# Django Web 框架讲稿

# 讲稿 1 Django 框架介绍与安装

## 介绍

本次课程将详细介绍 Django 框架在 Web 开发过程中的应用，包括 MVT 模型、模型类的编写、数据模型关系、创建数据表、路由基础知识、HTTP 请求方法、Cookie & session 、视图函数的创建、模板文件的配置与后台管理功能等知识点。

本部分主要来了解什么是 BS 架构、MVT 框架的构成、 Django 框架的简介与安装。

#### 知识点

* BS 架构简介
* MVT 结构
* Django 框架简介
* 安装 Django

## 什么是 BS 架构

要了解什么是 BS 架构，就要从什么是架构说起。

架构是什么意思？为什么要有架构？

编程的一个默认原则是 Don't Repeat Yourself ，即“不要重复编写代码”，也有翻译为：“不要重复造轮子（wheel）”，简称“DRY 原则”。编程之初，人们为了避免重复写代码，创建了函数：

# 举个简单的例子，以下为 Python 代码

>>> print('Hello, I am {}'.format('小明'))

Hello, I am 小明

>>> print('Hello, I am {}'.format('詹姆斯'))

Hello, I am 詹姆斯

>>> print('Hello, I am {}'.format('周芷若'))

Hello, I am 周芷若

>>>

# 为了避免重复，创建一个 hello 函数

>>> def hello(name):

... print('Hello, I am', name)

...

...

>>> hello('小明')

Hello, I am 小明

>>> hello('詹姆斯')

Hello, I am 詹姆斯

>>> hello('周芷若')

Hello, I am 周芷若

>>>

后来在一些编程语言中引入了面向对象的概念，将一些有共性的函数放到类里面，称之为方法。对类进行实例化，使用实例的属性或方法实现一些功能，这样在稍复杂场景中可以更加有效地避免编写重复代码：

>>> class Student:

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

...

... def hello(self):

... print('Hello, I am', self.name)

...

...

...

>>> st = Student('Nash', 12)

>>> st.hello()

Hello, I am Nash

>>>

场景更加复杂时，多个类之中还是会重复编写代码。为了进一步优化，又出现了继承的概念，子类继承父类并拥有父类全部属性和方法（以双下划线开头的私有属性或方法除外，这是Python语法的规定）。

以上所述在 Python 基础编程中随处可见，是自然而然的事情，它们是几代（其实，也没几代~~）编程人员总结归纳创新的结果。

在更加复杂的情况下，人们为了避免重复的劳动，提高工作效率，又提出来「设计模式」的概念。一个良好的设计模式在实践中不仅可以避免编写重复的代码，而且能够使程序的执行效率更高、更易于维护和更新功能。

设计模式是一个广泛的概念，其中包括一种叫做 “架构模式” 的概念。架构模式有很多种，例如分层模式是十分常见的一种，它在网络编程中一定会被提到，大家熟知的 OSI 七层模型，就是一种分层架构。分层架构把一个数据传输的模型分成若干层，每一层使用独特的协议对数据作一次包装或转换，然后传到下一层，最后合起来完成数据的传输。

设计模式中还有一个叫做 “客户端/服务器模式”（Client - Server Model）的概念，一个服务器对应多个客户端：客户端发起服务请求，服务器提供服务。例如日常使用的音视频软件，在手机或电脑上下载安装客户端 APP ，注册登录之后就可以听音乐看电影了。在这种模式中，客户端向服务器发送请求，服务器返回给客户端对应的资源。

这种模式就是被称作 CS 架构的模式， BS 架构是另一种设计模式（其实，是BS架构模式的一个特殊情况）。

BS 架构就是 Browser - Server Model ，以浏览器作为客户端向服务器发送请求。这种模式的优势就是不需要用户安装单独的客户端，只需要有一个浏览器即可访问任意服务器。CS 架构中客户端需要不定期更新，以支持服务器提供的最新功能，而 BS 模式下服务器只需要对浏览器进行适配开发即可。

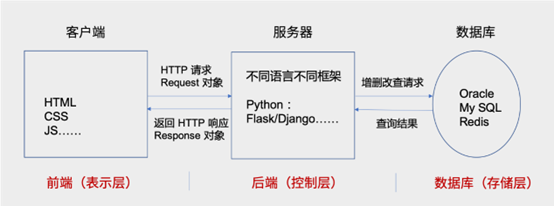
在手机上安装的各种 APP ，会频繁更新，有些时候需要更新才能用。在急需使用的时候还要等待下载新版本和安装，有些烦人。BS 架构下没有这种困扰，浏览器更新的频率也相对较低。

## MVT 模式

下面，简单描述一次完整的浏览器打开页面的操作：

* 在浏览器的地址栏输入网址，或者在搜索引擎的搜索框输入关键字，然后敲回车；
* 浏览器会把页面请求发送给对应的服务器；
* 服务器收到请求，处理请求，返回响应；
* 浏览器收到响应，解析成页面显示在屏幕上。

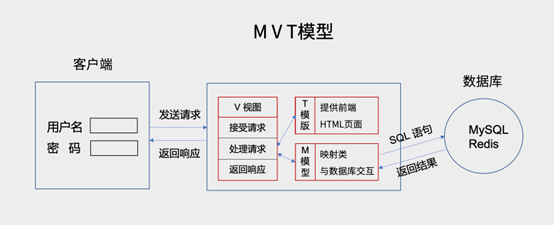
作为用户，使用浏览器操作即可，不关心服务器是怎么工作的。作为 Web 开发人员，要做的就是开发服务器（其上的应用程序）。



如上图所示，当使用浏览器打开网站的页面：

* 客户端浏览器向服务器发送一个页面请求，请求对象中包含 URL 信息、浏览器版本、Cookie 等信息。
* 服务器作为后端控制层，从请求对象中提取 URL 字符串，解析分配给对应的视图函数，由视图函数进行处理。
* 视图函数需要查询数据库中的课程数据，数据库将查询结果返回给视图函数。
* 视图函数将处理结果打包成 Response 对象返回给浏览器。
* 浏览器收到响应，存储 Cookie 、解析 JS 数据，将完整的页面显示在屏幕上。

MVT 框架如下图所示：



在收到浏览器的请求后，首先由视图函数来接受请求。视图函数在处理请求的过程中需要模板来配合，模板提供页面的所有静态资源，包括 HTML 、CSS 、JS 数据，它们是渲染页面的基础数据。页面中的动态数据（以发布课程的网站服务器为例），例如课程名称、简介、学习人数等数据需要从数据库中获取，这需要创建模型类（也叫映射类）来与数据库交互。

视图函数即 View ，模型类即 Model ，前端模板即为 Templates ，合起来简称 MVT 。这种设计模式就叫做 MVT 设计模式。

## Django 简介

Django 是一个由 Python 编写的开放源代码的 Web 应用程序开发框架。它采用了 MVT 软件设计模式，即 Model 模型、View 视图和 Templates 模板。它最初被开发出来用于管理某新闻网站，于 2005 年在 BSD 许可下发布开源。这套框架是以比利时的吉普赛吉他手的名字命名的。

Django 的主要目标是使得开发 “流程复杂、依赖数据库的网站” 变得简单。Django 注重各个组件间的低耦合性，遵循 DRY 原则。

Django 框架的核心包括：

* 面向对象的关系映射模型，支持编写对应数据表的映射类，这一点做得格外优秀；
* URL 分发器，基于正则表达式（Django 中支持 path 编写简洁路径字段）；
* 视图系统，用于接收客户端请求和返回响应；
* 模板系统，内置 Jinja2 模板引擎，便于编写 Python 风格的动态数据处理代码；
* 轻量、独立的 Web 服务器，便于开发测试；
* 缓存框架，支持多种缓存方式；
* 中间件，对请求处理阶段进行额外操作。

同时 Django 内置了一些针对新闻类网站的基础且实用的应用：

* 可扩展的认证系统
* 动态站点管理页面
* 灵活的评论系统
* CSRF Token 工具
* 标记语言模板库

在使用 Python 进行 Web 开发时，除了 Django 以外还有多种框架选择：

* Flask 很轻量，可以迅速搭建一个简单的 Web 网站。几乎没有组件限制，提供大量功能丰富且实用的插件，例如 Flask-Admin 后台管理插件、Flask-WTForm 表单管理插件、Flask-Login 登录功能插件、Flask-Migrate 数据表版本控制插件等。著名的科学网站果壳网就是使用 Flask 开发的。
* Tornado 性能较高，支持异步处理，这是它最大的亮点。对 Restful API 设计良好，但没有针对数据库设计的优秀的 ORM 模块，这是比较麻烦的。
* Bottle 更加轻量，只依赖于 Python 标准库，无法做到快速建站，在 Flask 面前几乎没有优势可言。
* Aiohttp 是比较新的框架，它需要基于 asyncio 编写异步代码，性能方面不必担心，但框架完善还需要较长时间。

此外还有数种小众框架，它们都有一些独到之处，但在商用上的前途比较渺茫。对 Python Web 开发而言，小的网站会使用 Flask ，大型的网站就要用 Django 。

## 换一种讲法

实际授课的时候，结合情况，可以换一种讲法。

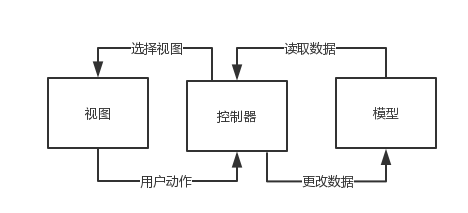
#### MVC

在说 MTV 模式之前，简单的说说著名的 MVC 模式。

MVC，是模型（Model）-视图（View）-控制器（Controller）的缩写。其具体定义如下：

* M：模型（Model），数据存取层，负责业务对象和数据库对象。
* V：视图（View），与用户的交互，负责显示与怎样显示。
* C：控制器（Controller），接受用户动作，调用模型，输出相应视图。

三者以一种插件似的，松耦合的方式连接在一起。



#### MTV

Django 的 MTV 设计模式是借鉴和遵循 MVC 的。

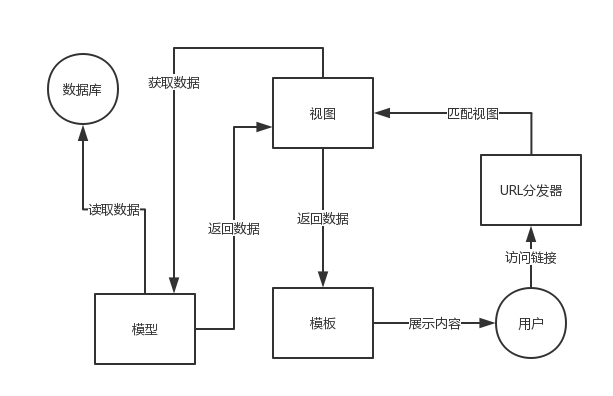
MTV 具体定义如下：

* M：模型（Model）,负责业务对象和数据库的关系映射。
* T：模板（Template）,负责如何把页面展示给用户。
* V：视图（View）,负责业务逻辑，并在适当时候调用模型和模板。

#### URL 分发器

URL 分发器的作用是将页面请求分发给不同的视图（View）处理，视图再调用相应的模型（Model）和模板（Template）。

Django Web 框架：



一个误区是把 MVC 模式与 MTV 模式等价替换。其实这样是不对的。

在 MTV 模式中，MVC 中的 View 分成了视图 View（展现哪些数据）和模板 Template（如何展现）2 个部分，而控制器（Controller）这个要素由框架自己来实现了，需要做的就是把 URL 对应到视图 V 就可以了，通过这样的 URL 配置，系统将一个请求发送到一个合适的视图。

## 安装 Django

作为 Python 编写的 Web 开发框架，安装自然是极简单的事，使用 pip 包管理工具即可一步到位。

本机环境中，应该提前已经安装了 Python 和 对应的包管理工具 pip ，在终端命令行更新 pip ：

pip install -U pip

为了周全考虑，针对一个工程创建一个虚拟环境。（以Python语言环境为例，Anaconda与之类似，上课时，针对学生需要，也可以演示下Anaconda的虚拟环境的创建和使用，btw：最好放在环境安装视频中演示下~~）

更新 virtualenv 库，

pip install -U virtualenv

切换到Python虚拟环境容器目录下（如：c:\Dev\Python\venvs）。

然后创建名为 venv 的虚拟环境。使用 virtualenv 创建虚拟环境，其中 -p 选项后面指定 Python 解释器，最后一个参数 venv 是虚拟环境的目录名：

virtualenv -p python venv

或者：

virtualenv venv

进入虚拟环境，安装 Django ：

在 venv/Scripts 目录下的命令行界面中：

activate

pip install django

pip show django

执行 pip show 命令可以查看虚拟环境中安装的指定工具包及其对应的版本号。

额外的，如果没有安装jupyter notebook，也需要安装，安装方法：

在虚拟环境中，执行：

pip install jupyter

或者，建议，再安装jupyter lab，安装方法：

在虚拟环境中，执行：

pip install jupyterlab

Django 已经安装完毕，它存在于虚拟环境 venv 中。

Python 中连接MySQL可选的第三方模块|工具包的选用：

[(27条消息) MySQLdb、Mysqlclient、PyMySQL 三个python的MySQL库的比较和总结\_mysqldb pymysql\_vicent\_uuid4的博客-CSDN博客](https://blog.csdn.net/weixin_48943299/article/details/122868470)

[(27条消息) mysqlclient和pymysql如何选择？\_gevent\_waiter的使用\_果汁华的博客-CSDN博客](https://blog.csdn.net/u011510825/article/details/86632598)

结论是：推荐选用 mysqlclient，然后:

[(27条消息) 使用pip install mysqlclient命令安装mysqlclient失败？\_Mr丶D的博客-CSDN博客](https://blog.csdn.net/cn_1937/article/details/81533544)

[Archived: Python Extension Packages for Windows - Christoph Gohlke (uci.edu)](https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient)

**注意：特别需要注意版面的对应关系。**

* **如果是：Python3.8，选用：**[mysqlclient‑1.4.6‑cp38‑cp38‑win\_amd64.whl](javascript:;)
* **如果是：Python3.9，选用：**[mysqlclient‑1.4.6‑cp39‑cp39‑win\_amd64.whl](javascript:;)

安装mysqlclient，直接安装（Windows下不推荐）：

# 安装 mysqlclient

pip install mysqlclient

推荐安装方法：

1. 下载 lfd 的安装文件，如：mysqlclient‑1.4.6‑cp38‑cp38‑win\_amd64.whl
2. 在对应的环境中（如上述激活的的 venv 中，执行：

pip install mysqlclient‑1.4.6‑cp38‑cp38‑win\_amd64.whl

到此为止，准备工作已经做完。

## 总结

本部分了解了什么是 BS 架构，以及为什么要使用架构，架构的作用是什么。此外介绍了 MVT 模式的设计思路以及视图函数、模板、模型类之间的协调关系。最后了解了 Django 框架的核心功能和优势以及在本机环境中安装 Django 的操作。

下一部分将学习使用 Django 创建一个 Web 项目。

## 作业：

在本机环境中，自行搭建虚拟环境并在其中完成Django和mysqlclient、jupyter notebook的安装，安装后，使用 pip show 命令查看并将查看的显示结果界面截图。

# 讲稿 2 最简单的 Django 项目

## 介绍

上一部分了解了 Django 的基本情况，并在本机环境中简单搭建了 Django 的工作环境。

从这一部分正式开始学习使用 Django 进行 Web 开发。本部分主要来熟悉一下 Django 基本命令和项目的目录结构。

#### 知识点

* 初始化项目
* 创建应用
* 注册应用
* 启动项目
* 创建视图函数
* 配置数据库

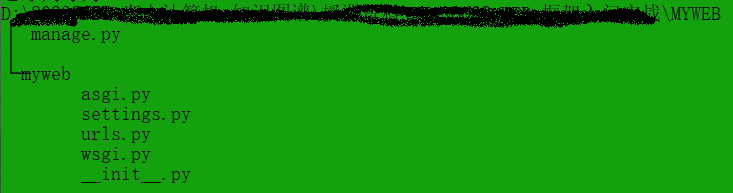
## 初始化 Django 项目

接续上一部分的操作，在虚拟环境容器（如：C:\Dev\Python\venvs\venv） 目录下执行 Scripts/activate 命令进入虚拟环境。然后执行如下命令初始化一个 Django 项目，项目的名称定为 myweb。此时当前目录下应有两个子目录 venv 和 myweb ，它们分别是虚拟环境目录和 Django 项目目录。最后执行 tree myweb 命令查看项目的目录结构：

django-admin startproject myweb

dir

tree myweb



创建项目的命令是 django-admin startproject ，后面的参数为项目名称。顺利的话，会在当前目录下生成一个项目的目录结构，目录名为 myweb ，它是项目的主目录。

对项目主目录下的文件和目录依次说明：

* manage.py 项目的入口文件，在后面的讲授中会大量使用它来执行一些命令用来创建应用、启动项目、控制数据表迁移等。
* myweb 主目录下的同名子目录，为项目的核心目录，它里面包含配置文件和管理应用的文件。
* myweb/\_\_init\_\_.py 每个子目录都会包含这样一个 \_\_init\_\_.py 文件，它是一个空文件，在需要的时候会引入目录下的对象。
* myweb/settings.py 配置文件，里面包含对数据库的设置项、CSRF Token 的设置项、模板的设置项等全部设置。
* myweb/urls.py 路由控制文件，处理客户端请求，分发到对应的视图函数去处理。
* myweb/wsgi.py 部署服务器处理请求和响应，很少去动它。
* myweb/asgi.py 异步部署服务器模块，这是 Django 最新版本中增加的功能，以支持 ASGI 异步部署，暂时用不到。

## 创建应用

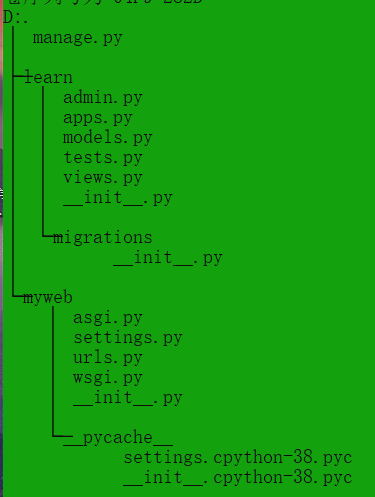
类似一个公司有生产部门、销售部门、财务部门，一个 Web 项目也可以分成多个部分，可以称它们为应用（application）。

首先在终端执行 cd myweb 命令进入到项目的主目录，然后执行如下命令创建一个名为 learn 的应用，注意本机环境里不可以创建名为 test 的应用，某些情况下它与 Python 模块冲突：

cd myweb

python manage.py startapp learn

操作截图如下：



如上所示，创建了名为 learn 的应用后，在项目的主目录下出现了名为 learn 的目录，这就是应用目录。

大型项目中，通常会有多个应用，learn1 、learn2 、learn3 等等，每个应用都会在主目录 myweb 下拥有自己的目录结构。

应用中的文件说明如下：

* learn/admin.py 用于控制后台管理的文件，在后面的讲授中会用到。
* learn/apps.py 用于管理应用本身的文件，包括应用的名字如何命名，默认就是 learn 。
* learn/\_\_init\_\_.py 空文件，前面已经介绍过。
* learn/migrations 这是用于记录数据库变更信息的目录，Django 中自带的数据库版本控制功能就体现在这个目录，在介绍数据存储时会详细介绍。
* learn/models.py 创建映射类的文件（问下学生是否熟悉 Flask ，如果熟悉，就一定不陌生）。
* learn/tests.py 编写测试代码的文件。
* learn/views.py 创建视图函数的文件，视图函数用于处理客户端发来的请求。

### 注册应用

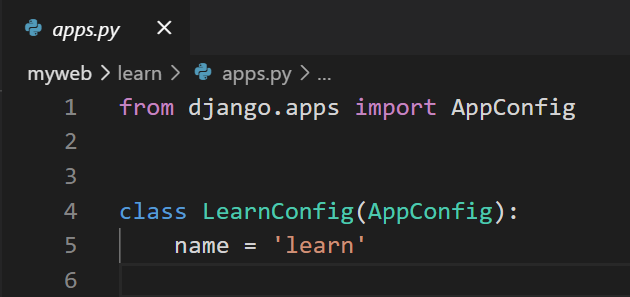
创建一个应用后，该应用并未与项目产生联系，需要在配置文件中添加此应用。

配置文件就是项目主目录 myweb 下的 myweb/settings.py 文件，打开它，修改其中的 INSTALLED\_APPS 项。注意，所有的配置项都是英文全大写，多个单词用下划线连接。INSTALLED\_APPS 是一个列表，用于保存所有注册应用，其中已经存在了一些默认的核心应用，只需要将 'learn' 字符串添加到其中：



注意：结尾需加上逗号。

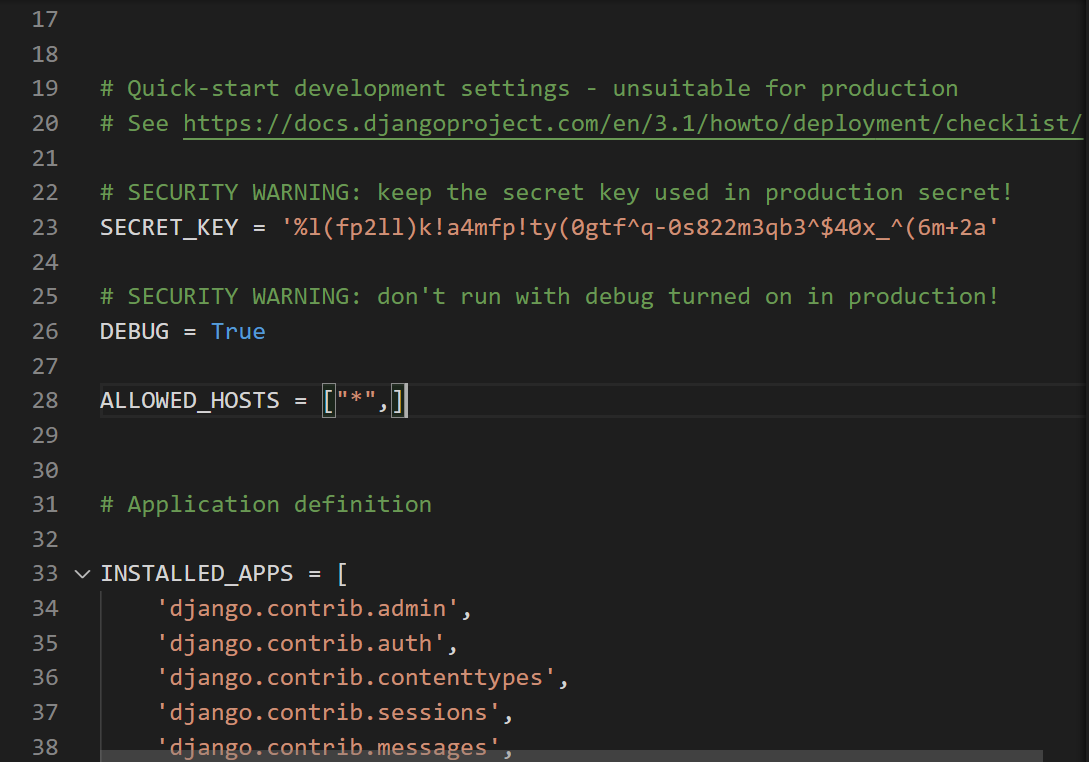
为什么是 learn 呢？因为应用名字就是 learn ，可以在 learn/apps.py 文件中查看，这是在创建应用时默认配置好的：



### 调整路由

由于希望是在本机环境中通过 Web 服务来查看项目，所以需要在配置文件中允许 Host 访问。

同样是项目主目录 myweb 下的 myweb/settings.py 文件，打开它，修改其中的 ALLOWED\_HOSTS 项。将其赋值为["\*",]。



## 启动项目

在项目的主目录下执行终端命令，启动项目：

终端命令行上执行此命令：

python manage.py runserver

显示界面：



其中 manage.py 作为入口文件，它有很多选项提供各种各样的功能。runserver 为启动项目的选项，上图倒数第二行是主页地址和端口号，默认的端口是 8000 。

此时，在浏览器中，使用 http://127.0.0.1:8000 访问。

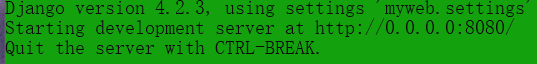
可以使用 python manage.py --help 命令查看全部功能：

推荐使用如下命令（问下学生，why）

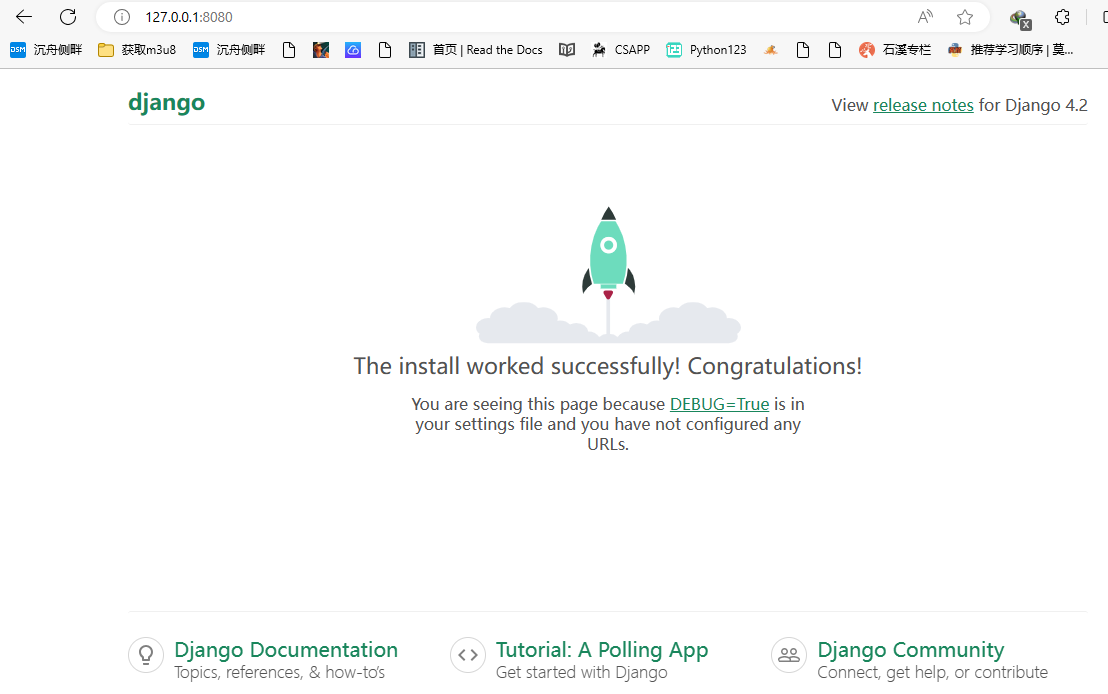
重新使用代码指定端口启动项目：

python manage.py runserver 0.0.0.0:8080

显示界面：



在浏览器中输入：http://127.0.0.1:8080，启动项目后，打开启动 **Web 服务** 即可呈现 Django 的欢迎页：



如上图所示，左上角为 Django 标识，右上角为版本号，中间是一个腾空而起的小火箭，底部是 Django 的文档、教程和社区链接。

Django 努力做到了国际通用，所以可以将欢迎页适配为中文。依然修改配置文件 myweb/settings.py ，将 LANGUAGE\_CODE 和 TIME\_ZONE 修改如下：



其中 'zh-hans' 表示语言为中文，'Asia/Shanghai' 表示时区为东八区，也就是北京时间。

修改完成后保存文件，刷新页面即可看到如下样式：



这就是启动项目的过程。

视情况，让学生完成上述操作。

## 创建主页的视图函数

截止目前，还没有写过任何有效代码，只是修改了配置文件。

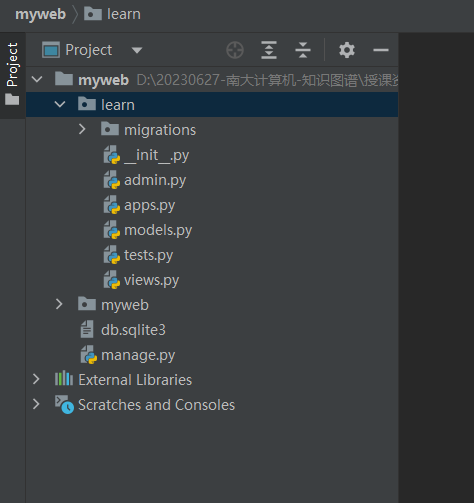
启动项目之后，之所以会显示欢迎页，是因为Web 主页没有设置，而且项目默认开启了 DEBUG 模式。

现在编写第一个视图函数，主页的视图函数。

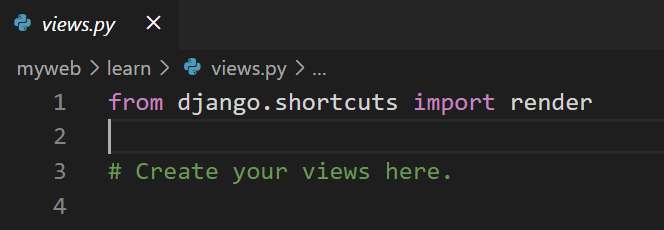
使用编辑器（学生自行选择VSCode或者PyCharm，讲授时，使用 PyCharm）

在PyCharm中打开项目主目录（myweb）：

**需要注意设置好 Python Interpreter**



打开 learn/views.py 文件：



如上图所示，文件中已经有了一行代码，补充视图函数代码如下：

from django.shortcuts import render

from django.http import HttpResponse

def index(request):

return HttpResponse('Hello World.')

首先引入 HttpResponse 类，它的实例即为服务器的响应对象。

创建视图函数 index ，参数为 request ，也就是客户端（浏览器）发送过来的请求对象。request 为必选的参数，即使视图函数中没有用到也要写上，参数名可以自定义，写成 req 也行。

函数的返回值须为响应对象，也就是 HttpResponse 的实例，它接收一个字符串作为参数，这个参数就是响应对象中的 content ，也就是文本内容。

保存修改，简易的视图函数就完成了。此处，需要引导学生思考一下，视图函数是有了，它映射到那个 URL 路径下呢？也就是说浏览器访问了哪个网址，才会调用这个视图函数处理？

：需要修改 myweb 目录下的 urls.py 文件，它是用来控制视图函数与 URLS 的匹配的文件。

打开 myweb/urls.py 文件，首先引入刚刚写好的视图函数，然后在 urlpatterns 列表中增加一项：

from django.contrib import admin

from django.urls import path

from learn.views import index # 引入视图函数

urlpatterns = [

path('admin/', admin.site.urls),

path('', index), # 新增路由映射

#path('hw', index), # 新增路由映射，换个映射url地址

]

最外层的 myweb 目录也就是 myweb 为项目的主目录，引入其它文件中的对象，可以使用绝对路径或相对路径。

urlpatterns 为路由映射到视图函数的控制列表，当服务器收到浏览器发送过来的请求时，首先到这里检查是否有对应的视图函数。如果是 [http://www.xxx.com](https://www.xxx.com/) 就让 index 来处理（对应空串’’的情况）， 如果是 [http://www.xxx.com/admin/](https://www.xxx.com/admin/) 就让 admin.site.urls 来处理（对应”admin/”字符串）。

保存修改后刷新页面，即可看到 “Hello World.” 字样：



这样就可以看到定义的 index 视图函数生效了。

## 配置数据库

一个网站通常要搭配一个或多个数据库用于存储各种数据，Django 提供 Sqlite3 、MySQL 、Oracle 等各种主流数据库的支持。

默认的配置是使用 Sqlite3 这个 Python 原生数据库，它就相当于一个单机不联网的数据库，通常用于测试项目。

发现启动项目之后，终端会打印一行红色的警告信息到屏幕上：



这是因为没有对数据库进行初始化。

关闭服务（在终端输入：ctrl+c），在终端执行如下语句即可创建数据库的迁移文件：

python manage.py makemigrations

结果界面：

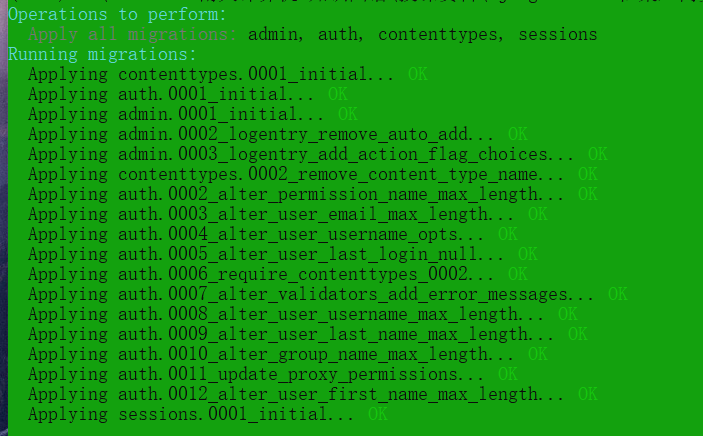


因为没有创建映射类，所以没有任何变化。

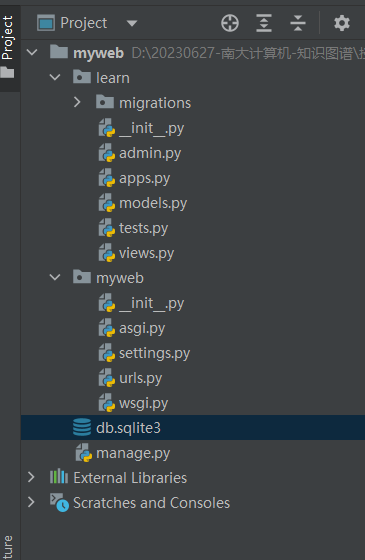
然后执行如下命令进行数据库的初始化：

python manage.py migrate

结果界面：



此时当前目录下会生成一个 db.sqlite3 文件，它就是 Sqlite3 的数据库文件：



初始化时，会在数据库中创建多个数据表，其中包括权限管理、后台管理等默认功能相关的表。

实际工程中，需要修改数据库的配置，使用 MySQL 作为数据库。

修改配置文件 myweb/settings.py 文件中的 DATABASES 项如下：

# 原来的代码

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

# 修改后

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'myweb',

'USER': 'root',

'PASSWORD': 'root',

'HOST': '127.0.0.1',

'PORT': 3306

}

}

对其中各项进行说明：

* ENGINE 是连接数据库的引擎，将其修改为 mysql
* NAME 是数据库的名字，设置为项目名称（myweb），或自行定义名称
* USER 为数据库的用户名，默认就是 root
* PASSWORD 为数据库用户 root 对应的密码，本机环境中设置密码，其值为root
* HOST 为主机名，或者叫 IP 地址，本地默认为 127.0.0.1
* PORT 为端口号，MySQL 启动后会默认监听在 3306 端口

改好了配置文件，需要启动本机环境中的 MySQL 服务的状态。启动成功后，创建 myweb 数据库：

# 启动 MySQL 服务，注意：需要在管理员权限的终端窗口中执行

net start mysql

# 创建 myweb 数据库，普通终端窗口中执行也可

# 登录进入 mysql 命令行界面，需要输入mysql的root用户的密码

mysql -u root -p

# 创建 myweb 数据库

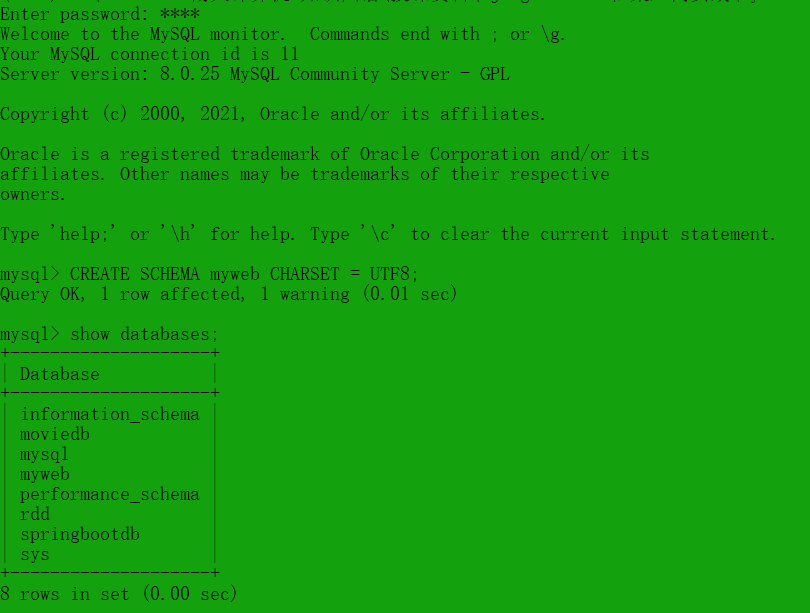
# 进入mysql命令行界面后，在 mysql> 状态下，执行：

CREATE SCHEMA myweb CHARSET = UTF8;

# 查看mysql中的数据库，执行：

show databases;

操作截图如下：



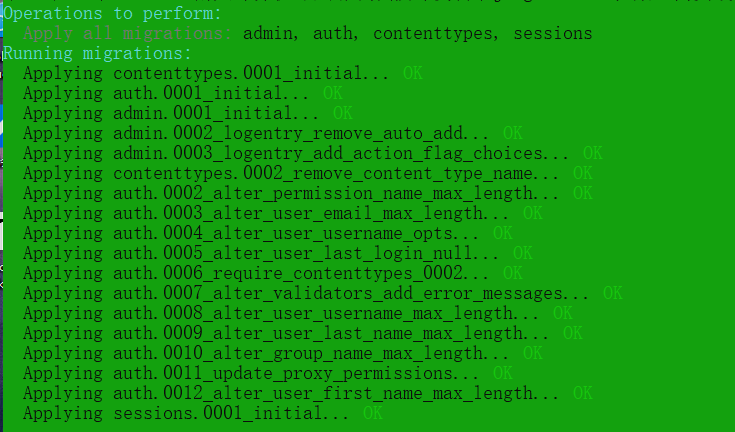
如上图所示，倒数第二个命令用来创建数据库，注意要增加 CHARSET = UTF8 这个参数，其作用是设置数据库的编码类似为 UTF-8 。（强调下）在创建数据库时记得加上这个参数。

完成以上操作后，再次执行创建数据表迁移文件和执行迁移的操作：

python manage.py makemigrations

python manage.py migrate

结果界面：



**特别注意：**

如果报错：django.db.utils.OperationalError: (2059, <NULL>)

原因：目前最新的mysql8.0对用户密码的加密方式为caching\_sha2\_password, django暂时还不支持这种新增的加密方式。只需要将用户加密方式改为老的加密方式即可。

解决方案：

以下命令在cmd窗口下完成。

1.登录mysql，连接用户为root。

> mysql -u root -p

2.执行命令查看加密方式

> use mysql;

> select user,plugin from user where user='root';

3.执行命令修改加密方式

> alter user 'root'@'localhost' identified with mysql\_native\_password by 'yourpassword'

即：

> alter user 'root'@'%' identified with mysql\_native\_password by 'root'

为啥是：root@%

问下学生，如果不知道，再讲下“root登录位置声明|指定”

4.属性权限使配置生效

> flush privileges

5.验证修改

select user,plugin from user where user='root';

重设mysql8.0的加密方式后，再次启动django服务器就没有任何问题了。

参考：[(27条消息) Django数据迁移:django.db.utils.OperationalError: (2059, ＜NULL＞) ----2059错误解决方案\_明 总 有的博客-CSDN博客](https://blog.csdn.net/qq_45720042/article/details/116459962)

进入 MySQL 客户端，执行：

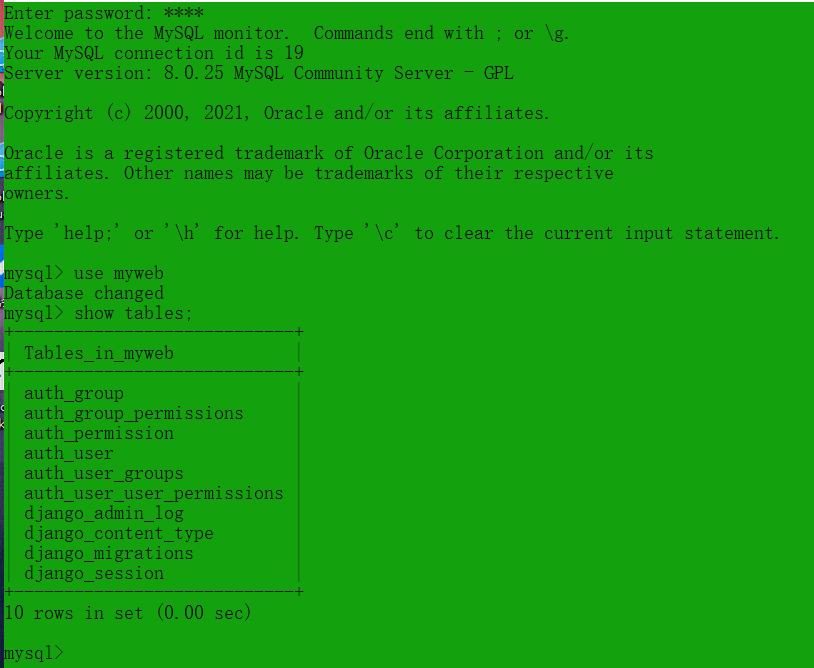
mysql -u root -p

输入root用户密码后，进入mysql命令行终端，输入：

use myweb;

show tables;

查看已经创建的数据表：



好，现在，完成了数据库的配置工作。

重新启动项目(python manage.py runserver 0.0.0.0:8080)，会发现之前的红色警告不见了。

这样就表示成功配置了数据库。

## 总结

本部分的项目示例代码：wgcode02.zip

本部分内容较为基础。首先创建了 myweb 项目，在项目中创建了 learn 应用并注册。启动项目时，记得修改配置文件中的 LANGUAGE\_CODE 、ALLOWED\_HOSTS 和 TIME\_ZONE 项。然后创建了首页的视图函数，并修改 myweb/urls.py 文件中的 urlpatterns 列表，添加一个路由配置项。最后修改配置文件并成功将项目的数据库修改为 MySQL 。

到此为止，对 Django 框架有了初步认识。

下一部分，将深入学习与数据库的交互操作。

## 作业：

自行总结创建Django项目的步骤和注意事项，并简述其中涉及的相关文件的作用。

# 讲稿 3 创建数据库映射类

## 介绍

本部分学习如何在 Django 项目中创建数据库的模型类（也叫映射类）。

首先了解 ORM 框架这个概念，知道映射类如何创建数据库中的数据表，然后学习使用映射类创建数据和管理数据。

#### 知识点

* ORM 框架
* 编写映射类
* 使用映射类创建数据表
* 向数据表中添加数据的多种方式

## ORM 框架

ORM 全称是 Object Relational Mapping ，翻译成中文就是 “对象关系映射”。其主要作用是在编程中，把面向对象的概念跟数据库中表的概念对应起来：

* Python 映射类 --> 数据表
* 映射类实例 --> 数据表中的一条数据
* 实例的属性 --> 数据的字段

[Orm - Wikipedia](https://en.wikipedia.org/wiki/Orm)

[Object–relational mapping - Wikipedia](https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping)

[对象关系映射 - 维基百科，自由的百科全书 (wikipedia.org)](https://zh.wikipedia.org/wiki/%E5%AF%B9%E8%B1%A1%E5%85%B3%E7%B3%BB%E6%98%A0%E5%B0%84)

百度百科（恶心也用着~~，学生未必能有上Wiki）中的解释如下：

对象关系映射（Object Relational Mapping，简称 ORM）是通过使用描述对象和数据库之间映射的元数据，将面向对象语言程序中的对象自动持久化到关系数据库中。本质上就是将数据从一种形式转换到另外一种形式，这也同时暗示着额外的执行开销。然而，如果 ORM 作为一种中间件实现，则会有很多机会做优化，而这些在手写的持久层并不存在。 更重要的是用于控制转换的元数据需要提供和管理；但是同样，这些花费要比维护手写的方案要少；而且就算是遵守 ODMG 规范的对象数据库依然需要类级别的元数据。

简言之就是使用 ORM 框架有助于编程人员省去编写复杂的 SQL 语句的麻烦，但将编程语言的代码转换成 SQL 语句也是有较大消耗的，不过这种消耗有优化的可能性。

如果在 Flask 中，需要安装 Flask-SQLAlchemy 插件来实现模型类和数据表的映射，同时需要安装 Flask-Migrate 插件来实现数据表版本控制。

在大而全的 Django 框架中自带了这些功能，而且做得极其优秀。

## 编写第一个映射类

在上一部分中，已经成功将 myweb 项目中使用的数据库配置成了 MySQL ，现在可以立即开始编写映射类了。

记得先启动 MySQL 服务（管理员终端界面中）：

net start mysql

提出问题（需求）：现在需要在 learn 应用下创建一个数据表 course ，对应的映射类名字为 Course 。表中数据设置四个字段，分别是主键 id 、课程名字 name 、发布时间 pub\_date 和学习人数 stu\_number 。

注（此处需要对学生强调）：下面文档中的相对路径基于项目主目录，learn/models.py 指的就是 myweb 目录下的子目录和子文件。后续不再重复说明。

用编辑器打开 learn/models.py ，该应用的所有映射类都写到这个文件中。将以下代码写入其中：

from django.db import models

class Course(models.Model):

'''课程表映射类，包括课程名字、发布时间和学习人数等属性

'''

name = models.CharField(max\_length=64)

pub\_date = models.DateField()

stu\_number = models.IntegerField(default=0)

def \_\_str\_\_(self):

return self.name

def \_\_repr\_\_(self):

return f'<Course: {self.name}>'

其中第 1 行为默认存在，引入 models 模块。创建映射类需要继承 models.Model 这个父类。三引号（块注释、多行注释）中写入映射类的描述信息，也叫类的文档。

定义属性的语法格式：属性 = models.字段类型(参数) 。

类内部的第 1 行代码定义了 name 属性，models.CharField 设置该属性对应到数据表中的字段类型为字符串型，参数 max\_length=64 设置字段最大长度。

类内部第 2 行代码定义了 pub\_date 属性，models.DateField 设置该属性对应到数据表中的字段类型为 Date 日期类型。

类内部的第 3 行代码定义了 stu\_number 属性，models.IntegerField 设置该属性对应到数据表中的字段类型为 Integer 类型，参数 default=0 设置字段的默认值为 0 。

注意 String 、Date 和 Integer 字段的非空约束 NULL 都是默认为 False ，也就是字段值不可以为空。

最后定义了 \_\_str\_\_ 和 \_\_repr\_\_ 方法，其目的是为了便于查看映射类实例的打印信息。

代码中并未设置 id 字段，因为 Django 的 ORM 会自动创建一个自增的 int 类型的 id 字段作为主键。

**字段的数据类型及其说明：**

| **数据类型** | **说明** |
| --- | --- |
| AutoField | 默认自增的 int 类型，通常不用指定，Django 会自动创建属性名为 id 的主键属性 |
| CharField | 字符串类型，必选参数 max\_length 表示最大字符个数 |
| IntegerField | 整数类型 |
| DecimalField | 十进制浮点数，可选参数 max\_digits 表示总位数，decimal\_places 表示小数位数 |
| FloatField | 浮点数 |
| DateField | 日期类型，可选参数 auto\_now 自动设置该字段为当前时间， auto\_now\_add 当对象第一次被创建时自动设置当前时间。 auto\_now\_add 和 auto\_now 互斥，选择其一即可 |
| TimeField | 时间类型，参数同上 |
| DateTimeField | 日期时间类型，参数同上 |
| FileField | 上传文件字段 |
| ImageField | 继承于 FileField ，对上传的内容进行校验，确保是合法的图片 |
| BooleanField | 布尔字段，值为 True 或 False |
| NullBooleanField | 支持 Null、True、False 三种值 |

**数据类型常用参数：**

| **参数** | **说明** |
| --- | --- |
| primary\_key | 设置主键约束，默认为 False，通常作为 AutoField 的默认参数 |
| unique | 设置唯一约束，默认为 False |
| default | 设置字段默认值 |
| db\_column | 指定字段的名称，缺省使用属性名 |
| db\_index | 表示为此字段创建索引，默认为 False |
| null | 设置非空约束，默认为 False ，即字段不可为空 |
| blank | 表示字段允许为空，默认为 False，即字段不可为空 |

注意：null 设置向数据库插入数据时字段能否为空，blank 设置前端填写表单时字段能否为空。通常 null = True 时 blank 也被设置为 True 。

### 创建数据表

（先提问学生）创建完映射类，如何使用它创建数据表呢？

下面的操作可以先停止Django服务后执行，也可以在PyCharm的Terminal中执行。

这步操作在上一部分中已经操作过，共分为两步：

# 创建数据库迁移文件

python manage.py makemigrations

# 执行数据库迁移

python manage.py migrate

何谓创建数据库迁移文件呢？在介绍 Django 项目的目录结构时，提到过 learn 目录下的 migrations 子目录，它就是用来存放数据库迁移文件的目录。此时该目录下只有一个空文件 \_\_init\_\_.py ：（记得给学生看一下）

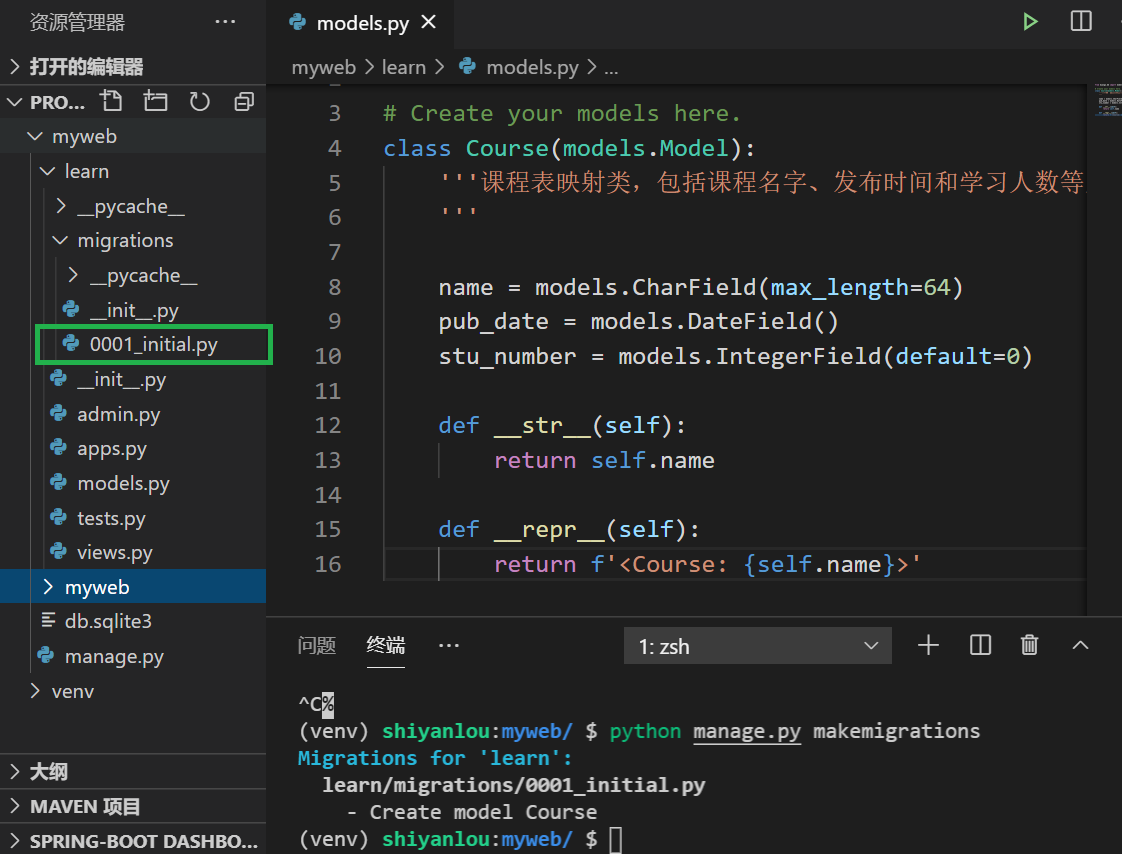
因为已经创建了一个 Course 映射类，现在执行创建数据库迁移文件的命令就会在 learn/migrations 目录下生成一个关于此映射类的迁移文件。

在终端执行如下命令：

cd myweb

python manage.py makemigrations

再次查看 learn/migrations 目录结构：



可以看到新增的 0001\_initial.py 文件，这就是迁移文件。打开它，查看文件内容：



如上图所示，第 14 行 name 变量的值就是映射类的类名，创建数据表时，表名即为类名的全小写，这是默认设置。

第 16 行出现了 id 字段，它是用 models.AutoField 定义的自动生成的字段，第 1 个参数 auto\_created 表示自增，第二个参数 primary\_key 表示主键，第三个参数 serialize=False 表示对类的实例进行序列化时不会带上这个字段，最后一个参数 verbose\_name 为前端页面显示时的列名。

创建好迁移文件，就可以执行迁移了：

python manage.py migrate

完成迁移后，再次查看数据库中的数据表：

mysql -uroot -p -e 'SHOW TABLES IN myweb'

在终端执行如上所示命令即可查看，无需进入 MySQL 命令行客户端。其中 -e 选项表示执行 SQL 语句，后面的字符串参数即为 SQL 语句。执行结果如下所示：



可以看到新增了一个数据表 learn\_course ，在最下面。

这张表的名字是根据应用名 learn 和映射类名 Course 来定义的，将应用名和类名全小写用下划线连接即为数据表的名字。

执行如下命令进入到 MySQL 客户端：

mysql -uroot -p myweb

看一下数据表 learn\_course 的详细信息：

DESC learn\_course;

>>>

+------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------+-------------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| name | varchar(64) | NO | | NULL | |

| pub\_date | date | NO | | NULL | |

| stu\_number | int(11) | NO | | NULL | |

+------------+-------------+------+-----+---------+----------------+

4 rows in set (0.01 sec)

如上所示，第 1 个字段为 id ，数据类型为 int ，非空，是主键，默认自增。所有字段默认带有非空约束。

这就是 Django 自带的 ORM 功能，它使用映射类生成对应的数据表。

小结：

在设置好MySQL连接的状态下，**先写实体类，再执行迁移命令**，就会对照创建数据库表。

关键是：学会写实体类的方法（类名、属性名、描述和数据表中数据类型对照的参数）。

提问：想想，一个实体类对应一个数据表吗？

## 向数据表中添加数据

向数据表中添加数据的方式有多种：

* 使用 MySQL 客户端直接添加
* 在 Python 命令行交互解释器中创建类的实例
* 将创建类的实例的代码写入脚本

接下来分别使用不同的方式演示操作。

### 使用 MySQL 客户端编写 SQL 语句

最基本的操作就是在 MySQL 命令行客户端中编写 SQL 语句。

执行如下命令向课程表中添加一条数据：

INSERT INTO learn\_course (name, pub\_date, stu\_number) VALUE ('Linux 基础教程', '2011-1-12', 123);

>>>

Query OK, 1 row affected (0.01 sec)

SELECT \* FROM learn\_course;

>>>

+----+--------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+--------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

+----+--------------------+------------+------------+

1 row in set (0.01 sec)

简单说明：

* id 字段为自增主键，无需提供值，当然也可以提供。
* pub\_date 字段为字符串，数据入库时 MySQL 会自动进行数据转换。

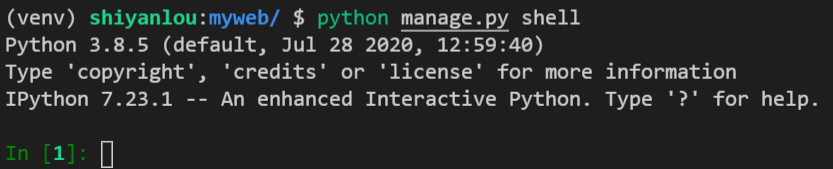
### 调用 Python Shell

向数据表中添加数据的另一种方式是使用 Python 的命令行交互解释器（在Django环境中），利用映射类创建实例，将实例作为一条数据写入数据表中，这就是 ORM 的作用。

可以开启一个新的终端执行如下语句启动交互解释器：

python manage.py shell

Django 会自动调用 ipython 这个交互解释器，这就是在第一部分安装 jupyter notebook 的目的：



警告信息可以忽略。首先引入映射类：

from learn.models import Course

向数据表中插入数据的方式有三种：

方法一 ：创建一个 Course 类实例，给实例的属性赋值，调用实例的 save 方法将数据写入数据表：

from learn.models import Course

c1 = Course()

c1.name = 'Git 与 Github 入门实践'

from datetime import datetime

c1.pub\_date = datetime(2018, 12, 22).date()

c1.stu\_number = 678

c1.save()

此时数据表的内容如下（下面的命令，在mysql命令行终端中执行）：

SELECT \* FROM learn\_course;

>>>

+----+-----------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+-----------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

+----+-----------------------------+------------+------------+

2 rows in set (0.01 sec)

方法二：创建 Course 类的实例时，直接将属性以参数的方式赋值：

c2 = Course(name='Python 异步编程', pub\_date=datetime(2019, 3, 4).date(),stu\_number=59)

c2.save()

此时数据表的内容如下（下面的命令，在mysql命令行终端中执行）：

SELECT \* FROM learn\_course;

>>>

+----+-----------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+-----------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

| 3 | Python 异步编程 | 2019-03-04 | 59 |

+----+-----------------------------+------------+------------+

3 rows in set (0.01 sec)

方法三：使用 Course 映射类的 objects.get\_or\_create 方法创建并添加数据。此方法会根据参数去数据库中搜索，如果搜索有结果，就将结果以实例返回，如果没有结果就创建一个新的实例并将数据写入数据表中。

objects 是映射类的属性，该属性指向 Django 框架的一个同名模块（也就是一个 Python 文件），objects 模块中有 get\_or\_create 函数实现这个功能。所以实际的代码应该是 Course.objects.get\_or\_create(.....) 。

现在使用此方法再次创建 Course 类的实例：

c3 = Course.objects.get\_or\_create(name = 'Julia 简明教程',pub\_date = datetime(2019,10,12).date(),stu\_number = 80)

此时数据表中的内容如下（下面的命令，在mysql命令行终端中执行）：

SELECT \* FROM learn\_course;

>>>

+----+-----------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+-----------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

| 3 | Python 异步编程 | 2019-03-04 | 59 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 |

+----+-----------------------------+------------+------------+

4 rows in set (0.00 sec)

这种方式的优点是避免创建重复数据导致报错。

说明：当在创建实例时，pub\_date 属性的值可以是表示日期的字符串，不一定是 Date 数据类型，因为数据库自己会进行数据类型转换。

### 编写创建数据的 Python 脚本

将创建数据的代码写入一个 Python 文件中，执行 Python 文件即可生成数据，这是最常规的方式。

接下来的操作可能需要伪造一些数据用于测试，先安装一个专业伪造数据的 Python 库 faker ：

pip install faker

在项目主目录myweb 下创建脚本 insert\_data.py 文件，将以下代码写入：

import django; django.setup()

from datetime import datetime

from faker import Faker

import random

from learn.models import Course

fake = Faker('zh-cn')

name\_list = ['Django 入门与实践', 'C++ 实现简易 Docker 容器', 'Flask 实现问答社区']

def create\_courses():

for name in name\_list:

course = Course(name=name, pub\_date=fake.date(), stu\_number=random.randint(100, 1000))

course.save()

if \_\_name\_\_ == '\_\_main\_\_':

create\_courses()

其中第 1 行代码也可以分两行写：

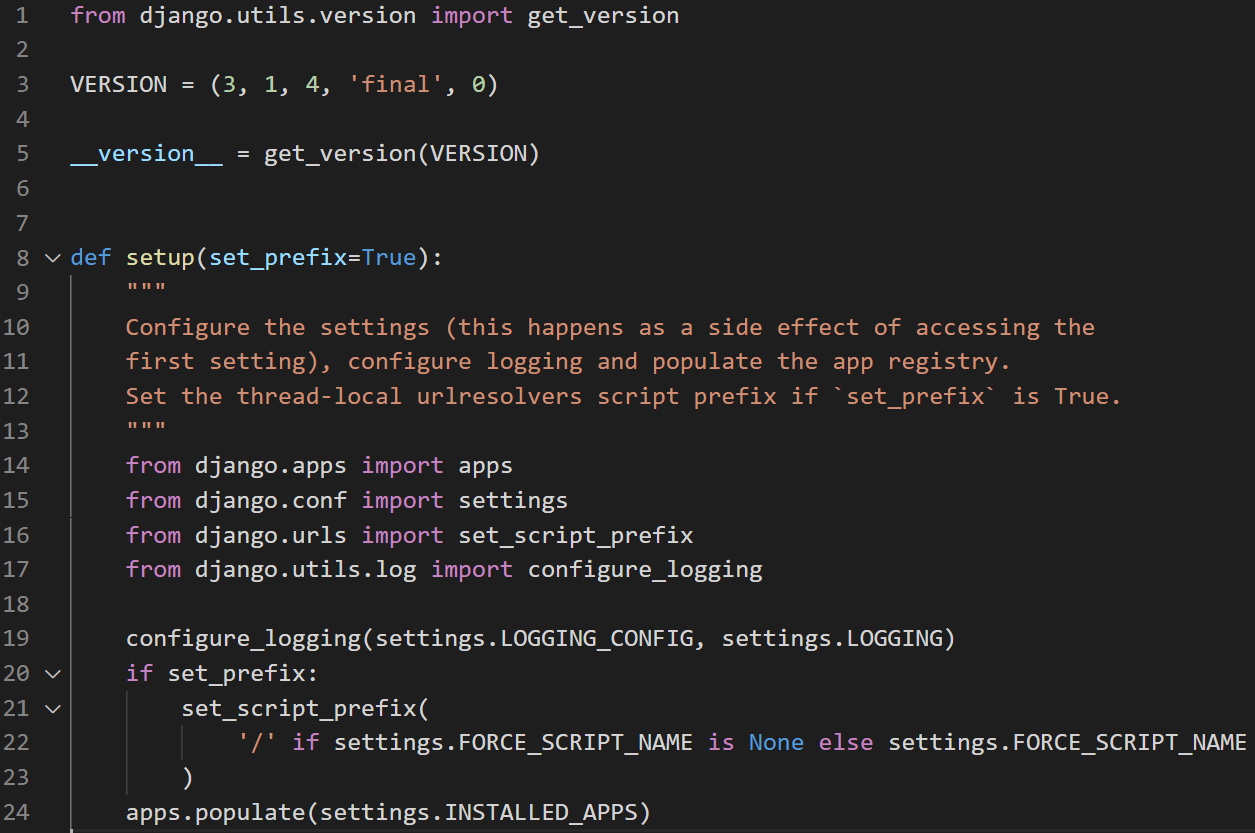
import django

django.setup()

其作用是调用 Django 框架的 \_\_init\_\_.py 文件中的 setup 函数，为脚本的执行进行一系列配置。

可以在终端查看该文件（位于当前虚拟环境下）的源码：

venv/lib/python3.8/site-packages/django/\_\_init\_\_.py



回到 insert\_data.py 文件，第 2 至 4 行引入所需模块，第 6 行引入映射类：

from datetime import datetime

from faker import Faker

import random

from learn.models import Course

然后创建 fake 对象，该对象有大量各种方法用于伪造各种类型的数据。name\_list 列表中写入三个字符串，作为 Course 类的实例的 name 属性值：

fake = Faker('zh-cn')

name\_list = ['Django 入门与实践', 'C++ 实现简易 Docker 容器', 'Flask 实现问答社区']

然后创建生成数据的函数 create\_courses ：

def create\_courses():

for name in name\_list:

course = Course(name=name, pub\_date=fake.date(), stu\_number=random.randint(100, 1000))

course.save()

最后调用函数生成 3 个 Course 类的实例并将数据存入数据库：

if \_\_name\_\_ == '\_\_main\_\_':

create\_courses()

脚本编写完成后，还需要在当前终端设置一个环境变量（在激活的当前虚拟环境的命令行终端界面中，执行）：

set DJANGO\_SETTINGS\_MODULE=myweb.settings

现在可以在终端使用 python -m 命令将 insert\_data.py 文件以模块的形式运行：

python -m insert\_data

运行结果没有任何输出信息。此时数据表中应新增 3 条数据，数据总数为 7 条（下面的命令，在mysql命令行终端中执行）：

SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+--------------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

| 3 | Python 异步编程 | 2019-03-04 | 59 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 |

| 5 | Django 入门与实践 | 1983-01-05 | 728 |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 |

| 7 | Flask 实现问答社区 | 1981-06-04 | 529 |

+----+--------------------------------+------------+------------+

7 rows in set (0.01 sec)

以上操作就是编写 Python 脚本在 Django 项目中运行的完整过程。

需要注意三点：

* 在当前终端命令行中添加环境变量（Windows下，使用set命令，Linux下使用export命令） DJANGO\_SETTINGS\_MODULE=myweb.settings ；

或者，可以在Python代码中设置环境变量（好处是，只是在当前python代码运行时，设置的环境变量才起作用。结束运行后，代码中设置的环境变量就被销毁释放不占用也不影响本机环境），具体参考insert\_data.py中的代码注释部分：

'''在代码中操作环境变量：  
# 配置临时环境变量，只在代码运行时生效  
import os  
  
# 设置环境变量  
os.environ['keyE']="value"  
# 获取环境变量方法1  
os.environ.get('key')  
#获取环境变量方法2(推荐使用这个方法)  
os.getenv('key')  
如：  
os.getenv('path')  
# 删除环境变量  
del os.environ['key']  
   
#设置常见环境变量  
os.environ['http\_proxy']:代理url(如127.0.0.1:8888）  
os.environ['HOMEPATH']:当前用户主目录  
os.environ['TEMP']:临时目录路径  
os.environ['PATHEXT']:可执行文件  
os.environ['SYSTEMROOT']:系统主目录  
os.environ['LOGONSERVER']:机器名  
os.environ['PROMPT']:设置提示符  
  
#永久设置环境变量，通过命令行将环境变量写入配置（不推荐）  
import os  
path=r"E:\env"  
# 设置环境变量 WORK1 的值为 e:\env  
command =r"setx WORK1 %s /m"%path  
os.system(command)  
# /m代表系统变量。  
# 不加/m为用户变量  
'''

* 在脚本中首先引入 django 并调用 django.setup 函数；
* 使用 python -m 以模块的形式执行脚本。

## 总结

本部分讲授了 ORM 框架的作用并编写了第一个映射类，然后使用映射类创建了一张数据表。向数据表中添加数据的三种方式：在 MySQL Shell 中执行 SQL 语句、调用 Python Shell 引入映射类创建其实例、编写创建数据的脚本并设置环境变量执行模块。

下一部分将学习映射类实例处理数据的 API 。

#### 参考代码

本阶段前几部分，代码比较简单，参考代码：code03.zip

## 作业：

0.简述对ORM的理解

1.总结Django框架中创建数据库映射类的相关概念

2.尝试编写一个映射（实体）类，名称和属性自定，并完成据此映射类创建数据库表以及插入至少3条数据。

3.简单了解下faker工具包的作用和用法（这个包，其实很有意思的）。

# 讲授 4 映射类的增删改查操作

## 说明

本部分内容较少，主要针对映射类操作数据库的 API 进行讲解。

上一部分中已经学习了添加数据的操作，本部分内容基本包括修改数据、删除数据以及最重要的查询数据，其实就是要掌握CRUD操作。

#### 知识点

* 利用 ORM 查询数据库
* 查询过滤器
* 条件运算符
* 聚合函数
* 修改和删除数据

## 查询数据

查询数据是 Django ORM 的强项，它有很多方式用于各种查询操作，本部分主要学习一些基本操作。

首先启动 MySQL 服务与虚拟环境，具体操作略。然后进入myweb的项目主目录位置。

cd myweb

进入 MySQL 客户端：

mysql -uroot -p myweb

根据前面的讲授内容，已经在数据库中创建了 learn\_course 数据表，并且向数据表中添加了 7 条数据：

mysql> SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+--------------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

| 3 | Python 异步编程 | 2019-03-04 | 59 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 |

| 5 | Django 入门与实践 | 1983-01-05 | 728 |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 |

| 7 | Flask 实现问答社区 | 1981-06-04 | 529 |

+----+--------------------------------+------------+------------+

7 rows in set (0.01 sec)

## 在 Python Shell 中操作

打开终端，切换到 myweb 项目的主目录 myweb ，执行如下语句进入 ipython ：

cd myweb

python manage.py shell

首先引入映射类：

from learn.models import Course

然后查询 learn\_course 数据表中的全部数据：

all\_courses = Course.objects.all()

all\_courses

Out[]: <QuerySet [<Course: Linux 基础教程>, <Course: Git 与 Github 入门实践>, <Course: Python 异步编程>, <Course: Julia 简明教程>, <Course: Django 入门与实践>, <Course: C++ 实现简易 Docker 容器>, <Course: Flask 实现问答社区>]>

第 2 行代码 all\_courses = Course.objects.all() 使用 Course.objects 的 all 函数只是返回了一个 QuerySet 对象赋值给变量 all\_courses ，这步并未执行查询数据库的操作。

当执行第 3 行代码查看变量 all\_courses 时，ORM 就会执行一个 SQL 语句查询 learn\_course 数据表的全部数据并返回。

变量 all\_courses 是 QuerySet 类的对象，可以称之为 “查询集”（类似Java JDBC的ResultSet） 。该对象为可迭代对象，使用 isinstance 函数可以判断：

# from collections import Iterable

#上面的方法已经降级，建议改为：

from collections.abc import Iterable

isinstance(all\_courses, Iterable)

Out[5]: True

可迭代对象，就可以使用 for 关键字对其进行迭代获取元素：

for course in all\_courses:

print(course)

>>>

Linux 基础教程

Git 与 Github 入门实践

Python 异步编程

Julia 简明教程

Django 入门与实践

C++ 实现简易 Docker 容器

Flask 实现问答社区

如上所示，打印每个元素会自动显示课程的名字，这是因为在创建映射类的时候，定义了 \_\_str\_\_ 方法：

当然啦，在 for 循环中 course 变量就是 Course 类的实例，可以打印它的其它属性：

for course in all\_courses:

print('Name: {}, Pub\_date: {}, Stu\_number: {}'.format(course.name, course.pub\_date, course.stu\_number))

Name: Linux 基础教程, Pub\_date: 2011-01-12, Stu\_number: 123

Name: Git 与 Github 入门实践, Pub\_date: 2018-12-22, Stu\_number: 678

Name: Python 异步编程, Pub\_date: 2019-03-04, Stu\_number: 59

Name: Julia 简明教程, Pub\_date: 2019-10-12, Stu\_number: 80

Name: Django 入门与实践, Pub\_date: 1983-01-05, Stu\_number: 728

Name: C++ 实现简易 Docker 容器, Pub\_date: 1974-12-02, Stu\_number: 158

Name: Flask 实现问答社区, Pub\_date: 1981-06-04, Stu\_number: 529

接下来要查询课程中学生人数超过 99 的所有课程的名字和发布日期，这要用到 Course.objects 的 filter 方法过滤数据：

for course in Course.objects.filter(stu\_number\_\_gt=99):

print('课程名字：{}, 发布日期：{}'.format(course.name, course.pub\_date))

课程名字：Linux 基础教程, 发布日期：2011-01-12

课程名字：Git 与 Github 入门实践, 发布日期：2018-12-22

课程名字：Django 入门与实践, 发布日期：1983-01-05

课程名字：C++ 实现简易 Docker 容器, 发布日期：1974-12-02

课程名字：Flask 实现问答社区, 发布日期：1981-06-04

其中 filter 方法的参数为 stu\_number\_\_gt ，它是使用双下划线连接属性名 stu\_number 和 gt 生成的，gt 的意思就是 “大于” ，stu\_number\_\_gt=99 的意思就是 stu\_number 这个字段的值大于 99 。这样就可以过滤出学生人数大于 99 的课程了。

filer方法等同于如下SQL语句：

SELECT `learn\_course`.`id`, `learn\_course`.`name`, `learn\_course`.`pub\_date`, `learn\_course`.`stu\_number` FROM `learn\_course` WHERE `learn\_course`.`stu\_number` > 99;

>>>

+----+--------------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+--------------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

| 5 | Django 入门与实践 | 1983-01-05 | 728 |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 |

| 7 | Flask 实现问答社区 | 1981-06-04 | 529 |

+----+--------------------------------+------------+------------+

这就是 ORM 的功能，它将 Python 代码转换成了 SQL 语句执行。

其中 filter 方法的参数 stu\_number\_\_gt 中双下划线后面的 gt 叫做 “条件运算符” ，而 filter 方法叫做过滤器。

## 过滤器

在 Django 中，过滤器有多种：all 、filter 、exclude 、get 和 order\_by ，它们都是 Course.objects 的方法。前面的讲授中已经接触了前两种(all/filter)，下面介绍一下后三种。

其中 exclude 方法的作用就是字面意思，过滤掉符合条件的数据，与 filter 正相反。

举例说明，filter(stu\_number\_\_gt=99) 是获取学生人数大于 99 的课程，exclude(stu\_number\_\_gt=99) 就是获取学生人数不大于 99 的课程：

for course in Course.objects.exclude(stu\_number\_\_gt=99):

print('课程名字：{}, 发布日期：{}'.format(course.name, course.pub\_date))

课程名字：Python 异步编程, 发布日期：2019-03-04

课程名字：Julia 简明教程, 发布日期：2019-10-1

filter 和 exclude 返回的是包含查询结果的 QuerySet 对象，而 get 方法只返回一个符合的映射类实例。如果查询结果是多个，会抛出 MultipleObjectsReturned 异常；如果查询结果是零个，会抛出 DoesNotExist 异常。

此外，get 方法不支持条件运算符，可以提供多个参数。filter 和 exclude 支持条件运算符，也可以提供多个参数。

举例说明：

# 查找 id 字段值为 2 的课程

c = Course.objects.get(id=2)

c

Out[12]: <Course: Git 与 Github 入门实践>

c.name

Out[13]: 'Git 与 Github 入门实践'

# 查找 id 字段值为 2 且 name 字段值为 "Git 与 Github 入门实践" 的课程

c = Course.objects.get(id=2, name='Git 与 Github 入门实践')

c

Out[15]: <Course: Git 与 Github 入门实践>

c.id

Out[16]: 2

c.name

Out[17]: 'Git 与 Github 入门实践'

最后介绍一下 order\_by ，它的作用很显然就是排序，与前几个过滤器不同之处是它针对 QuerySet 对象进行排序，也就是对查询结果进行排序，参数就是字段名的字符串。

例如使用 filter 获取到了学生人数大于 99 的所有课程，打印它们的发布日期：

for course in Course.objects.filter(stu\_number\_\_gt=99):

print('课程名字：{}, 发布日期：{}'.format(course.name, course.pub\_date))

课程名字：Linux 基础教程, 发布日期：2011-01-12

课程名字：Git 与 Github 入门实践, 发布日期：2018-12-22

课程名字：Django 入门与实践, 发布日期：1983-01-05

课程名字：C++ 实现简易 Docker 容器, 发布日期：1974-12-02

课程名字：Flask 实现问答社区, 发布日期：1981-06-04

现在希望这些课程按照发布日期排列，就可以在 QuerySet 对象后面使用 order\_by 方法：

for course in Course.objects.filter(stu\_number\_\_gt=99).order\_by('pub\_date'):

print('课程名字：{}, 发布日期：{}'.format(course.name, course.pub\_date))

课程名字：C++ 实现简易 Docker 容器, 发布日期：1974-12-02

课程名字：Flask 实现问答社区, 发布日期：1981-06-04

课程名字：Django 入门与实践, 发布日期：1983-01-05

课程名字：Linux 基础教程, 发布日期：2011-01-12

课程名字：Git 与 Github 入门实践, 发布日期：2018-12-22

很显然，查询结果已经按照 pub\_date 字段的值排序了，默认是由小到大升序排序。

如果要降序排序呢？只需在属性字符串前面加个减号 order\_by('-pub\_date') ：

for course in Course.objects.filter(stu\_number\_\_gt=99).order\_by('-pub\_date'):

print('课程名字：{}, 发布日期：{}'.format(course.name, course.pub\_date))

课程名字：Git 与 Github 入门实践, 发布日期：2018-12-22

课程名字：Linux 基础教程, 发布日期：2011-01-12

课程名字：Django 入门与实践, 发布日期：1983-01-05

课程名字：Flask 实现问答社区, 发布日期：1981-06-04

课程名字：C++ 实现简易 Docker 容器, 发布日期：1974-12-02

效果立现，发布日期从高到低排列了。

这就是 Python 的简洁与强大，吼吼，恶心……。

## 条件运算符

学习了 5 个过滤器方法，其实严格来说只有 filter 和 exclude 算是过滤器，因为它们支持条件运算符。

下面分别详细介绍常用的运算符的用法，包括：

* 比较大小
* 模糊查询
* 日期查询

仅演示如上三类条件运算符，这是其中最常用的三个。

### 比较大小

条件运算符有一个，表示 “大于” 的 gt，此外，

表示比较的运算符共有五个：

* gt 大于
* gte 大于等于
* exact 等于
* lt 小于
* lte 小于等于

举两个例子：

# 按学生数量倒序排序，打印课程 id 、pub\_date 和 stu\_number

from datetime import datetime

for course in Course.objects.filter(pub\_date\_\_lt=datetime(2000, 1, 1).date()).order\_by('-stu\_number'):

print('ID: {}, Date: {}, Stu: {}'.format(course.id, course.pub\_date, course.stu\_number))

ID: 5, Date: 1983-01-05, Stu: 728

ID: 7, Date: 1981-06-04, Stu: 529

ID: 6, Date: 1974-12-02, Stu: 158

# pub\_date\_\_lt 的值没必要构造 Date 类型的数据，直接使用字符串亦可

# 因为 MySQL 服务器自己会进行数据类型转换

for course in Course.objects.filter(pub\_date\_\_lt='2000-1-1').order\_by('-stu\_number'):

print('ID: {}, Date: {}, Stu: {}'.format(course.id, course.pub\_date, course.stu\_number))

ID: 5, Date: 1983-01-05, Stu: 728

ID: 7, Date: 1981-06-04, Stu: 529

ID: 6, Date: 1974-12-02, Stu: 158

# 由于数据由 faker 包生成，(提示学生)应根据生成的数据修改数字，上下均是。

for course in Course.objects.filter(stu\_number\_\_exact=728):

print(course.name)

# 打印学生数量为 728 的课程的名字

Django 入门与实践

### 模糊查询

模糊查询运算符有三个：

* contains 包含值的一部分
* startswith 以某值开头
* endswith 以某值结尾

这三个通常用于字符串数据类型的字段，就使用课程的名字来举例吧：

# 查询并打印课程名字中带大写字母 L 的课程的 id 和 name

for course in Course.objects.filter(name\_\_contains='L'):

print('ID: {}, Name: {}'.format(course.id, course.name))

>>>

ID: 1, Name: Linux 基础教程

# 如果字母不区分大小写呢？在 contains 前面加上一个字符 i 即可

for course in Course.objects.filter(name\_\_icontains='L'):

print('ID: {}, Name: {}'.format(course.id, course.name))

>>>

ID: 1, Name: Linux 基础教程

ID: 4, Name: Julia 简明教程

ID: 7, Name: Flask 实现问答社区

# 查询课程名字中包含 Docker 的课程，打印课程的 id 和 name

for course in Course.objects.filter(name\_\_icontains='docker'):

print('ID: {}, Name: {}'.format(course.id, course.name))

>>>

ID: 6, Name: C++ 实现简易 Docker 容器

# 查询课程名字以 “教程” 二字结尾的课程，打印课程的 id 和 name

for course in Course.objects.filter(name\_\_endswith='教程'):

print('ID: {}, Name: {}'.format(course.id, course.name))

>>>

ID: 1, Name: Linux 基础教程

ID: 4, Name: Julia 简明教程

# 查询课程名字以 “python” 开头的课程，不区分大小写，打印课程的 id 和 name

for course in Course.objects.filter(name\_\_istartswith='python'):

print('ID: {}, Name: {}'.format(course.id, course.name))

>>>

ID: 3, Name: Python 异步编程

范围运算符 in ，参数为列表，用于查询字段值在列表中的数据。

举例说明，查询 id 字段值为 1 或 3 或 7 的课程，打印课程的 id 和 name ：

for course in Course.objects.filter(id\_\_in=[1, 3, 7]):

print('ID: {}, Name: {}'.format(course.id, course.name))

>>>

ID: 1, Name: Linux 基础教程

ID: 3, Name: Python 异步编程

ID: 7, Name: Flask 实现问答社区

### 日期查询

对时间日期类型的数据进行过滤查询，运算符包括：

* year 年
* month 月
* day 日
* hour 时
* minute 分
* second 秒
* week\_day 星期

例如查询任意年 1 月份发布的课程：

for course in Course.objects.filter(pub\_date\_\_month=1):

print('Date: {}, Name: {}'.format(course.pub\_date, course.name))

>>>

Date: 2011-01-12, Name: Linux 基础教程

Date: 1983-01-05, Name: Django 入门与实践

除了以上介绍的运算符之外，还有空值查询、F 查询、Q 查询等等，考虑让学生自行拓展学习，或者在后续项目中遇到时再进行讲授。

## 聚合函数

至此已经接触了 Course.objects 的五个过滤方法和过滤方法中的各种运算符，现在来学习 Course.objects 的聚合函数 aggregate 。

聚合函数接收一些类的实例作为参数，这些类需要从 django.db.models 模块中引入。所以先引入 models 模块：

from django.db import models

这些类包括：

* Sum 求和
* Avg 求平均值
* Max 最大值
* Min 最小值
* Count 数量

Course.objects.aggregate 方法的返回值是一个字典，字典中只有一组键值对。其中 key 是使用双下划线把字段名和类名的全小写连起来生成的字符串，value 就是计算结果。

例如求学生人数的总和，aggregate 方法的返回值字典的 key 就是 stu\_number\_\_sum ，其中 stu\_number 是字段名，sum 是求和类 Sum 的全小写。

依次举例如下：

Course.objects.aggregate(models.Sum('stu\_number'))

>>>

{'stu\_number\_\_sum': 2355}

Course.objects.aggregate(models.Avg('stu\_number'))

>>>

{'stu\_number\_\_avg': 336.4286}

Course.objects.aggregate(models.Max('stu\_number'))

>>>

{'stu\_number\_\_max': 728}

Course.objects.aggregate(models.Min('stu\_number'))

>>>

{'stu\_number\_\_min': 59}

Course.objects.aggregate(models.Count('stu\_number'))

>>>

{'stu\_number\_\_count': 7}

In [43]:

其中 models.Count 类用得较少，因为 Course.objects 本身就有 count 方法获取数量：

Course.objects.count()

>>>

7

## 修改数据

修改数据十分简单，获取映射类实例，修改实例属性，调用实例的 save 方法即可。

举例说明，将 id 为 3 的课程的发布时间改成 4 月 3 号，年份不变：

c3 = Course.objects.get(id=3)

c3.id

>>>

3

c3.name

>>>

'Python 异步编程'

c3.pub\_date

>>>

datetime.date(2019, 3, 4)

# 修改属性值

c3.pub\_date = '2019-4-3'

# 调用 save 方法保存修改到数据库

c3.save()

此时查询数据库即可发现发布时间已经修改成功：

SELECT \* FROM learn\_course WHERE id = 3;

+----+---------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+---------------------+------------+------------+

| 3 | Python 异步编程 | 2019-04-03 | 59 |

+----+---------------------+------------+------------+

1 row in set (0.01 sec)

## 删除数据

与修改数据类似，依然是查询数据库获取类的实例对象，然后调用实例的 delete 方法删除自身。

现在删除 id 为 7 的课程：

c7 = Course.objects.get(id=7)

c7.id

Out[55]: 7

c7.name

Out[56]: 'Flask 实现问答社区'

c7.delete()

Out[57]: (1, {'learn.Course': 1})

在 MySQL 客户端中验证：

mysql> SELECT \* FROM learn\_course;

+----+--------------------------------+------------+------------+

| id | name | pub\_date | stu\_number |

+----+--------------------------------+------------+------------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 |

| 3 | Python 异步编程 | 2019-04-03 | 59 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 |

| 5 | Django 入门与实践 | 1983-01-05 | 728 |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 |

+----+--------------------------------+------------+------------+

6 rows in set (0.01 sec)

在 Django 项目中，在 MySQL 客户端中尽量只做一点点查询操作，数据库、数据表和数据的增删改查操作要在 Django 项目中使用 Python 代码来处理。

## 总结

本部分内容较多，但很简单。主要介绍了利用 Django 的 ORM 框架使用映射类的实例实现查询数据、修改数据和删除数据的操作。

其中查询数据的方法和规则较多，细如牛毛，本部分也只是介绍了常用的大部分。其中包括过滤器和条件运算符以及聚合函数，条件运算符特别多，比较难以掌握，需要大家多多练习。

下一部分，要求完成一个挑战作业来验证学习情况。

#### 参考代码

本阶段代码（主要在test\_orm\_crud.py中）比较简单，参考代码：code04.zip

## 作业：

1.编写Python代码，实现对之前作业中自行创建和生成并插入数据的数据表，执行增删改查的操作。（注意，不是在 python manage shell界面中执行）

# 挑战作业 1 创建课程作者数据表

# 创建课程作者数据表

## 挑战介绍

本次挑战作业要在上一部分的基础上创建一个 Author 映射类，并使用该映射类创建一张数据表。

#### 环境准备

先做一些准备工作，首先切换到 project 目录下创建虚拟环境：

venv

进入虚拟环境，安装所需工具包：

django

mysqlclient

基于上一部分的项目代码：code04.zip，并在IDE中打开。

启动 MySQL 服务，创建 myweb 数据库：

#### 挑战作业要求

Author 类从属于 learn 应用，所以数据表的表名应是 learn\_author 。

数据表包含如下字段：

* id 自增主键，int 类型
* name 课程作者，字符串类型，最大长度为 64 ，不可为空值
* gender 性别，布尔值类型，男性为 True，女性为 False ，可以为空值

创建好数据表之后，向表中添加三条数据：

| **ID** | **作者名字** | **性别** |
| --- | --- | --- |
| 1 | 卡卡西 | 男 |
| 2 | 娜美 | 女 |
| 3 | 宋神宗 |  |

注意第三条数据的 “性别” 字段为空。添加数据的方式不限，推荐利用映射类编写 Python 代码实现。

完成全部操作后，可以在数据库中获得如下查询结果：

SELECT \* FROM learn\_author;

+----+-----------+--------+

| id | name | gender |

+----+-----------+--------+

| 1 | 卡卡西 | 1 |

| 2 | 娜美 | 0 |

| 3 | 宋神宗 | NULL |

+----+-----------+--------+

3 rows in set (0.01 sec)

#### 涉及知识点

* Django 映射类的创建
* 数据迁移操作
* 利用映射类创建数据

## 提示信息

1、id 主键无需自定义，数据迁移时自动创建；

2、使用映射类生成数据表分两步，生成迁移文件和执行迁移；

3、字段非空约束的参数为 null=True 。

## 参考代码

向 learn/models.py 文件中增加如下代码：

class Author(models.Model):

'''课程作者映射类，包括作者名字、性别等属性

'''

name = models.CharField(max\_length=64)

gender = models.BooleanField(null=True)

def \_\_str\_\_(self):

return self.name

终端执行如下命令创建映射类的迁移文件并执行迁移：

cd myweb

python manage.py makemigrations

python manage.py migrate

执行 python manage.py shell 打开 Python Shell ：

from learn.models import Author

Author.objects.get\_or\_create(name='卡卡西', gender=1)

Out[3]: (<Author: 卡卡西>, True)

Author.objects.get\_or\_create(name='娜美', gender=0)

(<Author: 娜美>, True)

Author.objects.get\_or\_create(name='宋神宗')

(<Author: 宋神宗>, True)

# 讲授 5 数据模型关系之一对一和多对一

## 介绍

在前面的学习中，已经掌握了映射类的基本创建方法，包括字段属性以及属性的参数都已介绍。在利用映射类的 objects 查询数据的常用方法上做了大量演示练习，本部分继续深入学习数据模型关系的知识。

所谓数据模型关系，在数据库层面就是两张表之间的关系，有三种基本的关系（老生常谈）：

* **一对一关系**，例如课程作者表和作者详情表。课程作者表可能包含主键、名字、邮箱、登录密码之类的，作者详情表包括作者的性别、出生日期、简介、地址等等。这两张表是一对一的关系，一个作者对应唯一的详情数据。
* **多对一关系**，也可以说一对多关系，例如课程作者表与课程表。课程表可能包含课程的 id 、名字、发布时间、学习人数、关注人数、课程是谁写的等信息。一个课程只有一个作者，但一个作者可以编写多门课程，这就是一对多的关系。
* **多对多关系**，例如课程表与课程的标签表。课程的标签表仅有 id 、标签名字两个属性。一门课程可以有多个标签，而一个标签可能被贴在多门课程上，这就是多对多的关系了。多对多关系通常需要第三张表提供协助，这张表叫做关系表。

本部分来学习 “一对一” 和 “一对多” 关系，下一部分学习 “多对多” 关系。

#### 知识点

* 一对一关系
* 一对多关系

## 准备工作

接下来的内容需要接续上一个挑战的结果，也就是 myweb 数据库中已经有两张有自定义映射类创建的数据表 learn\_course 和 learn\_author ，前者有 6 条数据，后者有 3 条数据。

说明（对学生）：如果之前讲授的内容，已经照做并保持，该部分的准备工作可以不做，直接进入下一步骤。

或者，如果没有做过前面的讲授内容和挑战作业1：

在虚拟环境中运行，可以按照如下步骤操作：

解压项目代码challenge01.zip.zip，并在PyCharm中打开：

下面是数据库准备，如果数据库中有数据，可以略过。

执行如下语句创建数据库和迁移数据表：

net start mysql

mysql -uroot -p

CREATE SCHEMA myweb

# 退出mysql命令行终端界面

cd venv/Scripts

activate

cd myweb

python manage.py migrate

打开 MySQL 客户端：

# 方法有二

python manage.py dbshell # 这是 Django 提供的调用 MySQL Shell 的命令

mysql -uroot -p myweb # 这是常规操作

将以下代码复制到 MySQL Shell 中执行

INSERT INTO `learn\_author` VALUES

(1,'卡卡西',1),(2,'娜美',0),(3,'宋神宗',NULL);

INSERT INTO `learn\_course` VALUES

(1,'Linux 基础教程','2011-01-12',123),

(2,'Git 与 Github 入门实践','2018-12-22',678),

(3,'Python 异步编程','2019-04-03',59),

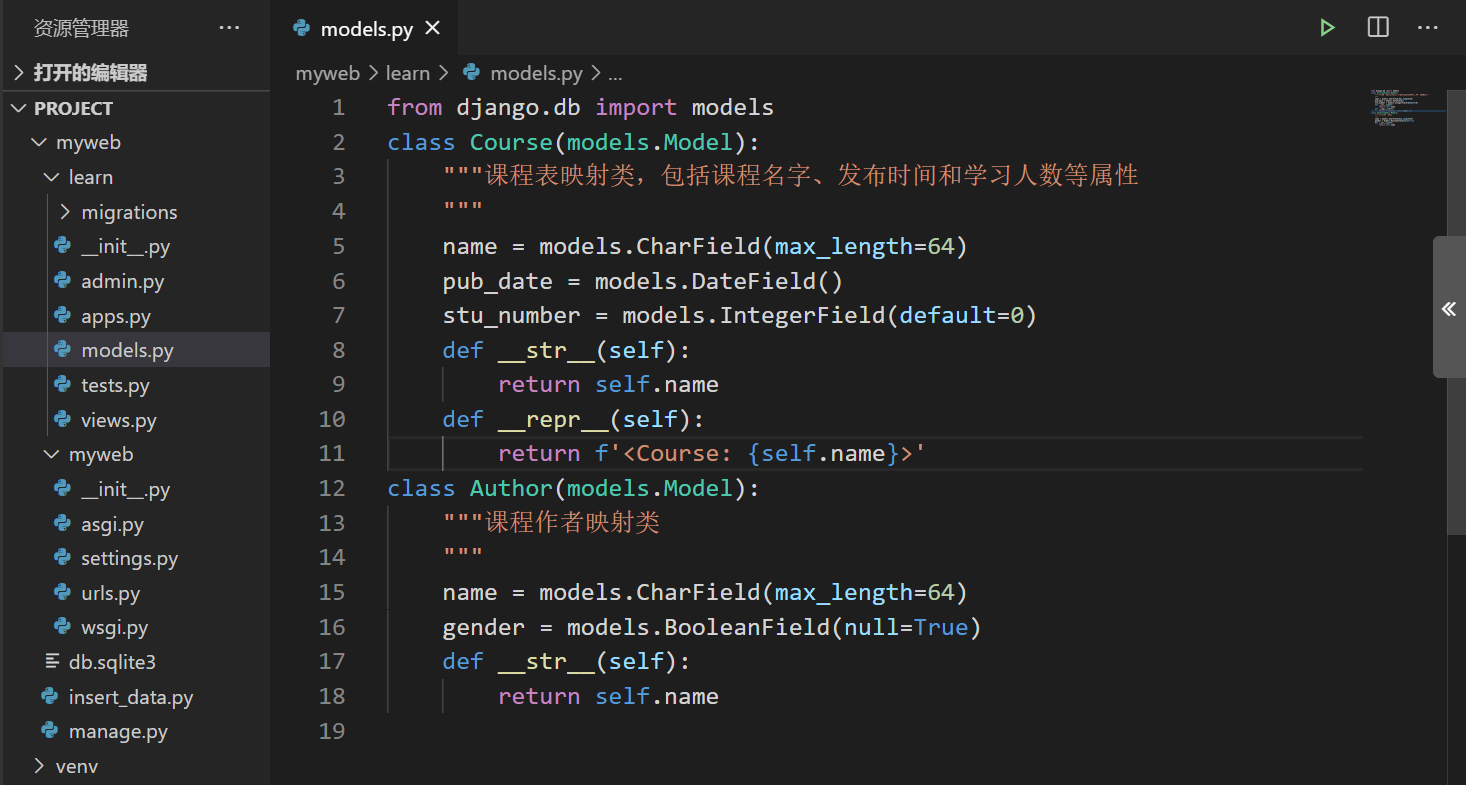
(4,'Julia 简明教程','2019-10-12',80),

(5,'Django 入门与实践','1983-01-05',728),

(6,'C++ 实现简易 Docker 容器','1974-12-02',158);

## 一对一关系

目前 myweb 项目中的 learn 应用已经编写了两个映射类，对应课程表和作者表：



下面编写第三个映射类用于存储作者的详细信息，然后生成 “作者信息表”。作者信息表与作者表是一对一的关系，每个课程作者有唯一对应的作者信息数据。

### 创建数据表

创建一对一关系的映射类，使用 OneToOneField 字段类型。

新建映射类的类名是 AuthorDetail 。

将以下代码补充到 learn/models.py 文件中：

class AuthorDetail(models.Model):

'''作者详情类，包括出生日期、地址、简介等属性

'''

birth\_date = models.DateField(null=True)

address = models.CharField(max\_length=64)

profile = models.TextField()

author = models.OneToOneField('Author', on\_delete=models.CASCADE)

def \_\_str\_\_(self):

return self.author.name

其中 birth\_date 字段设置为允许是空值，profile 字段为 TextField 。

最后一个 author 为一对一字段，它的第一个参数指向作者映射类 Author ，该字段可以直接写 Author 类，也可以写类名的字符串格式，建议采用后者。第二个参数 on\_delete 为级联删除设置项，models.CASCADE 的意思就是如果 Author 数据被删除，那么对应的 AuthorDetail 数据也被一并删除（即：级联删除）。

现在执行数据迁移的两步：

python manage.py makemigrations

python manage.py migrate

完成后，进入 MySQL Shell 查询已创建的数据表：

python manage.py dbshell

SHOW TABLES LIKE 'learn%';

+--------------------------+

| Tables\_in\_myweb (learn%) |

+--------------------------+

| learn\_author |

| learn\_authordetail |

| learn\_course |

+--------------------------+

3 rows in set (0.01 sec)

可以看到新增了一张名为 learn\_authordetail 的数据表，该表的名字是用下划线将应用名 learn 和映射类名 AuthorDetail 的全小写连起来生成的。

查看数据表详情：

DESC learn\_authordetail;

>>>

+------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------+-------------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| birth\_date | date | YES | | NULL | |

| address | varchar(64) | NO | | NULL | |

| profile | longtext | NO | | NULL | |

| author\_id | int(11) | NO | UNI | NULL | |

+------------+-------------+------+-----+---------+----------------+

5 rows in set (0.01 sec)

* 第 1 行为默认生成的 id 主键字段
* 第 2 行为出生日期，date 类型，可为空值
* 第 3 行为地址，字符串类型
* 第 4 行是简介，长文档类型
* 第 5 行需要着重说明，在映射类中定义的 author 属性为一对一外键关联属性，它只属于映射类的实例，在数据表中并不存在。而表中多出来的 author\_id 是由一对一外键生成的表属性，该属性同时属于映射类的实例。属性名字是由下划线连接 Author 类名的全小写和 learn\_author 表的主键字段 id 生成的。在实例化 AuthorDetail 类时，必须提供 author\_id 字段的值，且该值必须等于 learn\_author 表中某一个 id 字段的值。特别注意：Key列的值为UNI，表示唯一索引，即：使用唯一索引的字段做外键字段，表述一对一的关系。（需要提前在DBSigned中讲过RDBMS中三种关系的实现方式，学生就可以顺畅理解的。）

### 添加数据

创建好数据表，就可以增加数据了。显然对于 learn\_authordetail 表，目前最多可以增加三条数据，因为 learn\_author 表中只有三条数据。

在项目中打开 Python Shell ：

from learn.models import Author, AuthorDetail

# 先获取作者表中的所有三个作者，并赋值给 a1, a2, a3

a1, a2, a3 = Author.objects.all()

a1

Out[3]: <Author: 卡卡西>

a2

Out[4]: <Author: 娜美>

a3

Out[5]: <Author: 宋神宗>

如上所示第 2 行代码是 Python 语法中的拆包术，因为 Author.objects.all() 的返回值是一个可迭代对象，这一点在前面的讲授中已经使用 isinstance 方法验证过了。a1 、a2 和 a3 就是三个 Author 类的实例。

首先创建 a1 对应的 AuthorDetail 实例：

ad1 = AuthorDetail(birth\_date = '1520-9-15',address = '火之国木叶村可爱大道202号',profile = '暗部队长，第七班老师，六代火影，五五开绝技拥有者',author = a1)

# 保存入库

ad1.save()

如上所示，ad1 即为 a1 对应的作者详情实例。在 MySQL 客户端命令行中查询信息：

# \G  将结果 display result vertically（垂直显示）

SELECT \* FROM learn\_authordetail\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

birth\_date: 1520-09-15

address: 火之国木叶村可爱大道202号

profile: 暗部队长，第七班老师，六代火影，五五开绝技拥有者

author\_id: 1

1 row in set (0.00 sec)

在定义实例时，共有 4 个参数，前三个无需多解释。第 4 个参数 author = a1 的意思就是 ad1 的 author 属性值为 a1 这个实例。这里并未提供 author\_id 字段的值，因为已经提供了 author 属性，在执行 ad1.save() 时，ORM 会自动从 author 这个属性的值 a1 中获取 id 属性值。

当然也可以直接提供 author\_id 字段的值。

创建 a2 对应的 AuthorDetail 类的实例：

ad2 = AuthorDetail(birth\_date = '1640-7-3',address = '东海可可亚西村幸福大街007号',profile = '聪明又机灵，爱好金钱和橘子，精通航海术和气象学，会武术',author\_id = a2.id)

ad2.save()

如上所示，这次创建 AuthorDetail 实例时并未提供 author 参数，而是使用了 author\_id 。

创建最后一个实例：

ad3, result = AuthorDetail.objects.get\_or\_create(birth\_date = '1048-5-25',address = '濮安懿王宫邸睦亲宅18号',profile = '北宋皇帝，王安石变法支持者，击败安南，元丰改制，忧郁而逝',author = a3)

这次使用了 objects.get\_or\_create 方法创建实例并将数据写入数据库。

查询数据表如下：

mysql> SELECT \* FROM learn\_authordetail\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

birth\_date: 1520-09-15

address: 火之国木叶村可爱大道202号

profile: 暗部队长，第七班老师，六代火影，五五开绝技拥有者

author\_id: 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 2

birth\_date: 1640-07-03

address: 东海可可亚西村幸福大街007号

profile: 聪明又机灵，爱好金钱和橘子，精通航海术和气象学，会武术

author\_id: 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 3

birth\_date: 1048-05-25

address: 濮安懿王宫邸睦亲宅18号

profile: 北宋皇帝，王安石变法支持者，击败安南，元丰改制，忧郁而逝

author\_id: 3

3 rows in set (0.01 sec)

### 查询数据

数据已经添加完毕，接下来介绍查询数据的方法和属性。

（适当考虑提问：）现在已经有了 Author 类的实例，它们对应的作者详情信息怎么获取呢？

因为外键的缘故，Author 类的实例增加了一个 authordetail 属性，其属性值即为对应的 AuthorDetail 类的实例。很显然，这个属性的名字是作者详情类名的全小写。

获取 a1 对应的作者详情类实例：

a1.authordetail

Out[13]: <AuthorDetail: 卡卡西>

ad1

Out[14]: <AuthorDetail: 卡卡西>

这个实例其实就是 ad1 ，可以看到打印信息中带有作者的 name 属性值，这是由于在定义 AuthorDetail 类的时候定义了 \_\_str\_\_ 方法：

def \_\_str\_\_(self):

return self.author.name

该方法的返回值就是对应的作者的 name 属性。

不难猜到 ad1 的 author 属性值就是 a1 这个实例，author\_id 属性值就是 a1 的 id 属性值：

ad1.author

Out[15]: <Author: 卡卡西>

ad1.author\_id

Out[16]: 1

a1 的 authordetail 属性和 ad1 的 author 属性叫做查询接口，这些是由 Django 的 ORM 框架提供的。

🡪让学生自己周游一下类（Author、AuthorDetail）

## 一对多关系

一对多关系的映射类已经编写完了，就是 Author 和 Course 类。一个 Author 类实例对应多个 Course 类实例，

不过还没有给课程类增加外键使之形成一对多关系（记住：在多的一边添加一的一边的主键做外键字段，此处：多的一边是 Course）。

接下来的步骤中将修改 Course 映射类，使之与 Author 类形成一对多关系。然后演示一下一对多关系中添加和查询数据的操作。

### 创建数据表

现在打开 learn/models.py 文件修改 Course 类如下：

class Course(models.Model):

'''课程表映射类，包括课程名字、发布时间和学习人数等属性

'''

name = models.CharField(max\_length=64)

pub\_date = models.DateField()

stu\_number = models.IntegerField(default=0)

author = models.ForeignKey('Author', on\_delete=models.CASCADE, # 这两行

null=True) # 是新增代码

def \_\_str\_\_(self):

return self.name

因为 Course 类需要设置外键主动关联 Author ，所以 Author 不需要任何修改。

注意 models.ForeignKey 的参数要加上 null=True ，将其设置为可以是空值，因为目前 learn\_course 表中已经有数据，现在新增一列，这列是空值。

创建数据表修改部分的迁移文件，执行迁移操作：

python manage.py makemigrations

>>>

Migrations for 'learn':

learn/migrations/0006\_course\_author.py

- Add field author to course

python manage.py migrate

>>>

Operations to perform:

Apply all migrations: admin, auth, contenttypes, learn, sessions

Running migrations:

Applying learn.0006\_course\_author... OK

在执行 python manage.py migrate 时，因为只修改了 learn 相关的映射类，所以后面没有更多参数。假设有 learn2 应用，而且其中的映射类有变动，但我不想将这部分变动迁移到数据库中，可以在命令行后面加上 learn 参数，表示只迁移 learn 应用中的修改。

此时查看一下数据表：

DESC learn\_course;

>>>

+------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------+-------------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| name | varchar(64) | NO | | NULL | |

| pub\_date | date | NO | | NULL | |

| stu\_number | int(11) | NO | | NULL | |

| author\_id | int(11) | YES | MUL | NULL | |

+------------+-------------+------+-----+---------+----------------+

5 rows in set (0.00 sec)

SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+-----------+

| id | name | pub\_date | stu\_number | author\_id |

+----+--------------------------------+------------+------------+-----------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 | NULL |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 | NULL |

| 3 | Python 异步编程 | 2019-04-03 | 59 | NULL |

| 4 | Julia 简明教程 | 2019-10-12 | 80 | NULL |

| 5 | Django 入门与实践 | 1983-01-05 | 728 | NULL |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 | NULL |

+----+--------------------------------+------------+------------+-----------+

6 rows in set (0.01 sec)

可以看到 learn\_course 表新增了 author\_id 这列，且全部为空值。和一对一相同，列名是由下划线将外键关联的映射类 Author 的全小写和 Author 类的主键名 id 连起来生成的。

### 添加和查询数据

现在数据是有的：3 个 Author 类实例，6 个 Course 类实例。所谓 ”添加数据“ 就是让课程从属于某个作者，下面用一些方法将它们关联起来。

终端执行 python manage.py shell ，首先查询得到第一个作者，也就是卡卡西，卡殿：

from learn.models import Author, Course

# 可以使用下面的三种方法之一：

a1 = Author.objects.first() # 方法 1

a1 = Author.objects.get(id=1) # 方法 2

a1 = Author.objects.get(pk=1) # 方法 3

a1

<Author: 卡卡西>

着重说一下方法 3 ，pk 这个参数的意思就是 Primary Key 主键，因为 id 字段是主键，所以可以这样写。如果自定义了某个非 int 类型的主键，就不可以这样写了。

因为 Course 类使用 ForeignKey 外键关联了 Author 类，虽然后者什么都没做，但后者的实例依然增加了一个特别的属性（帅！） course\_set ，该属性的值是 RelatedManager 类，该类又有很多方法。这个属性的命名是 Course 类名的全小写 course 加上 \_set 拼起来的：

所以 a1.course\_set 属性就拥有很多方法，其中 add 方法可以将 Course 类的实例作为参数，执行结果就是将课程从属于该作者。

把前三个课程全部给卡卡西：

for c in Course.objects.all()[:3]:

a1.course\_set.add(c)

在数据表中，learn\_author 表是没有变动的，看下 leanr\_course 表：

SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+-----------+

| id | name | pub\_date | stu\_number | author\_id |

+----+--------------------------------+------------+------------+-----------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 | 1 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 | 1 |

| 3 | Python 异步编程 | 2019-04-03 | 59 | 1 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 | NULL |

| 5 | Django 入门与实践 | 1983-01-05 | 728 | NULL |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 | NULL |

+----+--------------------------------+------------+------------+-----------+

6 rows in set (0.00 sec)

前三个课程的 author\_id 变成了 1 ，符合预期。

a1.course\_set 的 all 方法可以获取属于自己的课程实例：

a1.course\_set.all()

Out[11]: <QuerySet [<Course: Linux 基础教程>, <Course: Git 与 Github 入门实践>, <Course: Python 异步编程>]>

a1.course\_set 的 count 方法可以获取作者所拥有的课程的数量：

a1.course\_set.count()

Out[12]: 3

a1.course\_set 的 remove 方法可以移除课程实例：

a1.course\_set.remove(Course.objects.get(name='Python 异步编程'))

a1.course\_set.count()

Out[14]: 2

这是作者实例添加课程实例的方法，（考虑提问？）课程主动添加作者该如何操作呢？

这就更简单了，Course 类的实例有 author 属性，直接赋值即可：

c3 = Course.objects.get(name='Python 异步编程')

c3.id

Out[16]: 3

c3.name

Out[17]: 'Python 异步编程'

c3.author

# 此时c3课程没有作者（因为刚才卡殿删除了），输出为空

a2 = Author.objects.get(pk=2)

Out[20]: <Author: 娜美>

# 简单而直接的关联

c3.author = a2

c3.save()

添加完毕，查看数据库：

SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+-----------+

| id | name | pub\_date | stu\_number | author\_id |

+----+--------------------------------+------------+------------+-----------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 | 1 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 | 1 |

| 3 | Python 异步编程 | 2019-04-03 | 59 | 2 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 | NULL |

| 5 | Django 入门与实践 | 1983-01-05 | 728 | NULL |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 | NULL |

+----+--------------------------------+------------+------------+-----------+

6 rows in set (0.01 sec)

这是使用 Course 类的 author 属性添加作者。

使用 author\_id 也是完全可以的。

将 id 为 4 和 5 的课程交给 id 为 2 的作者，也就是娜美：

for c in Course.objects.filter(id\_\_in=[4, 5]):

c.author\_id = 2

c.save()

查看数据表：

SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+-----------+

| id | name | pub\_date | stu\_number | author\_id |

+----+--------------------------------+------------+------------+-----------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 | 1 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 | 1 |

| 3 | Python 异步编程 | 2019-04-03 | 59 | 2 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 | 2 |

| 5 | Django 入门与实践 | 1983-01-05 | 728 | 2 |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 | NULL |

+----+--------------------------------+------------+------------+-----------+

6 rows in set (0.01 sec)

最后，就只能将 id 为 6 的课程交给宋神宗吧：

a3 = Author.objects.get(name='宋神宗')

a3.course\_set.add(Course.objects.get(id=6))

最后介绍 a3.course\_set 的 create 方法，此方法可以创建一个 Course 类的实例并且将其添加到数据表中，同时该课程实例属于 a3 ：

a3.course\_set.create(name = 'MySQL 基础课程',pub\_date = '2001-2-22',stu\_number = 333)

Out[29]: <Course: MySQL 基础课程>

此时 learn\_course 数据表中的情况：

SELECT \* FROM learn\_course;

>>>

+----+--------------------------------+------------+------------+-----------+

| id | name | pub\_date | stu\_number | author\_id |

+----+--------------------------------+------------+------------+-----------+

| 1 | Linux 基础教程 | 2011-01-12 | 123 | 1 |

| 2 | Git 与 Github 入门实践 | 2018-12-22 | 678 | 1 |

| 3 | Python 异步编程 | 2019-04-03 | 59 | 2 |

| 4 | Julia 简明教程 | 2019-10-12 | 80 | 2 |

| 5 | Django 入门与实践 | 1983-01-05 | 728 | 2 |

| 6 | C++ 实现简易 Docker 容器 | 1974-12-02 | 158 | 3 |

| 7 | MySQL 基础课程 | 2001-02-22 | 333 | 3 |

+----+--------------------------------+------------+------------+-----------+

7 rows in set (0.01 sec)

即：在给作者表中的作者添加课程的时候，同时将课程加入课程表。

## 总结

本部分详细介绍了两类数据模型关系：”一对一“ 和 ”一对多“ 。

创建数据表分别用什么类，如何添加数据、查询数据、删除数据等操作涉及到很多方法。这些方法并不需要死记硬背，关键是了解这两类模型的原理，提纲挈领，代码只是细节问题。

下一部分继续学习 ”多对多“ 关系模型。

#### 完整代码

本部分完成后的参考代码： code05.zip

## 作业：

结合讲授5，一起留置。

# 讲授 6 数据模型关系之多对多

## 介绍

上一部分了解到三种数据模型的关系并学习了其中的两种，本部分继续学习第三种模型关系 ：多对多关系。本部分接续上一部分的完成结果，在已有项目代码和数据的基础上继续添加新的操作。

#### 知识点

* 多对多关系
* 映射类多表查询

#### 完整代码

之前完成后的参考代码：code05.zip

**视需要，自行解压并在PyCharm中打开，同时需要同步数据库的数据表和数据。**

## 多对多关系

在前面的讲授中，已经创建了 Course 映射类并利用它生成了数据表 learn\_course ，而且已经向表中添加了 7 条数据。

多对多关系体现在课程和标签中，一门课程可以有多个标签，一个标签可以贴在多门课程上。

### 创建数据表

现在创建标签表的映射类 Tag ，将以下代码补充到 learn/models.py 文件中：

class Tag(models.Model):

'''课程的标签表，包括 name 属性，与 Course 为多对多关系

'''

name = models.CharField(max\_length=32)

course = models.ManyToManyField('Course')

def \_\_str\_\_(self):

return self.name

其中 course 即为多对多关系的属性，使用 ManyToManyField 创建，参数为 Course 类的类名字符串。

创建迁移文件和执行迁移操作：

cd venv/Scripts

activate

cd myweb

python manage.py makemigrations

python manage.py migrate

查看数据库：

SHOW TABLES LIKE 'learn%';

>>>

+--------------------------+

| Tables\_in\_myweb (learn%) |

+--------------------------+

| learn\_author |

| learn\_authordetail |

| learn\_course |

| learn\_tag |

| learn\_tag\_course |

+--------------------------+

5 rows in set (0.01 sec)

可以看到数据库中新增了两张表：learn\_tag 和 learn\_tag\_course ，后者的名字是由下划线将应用名 learn 、映射类 Tag 和 Course 的全小写连起来生成的。

（帅！）

learn\_tag 表的详情：

DESC learn\_tag;

>>>

+-------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| name | varchar(32) | NO | | NULL | |

+-------+-------------+------+-----+---------+----------------+

2 rows in set (0.00 sec)

只有 id 主键字段和 name 字段，并无新增字段。

learn\_tag\_course 表的详情：

DESC learn\_tag\_course;

>>>

+-----------+---------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-----------+---------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| tag\_id | int(11) | NO | MUL | NULL | |

| course\_id | int(11) | NO | MUL | NULL | |

+-----------+---------+------+-----+---------+----------------+

3 rows in set (0.01 sec)

（必须提问学生！？？）为什么会有这一张表呢？

这就是关系表，是自动生成的！因为 learn\_course 和 learn\_tag 是多对多关系，但它们之间并无相关字段，只有映射类有相关属性。所以在数据库中需要增加一张关系表，也叫中间表，以确定二者的多对多关系。

表中有三个字段：id 为主键自增字段，tag\_id 和 course\_id 分别对应 learn\_tag 和 learn\_course 表的主键，也就是这两个字段的值必须属于对应的表的主键值。

### 添加和查询数据

直接创建数据，开始展示。首先创建一些 Tag 类的实例并存入数据表：

inert\_tag\_data.py

# 配置临时环境变量，只在代码运行时生效  
import os  
# 设置环境变量，必须放在最开始  
os.environ['DJANGO\_SETTINGS\_MODULE']="myweb.settings"  
# 导入django，同时初始化 Django 环境  
import django; django.setup()  
from learn.models import Tag, Course  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 name\_list = ['Python', 'Linux', 'Web', '网络安全', 'C++', '数据库', 'Julia', 'Git']  
 for name in name\_list:  
 Tag(name=name).save()

右键🡪run

或者：python -m insert\_tag\_data.py

或者：上述代码，逐条在 python manage.py shell 中执行

如上所示，首先引入所需映射类，然后创建 name 属性值的列表，for 循环创建实例并添加到数据表中。

查看数据表中的数据：

SELECT \* FROM learn\_tag;

+----+--------------+

| id | name |

+----+--------------+

| 1 | Python |

| 2 | Linux |

| 3 | Web |

| 4 | 网络安全 |

| 5 | C++ |

| 6 | 数据库 |

| 7 | Julia |

| 8 | Git |

+----+--------------+

8 rows in set (0.01 sec)

现在课程数据和标签数据都已经准备好，接下来为课程添加标签：

进入 python manage.py shell，后，

from learn.models import Tag, Course

t1 = Tag.objects.get(pk=1)

t1

Out[7]: <Tag: Python>

t1.name

Out[8]: 'Python'

从 learn\_course 表中可知，id 值为 3 和 5 的课程与 Python 有关，将这两个课程贴上 a1 这个标签：

c3, c5 = Course.objects.filter(id\_\_in=[3, 5])

c3.name

Out[10]: 'Python 异步编程'

c5.name

Out[11]: 'Django 入门与实践'

t1.course.add(c3)

t1.course.add(c5)

Tag 类的实例有 course 属性，这是在映射类中定义的，该属性的值是 ManyRelatedManager 类，类有很多方法，add 方法用于添加课程，以使该课程贴上这个标签。（是不是有些熟悉？没错，）在前面讲 ”一对一“ 和 ”一对多“ 时就讲过了。

看下数据库，这个操作与 learn\_tag 和 learn\_course 无关，前面已经讲过，所以看下中间表：

SELECT \* FROM learn\_tag\_course;

>>>

+----+--------+-----------+

| id | tag\_id | course\_id |

+----+--------+-----------+

| 1 | 1 | 3 |

| 2 | 1 | 5 |

+----+--------+-----------+

2 rows in set (0.01 sec)

符合预期，learn\_tag\_course 表中新增了两条数据。这两条数据的 tag\_id 字段值都是 t1 的 id 值，并且 course\_id 字段值分别对应 c3 和 c5 这两个课程实例的 id 属性值。

可以使用 t1.course 的 all 方法获取对应的课程实例：

t1.course.all()

Out[19]: <QuerySet [<Course: Python 异步编程>, <Course: Django 入门与实践>]>

假设 c3 不仅与 Python 有关，还与 Linux 和 Git 有关，可以使用 c3 自己的方法来添加标签：

c3.tag\_set.add(Tag.objects.get(name='Linux'))

c3.tag\_set.add(Tag.objects.get(name='Git'))

同样地，Course 类没有设置多对多字段，它是被关联的一方，所以该类的实例所新增的属性就是 Tag 类名的全小写加上 \_set ，即 tag\_set 。如果把 Tag 类的 ManyToManyField 字段移到 Course 类中，是否可以？是可以的，完全没有问题，它们是平等关系。

现在查看一下 learn\_tag\_course 数据表：

SELECT \* FROM learn\_tag\_course;

+----+--------+-----------+

| id | tag\_id | course\_id |

+----+--------+-----------+

| 1 | 1 | 3 |

| 2 | 1 | 5 |

| 3 | 2 | 3 |

| 4 | 8 | 3 |

+----+--------+-----------+

4 rows in set (0.00 sec)

为 id 值为 3 的课程增加了两条数据，符合预期。

同样地，Course 类实例和 Tag 类实例都有自己的移除关联的方法：

c3.tag\_set.all()

Out[23]: <QuerySet [<Tag: Python>, <Tag: Linux>, <Tag: Git>]>

c3.tag\_set.remove(t1)

c3.tag\_set.all()

Out[25]: <QuerySet [<Tag: Linux>, <Tag: Git>]>

t1.course.all()

Out[27]: <QuerySet [<Course: Django 入门与实践>]>

t1.course.remove(c5)

t1.course.all()

Out[29]: <QuerySet []>

以上就是多对多的数据表的创建方式，以及数据的增删查操作。

## 映射类多表查询

前面介绍的三种数据模型关系中的查询方式都是先获取实例，然后使用实例的方法进行查询，这是符合正常的思维方式的。接下来讲授映射类本身具有的查询方法，也就是在前面的内容中讲解的 Class.objects.filter 方法过滤查询。

1、一对一关系，查询卡卡西的作者详情实例：

from learn.models import Author, AuthorDetail

result = AuthorDetail.objects.filter(author\_\_name='卡卡西')

result[0]

Out[37]: <AuthorDetail: 卡卡西>

想要查询的是什么？是 AuthorDetail 类的实例，所以一定是 AuthorDetail.objects.filter 来查询。

第 2 行代码中 filter 方法的参数是 author\_\_name ，它是用双下划线将 Author 类的全小写与 name 属性连接起来生成的，等号后面是属性值。查询结果为 QuerySet 实例，不要忽略这一点。

2、一对多关系，查询娜美的全部课程的列表：

result = list(Course.objects.filter(author\_\_name='娜美'))

result

Out[42]: [<Course: Python 异步编程>, <Course: Julia 简明教程>, <Course: Django 入门与实践>]

要查询的是课程实例的列表，那就一定是用 Course.objects.filter 来查询，参数名同上。

将查询结果的 QuerySet 对象作为 list 方法的参数返回 Course 实例列表。

查询课程名字中包含 ”教程“ 二字的作者的实例列表：

result = list(Author.objects.filter(course\_\_name\_\_contains='教程'))

result

Out[44]: [<Author: 卡卡西>, <Author: 娜美>]

如上所示，查询的是作者实例的列表，一定是用 Author.objects.filter 方法。参数是 course\_\_name ，再加上模糊查询的 contains 字段，同样用双下划线连接，所以完整的参数名就是 course\_\_name\_\_contains 。

这步查询操作转换成 SQL 语句，就是这样：

SELECT a.id, a.name, a.gender FROM learn\_author AS a INNER JOIN learn\_course AS c ON a.id = c.author\_id WHERE c.name LIKE '%教程%';

>>>

+----+-----------+--------+

| id | name | gender |

+----+-----------+--------+

| 1 | 卡卡西 | 1 |

| 2 | 娜美 | 0 |

+----+-----------+--------+

2 rows in set (0.01 sec)

3、多对多关系，查询 id 为 3 的课程有哪些标签：

result = list(Tag.objects.filter(course\_\_id=3))

result

Out[47]: [<Tag: Linux>, <Tag: Git>]

这与前面的查询方式是一样的，不再赘述（书不要简短…）。

接下来添加一些多对多关系：

t2 = Tag.objects.get(id=2)

t8 = Tag.objects.get(id=8)

t2.course.add(Course.objects.get(id=1))

t8.course.add(Course.objects.get(id=2))

此时关系表 learn\_tag\_course 中有如下数据：

SELECT \* FROM learn\_tag\_course;

+----+--------+-----------+

| id | tag\_id | course\_id |

+----+--------+-----------+

| 5 | 2 | 1 |

| 3 | 2 | 3 |

| 6 | 8 | 2 |

| 4 | 8 | 3 |

+----+--------+-----------+

4 rows in set (0.00 sec)

共 4 组关联，而且标签 t2 和 t8 都与 c3 关联，现在查询一下贴上 Linux 或 Git 标签的全部课程列表：

result = list(Course.objects.filter(tag\_\_name\_\_in=['Linux', 'Git']))

result

Out[53]:

[<Course: Linux 基础教程>,

<Course: Python 异步编程>,

<Course: Git 与 Github 入门实践>,

<Course: Python 异步编程>]

发现其中有重复数据，c3 出现了两次，这也符合预期。可以再调用一次 set 方法去重：

result = list(set(Course.objects.filter(tag\_\_name\_\_in=['Linux', 'Git'])))

result

Out[58]: [<Course: Linux 基础教程>, <Course: Git 与 Github 入门实践>, <Course: Python 异步编程>]

## 总结

本部分学习了多对多关系模型的映射类编写以及数据的增删改查操作，并且掌握了使用映射类进行多表联合查询的操作。

至此为止，对数据模型的学习就告一段落。

在接下来的讲授中，将学习 Django 中的视图与网址。

#### 完整代码

本部分完成后的参考代码：code06.zip

## 作业：

1. 请系统总结归纳Django中三种关系模型的实现的概念、要点、步骤。

2. 试用图书、出版社、作者、作者详细信息这四个实体，分析四个实体间的关系、设计字段，并实现映射类，描述各自的映射关系，完成数据库迁移和数据表中数据的插入。（要求使用Python编码实现）

3. 详述你对加入映射关系后的映射类实例中的额外属性（如set等）的理解，并结合代码演示论述。

4. 试举例说明（各至少举出6个例子）：

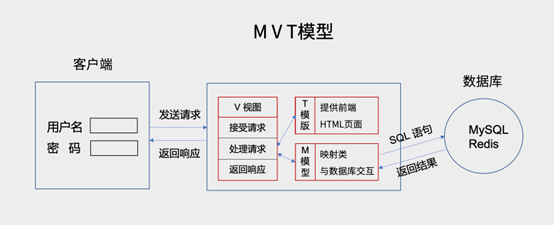
1）设置好映射关系（关联关系）后的查询实现

2）多表查询的实现

# 讲授 7 URL 路由基础

## 介绍

回顾：还记得第一部分中的 MVT 模型示意图吗？



前面几部分，其实介绍了MVT 模型中的 M 模型，接下来介绍其中的 V 视图。也就是负责接收浏览器请求并将处理结果返回给浏览器的那部分功能。

这部分被称为 “视图层” ，是 Django 框架的核心代码层。它接收浏览器的请求，并调用 T 模板和 M 模型将处理好的数据与前端部分结合，生成响应对象返回给浏览器。这是 BS 架构的真正后端层面。

#### 知识点

* URL 简介
* path 转换器
* 路由转发

#### 完整代码

之前部分完成后的参考代码：code06.zip

## URL 简介

URL 的英文全称是 **U**niform **R**esource **L**ocator ，中文名叫做 “统一资源定位符” ，俗称 “URL 地址”、“路由” 或 “网址” 。它是网络资源的地址，就像网络上的门牌号。

网址的标准格式如下：

[协议类型]://[服务器地址]:[端口号]/[资源层级 UNIX 文件路径]?[参数]#[书签]

以这个网址为例：

http://www.abc.com.cn/courses/9527/learning/?id=1314

* 协议类型：http ，后面需要跟随 :// 字样
* 服务器地址（也叫 “域名” ）：www.abc.com.cn
* 端口号：所有网址的默认端口号均为 80 ，如果写明端口号，前面使用 : 与域名连接
* 层级文件路径：/courses/9527/learning/ ，以 / 作为连接符
* 参数：id=1314 ，使用 ? 作为与路径的分隔符，参数名和参数值用 = 连接，多个参数使用 & 号连接

其中端口号、参数和书签是选填部分。

当浏览器访问上述地址，网站会收到请求，交由 courses/ 路径对应的视图函数进行处理。MVT 模型中的 V 就是用来处理这部分工作的。

## 准备

接续上一部分的操作，已经创建了 myweb Django 项目，并生成了 learn 应用，在应用中的 models.py 文件中创建了一些映射类。

在第二个讲授中创建一个简单的 Django 项目时，已经编写了一个视图函数，它在 learn/views.py 文件中：

from django.shortcuts import render

from django.http import HttpResponse

def index(request):

return HttpResponse('Hello World.')

其中index 函数就是视图函数，也叫做 “路由函数” 。

终端切换到 myweb 目录下，执行启动项目的命令：

cd myweb

net start mysql

venv/Scripts/activate

python manage.py runserver 0.0.0.0:8080

启动程序后，当使用 **Web 服务** 访问项目的主页时，服务器会调用这个函数进行处理，返回带有 'Hello World.' 字符串的响应对象，浏览器页面上就会显示这个字符串：



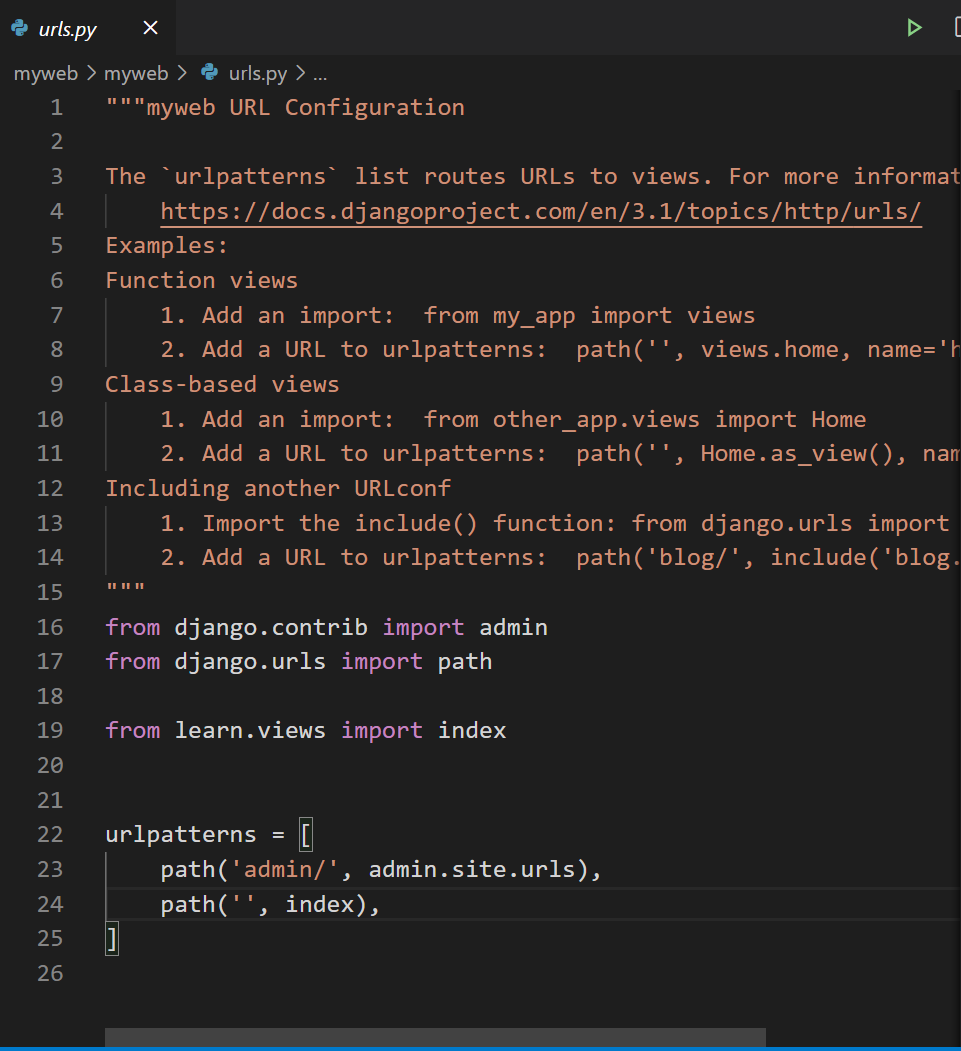
## Django 中的路由管理

在 Django 项目中用于管理路由的叫做 URLConf 模块，在 myweb/settings.py 文件中 ROOT\_URLCONF 字段用来设置它：

ROOT\_URLCONF = 'myweb.urls'

它的默认值就是 myweb/urls.py 文件这个模块的字符串。

在讲授 2 中曾经改写过这个文件，查看它的代码：



前 15 行是帮助信息，第 16 、17 行是默认存在的引入对象的代码，第 19 行引入 learn/views.py 文件中的 index 函数。

接下来在 learn/views.py 文件中再次写入另一个视图函数 courses ：

from django.shortcuts import render

from django.http import HttpResponse

# 引入所需映射类

from .models import Course

def index(request):

return HttpResponse('Hello Shiyanlou.')

# 新建视图函数

def courses(request):

# 从数据库中获取全部课程数据

courses = Course.objects.all()

s=""

for course in courses:

s+=str(course)

# 使用 </br> 标签将各个课程数据连起来生成字符串

# 其中 </br> 标签在浏览器页面上起到换行的作用

s+="</br>"

return HttpResponse(s)

下一步，修改 myweb/urls.py 文件：

from django.contrib import admin

from django.urls import path

from learn.views import index, courses # 引入 courses 视图函数

urlpatterns = [

path('admin/', admin.site.urls),

path('', index),

path('courses', courses) # 新增路由与对应的视图函数生成新的接口

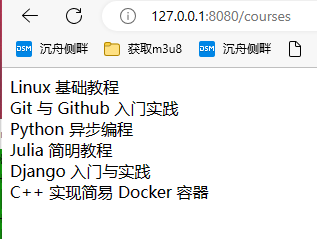
]

VSCode 在线环境可能会发生报错，这是由于 VSCode 中 Python 插件默认使用 pylint 而 pylint 是静态检查，模型未运行时不会有 objects 直接启动项目无视即可。

启动项目：

python manage.py runserver 0.0.0.0:8080

启动 **Web 服务，在浏览器中访问**[127.0.0.1:8080/courses](http://127.0.0.1:8080/courses) ：



现在修改 myweb/urls 中的 urlpatterns 列表最后一项：

urlpatterns = [

path('admin/', admin.site.urls),

path('', index),

path('courses/1', courses)

]

在 path 方法的字符串后面增加 '/1' ，此时就无法访问 http://127.0.0.1:8080/courses 地址了，

因为没有提供对应的视图函数，但是访问 http://127.0.0.1:8080/courses/1 地址，也会有问题。

此处不再贴图。

### 从 URL 中提取数据

想要展示每个课程的详细信息，提供这样的 URL ：http://127.0.0.1:8080/course/id ，其中 id 为 learn\_course 数据表中 id 字段的值，也就是数字。对应不同的 id 值，要展示相应的课程信息，所以视图函数在收到浏览器的请求时，需要从 URL 中提取 id 字段的值，从数据库中查找到相应的课程数据，处理后返回给浏览器。

首先编写对应的视图函数 course （不带 s ）：

def course(request, id):

course = Course.objects.get(id=id)

s = 'ID: {}</br>Name: {}</br>发布时间：{}</br>学生人数：{}'.format(

course.id, course.name, course.pub\_date, course.stu\_number)

return HttpResponse(s)

如何在 URL 的路径中提取数据呢？可以在 urlpatterns 列表的 path 方法的第一个参数字符串中添加尖括号，修改 myweb/urls.py 文件如下：

from django.contrib import admin

from django.urls import path

from learn.views import index, courses, course # 增加引入 course 函数

urlpatterns = [

path('admin/', admin.site.urls),

path('', index),

path('courses/1', courses),

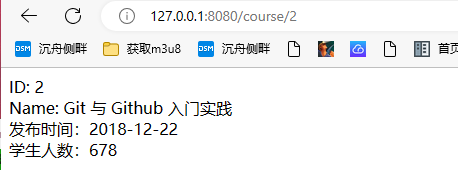
path('course/<int:id>', course), # 增加网址到视图函数的映射

]

如上所示，在 path 的第一个参数中添加一个 <int:id> ，冒号后面的是变量名，冒号前面是变量的数据类型。当浏览器发送 http://127.0.0.1:8080/course/2 请求时，项目的路由管理器会从 myweb/urls.py 文件中的 urlpatterns 列表中查询匹配，查到最后一项，就匹配到了。接下来调用 learn/views.py 文件中的 course 函数来处理请求，其中函数的 id 参数的值就是 2 。

course 函数内部从数据表中获取 id 字段值为 2 的那一行数据，处理成字符串后，生成 Response 对象返回给浏览器：

浏览器中访问：[127.0.0.1:8080/course/2](http://127.0.0.1:8080/course/2)



在 urlpatterns 列表中的 path 方法的第一个参数里，<int:id> 叫做**路径转换器**，它会把 URL 中的路径部分转换成指定的数据类型。如果不转换的话，可以直接使用尖括号中写入变量名的方式：http://127.0.0.1:8080/course/<id> ，那么 id 默认就是字符串数据类型，在视图函数 course 的 id 参数值也就是字符串类型了。

路径转换器有如下几种：

| **转换器类型** | **说明** |
| --- | --- |
| str | 匹配任何非空字符串，但不包含 / 。在不指定转换器的情况下，变量默认就是字符串数据类型。 |
| int | 匹配非负整数，返回 int 类型的变量值。 |
| slug | 可理解为注释、后缀、附属等概念，是在 URL 最后的一部分描述性性字符。该转换器匹配任何 ASCII 字符以及连接符和下划线，例如 hello-kitty-how-are-you 。 |
| uuid | 匹配一个 UUID 格式的对象，即通用唯一识别码，须符合识别码的格式规范，使用减号连接。 |
| path | 与 str 很像，也是匹配非空字符串，但它包含 / 。举例而言，a/b/c 可以被 path 匹配，但不可以被 str 匹配。 |

### 使用正则表达式

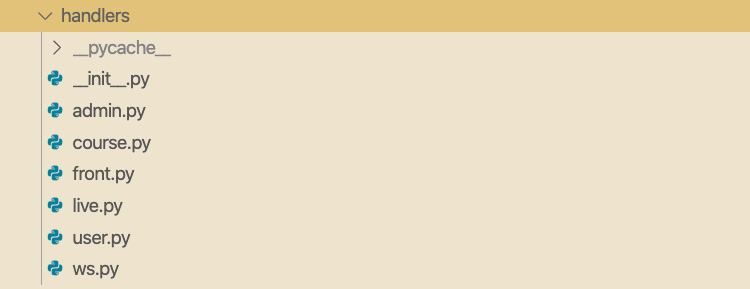
因为 Django2 以上的版本中已经支持了 path 方法创建 URL 与对应的视图函数进行映射，所以老旧的 url 方法就不再详细介绍。

在 Django1 中没有 path 函数，只有 url 函数，第一个参数可以使用正则表达式来匹配。有兴趣可以参阅 [这篇博客](https://kinegratii.github.io/2017/09/25/django2-url-path/) 。

### 路由转发

通常一个稍有规模的网站会提供成百上千个 URL ，也就是说，myweb/urls.py 文件中的 urlpatterns 列表中要写几百个 path ，这对于网站的维护和新增功能来说是相当麻烦的。

熟悉 Flask 框架的同学应该知道 Flask 有蓝图 Blueprint 的概念：



首先把项目的路由进行分类：

* 与首页相关的路由函数，写到 front.py 文件中。例如网站首页、注册、登录、错误页等。
* 与课程相关的路由函数，写到 course.py 文件中。例如课程首页、课程详情页、创建课程、修改课程等。
* 与用户相关的路由函数，写到 user.py 文件中。例如用户的主页、修改个人信息等。

在每个文件中先创建一个蓝图对象，然后使用蓝图对象的 route 方法作为装饰器创建视图函数。这样就可以起到分类管理路由的作用，使得项目分门别类，各司其职，有条不紊，管理有序。

为了达到同样的目的，在 Django 中实际上做得更彻底，就是将项目中各个功能分类，创建不同的应用以实现它们。现在myweb 项目中只有 learn 应用，后面可能会有 user 、admin 、auth 等各种应用。

（下面是关键，要缓缓的讲清楚。）

就路由管理而言，把每个应用提供的视图函数都写到 myweb/urls.py 文件中，肯定是混乱的。如何做到 Flask Blueprint 那样分门别类呢？答案就是**在每个应用中创建一个 urls.py 文件，在文件中创建 urlpatterns 列表用于管理该应用中的视图函数，最后再由 myweb/urls.py 进行统一管理**。

现在来改写代码以实现这种模式。现在所有 http://xxx/learn/xxx 样式的路由都交给 learn 应用中的视图函数来处理，未来所有 http://xxx/user/xxx 样式的路由都交给 user 应用中的视图函数来处理，以此类推。

首先 learn/views.py 是不需要变动的，它只负责创建视图函数，至于哪里要用视图函数，怎么用，它是不考虑的。

然后，在 learn 目录下新建 urls.py 文件，将以下代码写入其中：

from django.urls import path

from .views import index, courses, course

urlpatterns = [

path('', index),

path('courses', courses),

path('course/<int:id>', course)

]

最后修改 myweb/urls.py 文件如下：

from django.contrib import admin

from django.urls import path, include # 引入 include 方法

#from learn.views import index, courses, course

urlpatterns = [

path('admin/', admin.site.urls),

#path('', index),

#path('courses', courses),

#path('course/<int:id>', course),

path('learn/', include('learn.urls'))

]

第 2 行引入 include 方法，第 3 行注释掉从 learn/views.py 引入对象的代码。在 urlpatterns 路由表中添加一行 path('learn/', include('learn.urls')) ，这就是路由转发。

此时，可访问的url有：

http://127.0.0.1:8080/learn

http://127.0.0.1:8080/learn/courses

http://127.0.0.1:8080/learn/course/2 (可以是：/1,/2,/3…)

### 路由转发示例

举两个例子进行说明。

#### 例一

当浏览器发送 http://127.0.0.1:8080/learn/ 请求到服务器，服务器会从 URL 中去掉协议和域名，也就是去掉 http://127.0.0.1:8080/ 留下 'learn/' 字段，这叫做 URL 片段。

# myweb/urls.py

urlpatterns = [

path('admin/', admin.site.urls),

path('learn/', include('learn.urls'))

]

首先在 myweb/urls.py 中的根路由表也就是 urlpatterns 列表中查询 'learn/' ，发现最后一个 path 能够匹配 'learn/' 。这里 include 的作用就是路由转发，参数为 learn/urls.py 的模块格式的字符串，即 'learn.urls' ，也就是余下的 URL 片段交由 learn/urls.py 文件中的路由表来处理。

URL 片段在 myweb/urls.py 中匹配完毕后，去掉 'learn/' ，就没有内容了，只剩下空字符 '' 。最后在 learn/urls.py 的路由表中匹配到第一个 path ：

# learn/urls.py

urlpatterns = [

path('', index),

path('courses', courses),

path('course/<int:id>', course)

]

该 path 表示由 index 函数来处理，路由这部分就完成了任务，下一步交给 learn/views.py 文件中的 index 函数继续处理。

#### 例二

当浏览器发送 http://127.0.0.1:8080/learn/course/3 请求到服务器，服务器从 URL 中去掉协议和域名后，URL 片段为 'learn/course/3' 。该片段在 myweb/urls.py 的根路由表中匹配到最后一个 path ，URL 片段去掉 'learn/' ，剩下 'course/3' 。URL 片段被转发到 learn/urls.py 的路由表中进行匹配：

# learn/urls.py

urlpatterns = [

path('', index),

path('courses', courses),

path('course/<int:id>', course)

]

匹配到最后一个 path ，下一步交给 learn/views.py 文件中的 course 函数处理。其中的路由转换器将 '3' 转换为 int 类型的数值作为 id 参数的值传入 course 函数：

# learn/views.py

def course(request, id):

course = Course.objects.get(id=id)

s = 'ID: {}</br>Name: {}</br>发布时间：{}</br>学生人数：{}'.format(

course.id, course.name, course.pub\_date, course.stu\_number)

return HttpResponse(s)

需要注意的是 include 方法的参数为字符串，字符串是 learn/urls.py 对应的模块名。

## 总结

本部分讲授了 URL 的作用和组成，每一段的格式和功能以及是否必须存在。Django 中的路由管理在稍具规模的项目中都是由总路由表统管，在各个应用中创建自己的 urls.py 文件分管各自相关的路由。这种模式类似于 Flask 的蓝图模式，目的都是为了使路由的管理更加有条不紊，纲举目张（界门纲目科属种…杨柳青电厂…）。

下一部分继续学习 V 视图层的相关知识。

#### 完整代码

本部分完成后的参考代码：code07.zip

## 作业：

1. 总结Django中的路由转发的实现细节，并描述实现步骤、及其和views.py中函数的对应关系。

2. 尝试对比Flask蓝图和Django路由转发的异同。

# 挑战作业 2 创建求和视图函数

# 创建求和视图函数

## 挑战介绍

这次挑战的作业是在上一部分的基础上在 learn 应用中创建一个求和的视图函数，首先做一些准备工作。

打开环境，在终端获取项目代码、解压并在PyCharm中打开：

code07.zip

创建虚拟环境，安装所需工具包（如果已完成，可忽略）：

virtualenv venv

venv/Scripts/activate

pip install django

pip install mysqlclient

启动 MySQL 服务，创建数据库：

net start mysql

mysql -uroot -p

CREATE SCHEMA myweb CHARSET = UTF8

切换到项目主目录，执行数据迁移（其实就是创建数据表）：

cd myweb

python manage.py migrate

查看数据库中的数据表：

mysql -uroot -p

SHOW TABLES IN myweb

数据表中的数据与挑战无关，可以不必补充。

#### 挑战要求

在 learn 应用中增加一个函数计算 URL 中的值并显示到屏幕上。函数名为 add\_a\_b ，参数为 a 和 b ，返回两个数的和即可。

例如浏览器上的 URL 为 http://127.0.0.1:8080/learn/add/2/3 返回到屏幕上的数字就是 5 。

完成后截图如下：



#### 知识点

* 视图函数的编写
* URL 映射
* 路由转换器

## 提示信息

* 只需要修改和补充 learn 应用中的 Python 文件
* 注意视图函数的第一个参数须为 request
* 路由转换器要使用 < > 括号

## 参考代码

在 learn/views.py 文件中补充如下代码：

def add\_a\_b(request, a, b):

return HttpResponse(a+b)

将 learn/urls.py 文件修改如下：

from django.urls import path

from .views import index, courses, course, add\_a\_b

urlpatterns = [

path('', index),

path('courses', courses),

path('course/<int:id>', course),

path('add/<int:a>/<int:b>', add\_a\_b),

]

# 讲授 8 HTTP 请求方法

## 介绍

本部分继续介绍视图相关的操作。

HTTP 请求有很多方法，包括 GET 、POST 、HEAD 、DELETE 、PUT 等。在前面的讲授中使用的就是默认的 GET 请求。本部分来学习这些请求方法，详细讲解 GET 和 POST 请求。

#### 知识点

* GET 请求
* POST 请求

#### 完整代码

前面部分完成后的参考代码：challenge02.zip

## GET 请求

GET 方法用来请求已被 URL 识别的资源。指定的资源经服务器端解析后返回响应内容，最常用于向服务器查询某些信息。必要时，可以将查询字符串参数追加到 URL 末尾，以便将信息发送给服务器。

接下来的操作要在上一部分的基础上进行。上一部分中在 learn/views.py 文件中编写了 3 个视图函数，全部没有指定请求方法，即只接受（默认）浏览器发来的 GET 方法的请求。

如何设置视图函数接收哪种请求方法呢？可以在定义函数时增加一个具名参数 method ，以 course 函数为例，可以改写如下：

def course(request, id, method=['GET']):

course = Course.objects.get(id=id)

s = 'ID: {}</br>Name: {}</br>发布时间：{}</br>学生人数：{}'.format(

course.id, course.name, course.pub\_date, course.stu\_number)

return HttpResponse(s)

支持 GET 方法，就在 method 的列表中添加字符串 'GET' ；支持 GET 和 POST 方法就写成 method=['GET', 'POST'] 。如果只支持一种也可以简写为 method='HEAD' 这样，省去列表。

### 模拟登录的视图函数

现在来使用 GET 方法模拟用户登录的路由。

首先在 learn/views.py 文件中增加如下视图函数：

def login(request):

name = request.GET.get('name')

password = request.GET.get('password')

if name == 'test' and password == 'test':

return HttpResponse('登录成功')

return render(request, 'learn/login.html')

如上所示，login 函数只有一个必选参数 request ，默认支持的请求方法为 GET 。

第 2 、3 行从 request 的 GET 属性中获取 name 和 password 字段的值。其中 request.GET 属性值为字典类型的对象，字典对象可以使用 get 方法获取键值对的 value 值。request.GET 对象中存储了浏览器发来的 URL 中的参数数据。

第 4 、5 行判断 name 和 password 是否符合要求，如果符合返回 '登录成功' 字符串。否则执行最后一行代码。

最后一行代码返回的是 render 方法的调用，该方法接收多个参数，其中位置参数有两个：

* 1、request 对象，它是视图函数的第一个参数；
* 2、指定前端模板文件路径的字符串，'learn/login.html' 即为 learn/templates 目录下的相对路径。

接下来在 learn/urls.py 中增加 login 函数的路由映射：

from django.urls import path

from .views import index, courses, course, login

urlpatterns = [

path('', index),

path('courses', courses),

path('course/<int:id>', course),

path('login', login),

]

当浏览器发送 URL 为 http://127.0.0.1:8080/learn/login 的请求时，就会由 learn/views.py 中的 login 函数处理。

现在在 learn 目录下创建多级目录 templates/learn 以存储 HTML 模板文件（在 myweb 目录下执行）：

mkdir learn\templates\learn

在新建目录下创建 login.html 模板文件，将以下内容写入其中：

<h1>登录</h1>

<title>Login</title>

<form action="/learn/login" method="GET">

<p>用户名：<input type="text" name="name" /></p>

<p>密码：<input type="password" name="password" /></p>

<p><input type="submit" value="提交" /></p>

</form>

这就是 learn/views.py 文件中 login 函数的最后一行代码 return render(request, 'learn/login.html') render 方法的第二个参数指定的模板文件。

当浏览器发出 http://127.0.0.1:8080/learn/login 请求时，login 视图函数会调用这个模板文件并返回给浏览器。

现在启动项目，打开浏览器输入地址http://127.0.0.1:8080/learn/login：

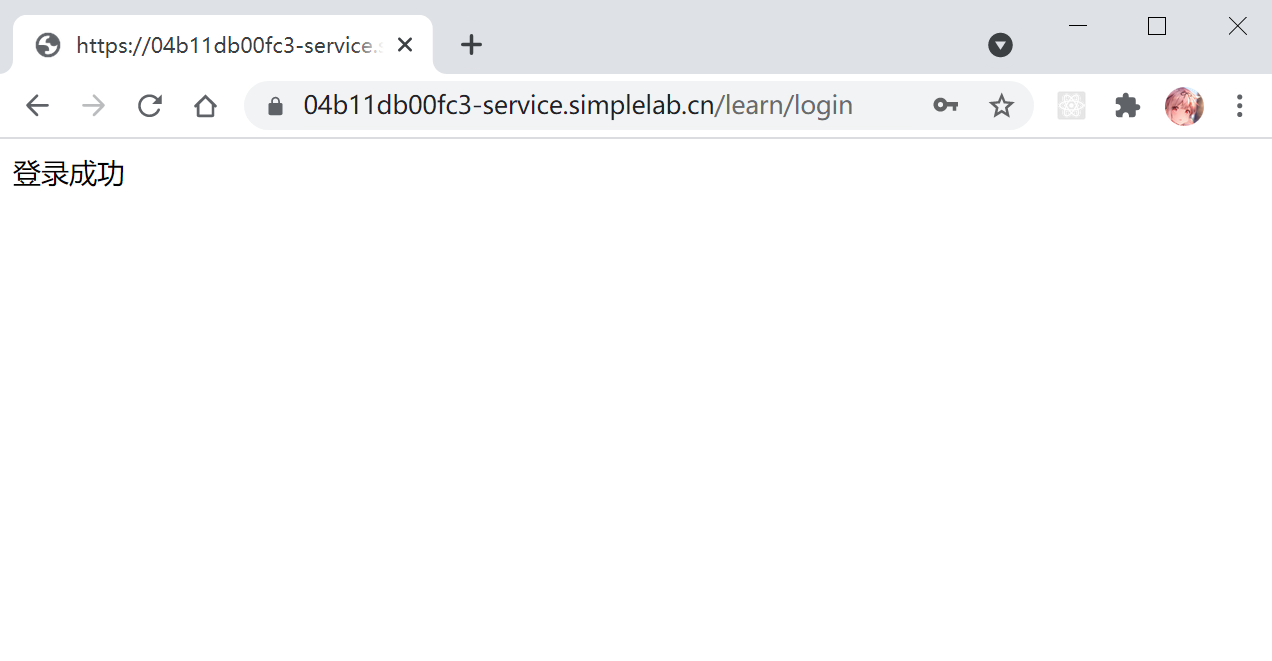


输入用户名 “test” 密码 “test” 点击提交：

这时，浏览器会将表单中输入的信息以参数的形式添加到 URL 的后面，以 ? 开头，多个参数以 & 分隔，用新生成的 URL 访问服务器。这是因为 HTML 中的 <form> 标签使用了 GET 方法发送请求：

<form action="/learn/login" method="GET"></form>

现在输入符合要求的用户名和密码（test/test），再次点击提交：



通过了 login 视图函数中的 if 判断，返回 '登录成功' 字符串。可以打开一个浏览器标签，主动构造一个 URL 向服务器发起请求。

根据上面的操作可以看出，GET 方法的请求会把数据以参数的形式拼接到 URL 后面，服务器中的视图函数可以使用 request.GET 属性获取这些数据（一般称为：明送）。

GET 方法有一个明显的缺点，所有信息都会暴露在 URL 上，URL 是公开的。因此，需要更加安全的方式，这就是下面要介绍的 POST 请求。

## POST 请求

与 GET 请求相对应的就是 POST 请求，也叫做表单请求。此方法的特点就是信息安全，只有收到请求的服务器通过 Request 对象才能获取数据。

将 GET 方法改为 POST 方法也很简单，首先修改 myweb/settings.py 中的配置项：

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',

]

将 MIDDLEWARE 中的第 4 行注释掉。这是一个中间件列表，第 4 行的作用是防止跨域伪造攻击（这是一种网络攻击的方式）。因为前面的模拟登录十分简陋，肯定通不过它的检测。

然后修改 learn/views.py 的 login 函数：

def login(request, methods=['GET', 'POST']):

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'shiyanlou' and password=='hello':

return HttpResponse('登录成功')

return render(request, 'learn/login.html')

首先增加参数 methods ，使该函数能够接受 GET 和 POST 方式的请求。在函数内部使用 request.POST 属性获取表单中的数据，该属性的值也是字典对象，所以仍然用 get 方法获取 value 值。

最后修改 learn/templates/learn/login.html 文件，只需修改 method 属性值为 POST：

<h1>登录</h1>

<title>Login</title>

<form action="/learn/login/" method="POST">

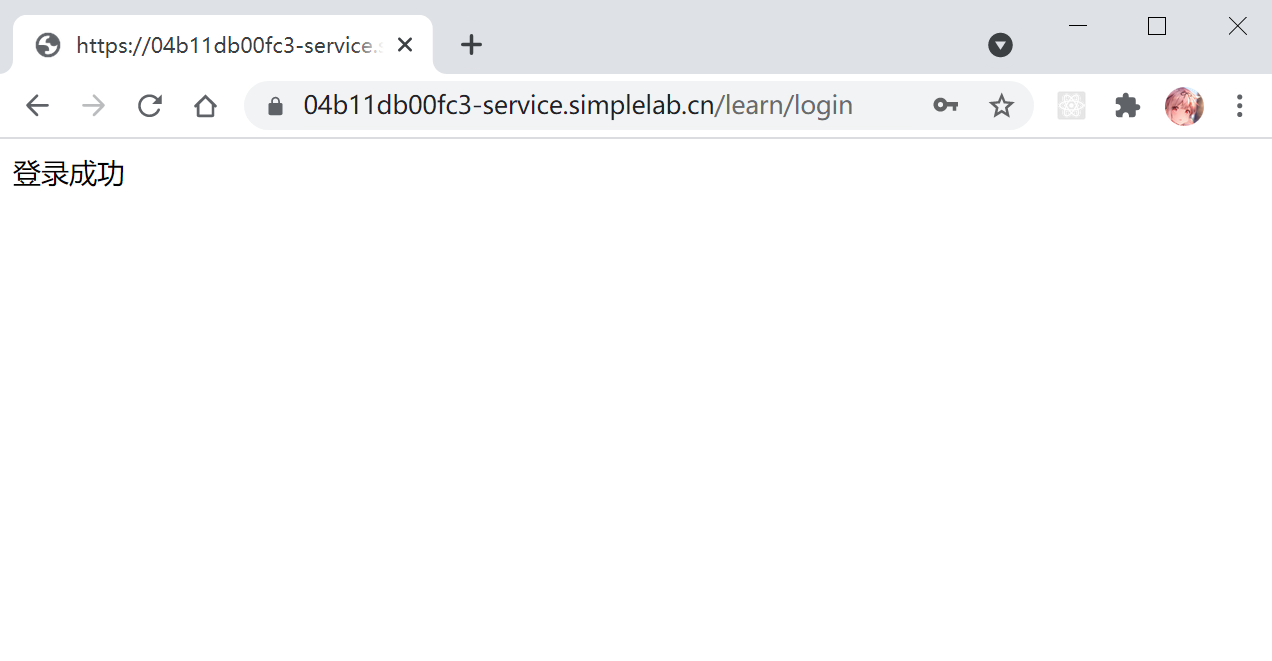
<p>用户名：<input type="text" name="name" /></p>

<p>密码：<input type="password" name="password" /></p>

<p><input type="submit" value="提交" /></p>

</form>

修改完毕后，打开浏览器，输入 http://127.0.0.1:8080/learn/login ，敲回车。填写表单，点击提交：



如上图所示，浏览器的地址栏中 URL 不再有参数显示出来，这样就保证了数据的安全。

## 其它请求

实际上大多数 HTTP 请求都是 GET 或 POST 方法，对于其它方法的请求，简单介绍一下（在爬虫中，其实都有讲过的）。

#### HEAD

HEAD 方法与 GET 方法一样，都是向服务器发出指定资源的请求。但是，服务器在响应 HEAD 请求时不会回传资源的内容部分，即：响应主体。这样，可以在不传输全部内容的情况下，就可以获取服务器的响应头信息。HEAD 方法常被用于客户端查看服务器的性能。

#### PUT

PUT 请求会向指定资源位置上传其最新内容，通过该方法客户端可以将指定资源的最新数据传送给服务器取代指定的资源的内容。

#### DELETE

DELETE 请求用于请求服务器删除所请求 URL 所标识的资源。DELETE 请求成功指定资源会被删除。

## 总结

本部分内容较简单，主要了解了 Django 框架中视图这部分如何处理 GET 、POST 请求，其实就是使用 request 对象的 GET 或 POST 属性来处理数据。

下一部分学习 Cookie 和 session 的使用。

#### 完整代码

本部分完成后的参考代码：code08.zip

## 作业：

1. 尝试完成用户注册功能，并将注册的用户信息插入数据库中。

# 讲授 9 状态保持

## 介绍

HTTP 协议是无状态的，每一次请求它们的关系都是相互独立的。但是在实际的应用中，确实有很多数据需要服务器记住，但又不适合存放在数据库中。比如说，一个登录页面需要在用户密码错误输入 3 次后要求输入验证码，服务器就需要一个计数器纪录错误次数，但把它放到数据库也不太合适（一般也不会放）。

如何处理实际应用中遇到的数据存储问题，这就需要服务器来想办法。目前在 BS 架构中解决数据存储问题的两个主要工具就是浏览器中添加 Cookie 和服务器中添加 Session 。

本部分来学习 Django 框架中 Cookie 和 Session 的创建与使用。

#### 知识点

* HTTP 协议简介
* Cookie 的创建与使用
* Session 的创建与使用

#### 完整代码

之前部分完成后的参考代码：code08.zip

## HTTP 协议简介

HTTP 是一个属于应用层的面向对象的协议，至于 “应用层” 是什么，可以搜一下 ”ISO 七层协议“ 。HTTP 协议一共有五大特点：1、支持客户端/服务器模式；2、简单快速；3、灵活；**4、无连接；5、无状态**。

#### 无连接

所谓 “无连接” 其实指的是无持续连接，也就是每次服务器通过 HTTP 收发请求后，与客户端的连接立即断开。与之对应的维持连接状态的场景，例如打电话，通话必须保持住，不可能我说完一句话就挂断，然后你再打回来说一句话。

#### 无状态

所谓 “无状态” 指的是协议对所处理的事务没有记忆功能。比方说用户 小明 在浏览器上登录了淘宝网，HTTP 是不知道的，下次访问淘宝，HTTP 仍然需要携带用户信息向淘宝发送请求。HTTP 不会记得 小明 曾经访问过哪些网站。

这就意味着 小明 每次访问淘宝都要输入账号和密码，实际使用中这么做的话就太麻烦了，而且网络传输的信息量变得冗长。

不过 HTTP 协议的设计模式也有其优点，简单灵活，反应迅速，不占用不必要的连接。

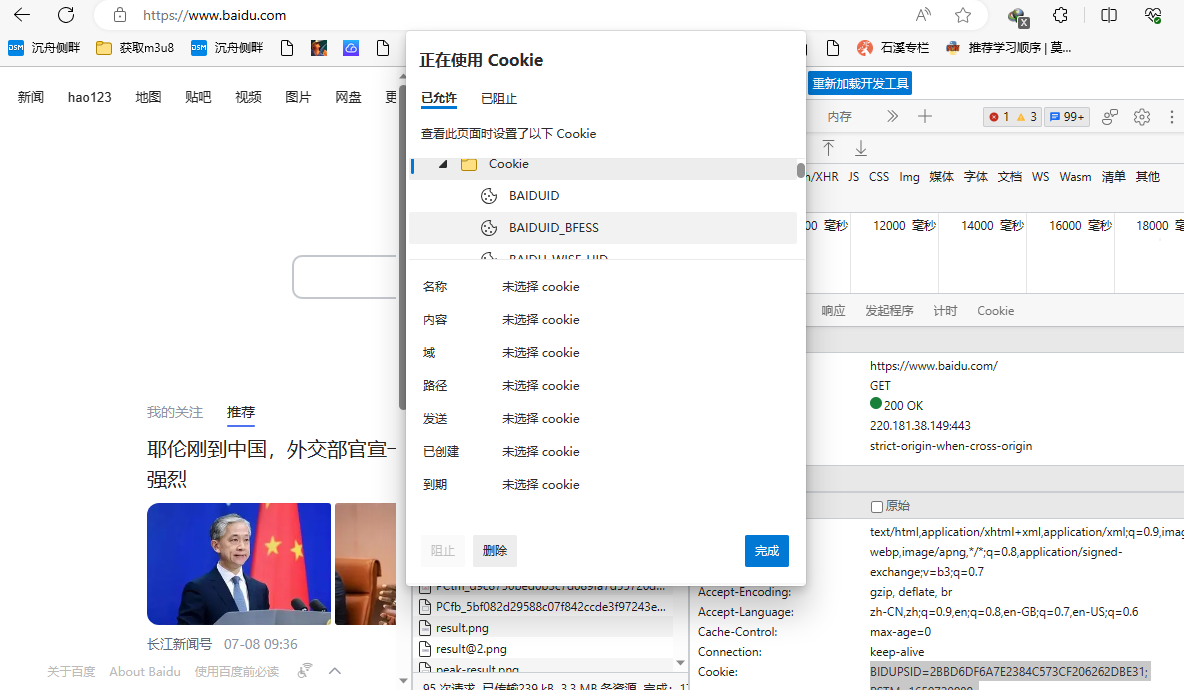
## Cookie 的创建和使用

提问：在浏览网站的过程中，经常会遇到需要登录的情况，有些页面只有登录之后才可以访问 ，而且登录之后可以连续访问很多次网站，但是有时候过一段时间就需要重新登录。 还有一些网站，在打开浏览器时就自动登录了，而且很长时间都不会失效，这种情况又是为什么？

本部分将了解这些。

几乎所有支持注册登录功能的网站都会使用 Cookie 保存用户的登录信息，而 Cookie 中的数据由服务器提供。

例如百度的用户登录后，浏览器中的 Cookie 如下：

### Cookie 的特点

Cookie 原意为小甜饼，它是存储在浏览器中的一段文本。当用户使用浏览器向服务器发送登录请求后，浏览器通过验证账号密码，确认信息无误后，返回给浏览器一个响应对象。浏览器收到响应后，解析数据，渲染页面到屏幕上，顺便将响应对象中提供的 Cookie 数据保存到本地。

当浏览器再次访问该网站，就会自动将 Cookie 数据携带。

Cookie 有时也用其复数形式 Cookies，指某些网站为了辨别用户身份、进行 session 跟踪而储存在用户本地终端上的数据（通常经过加密）。

Cookie 名称和值可以由服务器端自己定义，这样服务器可以知道该用户是否是合法用户以及是否需要重新登录等。服务器可以利用 Cookies 包含信息的任意性来筛选并经常性维护这些信息，以判断在 HTTP 传输中的状态。

通常服务器不会将用户的密码等敏感信息作为 Cookie 键值对中的数据发送给浏览器，因为浏览器可能被其他人使用，而且浏览器提供商也并非绝对可靠。

Cookie 的特点归纳如下：

* 以键值对的格式进行信息的存储
* 在服务端创建后，发送到客户端保存，客户端再次访问该网站时随着请求一起发送
* Cookie 基于域名安全，不同域名的 Cookie 是不能互相访问的（Cookie不能跨域）

### HttpRequest

在视图函数中，会看到第一个参数 request 为必选参数。这是服务器收到浏览器发送过来的请求后，根据报文（其实就是请求信息）自动创建的 HttpRequest 类的实例对象。

在上一部分中，已经接触了 request 参数的 GET 属性和 POST 属性，它们分别用于存储 GET 和 POST 方法发送的请求信息中的参数及其值的键值对。

HttpRequest 对象有很多属性和方法，常用的属性列表如下：

| **属性** | **说明** |
| --- | --- |
| GET | QueryDict 类型，存储 GET 方法请求的所有参数 |
| POST | QueryDict 类型，存储 POST 方法请求的所有参数 |
| path | 字符串类型，请求的完整路径，不含域名和参数 |
| method | 字符串类型，请求方法：GET / POST / HEAD / DELETE ... |
| encoding | 字符串类型，编码格式 |
| COOKIES | 字典类型，浏览器中保存的相关 Cookie 信息 |
| session | 字典类型，创建 session |
| headers | 字典类型，请求相关信息，包括 Content-Type 、User-Agent 等 |
| FILES | 字典类型，上传文件数据 |

其中 GET 和 POST 属性在上一部分中被描述为字典对象，这是因为 QueryDict 类型是 dict 类的子类。

### HttpResponse

视图函数收到请求后进行处理，最后返回一个 HttpResponse 对象，这一点在前面的讲授中已经介绍并应用过。

在 Django 项目中，创建 HttpResponse 实例的方法有两种：

* HttpResponse('xxx') ，引入 HttpResponse 类来创建实例
* render(request, 'xxx', context) ，调用 render 方法返回 HttpResponse 实例

这两种方法前面也已经应用过。

HttpResponse 对象也有很多属性和方法，以下为常用列表：

| **属性 / 方法** | **说明** |
| --- | --- |
| content | 返回的页面数据 |
| charset | 编码格式 |
| status\_code | HTTP 响应码 |
| content\_type | 返回的数据的类型 |
| set\_cookie | 设置 Cookie |

本部分将使用 HttpResponse 实例的 set\_cookie 方法设置 Cookie 。

### 设置 Cookie

修改 learn/views.py 中的 login 函数如下：

def login(request, methods=['GET', 'POST']):

if request.method == 'GET':

name = request.COOKIES.get('name') or 'Stranger'

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'test' and password=='test':

resp = HttpResponse('登录成功，{}'.format(name))

resp.set\_cookie('name', name)

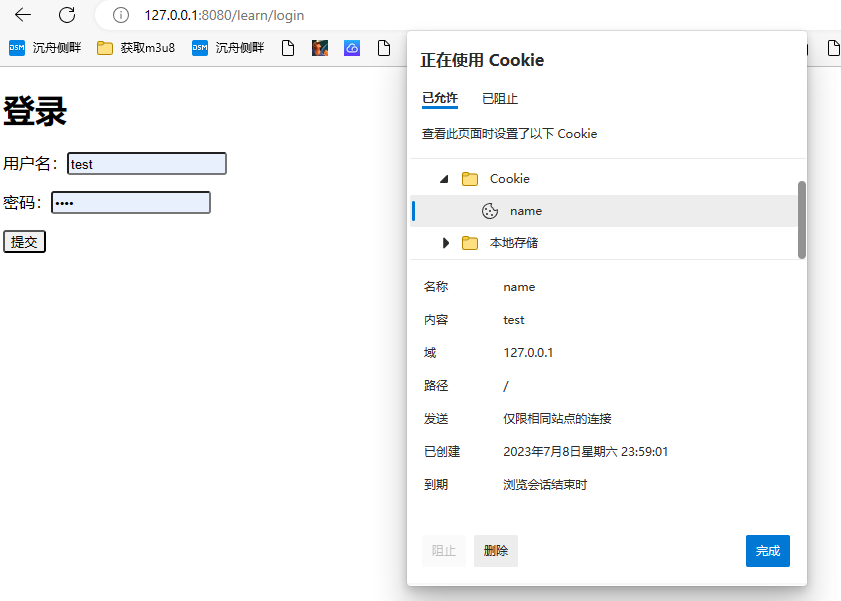
return resp

return render(request, 'learn/login.html', {'name': name})

首先使用 request.method 属性判断浏览器发送的请求类型。

如果是 GET 请求，从 request.COOKIES 属性中获取 name 字段的值，也就是用户名。第一次打开这个页面时，request.COOKIES 中没有设置 Cookie ，所以取值为 None ，这种情况下将 name 变量赋值为 'Stranger' 。这里用到了 Python 的三元表达式，意思是如果 or 前面的表达式的布尔值为 True （也就是 Cookie 字段已经设置好），那么 name 变量就是这个表达式的值；否则 name 变量的值为 'Stranger' (陌生人)。最后返回 render 方法的调用。注意 render 的最后一个参数为字典类型，它是位置参数，也是第三个参数，该参数的名字为 context 。因为是位置参数，所以可以省略参数名。也可以改写成：context={'name': name} 。这样，调用 learn/login.html 模板时，在模板内部可以直接使用变量 name （模板中暂时未显编写该变量的使用）。

如果是 POST 请求，从 request.POST 属性中获取 name 字段的值。当浏览器打开登录页面后，如下图所示：



此时没有浏览器中没有 Cookie ，填写登录表单：用户名为 test ，密码为 test 。点击提交后，浏览器再次向服务器发送请求。本次请求为 POST 方式，在 login 视图函数中运行第 2 个 if 语句块：

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'test' and password=='test':

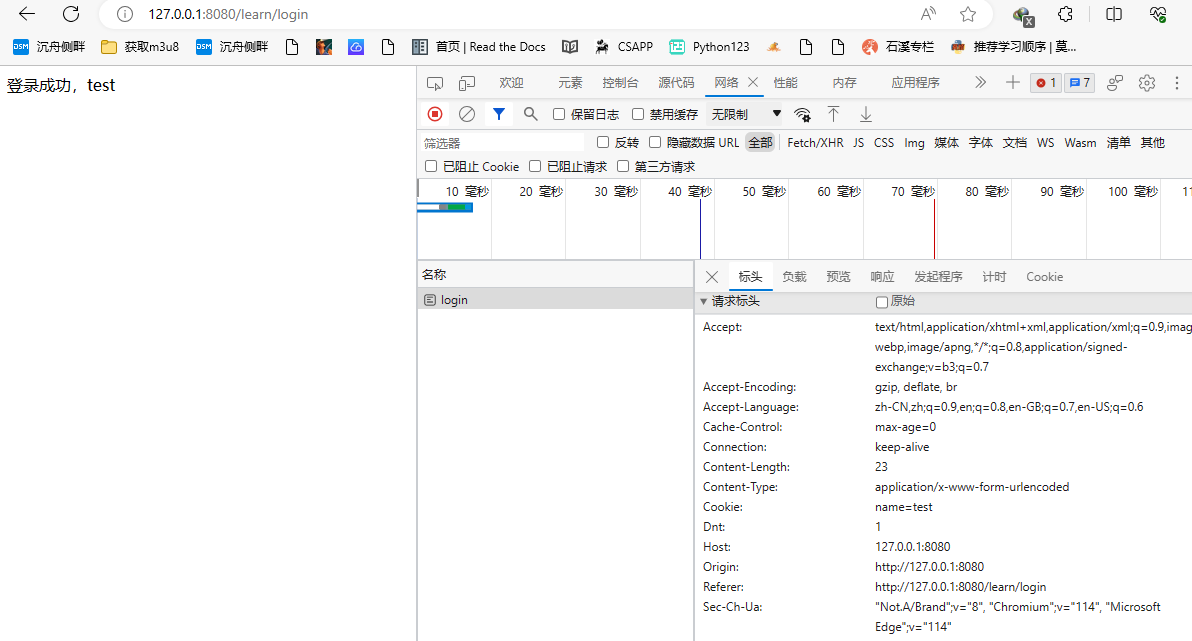
resp = HttpResponse('登录成功，{}'.format(name))

resp.set\_cookie('name', name)

return resp

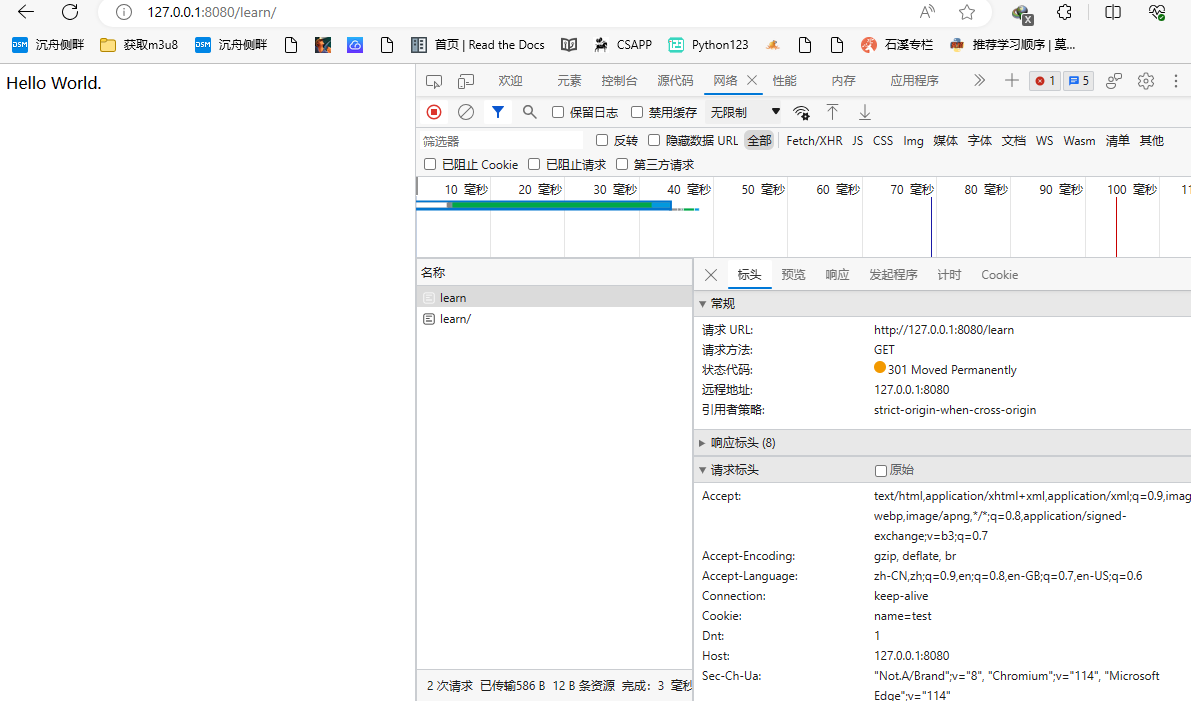
语句块内首先从 request.POST 中获取表单信息，然后嵌套 if 语句进行信息判断。符合要求的话，就创建 HttpResponse 类的实例，并赋值给 resp 变量。resp 调用 set\_cookie 方法设置 Cookie ，参数有两个，分别为 key 和 value 。最后返回 resp 对象给浏览器。

浏览器收到响应后，解析数据，将 “登录成功，test” 显示到屏幕上。同时保存数据中的 Cookie 信息到本地，如下图所示：



这就是完整的一次设置 Cookie 操作。如上图所示，Cookie 中的 name 对应两个键值对：value 是 'test' 这没有问题；其中还要一个 path 对应 '/' ，这是 name 这个 Cookie 的作用范围，表示该 Cookie 为全局有效。

此时再次使用浏览器访问 http://127.0.0.1:8080/learn/ ：



可以看到 Cookie 中依然存在 name ，这就是 path 的作用了。

此时再次访问 http://127.0.0.1:8080/learn/login ，这是默认的 GET 方法：



可以看到请求头中的 Cookie 字段中有 name=test 的键值对，说明浏览器在存储 Cookie 后，每次新的请求都会携带其中的信息。

### Cookie 的属性

Cookie 中的数据是由键值对组成的，每组键值对就是一个 Cookie ，所以 Cookie 其实应该是复数，就像 request.COOKIES 这个属性名字一样。下面说的 Cookie 就是指一组键值对。

有这样一段描述：

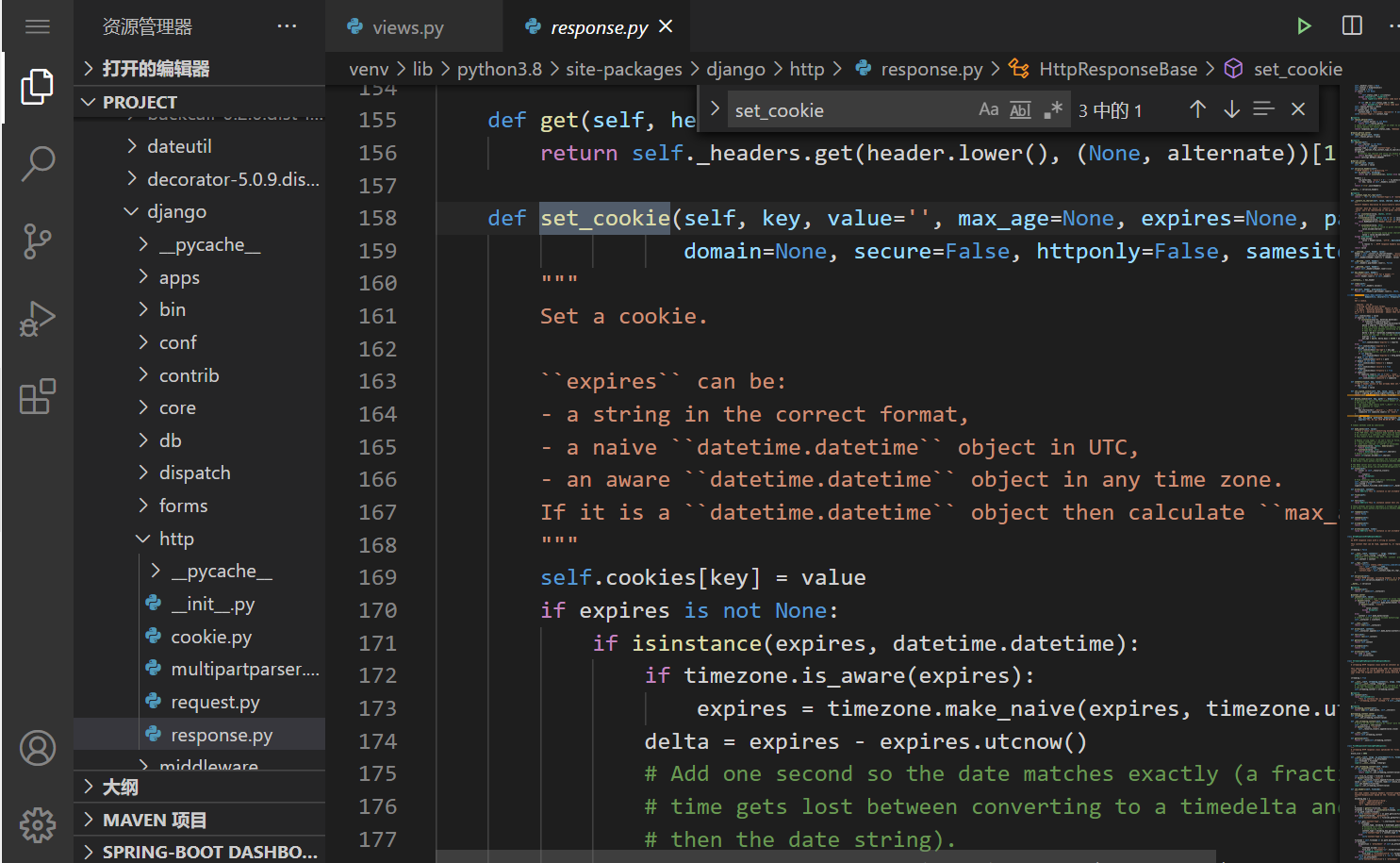
在浏览网站的过程中，经常会遇到需要登录的情况，有些页面只有登录之后才可以访问 ，而且登录之后可以连续访问很多次网站，但是有时候过一段时间就需要重新登录。 还有一些网站，在打开浏览器时就自动登录了，而且很长时间都不会失效，这种情况又是为什么？本部分将了解这些。

这个问题的答案就是 Cookie 是有有效期的。

前面已经了解到 path 决定 Cookie 的有效范围，而 max\_age 就用来决定 Cookie 的有效期。path 和 max\_age 都是 HttpResponse 实例的 set\_cookie 方法的参数。

看下 set\_cookie 的源码，找到如下路径的文件：

venv/lib/python3.8/site-packages/django/http/response.py



其中 set\_cookie 方法在 HttpResponseBase 类下面，而 HttpResponse 是 HttpResponseBase 的子类：

def set\_cookie(self, key, value='', max\_age=None, expires=None, path='/',

domain=None, secure=False, httponly=False, samesite=None):

"""

Set a cookie.

``expires`` can be:

- a string in the correct format,

- a naive ``datetime.datetime`` object in UTC,

- an aware ``datetime.datetime`` object in any time zone.

If it is a ``datetime.datetime`` object then calculate ``max\_age``.

"""

self.cookies[key] = value

if expires is not None:

if isinstance(expires, datetime.datetime):

如上所示，set\_cookie 接收多个参数，其中 path 的默认值就是 '/' ，所以才有前面的操作步骤的结果。其中第三个参数就是 max\_age ，同时它也是位置参数，调用函数时无需写这个参数名，只需将 Cookie 的有效时间写到第三个位置即可。（以上，大概说下即可）

下面修改 learn/view/login 函数：

def login(request, methods=['GET', 'POST']):

if request.method == 'GET':

name = request.COOKIES.get('name')

name = name if name else 'Stranger'

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'shiyanlou' and password=='hello':

resp = HttpResponse('登录成功，{}'.format(name))

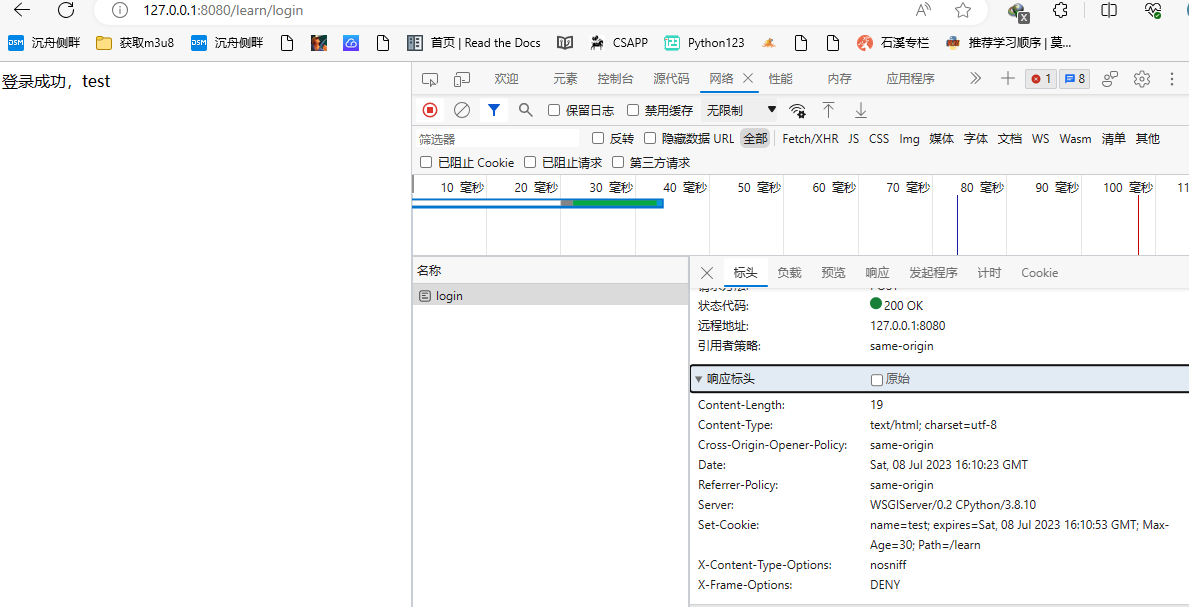
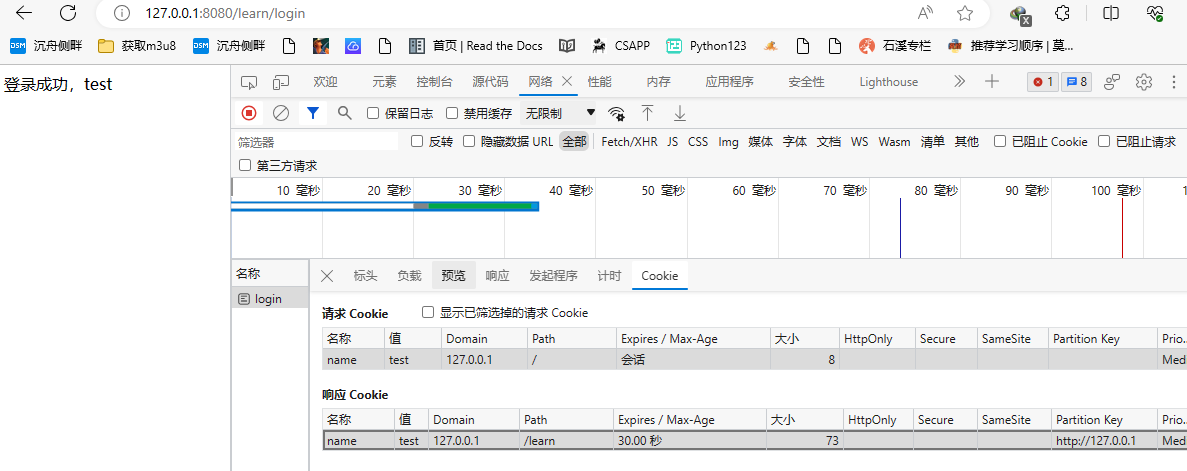
resp.set\_cookie('name', name, 30, path='/learn') # 修改这一行

return resp

return render(request, 'learn/login.html', {'name': name})

其中 set\_cookie 方法的第三个参数 30 即为 Cookie 的有效时间，单位是秒；path='/learn' 表示该 Cookie 只在 learn 应用相关的网址中有效，因为 myweb/urls.py 中的 urlpatterns 列表已经把所有 '/learn' 开头的路径全部转发给 learn/urls.py 来处理了。

现在重新填写登录表单，点击提交：

如上图所示，Cookie 中 name 新增了 expires/Max-Age ，它的值是一个时间，即 name 的失效时间（以秒为单位，有的浏览器，也会显示为失效的具体时间） 。此外可以看到 path 的值变成了 '/learn' 。

Cookie 是属于浏览器的，只要不手动删除，就会一直保存，即使过了有效期也不会被删除。

## session 的创建和使用

Cookie 由服务器创建，保存在浏览器中，每次浏览器发送请求给服务器都会携带 Cookie。所以 Cookie 从安全的角度讲，并不可靠。因此服务器不会将敏感信息作为 Cookie 发送给浏览器保存。

那么如何确定用户的登录状态呢？这就用到了 session 。session 也是以键值对的形式保存的。当谈论 session 时，通常 session 首字母是小写的。

服务器创建 session 并保存到自身的存储空间，不会向外发送。当用户第一次发起登录请求时，使用 POST 方式传递登录信息，包括用户名、密码等数据。服务器通过查询数据库比对数据，确认无误后，创建一条会话保存到自身的存储空间，也就是 session 。然后以加密的方式生成一个类似验证码的字符串，使用 set\_cookie 创建 Cookie 返回给浏览器。浏览器之后的访问都会带上这些数据（类似验证码之类），而不必再提供用户名和密码了。服务器收到后续访问请求，会识别其中的该类数据（类似验证码），确认无误后，返回给浏览器所需数据。

Cookie 是使用 HttpResponse 对象来创建的，而 session 是使用 HttpRequest 对象来创建的。原因是 Cookie 需要返回给浏览器进行存储，而 session 是要保存在服务器中的。所以在本部分前面的步骤中介绍的 HttpRequest 对象的属性中有一个就是 session 。

### 创建 session

在 learn/views.py 文件的 login 函数中，使用参数 request.session 来创建 session ，其实就是一组键值对。request.session 属性的默认值是空字典。使用字典对象的方式去添加键值对即可。

现在修改 login 视图函数如下：

def login(request, methods=['GET', 'POST']):

if request.method == 'GET':

name = request.COOKIES.get('name')

sessionid = request.COOKIES.get('sessionid')

if request.session.exists(sessionid):

resp = HttpResponse('您已处于登录状态，{}'.format(name))

return resp

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'test' and password=='test':

request.session['name'] = name

resp = HttpResponse('登录成功，{}'.format(name))

resp.set\_cookie('name', name, 60 \* 5, path='/learn')

return resp

return render(request, 'learn/login.html', {'name': name})

在介绍 Cookie 的步骤中，提交表单后显示登录成功，但实际上再次访问登录页面，依然需要填写表单，尽管 Cookie 中已经保存了信息。但是这种登录，只涉及 Cookie ，服务器无法识别。

如上所示的 login 视图函数，修改后增加了创建 session 的代码。以浏览器的操作过程来解释代码。

第一次访问 http://127.0.0.1/learn/login 地址，此时浏览器中无 Cookie ，请求方法为 GET 。所以 login 视图函数第 3 、4 行的 name 和 sessionid 变量值均为 None ，第 5 行的 if 语句也会判断失败。

现在以正确的数据填写表单，点击提交。服务器会收到 POST 请求，再次交给 login 视图函数来处理：

def login(request, methods=['GET', 'POST']):

if request.method == 'GET':

name = request.COOKIES.get('name')

sessionid = request.COOKIES.get('sessionid')

if request.session.exists(sessionid):

resp = HttpResponse('您已处于登录状态，{}'.format(name))

return resp

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'test' and password=='test':

request.session['name'] = name

resp = HttpResponse('登录成功，{}'.format(name))

resp.set\_cookie('name', name, 60 \* 5, path='/learn')

return resp

return render(request, 'learn/login.html', {'name': name})

直接跳到第 8 行执行第 2 个 if 语句块。第 9 、10 行获取表单中的数据，第 11 行进行判断，第 12 行添加 session 。此处需要详细讲解，当创建 session 的时候，session 被存到哪里了呢？在 Django 中，该数据被存到数据库的 django\_session 数据表中。

当第一次执行 python manage.py migrate 命令进行数据迁移时，django\_session 数据表就被创建了：

mysql> SHOW TABLES;

+----------------------------+

| Tables\_in\_myweb |

+----------------------------+

| auth\_group |

| auth\_group\_permissions |

| auth\_permission |

| auth\_user |

| auth\_user\_groups |

| auth\_user\_user\_permissions |

| django\_admin\_log |

| django\_content\_type |

| django\_migrations |

| django\_session |

| learn\_author |

| learn\_authordetail |

| learn\_course |

| learn\_tag |

| learn\_tag\_course |

+----------------------------+

15 rows in set (0.01 sec)

继续介绍 login 视图函数中的执行情况，执行到 12 行添加 session 时，django\_session 数据表中会新增一条数据：

mysql> SELECT \* FROM django\_session\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

session\_key: ys4phst4osrekjsidtkyqailqq9x2ack

session\_data: Njc4MjhlYjBlYjMzNjM2YmUyMWQwOGYzNWJmYjI1MDE3MmFjOGIyMTp7Im5hbWUiOiJzaGl5YW5sb3UifQ==

expire\_date: 2023-07-28 14:51:32.663683

1 row in set (0.01 sec)

mysql>

有三列数据，其中的 session\_key 就是用来与 Cookie 配合判断用户的登录状态的数据。第三列 expire\_date 是 session 的失效时间，默认的 session 有效期是两周。

回到 login 视图函数中，12 行执行完毕，13 行创建响应对象，14 行添加 Cookie 。此时 Cookie 中已经有了两个键值对，name 和 sessionid ，而 sessionid 的值就是 django\_session 数据表中 session\_key 字段的值。最后执行 15 行返回响应给浏览器。

### 使用 session

现在使用浏览器向服务器发送第 3 次请求，以 GET 方式发送请求到 http://127.0.0.1:8080/learn/login 请求。

本次请求中会携带上次请求后保存好的 Cookie 数据。服务器收到请求后，继续调用 learn/views.py 的 login 视图函数处理：

def login(request, methods=['GET', 'POST']):

if request.method == 'GET':

name = request.COOKIES.get('name')

sessionid = request.COOKIES.get('sessionid')

if request.session.exists(sessionid):

resp = HttpResponse('您已处于登录状态，{}'.format(name))

return resp

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

if name == 'test' and password=='test':

request.session['name'] = name

resp = HttpResponse('登录成功，{}'.format(name))

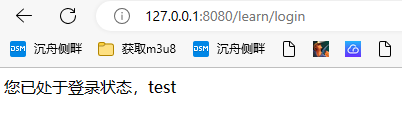
resp.set\_cookie('name', name, 60 \* 5, path='/learn')

return resp

return render(request, 'learn/login.html', {'name': name})

因为是 GET 请求，自然会执行第 2 行的 if 语句块。第 3、4 行从 Cookie 中获取 name 和 sessionid 值。第 5 行调用 request.session 的 exists 方法访问 django\_session 数据表，查询表中 session\_key 的值是否可以匹配 Cookie 中 sessionid 的值。如果匹配成功，就表示该浏览器中的用户已经在服务器中创建了会话且会话未过期。那么本次请求就属于登录成功之后的请求。第 6 行创建响应对象，第 7 行返回响应对象给浏览器。

以上就是一次完整的保持登录状态的操作。利用浏览器的 Cookie 与服务器的 session 配合来实现。前面的步骤中提到的 “相关数据（即：验证码类似物）” 在 Django 中就是 sessionid 这个 Cookie 。



这仅仅是一次演示操作，在实际应用中要比上面描述的步骤复杂得多，而 Django 也已经提供了完整的相关功能。

需要额外说明的是 session 的使用需要提供 SECRET\_KEY 字段的值，通常该值使用操作系统的环境变量来提供，在 Python 代码中是 os.environ.get('SECRET\_KEY') 来获取环境变量的值。在 Django 项目的配置文件中已经自动创建了该字段的值：

myweb/settings.py：

SECRET\_KEY = 'django-insecure-^uk=qf1(^o81t4&9092t\_lw8u#%39vtmf6v!oc%1+d2=(s\_qt-'

## 总结

本部分内容较多，但其实多是涉及HTTP 协议的一些特点，其中 “无连接” 和 “无状态” 促使 BS 架构的项目需要提供 Cookie 和 session 功能以维持用户的登录状态。

Cookie 是由服务器创建的保存在浏览器中的键值对数据，session 是由服务器创建的保存在服务器的存储空间的键值对数据。它们各司其职，相互配合以实现确认用户身份、维持用户登录状态的功能。

#### 完整代码

本部分的参考代码：code09.zip

## 作业：

1. 简述“打开浏览器自动登录“功能的实现原理和步骤。

# 讲授 10 HTTP 错误响应处理

## 介绍

本部分来学习 HTTP 错误响应的处理。错误响应的原因通常有两种，一种是浏览器导致的，一种是服务器导致的。下面分别了解它们的简单处理步骤。

#### 知识点

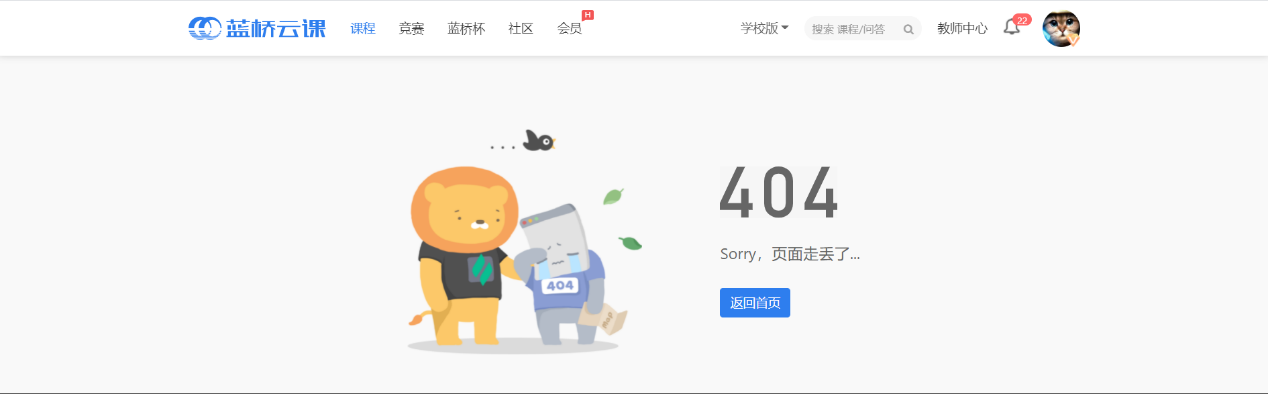
* HTTP 状态码
* 由浏览器导致的响应异常
* 由服务器导致的响应异常

#### 完整代码

前面部分，完成后的参考代码：code09.zip

## HTTP 状态码

在使用浏览器访问一些网站时，偶尔会出现页面找不到的情况：



如上图所示就是其中的一种。

浏览器向服务器发送请求后，服务器进行处理，返回响应。在整个流程中，可能会出现多种异常情况：视图函数代码错误、查询数据库出现连接问题、模板的渲染、中间件的处理异常等待。但不论任何异常，浏览器最后一定会收到响应。

不同的响应，在 HTTP 协议中设置了不同的状态码。正常响应和异常响应都有多种不同的状态码。各家网站可以自定义状态码表示不同的响应情况，大体来讲，状态码分为以下几类：

| **状态码** | **取值范围** | **含义** |
| --- | --- | --- |
| 1xx | 100-101 | 指示信息，表示请求已接收，继续处理 |
| 2xx | 200-206 | 成功，表示请求已被成功接收、理解、接受 |
| 3xx | 300-305 | 重定向，信息不完整需要进一步补充 |
| 4xx | 400-415 | 客户端错误，请求有语法错误或请求无法实现 |
| 5xx | 500-505 | 服务器端错误，服务器未能实现合法的请求 |

HTTP 状态码由三位十进制数字组成，第 1 位决定了状态码的类型，例如 2xx 一定表示请求已经被成功响应。

常见的 HTTP 状态码：

* 200 - 请求成功
* 301 - 资源（网页等）被永久转移到其它 URL
* 404 - 请求的资源（网页等）不存在
* 500 - 内部服务器错误

上面的图片就是 404 状态码的响应被显示到页面上，因为这个地址在服务器上是不提供的，所以它就是由浏览器导致的错误。

## 由浏览器导致的响应异常

打开本机环境，切换到 myweb 项目的主目录，启动 MySQL 服务，启动项目：

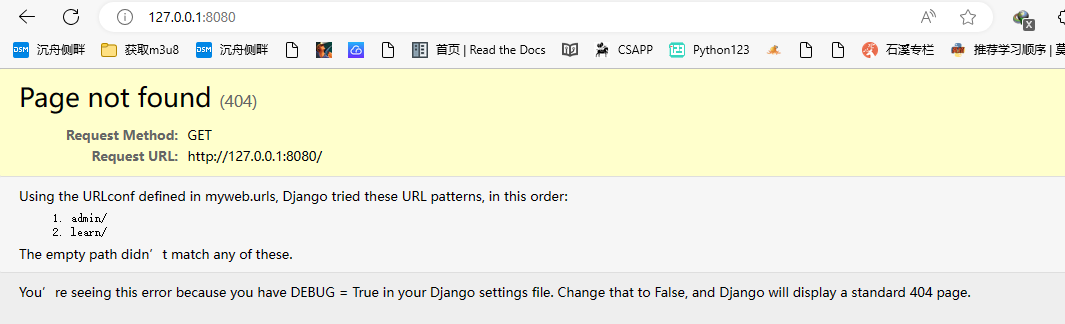
venv/Scripts/activate

net start mysql

cd myweb

python manage.py runserver 0.0.0.0:8080

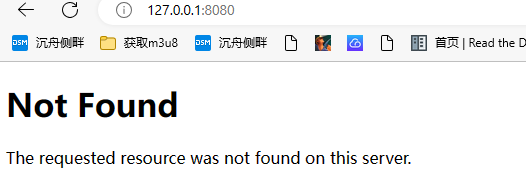
浏览器地址栏中输入： “http://127.0.0.1:8080” 打开页面：



可以看到如上图所示的信息，页面没有找到。因为现在 myweb 项目中只有 learn 应用，且 learn/views.py 中的全部视图函数用于处理 xxxx/learn/ 路径下的路由，没有提供主页根目录对应的视图函数。

所以浏览器访问了一个不存在的路由，因此收到了 404 错误响应。这个页面是由 Django 自动提供的排查信息组成的，最后一行提示：你能看到这个错误是因为你的项目设置了 DEBUG=True 。

修改 myweb/settings.py 中的 DEBUG 字段为 False （在第 26 行），然后刷新页面：



当浏览器访问服务器中任何不存在的 URL 时都会出现这个页面。

### 自定义 404 错误页面

如何自定义错误页面？首先，规范化前端设计：将首页、注册登录页、各种错误页面的视图函数都放到一个应用中。现在新建应用 front ：

python manage.py startapp front

然后将新建的应用添加到 myweb/settings.py 的 INSTALLED\_APPS 列表中：

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'learn',

'front',

]

修改 front/views.py 文件，添加处理 404 错误响应的视图函数 page\_not\_found ：

from django.shortcuts import render

def page\_not\_found(request, exception):

'''404 错误处理函数

'''

return render(request, '404.html')

注意，这里 page\_not\_found 函数有两个必填参数：request 和 exception ，其中 exception 是用来接收错误信息的，该参数为字典对象，暂不需要处理它。

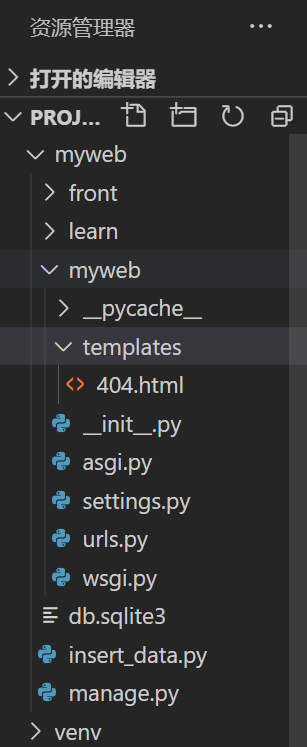
创建存储 '404.html' 文件的目录，将该目录创建在 myweb 目录下：

# myweb 目录下，执行：

mkdir myweb\templates

# 在 templates 文件夹下创建 404.html

此时 myweb 目录结构如下：



将以下代码写入 myweb/templates/404.html 文件中：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>404-无法找到文件</title>

</head>

<body>

<h1>404 错误，该页面不存在。</h1>

</body>

</html>

视图函数 page\_not\_found 是创建在 front/views.py 文件中的，函数执行时，如何找到 myweb/templates 目录下的 404.html 模板文件呢？

这需要在 myweb/settings.py 文件中进行设置，TEMPLATES 属性为字典对象，其中的 DIRS 键值对的值是列表，将 myweb/templates 以字符串的形式添加到列表中：

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': ['myweb/templates'], # 这一行被修改

'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',

],

},

},

]

如上所示第 4 行即为修改结果。

最后需要在 myweb/urls.py 中添加一项设置，以使得浏览器的请求出现 404 异常时能够交给 front/views.py 中的 page\_not\_found 函数来处理：

from django.contrib import admin

from django.urls import path, include

from front import views # 新增代码

urlpatterns = [

path('admin/', admin.site.urls),

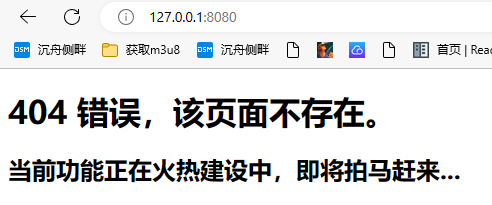
path('learn/', include('learn.urls'))

]

handler404 = views.page\_not\_found # 新增代码

第 3 行引入 views 模块，最后一行创建 handler404 变量，将 page\_not\_found 函数赋值给此变量。该变量名是固定的，Django 会对它进行处理。

至此就完成了全部代码修改，现在打开浏览器首页：



如上图所示，标签页的标题与页面信息均为 404.html 文件中设置的内容。

## 由服务器导致的响应异常

最常见的由服务器导致的响应异常就是浏览器发送的请求地址是合法的，但视图函数在处理过程中出现了错误。这种情况下浏览器收到的响应就是 500 错误。

在 learn/views.py 文件中提供了处理 http://localhost:8080/learn/course/id 地址的视图函数 course ：

def course(request, id, method=['GET']):

course = Course.objects.get(id=id)

s = 'ID: {}</br>Name: {}</br>发布时间：{}</br>学生人数：{}'.format(course.id, course.name, course.pub\_date, course.stu\_number)

return HttpResponse(s)

访问 http://127.0.0.1:8080/learn/course/1 时，可以展示课程 id 为 1 的课程信息。

仔细观察代码，第 2 行查询数据库获取 Course 类实例的代码可能会出现异常。在前面的讲授中，学习到 Course.objects 的 get 方法只能获取一条数据，如果出现查询结果为多个，则抛出 MultipleObjectsReturned 异常；如果数据为零，则抛出 DoesNotExist 异常。出现这种异常，会导致浏览器收到状态码为 500 的异常响应。

尝试在浏览器上访问 http://localhost:8080/learn/course/11 ：



出现了服务器异常，状态码为 500 。

### 自定义 500 错误页面

接下来针对这种状态码为 500 的错误，创建对应的视图函数。这与自定义 404 错误页面的操作是一样的。

首先，打开 front/views.py 文件，新建 server\_error 视图函数：

def server\_error(request):

'''500 服务器内部异常处理函数

'''

return render(request, '500.html')

注意此函数只提供一个 request 参数。

创建 myweb/templates/500.html 模板文件，将以下代码写入其中：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>500-服务器错误</title>

</head>

<body>

<h1>由服务器内部异常导致 500 响应</h1>

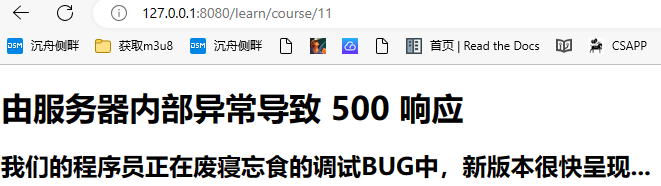
</body>

</html>

在 myweb/urls.py 文件中新增一行代码，以使得服务器出现内部错误时将请求交由 front/views.py 中的 server\_error 视图函数来处理：

handler500 = views.server\_error

代码已经编写完毕，现在使用浏览器访问 http://127.0.0.1:8080/learn/course/11 ：



页面显示符合预期。

为了便于快速演示，对于错误页面设置得十分简陋，大家有兴趣可以美化一下（本来前端就是我的软肋嘛…）。

## 总结

本部分主要讲授了 HTTP 状态码的种类、作用和含义。

了解到在 Django 项目中如何自定义由浏览器导致的 404 响应异常，以及自定义由服务器导致的 500 响应异常。

在实际项目中，还有其它多种异常需要定义，大家可以根据本部分了解的知识举一反三。

#### 完整代码

本部分的参考代码：code10.zip

## 作业：

1. 自行总结归纳创建自定义404/500等状态码响应页面的实现步骤。

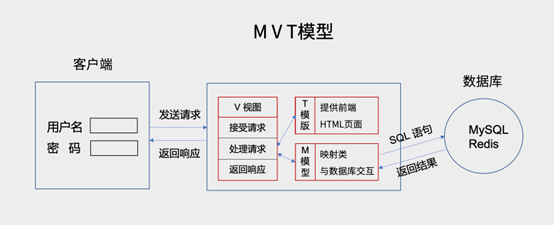
# 讲授 11 模板引擎

## 介绍

（开始进入我的软肋了，那就是前端呗…）

这部分开始学习 Django 框架 MVT 模型中的 T 模板，在前面的内容中，已经稍稍接触了一些模板相关的内容，创建了一些模板文件，其实就是 html 后缀的文件。

再次展示一下 MVT 模型示意图：



接触过前端模板的同学一定知道 Jinja2 模板（在 Flask 框架中也有模板的概念），它是功能齐全、性能优异的模板引擎。Django 自带的模板引擎在用法上与 Jinja2 十分相似。本部分来学习 Django 框架中的模板引擎。

#### 知识点

* 模板概述
* 模板变量
* 模板标签
* 模板过滤器
* 模板注释

#### 完整代码

之前内容完成后的参考代码：code10.zip

## 概述

在前面的讲授中，对于极简单的页面，通常使用 HttpResponse 返回一个字符串作为页面内容。例如 learn/views.py 文件中的 courses 和 course 视图函数。在实际应用中，页面通常都是内容丰富、画面精美的。这需要使用 HTML 标记语言来配合实现。完整的网页通常由 HTML 、CSS 和 JavaScript 三部分构成静态部分（我的天，这就是我的软肋…），由模板引擎提供动态数据。

在 Django 和大多数 Web 开发框架中，大体流程是由视图函数处理请求后返回数据，HTML 模板文件在内部加载 CSS 和 JavaScript 数据，由模板引擎加载视图函数提供的动态数据，然后一并返回给视图函数。

而在 HTML 的标签代码中，需要对一些动态数据进行额外的处理，其中主要是数据库查询得到的数据。处理动态数据，就需要模板引擎提供支持。

模板的设计实现了业务逻辑 View 与显示内容 Template 的分离，一个视图函数可以使用任意一个模板，一个模板可以供多个视图函数调用。

模板语言包括四大类型：变量、标签、过滤器和注释。

## 模板变量

模板变量用于调用视图函数传递过来的字典对象。下面举例说明。

在 myweb 项目中，learn/views.py 的 course 视图函数由 HttpResponse 提供返回的响应对象，参数为课程信息的字符串。

修改该视图函数如下：

from datetime import datetime

def course(request, id, method=['GET']):

course = Course.objects.get(id=id)

date = datetime.now()

context = {'course': course, 'date': date}

return render(request, 'learn/course.html', context)

1、首先引入 datetime 模块用户创建时间对象。

2、在 course 函数内部创建两个变量 course 和 date ，分别对应查询数据库得到的 Course 类的实例和当前北京时间。

3、将 course 和 date 对象作为 value 生成字典对象 context 。

4、调用 render 方法创建返回值，第 1 个参为 request ，第 2 个参数为 HTML 模板文件的相对路径，第三个参数为字典对象传递给模板文件。

现在创建模板文件 learn/templates/learn/course.html ，将以下代码写入其中：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>{{ course.name }} 详情页</title>

</head>

<body>

<h1>课程信息</h1>

<p>课程 ID ：{{ course.id }}</p>

<p>课程名称：{{ course.name }}</p>

<p>发布日期：{{ course.pub\_date }}</p>

<p>学生人数：{{ course.stu\_number }}</p>

<p>{{ date }}</p>

</body>

</html>

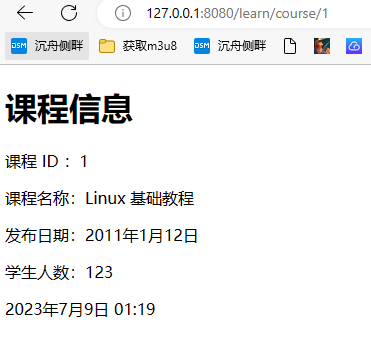
在 HTML 模板文件内部即可使用视图函数的第三个参数传过来的数据。字典 context 中键值对的所有 key 值可以直接使用，但需要用双大括号括起来 {{ var }} 。

根据 key 对应的 value 的数据类型不同，有如下几种获取数据的方式：

* 数字、字符串、日期类型：{{ 变量名 }}
* 列表类型：{{ 变量名.下标 }}
* 字典类型：{{ 变量名.键 }}
* 对象类型：{{ 变量名.属性 }} {{ 变量名.方法() }}

在 course.html 文件中，date 属于日期类型变量，course 属性对象类型变量。所以第 5 、9 ~ 11 行中使用了 {{ course.xxx }} 格式获取 course 的属性值，第 12 行直接调用 date 变量获取时间数据。

启动 myweb 项目，在浏览器上访问 http://127.0.0.1:8080/learn/course/1 ，结果如下：



## 模板标签

模板标签这个称谓不是太准确，标签只是对 HTML 标记语言中的代码的称呼。

最常用的模板标签是 for 循环语句和 if 判断语句，语法结构与 Python 语法很相似。

* for 循环语句：

{% for item in 可迭代对象 %} 循环逻辑 {{ forloop.counter }}

表示当前是第几次循环，从 1 开始 {% empty %} 列表为空或不存在时执行此逻辑 {%

endfor %}

* if 判断语句：

{% if 条件 A %} 逻辑 A {% elif 条件 B %} 逻辑 B {% else %} 逻辑 C {% endif %}

* 比较运算符：==、!=、<、>、<=、>= ，运算符左右两边须有空格。
* 逻辑运算符：and、or、not

现举例说明。修改 learn/views.py 文件的 courses 视图函数如下：

def courses(request):

courses = Course.objects.all()

return render(request, 'learn/courses.html', {'courses': courses})

将全部课程的查询对象 courses 传递给 learn/courses.html 模板文件。

创建 learn/templates/learn/courses.html 文件并将如下代码写入其中：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>全部课程</title>

</head>

<body>

<h1>course List</h1>

<table>

<tr>

<td>书名</td>

<td>作者</td>

<td>出版时间</td>

<td>学生人数</td>

</tr>

{% for course in courses %}

<tr>

<td>{{ course.name }}</td>

<td>{{ course.author.name }}</td>

<td>{{ course.pub\_date }}</td>

<td>{{ course.stu\_number }}</td>

</tr>

{% endfor %}

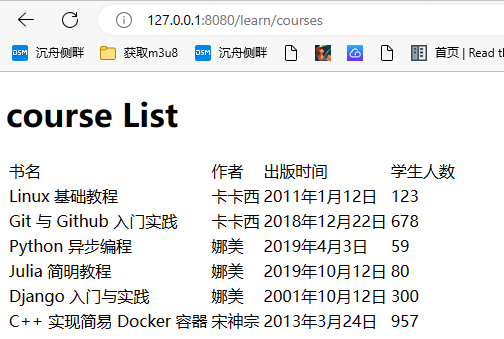
</table>

</body>

</html>

注意模板标签是成对出现的，有 {% for ... %} 必有 {% endfor %} 。倒数第 8 行 course.author 属性值为 Author 映射类的实例，course.author.name 为实例的 name 属性值，即作者的名字。

打开浏览器，访问 http://127.0.0.1:8080/learn/courses ：



## 模板过滤器

通常会使用模板过滤器对传入模板文件的数据进行进一步处理，包括计算、格式转换等操作。

过滤器的语法格式为 {{ 变量 | 过滤器:参数 }} 。

常用过滤器有很多：

* length，返回字符串包含字符的个数，或列表、元组、字典的元素个数。
* default，如果变量不存在时则返回默认值。
* date，用于对日期类型的值进行字符串格式化。

用 learn/views.py 文件中的视图函数及其模板文件演示其中的几个。

#### add 增加数值的过滤器

修改 learn/course.html 文件如下：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>{{ course.name }} 详情页</title>

</head>

<body>

<h1>课程信息</h1>

<!-- 修改下面这一行 -->

<p>课程 ID ：{{ course.id | add:'2' }}</p>

<p>课程名称：{{ course.name }}</p>

<p>发布日期：{{ course.pub\_date }}</p>

<p>学生人数：{{ course.stu\_number }}</p>

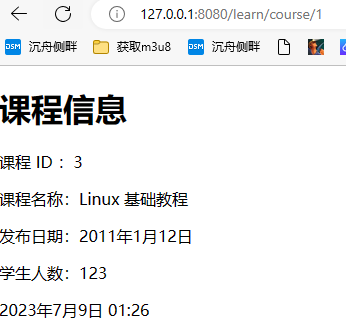
<p>{{ date }}</p>

</body>

</html>

该过滤器将冒号后面的参数转换成 int 类型并加到 course.id 这个变量的值上。

打开浏览器，访问 http://127.0.0.1:8080/learn/course/1 ：



课程 ID 由原值 1 变成了 3 。

需要注意的是过滤器的 : 冒号左右两边不允许有空格，| 左右两边允许有空格。

#### cut 去掉指定字符

现有如下要求：如果课程名字最后两个字是 “教程” ，那么去掉这两个字。

修改 learn/templates/learn/course.html 文件如下：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>{{ course.name }} 详情页</title>

</head>

<body>

<h1>课程信息</h1>

<p>课程 ID ：{{ course.id | add:'2' }}</p>

<!-- 修改下面这一行 -->

<p>课程名称：{{ course.name | cut:'教程' }}</p>

<p>发布日期：{{ course.pub\_date }}</p>

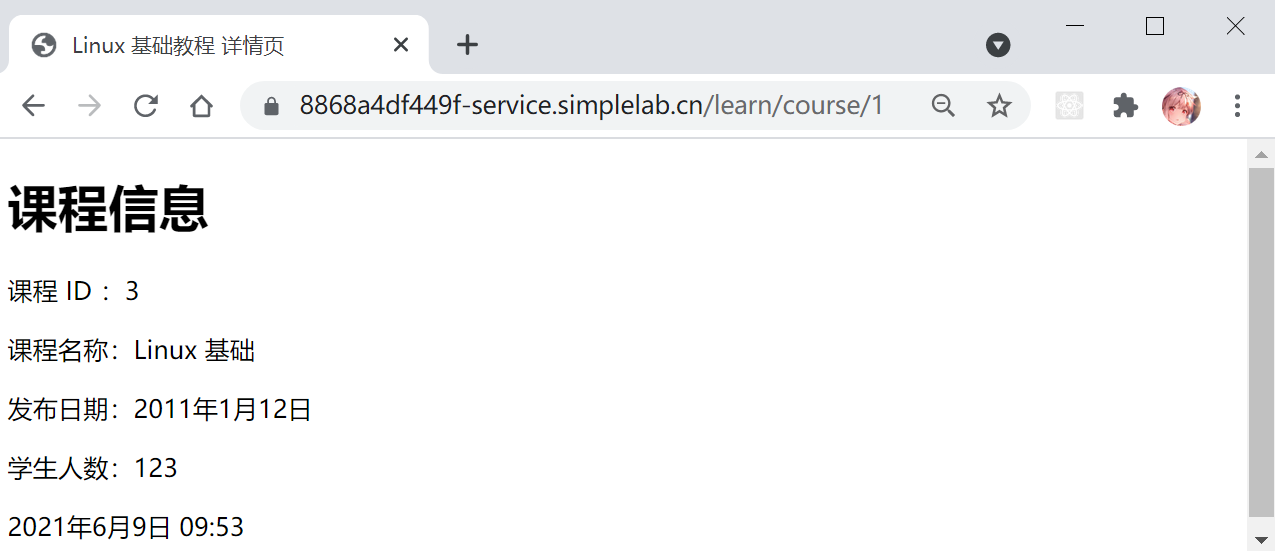
<p>学生人数：{{ course.stu\_number }}</p>

<p>{{ date }}</p>

</body>

</html>

在浏览器上刷新页面：



#### date 日期时间过滤器

该过滤器极其繁杂，这里仅举例说明它的用法。

修改 learn/templates/learn/course.html 文件如下：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>{{ course.name }} 详情页</title>

</head>

<body>

<h1>课程信息</h1>

<p>课程 ID ：{{ course.id | add:'2' }}</p>

<p>课程名称：{{ course.name | cut:'教程' }}</p>

<p>发布日期：{{ course.pub\_date }}</p>

<p>学生人数：{{ course.stu\_number }}</p>

<!-- 修改下面这一行 -->

<p>{{ date | date:'Y年 F d日 H:i' }}</p>

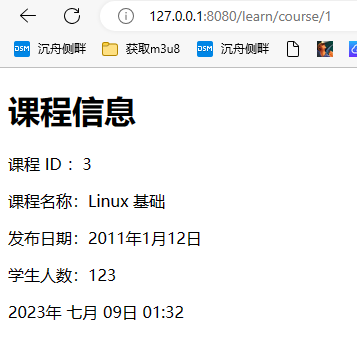
</body>

</html>

倒数第 3 行 date: 后面的字母分别表示：

* Y 四位数年份
* F 文字月份（如果是英文，就是 January 、February ...）
* d 两位数日期（如果是 2 号就是 02 ）
* H:i 24 小时制的时分，中间用冒号连接

浏览器刷新页面：



## 模板注释

注释是比较简单的，HTML 的注释是 <!-- --> 这样的格式，而 HTML 的注释对模板标签无效，模板有自己的注释方式：{# #} 。

简单例子，在 learn/templates/learn/courses.html 文件中添加注释：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>全部课程</title>

</head>

<body>

<h1>course List</h1>

<table>

<tr>

<td>书名</td>

<td>作者</td>

<td>出版时间</td>

<td>学生人数</td>

</tr>

{% for course in courses %} {# 如果 course

的学生数量低于100，则不显示它的信息 #} {% if course.stu\_number >= 100 %}

<tr>

<td>{{ course.name }}</td>

<td>{{ course.author.name }}</td>

<td>{{ course.pub\_date }}</td>

<td>{{ course.stu\_number }}</td>

</tr>

{% endif %} {% endfor %}

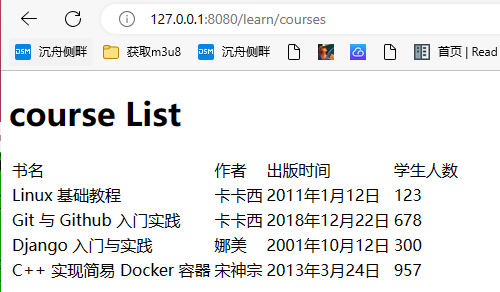
</table>

</body>

</html>

修改 for 循环，增加 if 判断，目的如第 17 行注释所述。

打开浏览器，访问 http://127.0.0.1:8080/learn/courses ：



## 总结

本部分内容较为简单（其实也是因为我的软肋的原因~~），主要学习了模板的概念，以及模板的四大类型：变量、标签、过滤器和注释。其中模板过滤器是比较复杂的，过滤器的种类繁多，无法依次举例。

大家在实际项目中根据需求自行搜索相应的过滤器，在一些时候，过滤器是特别有效的（其实，前端的业务逻辑完全可以在后端的代码中实现，然后将满足业务逻辑的数据作为返回数据封装在返回的响应中，这样其实更实用）。

#### 完整代码

本部分的参考代码：code11.zip

## 作业：

1. 请自行美化当前项目的前端页面。

# 挑战作业 3：查询作者的全部课程信息

# 查询作者的全部课程信息

## 挑战介绍

本次挑战作业要求实现一个页面，该页面能够查询课程信息并展示。

如有当前项目，以下步骤可以略过。

前面部分实现后的项目代码：code10.zip

创建虚拟环境，安装所需模块：

virtualenv venv

venv/Scripts/activate

pip install django

pip install mysqlclient

切换到 myweb 项目主目录，启动 MySQL 服务，创建数据库：

cd myweb

net start mysql

mysql -uroot -p

CREATE SCHEMA myweb CHARSET = UTF8

数据迁移（创建数据表）：

python manage.py migrate

进入 MySQL 客户端，添加数据：

mysql -uroot myweb

INSERT INTO `learn\_author` VALUES (1,'卡卡西',1),(2,'娜美',0),(3,'宋神宗',NULL);

INSERT INTO `learn\_course` VALUES

(1,'Linux 基础教程','2011-01-12',123,1),

(2,'Git 与 Github 入门实践','2018-12-22',678,1),

(3,'Python 异步编程','2019-04-03',59,2),

(4,'Julia 简明教程','2019-10-12',80,2),

(5,'Django 入门与实践','1983-01-05',728,2),

(6,'C++ 实现简易 Docker 容器','1974-12-02',158,3),

(7,'MySQL 基础课程','2001-02-22',333,3);

#### 挑战要求

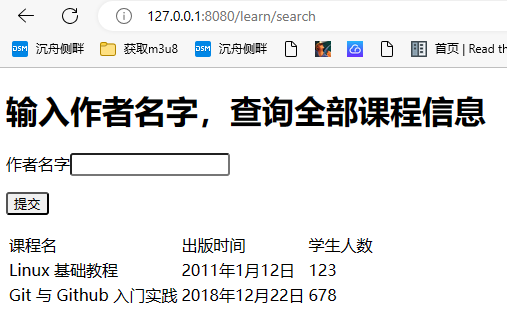
1、在 learn/views.py 文件中增加 search 视图函数，用于处理 http://127.0.0.1:8080/learn/search 路由。

2、search 视图函数使用 render 方法返回响应数据，需要调用 learn/templates/learn/search.html 模板文件。

3、实现页面后，输入作者名字点击提交，在同一页面显示相关课程的名字、出版时间和学生人数。

4、适当考虑页面的美观。

#### 实现结果



#### 知识点

* 模板变量
* 模板标签

## 提示信息

1、在项目已有框架的基础上添加代码即可。

2、前端代码可以参考 login.html 和 courses.html 文件。

3、记得在 learn/urls.py 中补充路由与视图函数的映射。

## 参考答案

参考步骤：

1. 修改 learn/views.py

2. 新建 learn/templates/learn/search.html

3. 修改 learn/urls.py

可以参考代码：code35.zip

# 讲授 12 添加静态资源

## 介绍

本部分主要讲授模板文件中所需要的静态资源如何保存在项目的目录结构中。静态文件是指网站中的 js 、css 、图片、视频等文件。

#### 知识点

* 什么是静态资源
* 配置静态文件路径
* 添加示例静态文件
* 模板文件代码介绍

## 配置静态文件路径

模板文件被分为两类：一类是 HTML 文件，它们构成一个页面的骨架；另一类是渲染美化页面的 js 和 css 文件，它们被称为静态文件。静态文件还包括图片和视频等资源，简单理解就是放那里不动，等着被调用的文件。HTML 文件通常被放在 templates 目录下，静态文件被统一放在 static 目录下，该目录可以创建一些子目录以分别存放不同类型的静态文件，例如 static/css 存放 CSS 文件，static/img 存储图片文件。

在 Django 项目中，静态文件默认由各个应用自行管理和配置。但实际上，肯定是会有一些共用的资源，这些资源应该被放到公共区域，可以方便地被任何应用调用。

这个公共区域选在哪呢？

可以选择在项目的主目录 myweb 下，与各个应用的主目录同级。

也可以选择放到 myweb/myweb 目录下，该目录下的文件用于统管全局，是项目的核心地段。

这两种方式都可以。

一般推荐：把 templates 和 static 放到后一个选择上。这样的好处就是主目录下只有应用目录和项目核心目录，还有一个 manage.py 文件，显得项目结构更加整齐。

当提到项目主目录下的某个路径时，指的就是 myweb 目录下的内容。

在前面的讲授中，已经在 myweb 这个目录下创建了 templates 子目录，而且在其中写了两个处理错误状态码的 HTML 模板文件：

cd myweb/myweb

tree /f templates

templates

├── 404.html

└── 500.html

这个 templates 目录就是模板的公共目录。

在myweb目录下再创建 static 目录及其子目录：

mkdir static

cd static

mkdir js

mkdir css

mkdir images

# 转回到 myweb\myweb 目录

tree /f static

static

├── css

├── images

└── js

接下来在配置文件中添加配置项，在 myweb/settings.py 文件中增加如下内容：

# 项目创建时默认生成的配置项，这是浏览器直接打开静态文件用的前缀目录

# 此种设置，默认会在web项目中含有当前项目（如：front、learn）中查找静态资源

STATIC\_URL = 'static/'

# 如果是下面的设置，所有的静态资源均会到 项目（myweb）目录下的静态资源文件夹中查找

#STATIC\_URL = '/static/'

# 静态文件放在对应的 app 下的 static 目录中或者 STATICFILES\_DIRS 列表里的目录中

# 当 DEBUG = True 时，Django 就能自动找到放在里面的静态文件

# 如果不想用 STATICFILES\_DIRS 可以不用，都放在 app 的 static 目录下也可以

STATICFILES\_DIRS = ['myweb/static']

STATICFILES\_DIRS 列表中的字符串就是全局静态文件的存储主目录。配置完毕，当模板文件中有如下代码时：

<link href="{% static 'css/xxx.css' %}" rel="stylesheet" />

Django 自己就会到 myweb/static 目录下的 css 子目录中寻找 xxx.css 并将其加载到内存中。

## 添加静态文件

接下来在已有的项目基础上增加一些静态文件，并简单地显示网站主页。这些静态文件涉及复杂的 JavaScript 和 CSS 语法，（反正我了解的不够好）不需要了解文件中的代码，只需知道它们是用来渲染页面的就可以了。

首先，切换到 myweb 项目主目录，删除 myweb/static 目录：

rd /s /q myweb\static

解压 static.zip：static.zip

将解压得到的 static 静态文件目录全部移动到 myweb 目录下，删除压缩包：

move static myweb\myweb

cd myweb

tree /f myweb\static

界面截图：

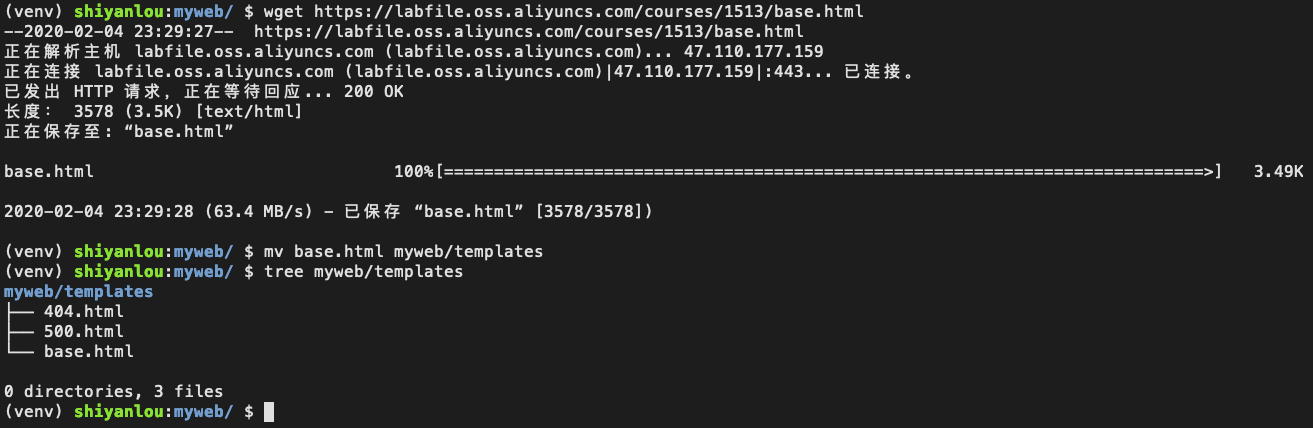


复制模板 base.html 文件到 myweb/templates 目录下：

copy base.html myweb\templates

tree /f myweb\templates

界面截图：



静态文件准备完毕，下面编写代码。

## 添加主页视图函数

在 front/views.py 文件中添加主页视图函数，代码如下：

def home(request):

return render(request, 'front/home.html')

修改 myweb/urls.py 文件如下，注册路由：

from django.contrib import admin

from django.urls import path, include

from front import views

urlpatterns = [

path('admin/', admin.site.urls),

path('learn/', include('learn.urls')),

path('', views.home) # 添加这一行代码，其余不变

]

handler404 = views.page\_not\_found

handler500 = views.server\_error

在 front 应用中创建 templates/front 目录，并创建 home.html 文件：

mkdir front/templates/front/

touch front/templates/front/home.html

将如下代码写入 front/templates/front/home.html 文件：

{% extends 'base.html' %}

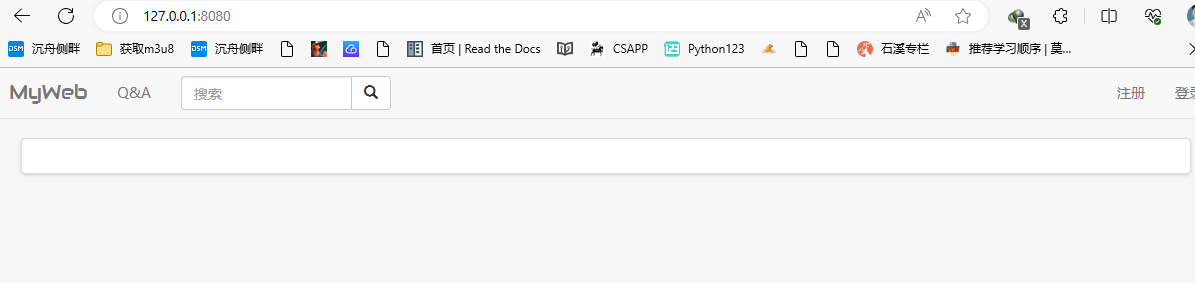
就是引入 myweb/templates/base.html 文件，没有其它代码。

## 启动项目

完成以上准备工作，在终端执行如下命令启动项目：

python manage.py runserver 0:8080

打开浏览器，输入127.0.0.1:8080：



如上图所示，主页已经被正确打开，同时加载了 css 和 js 文件渲染了页面。

## 模板文件代码说明

打开 myweb/templates/base.html 文件，接下来对文件中的一些代码进行解释（尽我所能…）。

#### 加载静态文件

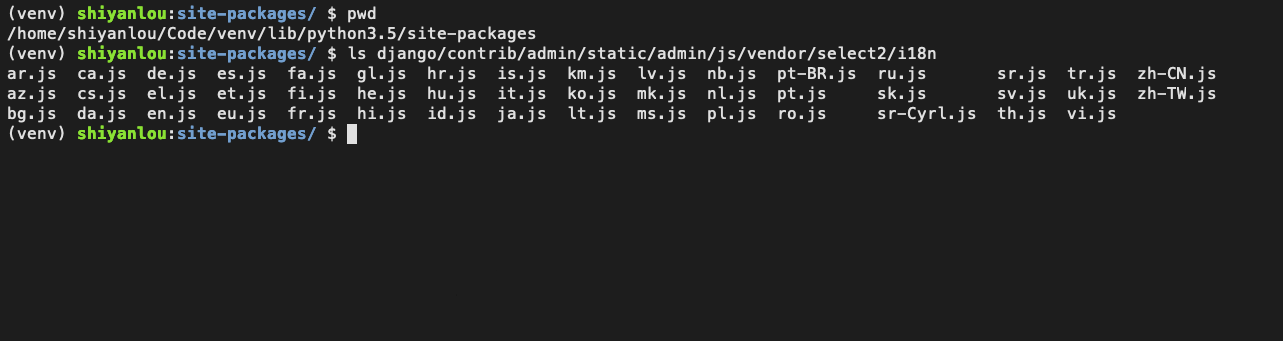
{# 引入模板标签库或过滤器库 #} {% load i18n %} {% load static %}

第 1 行为注释，这在上一部分中已有介绍。

第 2 、3 行为加载模板标签库，如注释所述。那么 i18n 是什么？

当 Web 项目搭建好以后，可以接收来自全球不同国家的用户的访问。这样就要求开发人员调整软件，使之能适用于不同的语言，即国际化和本地化。国际化的英文单词为 Internationalization，i 和 n 之间有 18 个字母，简称 I18N 。本地化的英文单词为 Localization， l 和 n 之间有 10 个字母，简称 L10N。国际化意味着 Web 产品有适用于任何地方的潜力，针对程序开发人员；本地化则是指使一个国际化的程序为了在某个特定地区使用而进行实际翻译的过程，针对翻译人员而言。

此处加载 i18n 指的是加载如下所示的由 Django 框架提供的静态文件目录：



这些 js 文件分别用于处理各个国家的语言转换。将整个 i18n 目录中的所有文件加载至内存中，随时用于处理字符串的翻译任务。可以看出最右侧有两个 zh 开头的 js 文件，分别用于处理简体中文和台湾繁体中文。

第 3 行的 static 又是什么呢？还记得本部分前面的步骤 “配置静态文件路径” 中修改 myweb/settings.py文件的操作吗？

# 项目创建时默认生成的配置项，这是浏览器直接打开静态文件用的前缀目录

STATIC\_URL = 'static/'

# 静态文件放在对应的 app 下的 static 目录中或者 STATICFILES\_DIRS 列表里的目录中

# 当 DEBUG = True 时，Django 就能自动找到放在里面的静态文件

# 如果不想用 STATICFILES\_DIRS 可以不用，都放在 app 的 static 目录下也可以

STATICFILES\_DIRS = ['myweb/static']

第 3 行的作用就是将 STATICFILES\_DIRS 列表中的目录字符串指向的静态文件目录中所有的静态文件都加载到内存中备用。

特别说明：

{#{% load staticfiles %}#}  
{#https://docs.djangoproject.com/en/dev/internals/deprecation/#deprecation-removed-in-3-0#}  
{#修改前端模板为{% load static %}#}  
{#因为在django3.x以后，对于：#}  
{#{% load staticfiles %}#}  
{#{% load static from staticfiles %}#}  
{#{% load admin\_static %}#}  
{#都统一改为{% load static %}#}  
{% load static %}

### 设置语言

看一下 base.html 文件的第 4、5 行：

<!DOCTYPE html>

<html lang="{% get\_current\_language as LANGUAGE\_CODE %}"></html>

第 4 行的作用是声明这是 HTML5 文件。

第 5 行的作用是设置模板语言，其中的 LANGUAGE\_CODE 是在 myweb/settings.py 中配置过的：

# Internationalization

# https://docs.djangoproject.com/en/2.2/topics/i18n/

LANGUAGE\_CODE = 'zh-hans'

TIME\_ZONE = 'Asia/Shanghai'

在加载了 i18n 之后生效。

### 头部标签

继续查看 base.html 文件中的代码：

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

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

<link rel="icon" type="image/png" href="{% static 'img/favicon.png' %}" />

<link

href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css"

rel="stylesheet"

/>

<link href="{% static 'css/bootcamp.css' %}" rel="stylesheet" />

<script src="https://cdn.bootcss.com/jquery/2.2.4/jquery.min.js"></script>

<script src="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

{% block head %} {% endblock head %}

</head>

如上所示，这是文件中的 head 标签的全部代码。<head> 标签用于定义文档的头部，它是所有头部元素的容器。<head> 中的元素可以引用脚本、指示浏览器在哪里找到样式表、提供元信息等等。

* 第 1 个 meta 标签定义了文件的编码格式。
* 第 5 行定义了浏览器标签的标题，其中的 block title 是模板引擎的 “块语句” ，这里定义的 block 块，可以在继承该文件的其它模板文件中使用。
* 第 6 行定义浏览器标签的图标，也就是标签左边的小图标。
* 第 7 、9 、10 行下载网上的静态资源。
* 第 8 行使用本地的静态资源，其中 static 标签关键字之所以生效，是因为前面已经加载了指定目录中的静态文件，所以这里的 'css/bootcamp.css' 指的就是 myweb/static/css/bootcamp.css 文件。
* 倒数第 2 、3 行定义头部 block 块备用。

文档的头部描述了文档的各种属性和信息，包括文档的标题、在 Web 中的位置以及和其它文档的关系等。绝大多数文档头部包含的数据都不会真正作为内容显示给读者。

### 主体标签

<body> 标签定义文档的主体。其中包含文档的所有内容，比如文本、超链接、图像、表格和列表等等。

简单介绍这一段代码：

<ul class="nav navbar-nav navbar-right">

{% if not user.is\_anonymous %}

<li><a href="#">{{ user.get\_username }}</a></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown">

<b class="caret"></b>

</a>

<ul class="dropdown-menu">

<li><a href="#">{% trans 'profile' %}</a></li>

<li class="divider"></li>

<li><a href="">{% trans 'Log out' %}</a></li>

</ul>

</li>

{% else %}

<li><a href="">注册</a></li>

<li><a href="">登录</a></li>

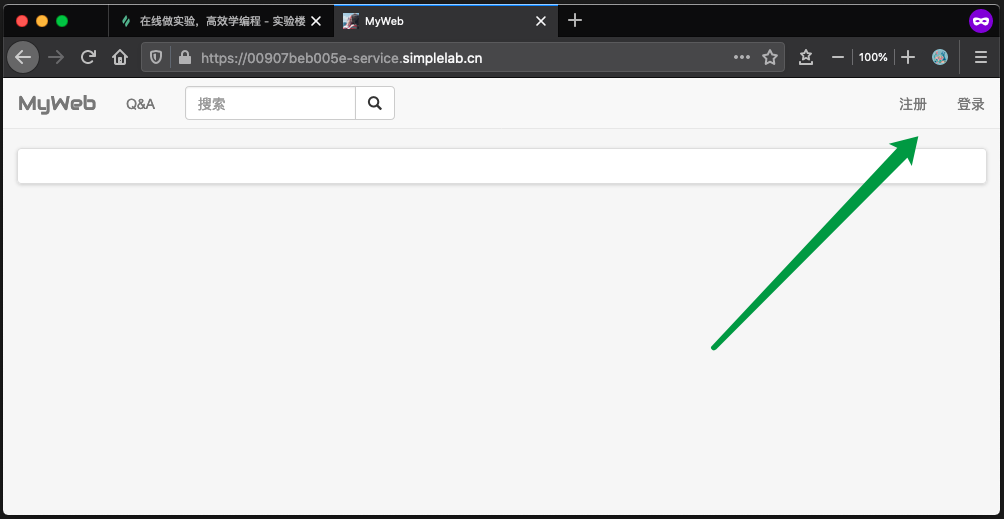
{% endif %}

</ul>

这是 body 标签中的 ul 标签，用于定义无序列表。

第 2 行，当用户登录后，user.is\_anonymous 值为 False ，显示后面这些 li 标签。其中包括用户名、个人详情、登出，后两个是 <a> 标签定义的链接，点击后跳转到用户详情页和退出登录。

如果用户未登录，user.is\_anonymous 值为 True ，也就是匿名用户，显示后面的 li 标签，也就是当前页面的状态，在右侧显示 “注册” 和 “登录” 按钮：



关于 base.html 文件就介绍这些，大家有兴趣可以根据当前已有的静态文件去创建一些其它视图函数。

## 总结

本部分内容较为简单（也是因为是我软肋的缘故…），主要介绍了如何配置静态文件路径。后半部分使用了一些示例静态文件并使用它们渲染了主页。最后讲解了模板文件中的一些标签的工作原理。

关于 Django 的静态文件处理，还有很多复杂的操作。希望大家能够根据本部分的基础讲解，继续深入学习高阶用法（建议适当尝试前端使用Vue框架，后端向前前端返回json字符串，并在前端获取后渲染展示）。

本部分的参考代码：code12.zip

## 作业：

请自行扩展前端界面并美化，尝试使用最新的主流前端框架技术作为自己Web项目的前端模块。

# 讲授 13 后台管理

## 介绍

Django 自身带有一个功能强大的后台管理系统，这是 Django 与其它 Web 框架相比很大的一个优势！

使用一些 admin 自带的参数，可以定制出一套功能丰富的后台管理系统。

当创建了一个 Web 项目，对于用户和其它数据的管理不会直接在数据库的客户端中操作，而是需要对用户分配相应的权限，登录用户在 Web 项目的后台页面操作。例如网站的外部作者，他们的账号拥有部分后台管理的权限，包括创建、删除、添加课程等操作。

本部分讲授 Django 框架的后台管理功能。

#### 知识点

* 创建管理员账号
* 后台管理
* 编写模板类对应的管理类

## 创建管理员账号

Django 提供了基于 web 的管理工具。Django 自动管理工具是 django.contrib 的一部分。可以在项目的 myweb/settings.py 中的 INSTALLED\_APPS 看到它：

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'learn',

'front',

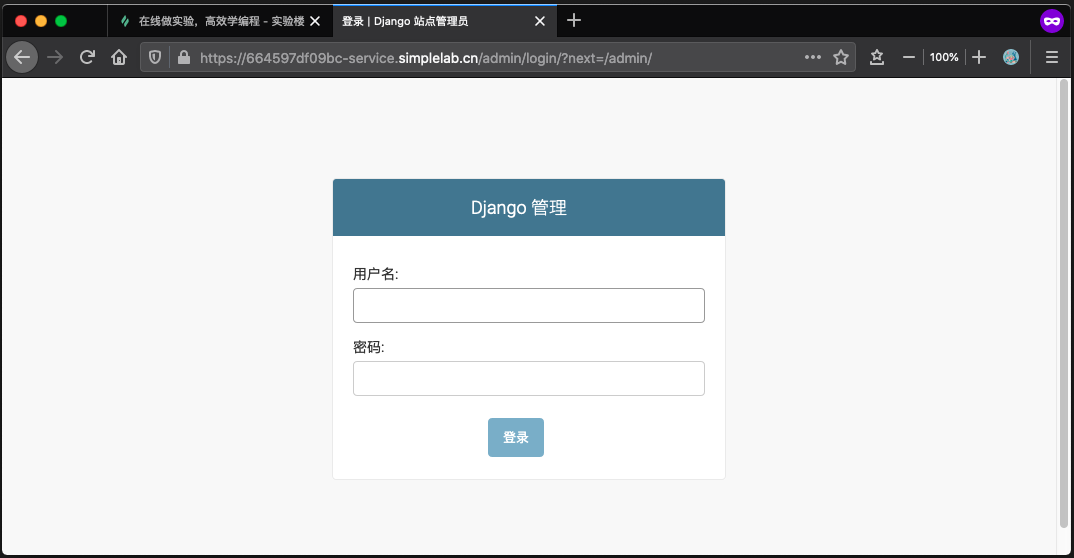
]

其中前两个字符串引入的应用就是管理后台和管理用户权限的包。

在创建项目之处，myweb/urls.py 文件中的 urlpatterns 列表中就有这样的路由映射：

path('admin/', admin.site.urls),

所以可以在项目启动后随时访问 http://127.0.0.1:8080/admin 地址，它会自动跳转到管理员登录页：



这就是 Django 提供的功能完善的后台管理功能，此时只需要创建一个管理员账号进行登录即可进入到后台管理页面。

在项目主目录下执行如下命令创建一个管理员账号：

python manage.py createsuperuser

最后一个参数为 create super user ，意为创建超级用户，即拥有全部权限的管理员用户。Django 提供了默认的表单类用于存储管理员用户的信息，其中包括用户名、邮箱、密码、管理权限等字段。

按照命令提示信息依次写入管理员账户的用户名、邮箱、密码，注意密码在输入时是不显示的，如，管理员名/管理员邮箱/管理员密码：admin/admin@abc.com/admin。

在对项目关联数据后，第一次执行 python manage.py migrate 命令时，就已经创建了存储管理员用户信息的数据表 auth\_user ，可以在 MySQL 客户端中查看：

mysql> SHOW TABLES;

+----------------------------+

| Tables\_in\_myweb |

+----------------------------+

| auth\_group |

| auth\_group\_permissions |

| auth\_permission |

| auth\_user |

| auth\_user\_groups |

| auth\_user\_user\_permissions |

| django\_admin\_log |

| django\_content\_type |

| django\_migrations |

| django\_session |

| learn\_author |

| learn\_authordetail |

| learn\_course |

| learn\_tag |

| learn\_tag\_course |

+----------------------------+

15 rows in set (0.01 sec)

mysql> DESC auth\_user;

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| id | int(11) | NO | PRI | NULL | auto\_increment |

| password | varchar(128) | NO | | NULL | |

| last\_login | datetime(6) | YES | | NULL | |

| is\_superuser | tinyint(1) | NO | | NULL | |

| username | varchar(150) | NO | UNI | NULL | |

| first\_name | varchar(30) | NO | | NULL | |

| last\_name | varchar(150) | NO | | NULL | |

| email | varchar(254) | NO | | NULL | |

| is\_staff | tinyint(1) | NO | | NULL | |

| is\_active | tinyint(1) | NO | | NULL | |

| date\_joined | datetime(6) | NO | | NULL | |

+--------------+--------------+------+-----+---------+----------------+

11 rows in set (0.00 sec)

mysql>

创建好管理员账号，可以在数据库中查看：

mysql> SELECT \* FROM auth\_user\G

## 登录管理员账号

启动项目，打开后台管理页面，输入用户名和密码：



点击 “登录” 按钮，即可进入到后台页面：



这些页面都是由 Django 自动生成的，免去了很多工作。

## 后台管理

在前面的讲授中，在 learn 应用下创建了一些映射类，并使用它们创建了数据表，而且在数据表中添加了少量数据。后台管理主要就是针对这些数据表中的数据进行增删改查操作，使得非程序员用户可以在页面中点击按钮完成工作。

管理员用户在后台页面管理数据表需要依托映射类来实现。

修改 learn/admin.py 文件如下：

from django.contrib import admin

from .models import Author, AuthorDetail, Course, Tag

for model in (Author, AuthorDetail, Course, Tag):

admin.site.register(model)

首先从当前目录下的 models 模块中引入需要管理的映射类，然后将映射类依次注册到 admin 站点。

现在刷新后台管理页面，即可看到相关表单的管理按钮：



如上图所示，在 LEARN 栏目下出现了 4 个表单的管理项。显然 LEARN 就是 learn 应用的全大写。

将鼠标移至 Authors 文字上，点击鼠标左键进入作者数据表的管理页面：



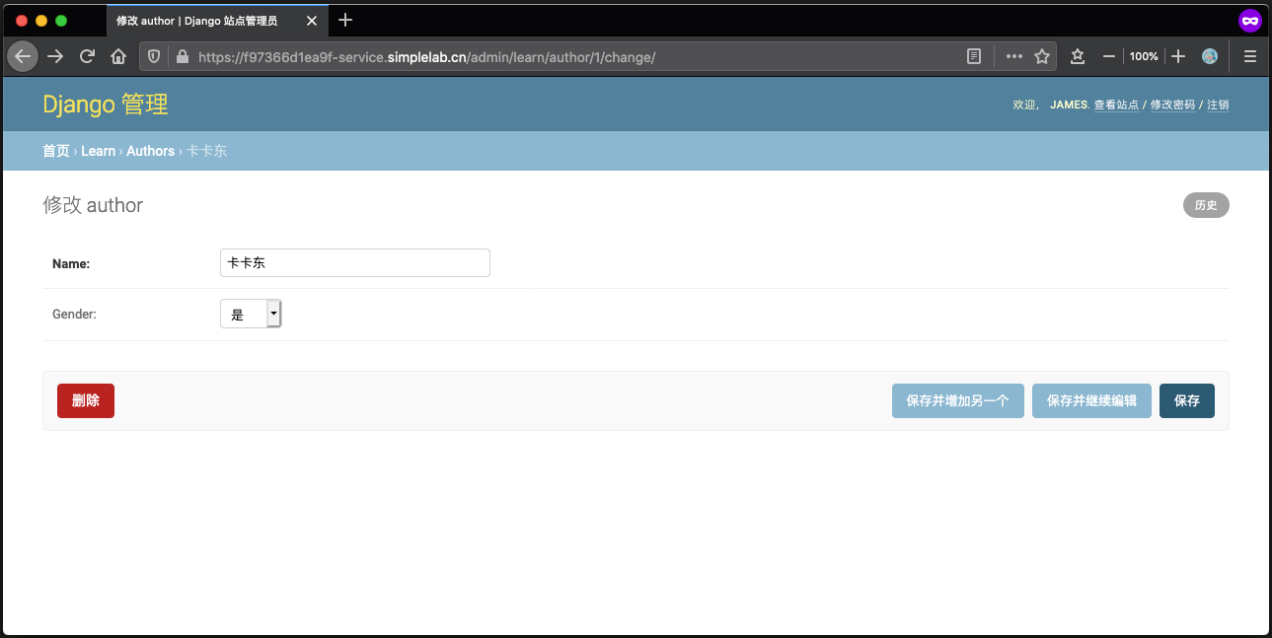
可以看到 URL 的路径中增加了 author/ （即：http://127.0.0.1:8080/admin/learn/author/），点击任意作者的名字可以查看某一条数据的详情，右侧有增加新作者的按钮。选中任意用户，可以执行删除操作：



进入某一条数据的详情页，可以执行编辑和删除操作：



例如将卡卡西的名字改成 “卡卡东” ：



点击保存按钮后，回到本机环境里，打开 MySQL 客户端查询 learn\_author 数据表：

mysql> SELECT \* FROM learn\_author;

+----+-----------+--------+

| id | name | gender |

+----+-----------+--------+

| 1 | 卡卡东 | 1 |

| 2 | 娜美 | 0 |

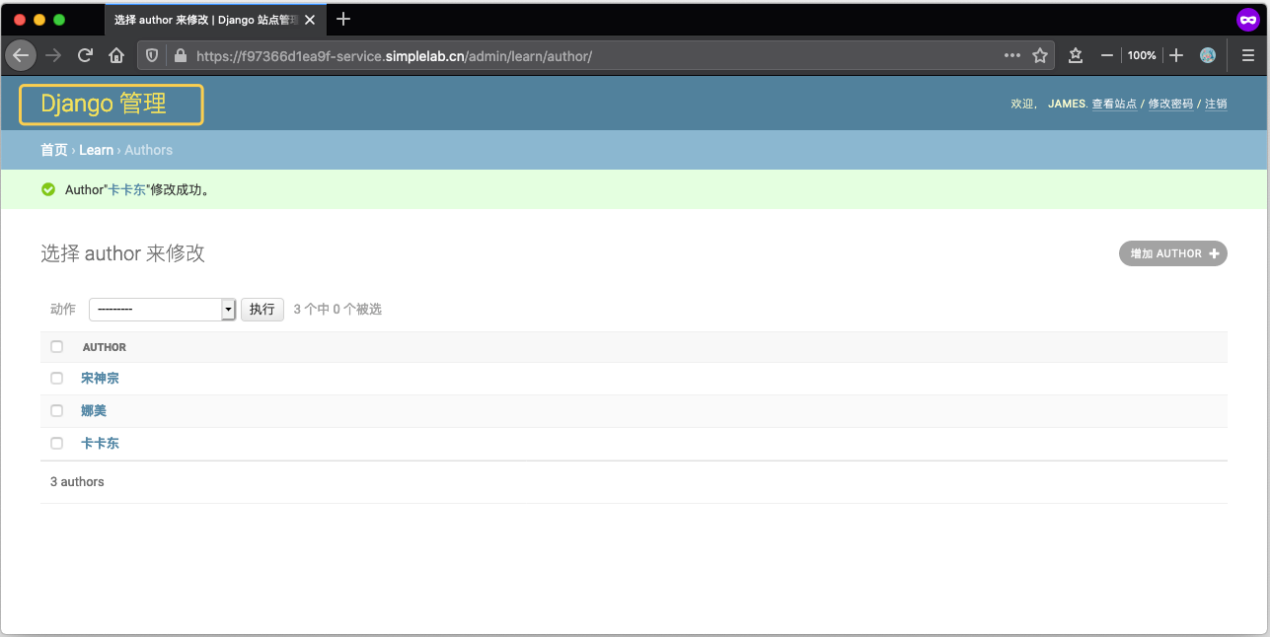
| 3 | 宋神宗 | NULL |

+----+-----------+--------+

3 rows in set (0.01 sec)

可以看到数据库中的数据已经被修改成功。

点击左上角的 “Django 管理” 或者下一行的导航栏中的 “首页” 可以回到后台管理的主页面（现在无需这样做）：



在当前页面只能看到 learn\_author 数据表中 name 这一个字段的值，也就是作者名字。再次修改 learn/admin.py 文件，能看到其它字段的值。

在 learn/admin.py 中创建一个类，类名为 AuthorAdmin ，这个类名是固定的写法，即映射类的类名加上 'Admin' ：

from django.contrib import admin

from .models import Author, AuthorDetail, Course, Tag

class AuthorAdmin(admin.ModelAdmin):

'''映射类对应的后台管理类

'''

list\_display = ['id', 'name', 'gender']

admin.site.register(Author, AuthorAdmin)

for model in (AuthorDetail, Course, Tag):

admin.site.register(model)

如上所示，定义 AuthorAdmin 类时，需要继承 admin.ModelAdmin 类，在新定义的类内部只需写一个 list\_display 列表即可，在列表中写上需要在后台管理页中显示的字段。注意倒数第 4 行注册映射类时，需要将对应的管理类一并注册。

此时刷新页面：



这样就会显示 list\_display 列表中所列的字段值了。字段的名字为全大写的英文，下面演示自定义它们。

方法有两个，特别简单，只需要修改相关映射类的属性定义就可以了。将 learn/models.py 文件中的 Author 类修改如下：

class Author(models.Model):

'''课程作者映射类，包括作者名字、性别等属性

'''

name = models.CharField(max\_length=64, verbose\_name='名字')

gender = models.BooleanField('性别', null=True)

def \_\_str\_\_(self):

return self.name

为字段类增加 verbose\_name 参数或者在第一个位置参数上写上字符串，这两个方法作用相同，都是定义前端页面显示的字段名。这次修改无需执行数据迁移操作，因为它们不涉及数据库的改动，只与前端页面的显示有关。

刷新页面即可看到变动：



以上就是 Django 框架提供的后台管理功能的简单介绍，对于其它映射类，（一定要想清楚后）可以自行定义对应的后台管理类。

## 总结

本部分为介绍了 Django 自带的功能强大的后台管理模块。课程到这里就暂告一段落。

以上共包含 13 部分的讲授内容和 3 个挑战作业。

介绍了 MVT 模型中的三大部分，其中 Model 和 Veiw 介绍得较为详细，Templates 这部分涉及前端的知识较多（也是我的软肋…），希望同学们能够在上述文档的基础上进一步接触更多相关知识。

Django 框架涉及的内容十分复杂，上面也只是介绍了其中的一点皮毛，目的是为了让大家能够了解框架的主体结构，并完成基本的web项目。建议更多的参阅官方文档和源码。

[Django documentation | Django documentation | Django (djangoproject.com)](https://docs.djangoproject.com/en/4.2/)

[Django 文档内容 | Django 文档 | Django (djangoproject.com)](https://docs.djangoproject.com/zh-hans/4.2/contents/)

本部分的参考代码：/code13.zip

JZ1S

the buck stops here