_one_get),
]
```

(9) 打开命令提示符窗口,登入 MySQL。

(10) 在 MySQL 命令行窗口中输入 SQL 语句“create database django_db;”,创建数据库 django_db,其结果如图 5-153 所示。

(11) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py makemigrations,生成迁移脚本文件,如图 5-154 所示。

```
mysql> create database django_db;
Query OK, 1 row affected (0.00 sec)
```

图 5-153 创建数据库

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'user':
  user\migrations\0001_initial.py
    - Create model User
    - Create model UserExtend
```

图 5-154 生成迁移脚本文件

(12) 再次输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-155 所示。

(13) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/one_to_one_

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, user
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying user.0001_initial... OK
```

图 5-155 迁移脚本文件映射到数据库中

set/，其显示内容如图 5-156 所示。

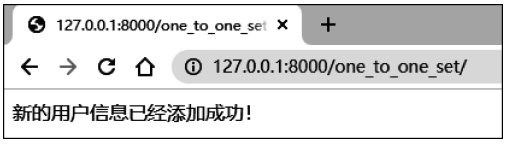


图 5-156 浏览器中的显示内容

(14) 在 MySQL 命令行窗口中输入 SQL 语句“select * from user_user;”，其结果如图 5-157 所示。

(15) 再次输入 SQL 语句“select * from user_userextend;”，其结果如图 5-158 所示。

```
mysql> select * from user_user;
+----+----------+
| id | username |
+----+----------+
|  1 | 夏正东   |
+----+----------+
1 row in set (0.00 sec)
```

图 5-157 数据表 user_user 中的数据

```
mysql> select * from user_userextend;
+----+--------+---------+
| id | job    | user_id |
+----+--------+---------+
|  1 | 老师   |       1 |
+----+--------+---------+
1 row in set (0.00 sec)
```

图 5-158 数据表 user_userextend 中的数据

(16) 在浏览器中的网址栏输入 http://127.0.0.1:8000/one_to_one_get/，其显示内容如图 5-159 所示。

图 5-159 浏览器中的显示内容

3. 多对多关系

多对多关系表示两个数据模型之间存在多个对应关系，例如，有两个数据模型 Article 和 Tag，即一篇文章可以对应多个标签，并且一个标签也可以对应多篇文章。可以通过数据模型中的 ManyToManyField 字段实现。

下面通过一个示例演示一下如何实现多对多关系。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp article，创建名为 article 的应用，如图 5-160 所示。

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp article
```

图 5-160　创建应用

(3) 打开应用 djangoProject 中的文件__init__.py,代码如下:

```
#资源包\Code\chapter5\5.7\6\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(4) 打开应用 djangoProject 中的配置文件 settings.py,进行数据库的相关配置,用于连接数据库,代码如下:

```
#资源包\Code\chapter5\5.7\6\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接 MySQL 的用户名
        'USER': 'root',
        #连接 MySQL 的密码
        'PASSWORD': '12345678',
        #MySQL 的主机地址
        'HOST': '127.0.0.1',
        #MySQL 的端口号
        'PORT': '3306',
    }
}
```

(5) 打开应用 djangoProject 中的配置文件 settings.py,将应用 article 添加到变量 INSTALLED_APPS 中,用于完成应用注册,代码如下:

```
#资源包\Code\chapter5\5.7\6\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'article',
]
```

(6) 打开应用 article 中的文件 models.py,代码如下:

```
#资源包\Code\chapter5\5.7\6\djangoProject\article\models.py
from django.db import models
class Article(models.Model):
    title = models.CharField(max_length=100)
    username = models.CharField(max_length=100)
class Tag(models.Model):
    name = models.CharField(max_length=100)
    articles = models.ManyToManyField("Article", related_name='tags')
```

(7) 打开应用 article 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.7\6\djangoProject\article\views.py
from article.models import Article, Tag
from django.http import HttpResponse
def many_to_many_set(request):
    tag1 = Tag(name='计算机文章')
    tag1.save()
    tag2 = Tag(name='热门文章')
    tag2.save()
    article1 = Article(title='Django 开发的流程', username='夏正东')
    article1.save()
    article2 = Article(title='如何养成缜密的逻辑思维', username='夏正东')
    article2.save()
    #通过 add()方法添加数据
    article1.tags.add(tag1)
    article1.tags.add(tag2)
    article2.tags.add(tag2)
    return HttpResponse('新的文章已经添加成功!')
def many_to_many_get(request):
    article = Article.objects.first()
    #获取当前用户所对应的扩展信息,即反向引用
    tags = article.tags.all()
    tag_lt = []
    for tag in tags:
        tag_lt.append(tag.name)
    return HttpResponse(tag_lt)
```

(8) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.7\6\djangoProject\djangoProject\urls.py
from django.urls import path
from article import views
urlpatterns = [
    path('many_to_many_set/', views.many_to_many_set),
    path('many_to_many_get/', views.many_to_many_get),
]
```

(9) 打开命令提示符窗口,登入 MySQL。

(10) 在 MySQL 命令行窗口中输入 SQL 语句“create database django_db;”,创建数据库 django_db,其结果如图 5-161 所示。

(11) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py makemigrations,生成迁移脚本文件,如图 5-162 所示。

```
mysql> create database django_db;
Query OK, 1 row affected (0.00 sec)
```

图 5-161 创建数据库

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'article':
  article\migrations\0001_initial.py
    - Create model Article
    - Create model Tag
```

图 5-162 生成迁移脚本文件

(12) 再次输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-163 所示。

(13) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/many_to_many_set/,其显示内容如图 5-164 所示。

```
Operations to perform:
  Apply all migrations: admin, article, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying article.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

图 5-163　迁移脚本文件映射到数据库中

(14) 在 MySQL 命令行窗口中输入 SQL 语句“select * from article_article;”,其结果如图 5-165 所示。

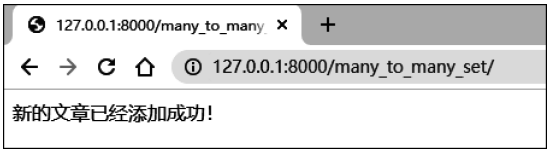


图 5-164　浏览器中的显示内容

```
mysql> select * from article_article;
+----+-----------------------------------+----------+
| id | title                             | username |
+----+-----------------------------------+----------+
|  1 | Django开发的流程                  | 夏正东   |
|  2 | 如何养成缜密的逻辑思维            | 夏正东   |
+----+-----------------------------------+----------+
2 rows in set (0.00 sec)
```

图 5-165　表 article_article 中的数据

(15) 再次输入 SQL 语句“select * from article_tag;”,其结果如图 5-166 所示。

(16) 此时,Django 会在数据库中自动生成关联表 article_article_tag,输入 SQL 语句“select * from article_article_tag;”,其结果如图 5-167 所示。

```
mysql> select * from article_tag;
+----+----------------+
| id | name           |
+----+----------------+
|  1 | 计算机文章     |
|  2 | 热门文章       |
+----+----------------+
2 rows in set (0.00 sec)
```

图 5-166　表 article_tag 中的数据

```
mysql> select * from article_tag_articles;
+----+--------+------------+
| id | tag_id | article_id |
+----+--------+------------+
|  1 |      1 |          1 |
|  2 |      2 |          1 |
|  3 |      2 |          2 |
+----+--------+------------+
3 rows in set (0.00 sec)
```

图 5-167　表 article_article_tag 中的数据

(17) 在浏览器中的网址栏输入 http://127.0.0.1:8000/many_to_many_get/,其显示内容如图 5-168 所示。

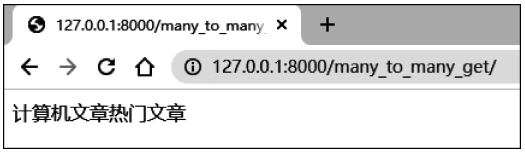


图 5-168　浏览器中的显示内容

5.8　表单验证

5.8.1　HTML 表单验证

在 Django 中表单验证可以分为 4 步:

(1) 导入 django.forms 模块和 django.core.validators 模块中的相关类,其中,django.forms 模块包含支持 HTML 的标准字段类(如表 5-11 所示);django.core.validators 模块包含对 HTML 表单进行验证的验证器类,包括内置验证器类和自定义验证器类。

表 5-11 HTML 标准字段类

字段类	参数	描述
CharField	label,表示字段别名; validators,表示验证规则组成的列表; help_text,表示字段帮助信息; required,表示是否为必填字段; widget,表示 HTML 插件; error_messages,表示自定义错误的字典; disabled,表示是否为禁用字段; label_suffix;表示所有字段的标签后缀,在默认情况下为冒号; initial,表示字段的初始值	字符串字段
FloatField		浮点数字段
IntegerField		整型字段
EmailField		电子邮件地址字段
URLField		URL 字段
FileField		文件上传字段
ImageField		图片字段
BooleanField		复选框字段
ChoiceField		下拉列表字段
MultipleChoiceField		多选下拉列表字段
DateField		日期字段
DateTimeField		日期时间字段

(2) 创建表单类:表单类主要用于定义 HTML 表单中的字段,该类中的属性对应 HTML 表单中的每个字段。需要注意的是,表单类必须继承自 django.forms 中的 Form 类,并且其属性必须与表单字段中属性 name 的值一致。

(3) 处理验证数据:在视图中,通过表单对象的 is_valid()方法处理数据验证之后的结果。

(4) 指定验证器:在验证表单中的字段时,还可以通过给字段类中的参数 validators 传递指定验证器,从而进一步对数据进行过滤,其常用的内置验证器类如表 5-12 所示。

表 5-12 常用的内置验证器类

内置验证器类	描述
MaxValueValidator	用于确保数据的值不超过指定的最大值
MinValueValidator	用于确保数据的值不低于指定的最小值
MinLengthValidator	用于验证字段输入长度是否达到最小长度
MaxLengthValidator	用于确保字段的长度不会超过指定的最大长度
EmailValidator	用于验证电子邮件地址
URLValidator	用于验证输入是否为有效的 URL
RegexValidator	用于确保字段的输入匹配一个正则表达式

其中,自定义验证器类需要在表单类中创建一种方法,如果验证单一字段,则该方法的命名规则需为“clean_字段名”,如果验证多字段,则该方法的命名规则需为 clean。

下面通过一个示例,演示一下如何进行 HTML 表单验证。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-169 所示。

(3) 在应用 book 中创建表单验证文件 forms.py,代码如下:

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-169　创建应用

```
#资源包\Code\chapter5\5.8\1\djangoProject\book\forms.py
from django import forms
from django.core import validators
class RegForm(forms.Form):
    book_name = forms.CharField(max_length=8, min_length=2)
    book_price = forms.FloatField(max_value=200, min_value=50)
    author_tel = forms.CharField(validators=[validators.RegexValidator(r'1[3456789]\d{9}
')])
    author_email = forms.EmailField()
    web_site = forms.urlField()
    pwd1 = forms.CharField(max_length=10, min_length=6)
    pwd2 = forms.CharField(max_length=10, min_length=6)
    #单一字段验证(自定义)
    def clean_author_email(self):
        #cleaned_data 含通过验证的表单数据所组成的字典
        author_email = self.cleaned_data.get('author_email')
        if author_email == 'oldxia1@qq.com':
            raise forms.ValidationError(message='此邮箱不被允许')
        return author_email
    #多字段验证(自定义)
    def clean(self):
        result = super().clean()
        pwd1 = self.cleaned_data.get('pwd1')
        pwd2 = self.cleaned_data.get('pwd2')
        if pwd1 != pwd2:
            raise forms.ValidationError(message='两次密码不一致')
        return result
```

(4) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.8\1\djangoProject\book\views.py
from django.http import HttpResponse
from django.shortcuts import render
from django.views.generic import View
from book.forms import RegForm
class RegView(View):
    def get(self, request):
        return render(request, 'index.html')
    def post(self, request):
        form = RegForm(request.POST)
        #处理验证数据
        if form.is_valid():
            return HttpResponse('数据已经提交!')
        else:
            return HttpResponse('提交的数据有误,请重新提交!')
```

(5) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.8\1\djangoProject\templates\index.html
<html lang="en">
    <head>
```

```
        <meta charset = "UTF - 8">
        <title > Title </title >
    </head >
    < body >
        < form action = "" method = "post">
            { % csrf_token % }
            < p >书籍名称: < input type = "text" name = "book_name"></p >
            < p >书籍价格: < input type = "text" name = "book_price"></p >
            < p >作者手机号码: < input type = "text" name = "author_tel"></p >
            < p >作者 Email: < input type = "text" name = "author_email"></p >
            < p >作者个人博客: < input type = "text" name = "web_site"></p >
            < p >密码 1: < input type = "password" name = "pwd1"></p >
            < p >密码 2: < input type = "password" name = "pwd2"></p >
            < input type = "submit" value = "提交">
        </form >
    </body >
</html >
```

(6) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
# 资源包\Code\chapter5\5.8\1\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
   path('', views.RegView.as_view()),
]
```

(7) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-170 所示。

(8) 此时,在表单中填入相关数据,其显示内容如图 5-171 所示。

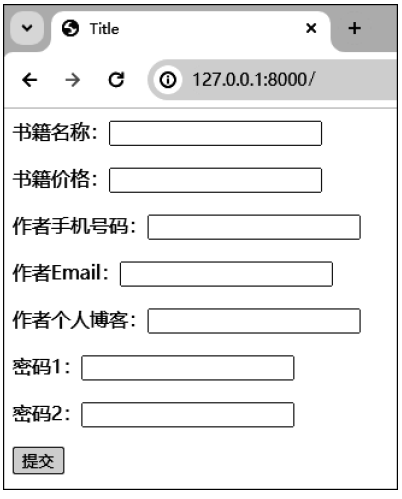


图 5-170 浏览器中的显示内容(1)

图 5-171 浏览器中的显示内容(2)

由于填入的“作者手机号码”格式错误,所以表单数据无法成功提交,其显示内容如图 5-172 所示。

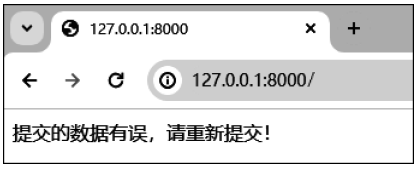


图 5-172　浏览器中的显示内容(3)

5.8.2　上传文件验证

上传文件验证需要注意两点：一是表单类中对应属性的字段类型需为 FileField；二是通过验证器类中的参数 allowed_extensions，可以对文件类型进行限制。

下面通过一个示例演示一下如何进行上传文件验证。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-173 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-173　创建应用

(3) 在应用 book 中创建表单验证文件 forms.py，代码如下：

```
#资源包\Code\chapter5\5.8\2\djangoProject\book\forms.py
from django import forms
from django.core import validators
class UploadFileForm(forms.Form):
    myfile = forms.FileField(validators=[validators.FileExtensionValidator(allowed_
extensions=['pdf', 'docx', 'doc'])])
```

(4) 打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.8\2\djangoProject\book\views.py
from django.views.generic import View
from django.shortcuts import render
from django.http import HttpResponse
from book.forms import UploadFileForm
class IndexView(View):
    def get(self, request):
        return render(request, 'index.html')
    def post(self, request):
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            myfile = request.FILES.get('myfile')
            #此时,myfile 类型为 InMemoryUploadedFile,可以通过 read()方法读取文件内容,并
#返回二进制数据
            with open('file.txt', 'wb') as fb:
                fb.write(myfile.read())
            return HttpResponse('电子书上传成功!')
        else:
            return HttpResponse('电子书上传失败!')
```

(5) 在项目根目录下的文件夹 templates 中创建模板文件 index.html，代码如下：

```
#资源包\Code\chapter5\5.8\2\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <form action="" method="post" enctype="multipart/form-data">
            {% csrf_token %}
            <p>文件:<input type="file" name="myfile" id="">></p>
            <input type="submit" value="提交">
        </form>
    </body>
</html>
```

(6) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.8\2\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
   path('', views.IndexView.as_view()),
]
```

(7) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-174 所示。

(8) 此时,上传 PDF 文件并单击“提交”按钮,其显示内容如图 5-175 所示。

图 5-174 浏览器中的显示内容(1)

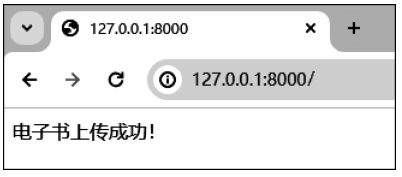


图 5-175 浏览器中的显示内容(2)

5.8.3 ModelForm 类

ModelForm 类是一种自动生成表单的工具,其是以数据模型为基础并在数据模型类上定义的表单。

在使用 ModelForm 类时,只需指定数据模型类作为表单数据的基础,就可以自动生成表单并进行验证。

下面通过一个示例,演示一下如何使用 ModelForm 类。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-176 所示。

(3) 在项目根目录下创建文件夹 media,用于存放上传的文件。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-176　创建应用

(4) 在应用 book 中创建表单验证文件 forms.py,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\book\forms.py
from django import forms
from book.models import Book
class AddBookForm(forms.ModelForm):
    def clean_author(self):
        author = self.cleaned_data.get('author')
        if author != '夏正东':
            print(author)
            raise forms.ValidationError('作者信息错误!')
        return author
    class Meta:
        model = Book
        #需要验证的字段
        fields = "__all__"
        #不需要验证的字段
        #Exclude = "author"
```

(5) 打开应用 book 中的文件 models.py,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\book\models.py
from django.db import models
from django.core import validators
class Book(models.Model):
    name = models.CharField(max_length = 100)
    author = models.CharField(max_length = 20)
    price = models.FloatField(validators = [validators.MinValueValidator(limit_value = 50)])
    #upload_to 定义上传文件存放的位置
    pdf = models.FileField(upload_to = '%Y/%m/%d', validators = [validators.
FileExtensionValidator(['pdf'])])
```

(6) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\book\views.py
from django.http import HttpResponse
from django.shortcuts import render
from book.forms import AddBookForm
from django.views.generic import View
class AddBookView(View):
    def get(self, request):
        return render(request, 'index.html')
    def post(selfm, request):
        form = AddBookForm(request.POST, request.FILES)
        myfile = request.FILES.get('pdf')
        print(myfile)
        if form.is_valid():
            form.save()
            return HttpResponse("书籍添加成功!")
        else:
            print(form.errors.get_json_data())
            return HttpResponse("书籍添加失败!")
```

(7) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
    <form action="" method="post" enctype="multipart/form-data">
        {% csrf_token %}
        <p>书籍名称:<input type="text" name="name"></p>
        <p>书籍作者:<input type="text" name="author"></p>
        <p>书籍价格:<input type="text" name="price"></p>
        <p>PDF 书籍:<input type="file" name="pdf"></p>
        <p><input type="submit" value="添加书籍"></p>
    </form>
    </body>
</html>
```

(8) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\djangoProject\urls.py
from Django.urls import path
from book import views
urlpatterns = [
   path('', views.AddBookView.as_view()),
]
```

(9) 打开应用 djangoProject 中的文件__init__.py,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(10) 打开应用 djangoProject 中的配置文件 settings.py,进行数据库的相关配置,用于连接数据库,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\djangoProject\settings.py
DATABASES = {
   'default': {
       #数据库引擎
       'ENGINE': 'django.db.backends.mysql',
       #数据库名称
       'NAME': 'django_db',
       #连接 MySQL 的用户名
       'USER': 'root',
       #连接 MySQL 的密码
       'PASSWORD': '12345678',
       #MySQL 的主机地址
       'HOST': '127.0.0.1',
       #MySQL 的端口号
       'PORT': '3306',
   }
}
```

(11) 打开应用 djangoProject 中的配置文件 settings.py,将应用 book 添加到变量 INSTALLED_APPS 中,用于完成应用注册,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\djangoProject\settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'book',
]
```

(12) 打开应用 djangoProject 中的配置文件 settings.py,设置上传文件的位置和访问上传文件的 URL,代码如下:

```
#资源包\Code\chapter5\5.8\3\djangoProject\djangoProject\settings.py
#上传文件的位置
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
#访问上传文件的 URL
MEDIA_URL = '/media/'
```

(13) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py makemigrations,生成迁移脚本文件,如图 5-177 所示。

```
(django_env) E:\Python全栈开发\djangoProject>python manage.py makemigrations
Migrations for 'book':
  book\migrations\0001_initial.py
    - Create model Book
```

图 5-177 生成迁移脚本文件

(14) 再次输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-178 所示。

```
Operations to perform:
  Apply all migrations: admin, auth, book, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying book.0001_initial... OK
  Applying sessions.0001_initial... OK
```

图 5-178 迁移脚本文件映射到数据库中

(15) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-179 所示。

(16) 此时,在表单中填入相关数据,其显示内容如图 5-180 所示。

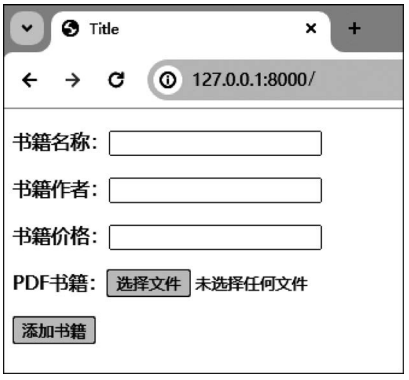


图 5-179 浏览器中的显示内容(1)

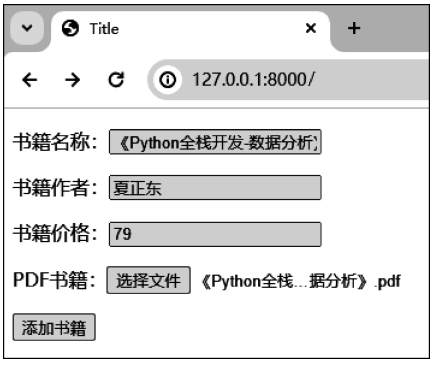


图 5-180 浏览器中的显示内容(2)

单击“添加书籍”按钮后,其显示内容如图 5-181 所示。

上传文件在项目中的存储位置如图 5-182 所示。

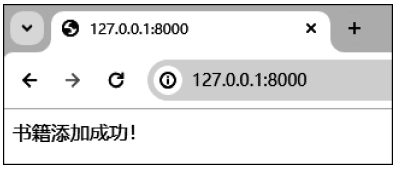


图 5-181 浏览器中的显示内容(3)

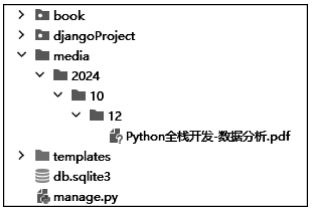


图 5-182 上传文件的存储位置

5.9 Cookie 和 Session

5.9.1 设置、获取和删除 Cookie

1. 设置 Cookie

通过 HttpResponse 对象的 set_cookie()方法对 Cookie 进行设置,其语法格式如下:

```
set_cookie(key, value, max_age, expires, path, domain, secure, httponly)
```

其中,参数 key 表示 Cookie 的键,参数 value 表示 Cookie 的值,参数 max_age 表示 Cookie 被保存的时间,单位为秒,参数 expires 表示 Cookie 的具体过期时间,参数 path 用于限制 Cookie 的有效路径,参数 domain 用于设置 Cookie 可用的域名,参数 secure 用于设置 Cookie 是否仅通过 HTTPS 发送,参数 httponly 用于设置是否禁止 JavaScript 获取 Cookie。

2. 获取 Cookie

通过 HttpRequest 对象的属性 COOKIES 来获取指定 Cookie 的值。

3. 删除 Cookie

通过 HttpResponse 对象的 delete_cookie()方法对 Cookie 进行删除,其语法格式如下:

```
delete_cookie(key)
```

其中，参数 key 表示 Cookie 的键。

下面通过一个示例演示一下如何设置、获取和删除 Cookie。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-183 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-183　创建应用

(3) 打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.9\1\djangoProject\book\views.py
from django.http import HttpResponse
def setcookie(request):
    response = HttpResponse('Cookie 已经设置成功!')
    response.set_cookie('web_site', 'http://www.oldxia.com')
    return response
def getcookie(request):
    cookies = request.COOKIES
    web_site = cookies.get('web_site')
    return HttpResponse(web_site)
def deletecookie(request):
    response = HttpResponse('Cookie 已经删除成功!')
    response.delete_cookie('web_site')
    return response
```

(4) 打开应用 djangoProject 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.9\1\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('setcookie/', views.setcookie),
    path('getcookie/', views.getcookie),
    path('deletecookie/', views.deletecookie),
]
```

(5) 运行上述项目，打开浏览器并在网址栏输入 http://127.0.0.1:8000/setcookie/，即可设置 Cookie，其显示内容如图 5-184 所示。

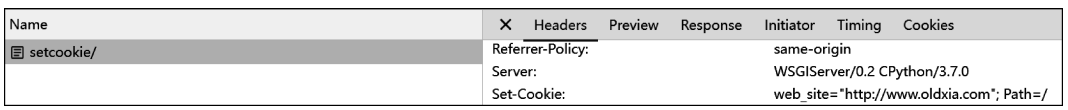


图 5-184　浏览器中的显示内容(1)

(6) 在浏览器中的网址栏输入 http://127.0.0.1:8000/getcookie/，即可获取 Cookie 的值，其显示内容如图 5-185 所示。

(7) 在浏览器中的网址栏输入 http://127.0.0.1:8000/deletecookie/，即可删除 Cookie，其显示内容如图 5-186 所示。

图 5-185 浏览器中的显示内容(2)

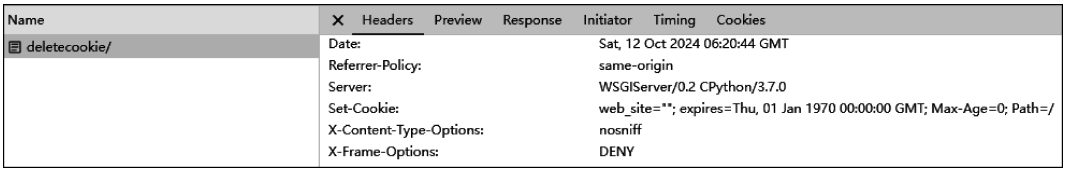


图 5-186 浏览器中的显示内容(3)

5.9.2 设置、获取和删除 Session

在默认情况下,由于 Session 存储在数据库的 django_session 数据表之中,所以在使用 Session 之前,需要在命令提示符中执行命令“python manage.py migrate”。

1. 设置 Session

通过 HttpRequest 对象的属性 session 对 Session 进行设置。

2. 获取 Session

通过 Session 对象的 get()方法来获取指定的 Session 值,其语法格式如下:

```
get(key)
```

其中,参数 key 表示 Session 的键。

3. 删除 Session

删除 Session 有两种方式。

一是通过 Session 对象的 pop()方法删除指定的 Session,其语法格式如下:

```
pop(key)
```

其中,参数 key 表示 Session 的键。

二是通过 Session 对象的 clear()方法删除全部的 Session,其语法格式如下:

```
clear()
```

下面通过一个示例演示一下如何设置、获取和删除 Session。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-187 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-187 创建应用

(3) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.9\2\djangoProject\book\views.py
from django.http import HttpResponse
def setsession(request):
    request.session['web_site'] = 'http://www.oldxia.com'
    return HttpResponse('Session已经设置成功!')
def getsession(request):
    web_site = request.session.get('web_site')
    return HttpResponse(web_site)
def deletesession(request):
    request.session.pop('web_site')
    return HttpResponse('Session已经删除!')
```

(4) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.9\2\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('setsession/', views.setsession),
    path('getsession/', views.getsession),
    path('deletesession/', views.deletesession),
]
```

(5) 打开应用 djangoProject 中的文件__init__.py,代码如下:

```
#资源包\Code\chapter5\5.9\2\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(6) 打开应用 djangoProject 中的配置文件 settings.py,进行数据库的相关配置,用于连接数据库,代码如下:

```
#资源包\Code\chapter5\5.9\2\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接MySQL的用户名
        'USER': 'root',
        #连接MySQL的密码
        'PASSWORD': '12345678',
        #MySQL的主机地址
        'HOST': '127.0.0.1',
        #MySQL的端口号
        'PORT': '3306',
    }
}
```

(7) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py migrate,将迁移脚本文件映射到数据库中,如图 5-188 所示。

(8) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/setsession/,

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

图 5-188 迁移脚本文件映射到数据库中

即可设置 Session,其显示内容如图 5-189 所示。

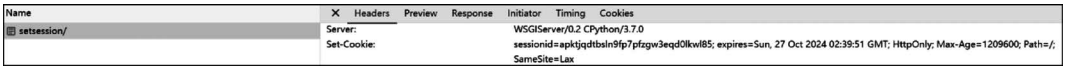


图 5-189 浏览器中的显示内容(1)

(9) 在浏览器中的网址栏输入 http://127.0.0.1:8000/getsession/,即可获取 Session 的值,其显示内容如图 5-190 所示。

(10) 在浏览器中的网址栏输入 http://127.0.0.1:8000/deletesession/,即可删除 Session。此时,在浏览器中的网址栏输入 http://127.0.0.1:8000/getsession/,其显示内容如图 5-191 所示。

图 5-190 浏览器中的显示内容(2)

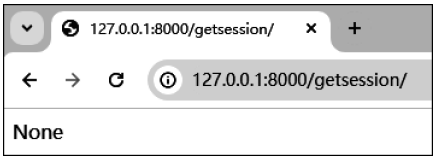


图 5-191 浏览器中的显示内容(3)

此外,可以在配置文件 settings.py 中添加变量 SESSION_ENGINE,用于对 Session 的存储机制进行修改。

(1) 数据库存储(默认),代码如下:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
```

(2) 文件存储,代码如下:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.file'
```

(3) 缓存存储,代码如下:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
```

(4) 缓存加数据库存储,代码如下:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cached_db'
```

(5) Cookie 存储,代码如下:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.signed_cookies'
```

5.10 上下文处理器

上下文处理器是一种函数,用于在模板的上下文中添加额外的数据。

通过上下文处理器,开发人员可以方便地向模板中传递额外的数据,以便模板能够渲染所需的信息。

在 Django 中,上下文处理器分为内置上下文处理器和自定义上下文处理器。

1. 内置上下文处理器

在 Django 中,内置上下文处理器均在配置文件 settings.py 中的 TEMPLATES 配置的 OPTIONS 字典的 context_processors 列表中进行添加,其常用的内置上下文处理器如表 5-13 所示。

表 5-13 常用的内置上下文处理器

内置上下文处理器	描 述
django.template.context_processors.debug	添加一个 debug 变量,当设置为 True 时,如果出现异常,则将在模板中显示详细的错误信息
django.template.context_processors.request	添加 HttpRequest 对象,用于在模板中获取一些与请求相关的信息,例如请求的方法、URL 或查询参数等
django.contrib.auth.context_processors.auth	添加一个 user 变量,用于在模板中添加有关当前用户的信息
django.contrib.messages.context_processors.messages	添加一个 messages 变量,用于在模板中添加错误、警告或任何需要展示给用户的消息

2. 自定义上下文处理器

自定义上下文处理器的步骤可以分为两步:一是在应用中创建一个名为 context_processors.py 的文件,用于自定义上下文处理器;二是在配置文件 settings.py 中注册该上下文处理器,即需要在配置文件 settings.py 中的 TEMPLATES 的 OPTIONS 字典的 context_processors 列表中添加该处理器。

下面通过一个示例演示一下如何使用上下文处理器。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令 python manage.py startapp book,创建名为 book 的应用,如图 5-192 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-192 创建应用

(3) 打开应用 book 中的文件 views.py,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\book\views.py
from django.shortcuts import render
```

```
from django.contrib import messages
def index(request):
    messages.info(request, '这是内置上下文处理器')
    return render(request, 'index.html')
```

(4) 在应用 book 中创建自定义上下文处理器文件 context_processors.py,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\book\context_processors.py
import datetime
def get_daytime(request):
    now = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    my_hour = int(datetime.datetime.now().strftime('%H'))
    if my_hour >= 5 and my_hour <= 7:
        now_message = '早上好'
    elif my_hour > 7 and my_hour <= 11:
        now_message = '上午好'
    elif my_hour > 11 and my_hour < 18:
        now_message = '下午好'
    else:
        now_message = '晚上好'
    #给模板传递所有变量
    return locals()
```

(5) 在项目根目录下的文件夹 templates 中创建模板文件 index.html,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <p>******* 内置上下文处理器 *******</p>
        <p>变量 debug: {{ debug }}</p>
        <p>HttpRequest 对象: {{ request.method }}</p>
        <p>变量 user: {{ user }}</p>
        <p>变量 messages:
            {% for message in messages %}
                {{ message }}
            {% endfor %}
        </p>
        <p>******* 自定义上下文处理器 *******</p>
        <p>
            现在时间: {{ now }}<br>
            {{ now_message }}
        </p>
    </body>
</html>
```

(6) 打开应用 djangoProject 中的配置文件 settings.py,对自定义上下文处理器进行注册,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\djangoProject\settings.py
TEMPLATES = [
```

```
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [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',
                'book.context_processors.get_daytime',
            ],
        },
    },
]
```

(7) 打开应用 djangoProject 中的文件 urls.py,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('', views.index),
]
```

(8) 打开应用 djangoProject 中的文件__init__.py,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\djangoProject\__init__.py
import pymysql
pymysql.install_as_Mysqldb()
```

(9) 打开应用 djangoProject 中的配置文件 settings.py,进行数据库的相关配置,用于连接数据库,代码如下:

```
#资源包\Code\chapter5\5.10\1\djangoProject\djangoProject\settings.py
DATABASES = {
    'default': {
        #数据库引擎
        'ENGINE': 'django.db.backends.mysql',
        #数据库名称
        'NAME': 'django_db',
        #连接 MySQL 的用户名
        'USER': 'root',
        #连接 MySQL 的密码
        'PASSWORD': '12345678',
        #MySQL 的主机地址
        'HOST': '127.0.0.1',
        #MySQL 的端口号
        'PORT': '3306',
    }
}
```

(10) 打开命令提示符窗口,进入项目所在的根目录中,激活虚拟环境,并输入命令

python manage.py migrate，将迁移脚本文件映射到数据库中，如图 5-193 所示。

(11) 运行上述项目，打开浏览器并在网址栏输入 http://127.0.0.1:8000/，其显示内容如图 5-194 所示。

图 5-193 迁移脚本文件映射到数据库中

图 5-194 浏览器中的显示内容

5.11 中间件

中间件是一个轻量级、可重用的组件，用于处理 Django 请求和响应的过程，其提供了对请求和响应进行全局处理的机制，可以在请求达到视图之前进行预处理或在响应返回客户端之前进行后处理。

中间件是按顺序依次执行的，每个中间件都可以对请求和响应进行修改、补充或处理。

中间件通过配置文件 settings.py 中的变量 MIDDLEWARE 进行配置，包括注册和执行顺序。

在 Django 中，不仅内置了许多中间件，用于实现各自所对应的功能，同时还支持自定义中间件。

1. 内置中间件

内置中间件的作用主要包括认证和授权、请求和响应处理、异常处理和性能优化等，其常用的内置中间件如表 5-14 所示。

表 5-14 常用的内置中间件

内置中间件	描 述
django.middleware.security.SecurityMiddleware	安全中间件负责处理与网站安全相关的任务
django.contrib.sessions.middleware.SessionMiddleware	会话中间件负责处理用户会话创建和检索用户数据
django.middleware.common.CommonMiddleware	通用中间件提供了一些常见而关键的 HTTP 请求处理功能
django.middleware.csrf.CsrfViewMiddleware	CSRF 中间件用于防止跨站请求伪造攻击
AuthenticationMiddleware	认证中间件负责处理用户身份认证相关的任务

续表

内置中间件	描　　述
MessageMiddleware	消息中间件用于在请求处理过程中存储和传递临时的一次性的用户消息
XFrameOptionsMiddleware	单击劫持中间件用于防止页面被嵌入其他网站中，从而提供一定的单击劫持保护

2. 自定义中间件

在 Django 中，可以通过函数或类进行中间件的自定义，并且必须将该自定义的中间件添加到配置文件 settings.py 的变量 MIDDLEWARE 中，以完成注册操作。

下面通过一个示例演示一下如何使用中间件。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-195 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-195　创建应用

(3) 打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.11\1\djangoProject\book\views.py
from django.http import HttpResponse
def index(request):
    print('视图中执行的代码')
    return HttpResponse('项目运行成功!')
```

(4) 在应用 book 中创建自定义中间件文件 middlewares.py，代码如下：

```
#资源包\Code\chapter5\5.11\1\djangoProject\book\middlewares.py
#函数
#def django_middleware(get_response):
#print('自定义中间件初始化的相关代码')
#def middleware(request):
#print('请求到达视图之前所执行的代码')
#response = get_response(request)
#print('响应到达客户端浏览器之前所执行的代码')
#return response
#return middleware
#类
class DjangoMiddleware(object):
    def __init__(self, get_response):
        print('自定义中间件初始化的相关代码')
        self.get_response = get_response
    def __call__(self, request):
        print('请求到达视图之前所执行的代码')
        response = self.get_response(request)
        print('响应到达客户端浏览器之前所执行的代码')
        return response
```

(5) 打开应用 djangoProject 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.11\1\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
    path('', views.index),
]
```

(6) 打开应用 djangoProject 中的配置文件 settings.py,对自定义中间件进行注册,代码如下:

```
#资源包\Code\chapter5\5.11\1\djangoProject\djangoProject\settings.py
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'book.middlewares.DjangoMiddleware'
]
```

(7) 运行上述项目,打开浏览器并在网址栏输入 http://127.0.0.1:8000/,其显示内容如图 5-196 所示。

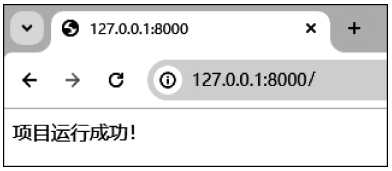


图 5-196 浏览器中的显示内容

此时,PyCharm 中的输出结果如图 5-197 所示。

图 5-197 PyCharm 中的输出结果

5.12 CSRF 防御

CSRF(Cross-Site Request Forgery,跨站请求伪造)是一种挟制用户在当前已登录的 Web 应用程序上执行非本意的操作的攻击方法。攻击者通过 HTTP 请求将数据发送到服

务器，进而盗取 Cookie。在盗取到 Cookie 之后，攻击者不仅可以获取用户的相关信息，还可以修改该 Cookie 所关联的账户信息。

Django 提供了一套基于 Token 校验的完善的 CSRF 防护体系，仅需两步即可非常简单地解决 CSRF 攻击问题，一是开启 CSRF 中间件；二是在模板的表单当中添加模板变量{% csrf_token %}。

下面通过一个示例演示一下如何进行 CSRF 防御。

(1) 创建名为 djangoProject 的 Django 项目。

(2) 打开命令提示符窗口，进入项目所在的根目录中，激活虚拟环境，并输入命令 python manage.py startapp book，创建名为 book 的应用，如图 5-198 所示。

```
E:\Python全栈开发\djangoProject>workon django_env
(django_env) E:\Python全栈开发\djangoProject>python manage.py startapp book
```

图 5-198　创建应用

(3) 打开应用 book 中的文件 views.py，代码如下：

```
#资源包\Code\chapter5\5.12\1\djangoProject\book\views.py
from django.shortcuts import render
def index(request):
   return render(request, 'index.html')
```

(4) 在项目根目录下的文件夹 templates 中创建模板文件 index.html，代码如下：

```
#资源包\Code\chapter5\5.12\1\djangoProject\templates\index.html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
    <form action="">
        {% csrf_token %}
        <p>姓名：<input type="text"></p>
        <p><input type="submit" name="提交"></p>
    </form>
    </body>
</html>
```

(5) 打开应用 djangoProject 中的文件 urls.py，代码如下：

```
#资源包\Code\chapter5\5.12\1\djangoProject\djangoProject\urls.py
from django.urls import path
from book import views
urlpatterns = [
   path('', views.index),
]
```

(6) 打开应用 djangoProject 中的配置文件 settings.py，确保 CSRF 中间件已开启。

(7) 运行上述项目，打开浏览器并在网址栏输入 http://127.0.0.1:8000/，其显示内容如图 5-199 所示。

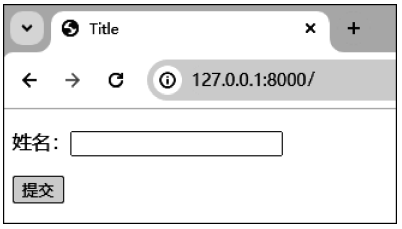


图 5-199 浏览器中的显示内容

(8) 右击,单击“查看网页源代码”。此时,在 HTML 中创建了一个隐藏的字段,其中包含了 CSRF 令牌,其显示内容如图 5-200 所示。

图 5-200 浏览器中的显示内容