链滴

# Django 3.1 官网学习路线

作者：cuijianzhe

原文链接：https://ld246.com/article/1597665829815

来源网站：链滴

# Django 3.1 官网学习路线

开始按照官网进行学习Django 框架

# 第一部分

## 安装 Django

```
D:\Django_study>python -m django --version
3.1
```

## 创建项目

如果这是您第一次使用 Django，那么您必须进行一些初始设置。也就是说，您需要自动生成一些建立 Django 项目的代码——Django 实例的设置集合，包括数据库配置、特定于 Django 的选项和特定应用程序的设置。

从命令行，cd 到您想存储代码的目录，然后运行以下命令：

```
django-admin startproject Django_study
```

## 开发服务

```
python manage.py runserver
```

默认情况下，runserver 命令在端口 8000 的内部 IP 上启动开发服务器。

如果您想更改服务器的端口，请将其作为命令行参数传递。例如，这个命令在端口 8080 上启动服务：

```
python manage.py runserver 8080
```

如果您想更改服务器的 IP，请将其与端口一起传递。例如，要监听所有可用的公共 ip(如果你正在运行 Vagrant 或想在网络上的其他计算机上监听你的工作，这很有用)，使用：

```
python manage.py runserver 0:8000
```

## 创建 polls 应用

现在您的环境——一个“项目”——已经设置好了，您可以开始工作了。

用 Django 编写的每个应用程序都由一个遵循特定约定的 Python 包组成。Django 附带一个工具，可以自动生成应用程序的基本目录结构，因此您可以专注于编写代码，而不是创建目录。

```
python manage.py startapp polls
```

## 编写 views.py 文件

我们来写第一个视图。打开文件 polls/views.py，并在其中放入以下 Python 代码：

```
from django.http import HttpResponse
```

```
def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

这是 Django 中最简单的视图。要调用视图，我们需要将其映射到一个 URL—为此我们需要一个 URL onf。

要在轮询目录中创建 URLconf，请创建一个名为 urls.py 的文件。你的 app 目录现在应该是这样的：

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    urls.py
    views.py
```

在 poll /urls.py 文件中包含以下代码：

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

下一步是将根 URLconf 指向 polls.urls 模块。在 mysite / urls.py 中，为 django.urls.include 添加个导入，并在 urlpatterns 列表中插入一个 include()，这样您就可以：

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

- include()函数允许引用其他 URLconf。每当 Django 遇到 include（）时，它都会截断匹配到该点 URL 的任何部分，并将剩余的字符串发送到包含的 URLconf 中以进行进一步处理。
- include()背后的想法是使即插即用 url 变得容易。因为轮询是在它们自己的 URLconf (polls/urls.py 中，它们可以被放在“/polls/”下，或“/fun_polls/”下，或“/content/polls/”下，或任何其他径根目录下，应用程序仍然可以工作。

什么时候用 include() ?

在包含其他 URL 模式时，应该始终使用 include()。admin. site .url 是唯一的例外。

现在已经将索引视图连接到 URLconf。验证它的工作与以下命令：

```
python manage.py runserver
```

浏览器访问： http://127.0.0.1/polls/

**path()函数传递了四个参数，两个是必需的：路由和视图，两个是可选的：kwargs 和 name。在这一点上，有必要回顾一下这些论点的意义。**

**path() argument: route**

route 是一个包含 URL 模式的字符串。在处理请求时，Django 从 urlpatterns 中的第一个模式开始沿着列表向下移动，将所请求的 URL 与每个模式进行比较，直到找到一个匹配的。

模式不搜索 GET 和 POST 参数或域名。例如，在对https://www.example.com/myapp/的请求中，RLconf 将查找 myapp/。在请求https://www.example.com/myapp/?页面 =3 时，URLconf 也会找 myapp/。

**path() argument: view**

当 Django 找到匹配的模式时，它调用指定的视图函数，第一个参数是 HttpRequest 对象，从路由“捕获”的任何值都是关键字参数。我们会给出一个例子。

**path() argument: kwargs**

可以在字典中将任意关键字参数传递给目标视图。在本教程中，我们不会使用 Django 的此功能。

**path() argument: name**

通过命名 URL，您可以从 Django 的其他地方明确地引用它，特别是在模板中。这个强大的特性允许在只修改单个文件的同时对项目的 URL 模式进行全局更改。

当您熟悉了基本的请求和响应流后，请阅读本教程的第 2 部分，开始使用数据库。

# 第二部分

设置数据库，创建您的第一个模型，并快速介绍 Django 自动生成的管理网站。

## 数据库设置

打开 Django_study / settings.py。这是一个普通的 Python 模块，带有表示 Django 设置的模块级量。

如果您不使用 SQLite 作为数据库，则必须添加其他设置，例如 USER，PASSWORD 和 HOST。有更多详细信息看下请参见 DATABASES 的参考文档

**ENGINE**

'django.db.backends.postgresql'

'django.db.backends.mysql'

'django.db.backends.sqlite3'

'django.db.backends.oracle'

当你编辑 mysite/setting.py。将 TIME_ZONE 设置为您的时区。

另外，请注意文件顶部的 INSTALLED_APPS 设置。它包含这个 Django 实例中激活的所有 Django 用程序的名称。应用程序可以在多个项目中使用，您可以将它们打包并分发给他们项目中的其他人使。

默认情况下，INSTALLED_APPS 包含以下应用程序，所有这些跟 Django:

- django.contrib.admin – 后台管理页面
- django.contrib.auth – 认证系统。
- django.contrib.contenttypes – 内容类型框架。
- django.contrib.sessions – 会话框架。
- django.contrib.messages – 消息传递框架。
- django.contrib.staticfiles – 静态文件管理框架。

但是，其中一些应用程序至少使用了一个数据库表，因此在使用表之前，我们需要在数据库中创建表为此，运行以下命令：

```
python manage.py migrate
```

他的 migrate 命令会查看 INSTALLED_APPS 设置，并根据 mysite/settings.py 文件中的数据库设和应用附带的数据库迁移(我们稍后将讨论这些)创建任何必要的数据库表。对于它应用的每个迁移，将看到一条消息。如果你感兴趣，运行数据库的命令行客户端，输入\dt (PostgreSQL)，SHOW TABES;(MariaDB, MySQL)， .schema (SQLite)，或从 USER_TABLES 中选择 TABLE_NAME;(Oracle)来示 Django 创建的表。

如前所述，默认应用程序是为常见情况而包含的，但不是每个人都需要它们。如果您不需要它们中的何一个或全部，那么可以在运行 migrate 之前随意地注释或删除 INSTALLED_APPS 中的适当行。migate 命令只会在 INSTALLED_APPS 中运行应用程序的迁移。

## 创建模型

在我们的投票应用程序中，我们将创建两个模型：

**问题和选择。**问题有问题和发布日期。 “选择”具有两个字段：选择的文本和投票提示。每个选择都一个问题相关联

编辑 polls/models.py file 如下：

```
from django.db import models
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

在这里，每个模型都由一个子类**django.db.models.Model**表示。每个模型都有许多类变量，每个变量表示模型中的一个数据库字段

每个字段都由**Field**类的实例表示-例如，**CharField**用于字符字段，**DateTimeField**用于日期时间。告诉 Django 每个字段保存什么类型的数据。

每个 Field 实例的名称（例如**question_text**或**pub_date**）是该字段的名称，采用机器友好的格式您将在 Python 代码中使用此值，数据库将使用它作为列名。

可以对字段使用可选的第一个位置参数来指定我们可读的名称。它在 Django 的几个内省部分中使用同时也用作文档。如果没有提供这个字段，Django 将使用机器可读的名称。在本例中，我们仅为**Question.pub_date**定义了一个人类可读的名称。对于此模型中的所有其他字段，该字段的机器可读名称足以作为其人类可读的名称。

一些 Field 类具有必需的参数。例如，CharField 要求您给它一个 max_length。我们将很快看到，不仅用于数据库架构，而且用于验证。

字段还可以有各种可选参数;在本例中，我们将投票的默认值设置为 0。

最后，请注意使用外键定义了关系。这告诉 Django 每个选择都与一个问题相关。Django 支持所有见的数据库关系：多对一、多对多和一对一。

## 激活模型

这一小段模型代码为 Django 提供了大量信息。有了它，Django 可以：

- 为这个应用程序创建一个数据库模式(创建表语句)。
- 创建用于访问问题和选择对象的 Python 数据库访问 API。

**但是首先我们需要告诉我们的项目已经安装了投票应用程序。**

要在我们的项目中包含应用程序，我们需要在 INSTALLED_APPS 设置中添加对其配置类的引用。PollConfig 类在 polls/apps.py 文件中，因此它的点状路径是'poll .apps.PollsConfig'。编辑 mysite/settngs.py 文件，并将这个虚线路径添加到 INSTALLED_APPS 设置中。它是这样的：

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

现在 Django 知道如何包含这个 polls 应用程序了。然后执行：

```
python manage.py makemigrations polls
```

将会看到如下：

```
D:\Django_study>python manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    - Create model Question
    - Create model Choice
```

通过运行 makemigrations，您将告诉 Django 您对模型进行了一些更改（在这种情况下，您进行了的更改），并且希望将更改存储为迁移。

迁移是 Django 将更改存储到您的模型(以及您的数据库模式)的方式——它们是磁盘上的文件。如果愿意，你可以阅读你的新模型的迁移;它是 poll /migrations/0001_initial.py 文件。不用担心，不必次 Django 生成一个时都读取它们，但是如果您想手动调整 Django 的更改方式，它们是可人工编辑。

有一个命令可以为您运行迁移并自动管理您的数据库模式——这叫做 migrate，我们马上就会讲到—但首先，让我们看看迁移会运行什么 SQL。sqlmigrate 命令接受迁移名称并返回它们的 SQL.

```
python manage.py sqlmigrate polls 0001
```

看到如下信息：

```
D:\Django_study> python manage.py sqlmigrate polls 0001
--
-- Create model Question
--
CREATE TABLE `polls_question` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `qu
stion_text` varchar(200) NOT NULL, `pub_date` datetime(6) NOT NULL);
--
-- Create model Choice
--
CREATE TABLE `polls_choice` (`id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY, `choic
_text` varchar(200) NOT NULL, `votes` integer NOT NULL, `question_id` integer NOT NULL);
ALTER TABLE `polls_choice` ADD CONSTRAINT `polls_choice_question_id_c5b4b260_fk_polls_
uestion_id` FOREIGN KEY (`question_id`) REFERENCES `polls_question` (`id`);
```

请注意以下几点：

确切的输出将根据所使用的数据库而有所不同。上面的例子是为 PostgreSQL 生成的。

表名是通过结合应用程序的名称(投票)和模型的小写名称——问题和选择——自动生成的。(您可以重此行为。)

主键(id)会自动添加。(你也可以忽略这个。)

按照惯例，Django 会将"_id"附加到外键字段名。(是的，你也可以重写这个。)

外键关系是通过外键约束来显式的。不要担心可延期的部分;它告诉 PostgreSQL 在事务结束之前不强制执行外键。

它是为你使用的数据库量身定制的，所以数据库特定的字段类型，如 auto_increment (MySQL)，串(PostgreSQL)，或整数主键 autoincrement (SQLite)会自动为你处理。字段名的引号也是一样——如，使用双引号或单引号。

sqlmigrate 命令实际上并没有在数据库上运行迁移—相反，它将迁移结果打印到屏幕上，以便您可看到 SQL Django 认为需要什么迁移。它对于检查 Django 要做什么，或者您的数据库管理员是否需 SQL 脚本进行更改非常有用。

如果您感兴趣，还可以运行**python manager .py check**;这将检查项目中的任何问题，而不进行迁或接触数据库。

现在，再次运行 migrate 在数据库中创建这些模型表：

```
D:\Django_study> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0001_initial... OK
```

migrate 命令获取所有还没有应用的迁移(Django 跟踪哪些迁移是使用数据库中名为 django_migrations 的特殊表应用的)，并在数据库上运行它们——本质上，就是将您对模型所做的更改与数据库中的式同步。

迁移功能非常强大，它允许您在开发项目时随着时间的推移更改模型，而不需要删除数据库或表并创新表——它专门用于实时升级数据库，而不会丢失数据。我们将在本教程的后面部分更深入地介绍它，但是现在，请记住进行模型更改的三步指南

- 更改您的模型(在 **models.py**中)。
- 运行 **python manager .py makmigration******来为这些更改创建迁移

● 运行 **python manager .py migrate**将这些更改应用到数据库。

之所以要用单独的命令来进行迁移是因为你要将迁移提交到版本控制系统，并与应用一起发布;它们不使您的开发更容易，还可用于其他开发人员和生产环境中。

请阅读django-admin 文档以获得关于 manager .py 实用程序可以做什么的完整信息。

## 玩转 API

现在，让我们跳入交互式 Python shell 并尝试使用 Django 提供的免费 API。要调用 Python Shell 请使用以下命令：

```
python manage.py shell
```

我们使用它，而不是简单地输入“python”，因为 manager .py 设置了 DJANGO_SETTINGS_MODULE 环境变量，它为 Django 提供了 mysite/settings.py 文件的 python 导入路径。

```
D:\Django_study>python manage.py shell
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.9.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from polls.models import Choice, Question
#现在系统里还没有问题
In [2]: Question.objects.all()
Out[2]: <QuerySet []>
#创建一个新问题。
#在默认设置文件中启用了对时区的支持，因此
# Django期望为pub_date使用tzinfo的日期时间。使用timezone.now ()
#而不是date .date .now()，它将做正确的事情
In [3]: from django.utils import timezone

In [4]: q = Question(question_text="What's new?", pub_date=timezone.now())
#将对象保存到数据库中。您必须显式调用save()。
In [5]:  q.save()
#现在它有一个ID。
In [6]: q.id
Out[6]: 1
#通过Python属性访问模型字段值。
In [7]: q.question_text
Out[7]: "What's new?"

In [8]: q.question_text
Out[8]: "What's new?"

In [9]: q.pub_date
Out[9]: datetime.datetime(2020, 8, 17, 7, 13, 52, 104000, tzinfo=<UTC>)
#通过更改属性来更改值，然后调用save()。
In [10]: q.question_text = "What's up?"

In [11]: q.save()
#objects.all（）显示数据库中的所有问题。
In [12]: Question.objects.all()
Out[12]: <QuerySet [<Question: Question object (1)>]>
```

等一下。<Question: Question object(1)> 不是这个对象的有用表示。让我们通过编辑问题模型(在 olls/models.py 文件中)并在问题和选择中添加其他的__str__()方法来解决这个 **Question** 和 **Choice**

```
from django.db import models
# Create your models here.
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __str__(self):
        return self.question_text
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    def __str__(self):
        return self.choice_text
```

向模型中添加__str__()方法非常重要，这不仅是为了方便您在处理交互式提示时使用，而且因为在 Djago 的自动生成的管理员中都使用了对象的表示形式。

让我们也为这个模型添加一个自定义方法：

注意，添加了 **import datetime** 和 **from django.utils import timezone**。utils 导入 timezone 以引用 Python 的标准 datetime 模块和 Django 的与时间区域相关的实用程序。分别时区。如果您熟悉 Python 中的时区处理，可以在时区支持文档中了解更多内容

保存这些更改，并通过运行**Python manager .py shell**再次启动一个新的 Python 交互式 shell:

(略)

## 创建管理员账号

```
python manage.py createsuperuser
python manage.py runserver
```

## 让投票应用程序在管理可修改

但我们的投票应用在哪？它没有显示在管理索引页上。

还有一件事要做：我们需要告诉管理员**Question**对象有一个管理接口。为此，打开 poll/admin.py 件，并编辑它，使其看起来像这样：

```
from django.contrib import admin
from .models import Question

# Register your models here.
admin.site.register(Question)
```

## 探索免费的管理功能

现在我们已经注册了 Question，Django 知道应该在 admin 索引 pag 上显示它


点击“Questions”。现在您在“更改列表”页面查看问题。此页面显示数据库中的所有问题，并允您选择一个进行更改。有“What's up?”“我们之前提出的问题是：


**这里需要注意的是：**

- 表单是根据问题模型自动生成的。
- 不同的模型字段类型(DateTimeField、CharField)对应于适当的 HTML 输入小部件。每种类型的段都知道如何在 Django 管理中显示自己。
- 每个 DateTimeField 都有免费的 JavaScript 快捷键。日期有一个“今天”快捷方式和日历弹出，间有一个“现在”快捷方式和一个方便的弹出，列出了通常输入的时间。
- 页面的底部提供了几个选项：
- 保存-保存更改并返回此类型对象的更改列表页。
- 保存并继续编辑——保存更改并重新加载此对象的管理页面。
- 保存并添加另一个——保存更改并为这种类型的对象加载一个新的空白表单。
- 删除-显示删除确认页面。
- 如果“Date published”的值与教程 1 中创建问题时的时间不匹配，这可能意味着您忘记为 TIME_ONE 设置正确的值。更改它，重新加载页面并检查正确的值出现。
- 通过点击“今天”和“现在”快捷键更改“发布日期”。然后点击“保存并继续编辑”。然后点击上角的“历史”。您将看到一个页面，其中列出了通过 Django 管理员对这个对象所做的所有更改，及更改者的时间戳和用户名：

# 第三部分

## 概览

视图是 Django 应用程序中的 Web 页面的“类型”，通常提供特定的功能和特定的模板。例如，在个博客应用程序中，您可能有以下视图：

- 博客首页-显示最近的几个条目。
- 条目“详细信息”页面——一个条目的永久链接页面。
- 基于年份的归档页面——显示给定年份中的所有月份和条目。
- 基于月份的归档页面——显示给定月份中的所有天数和条目。
- 基于天的归档页面——显示给定天中的所有条目。
- 评论操作——处理向给定条目发布评论。

在我们的投票应用程序中，我们将有以下四个视图：

- 问题“索引”页面-显示最近的几个问题。
- 问题“细节”页面-显示一个问题文本，没有结果，但有一个表格来投票。
- 问题“结果”页面-显示特定问题的结果。
- 投票行动-处理对特定问题中的特定选择进行投票。
- 在 Django 中，Web 页面和其他内容是通过视图传递的。每个视图都由一个 Python 函数(或方法对于基于类的视图)表示。Django 将通过检查被请求的 URL(确切地说，是域名后面的 URL 部分)来择一个视图。

现在，在你上网的时候，你可能会遇到这样 **ME2/Sites/dirmod.htm?sid=&type=gen&mod=Cor+Pages&gid=A6CD4967199A42D9B65B1B**您会很高兴地知道，Django 允许我们使用比这更优的 URL 模式。

URL 模式是 URL 的一般形式，例如：**/newsarchive/<year>/<month>/**。

为了从 URL 到视图，Django 使用了所谓的“**URLconfs**”。URLconf 将 URL 模式映射到视图。

本教程提供了使用 URLconfs 的基本指导，您可以参考URL 分派器了解更多信息。

## 添加更多的视图

现在让我们向 poll /views.py 添加更多的视图。这些观点略有不同，因为他们有一个论点：

```
def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)

def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)

def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```

**将这些新视图 polls.urls 中。通过添加以下 path()调用的 polls.urls 模块(polls/urls.py):**

polls/urls.py¶

```
from django.urls import path
from . import views

urlpatterns = [
    # ex: /polls/
    path('', views.index, name='index'),
    # ex: /polls/5/
    path('<int:question_id>/', views.detail, name='detail'),
    # ex: /polls/5/results/
    path('<int:question_id>/results/', views.results, name='results'),
    # ex: /polls/5/vote/
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

看看你的浏览器，" /polls/34/ "。它将运行 detail()方法并显示 URL 中提供的任何 ID。尝试“/polls/4/results/”和 /polls/34/vote/——这些将显示占位符结果和投票页面。

当有人从您的网站请求页面时（例如“ /polls/34 /"），Django 将加载 mysite.urls Python 模块，为它由 ROOT_URLCONF 设置指向。它找到名为 urlpatterns 的变量，并按顺序遍历模式。在“ poll /”找到匹配项后，它将剥离匹配的文本（“ polls /”），并将剩余的文本“ 34/”发送到“ polls.url ” URLconf，以进行进一步处理。在那里，它与“ <int：question_id> /”匹配，从而导致对 detail(视图的调用，如下所示：

```
detail(request=<HttpRequest object>, question_id=34)
```

question_id=34 部分来自int:question_id。使用尖括号“捕获”URL 的一部分，并将其作为关键字数发送给视图函数。字符串的：question_id> 部分定义了将用于标识匹配模式的名称，而 <int:部分一个转换器，用于确定哪些模式应该匹配 URL 路径的这一部分。

## 编写实际应用的视图

每个视图负责做两件事中的一件：返回一个包含被请求页面内容的 HttpResponse 对象，或者引发一异常，比如 Http404。剩下的就看你了。

为方便起见，让我们使用 Django 自己的数据库 API，我们在教程 2 中进行介绍。这是新 index()视的一个尝试，它显示系统中最新的 5 个投票问题，根据发布日期用逗号分隔：：

polls/views.py¶

```
from django.http import HttpResponse

from .models import Question


def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    output = ', '.join([q.question_text for q in latest_question_list])
    return HttpResponse(output)

# Leave the rest of the views (detail, results, vote) unchanged 其余视图函数（detail, results, vo
e）保持不变
```

但这里有一个问题：页面的设计是在视图中硬编码的。如果您想改变页面的外观，您必须编辑此 Pyth n 代码。因此，让我们使用 Django 的模板系统来创建视图可以使用的模板，从而将设计与 Python 离开来。

首先，在您的轮询目录中创建一个名为 templates 的目录。Django 会在其中寻找模板。

项目的模板设置描述了 Django 如何加载和呈现模板。默认设置文件配置一个 DjangoTemplates 后，其 APP_DIRS 选项设置为 True。按照惯例，DjangoTemplates 在每个 INSTALLED_APPS 中寻找个“templates”子目录。

在刚刚创建的模板目录中，创建另一个名为 polls 的目录，并在该目录中创建一个名为 index.html 文件。换句话说，您的模板应该在 polls/templates/polls/index.html 中。由于 app_directory 模板载器的工作方式如上所述，您可以在 Django 中将这个模板引用为 poll /index.html。

polls/templates/polls/index.html¶

```
{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href="/polls/{{ question.id }}/">{{ question.question_text }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

现在让我们用模板更新 poll/views.py 中的索引视图：

```
from django.http import HttpResponse
from django.template import loader

from .models import Question


def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = {
        'latest_question_list': latest_question_list,
    }
    return HttpResponse(template.render(context, request))
```

该代码加载名为 polls/index.html 的模板，并传递一个上下文。上下文是模板变量名到 Python 对象映射字典。

通过将浏览器指向" /polls/ "来加载页面，您应该会看到一个项目符号列表，其中包含教程第二部分的" What 's up "问题。

## A shortcut: render

加载模板、填充上下文并将呈现模板的结果返回 HttpResponse 对象是一种非常常见的习惯用法。Django 提供了一个快捷方式。这是完整的 index()视图，重写：

```
from django.shortcuts import render
from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)
```

注意，一旦我们在所有这些视图中完成了这些操作，我们就不再需要 import loader 和 HttpRespons
(如果您仍然拥有用于 detail, results, and vote 的函数方法，那么您将希望保留 HttpResponse)。

render()函数将请求对象作为第一个参数，将模板名称作为第二个参数，将字典作为可选的第三个参
。它返回使用给定上下文呈现的给定模板的 HttpResponse 对象。

# Raising a 404 error

现在，让我们处理问题细节视图——显示给定投票的问题文本的页面。视图：

```
from django.http import Http404
from django.shortcuts import render

from .models import Question
# ...
def detail(request, question_id):
    try:
        question = Question.objects.get(pk=question_id)
    except Question.DoesNotExist:
        raise Http404("Question does not exist")
    return render(request, 'polls/detail.html', {'question': question})
```

这里的新概念是：如果请求的 ID 不存在问题，视图就会抛出 Http404 异常。

稍后我们将讨论你可以把什么放到那个 poll/detail.html 模板中，但如果你想快速得到上面的例子，
个文件包含：

polls/templates/polls/detail.html¶

```
{{ question }}
```

# Namespacing URL names

polls/urls.py

```
from django.urls import path

from . import views

app_name = 'polls'
urlpatterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/', views.detail, name='detail'),
    path('<int:question_id>/results/', views.results, name='results'),
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

然后修改模板语言

```
<li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
```

指向命名空间详细信息视图

```
<li><a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a></li>
```

# 第四部分

## 优化代码

polls/templates/polls/detail.html¶

```
<h1>{{ question.question_text }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
{% for choice in question.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}"

    <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
{% endfor %}
<input type="submit" value="Vote">
</form>
```

polls/urls.py¶

```
path('<int:question_id>/vote/', views.vote, name='vote'),
```

polls/views.py¶

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse

from .models import Choice, Question
# ...
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

现在，创建一个 polls/results.html 模板：

```
<h1>{{ question.question_text }}</h1>
```

```
<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="{% url 'polls:detail' question.id %}">Vote again?</a>
```

# 第五部分：自定义管理表单

通过将问题模型注册为 admin.site.register(Question)，Django 能够构造一个默认的表单表示。通，您需要定制管理表单的外观和工作方式。可以通过在注册对象时告诉 Django 所需的选项来实现。

通过重新排列编辑表单中的字段来了解其工作原理。将 admin.site.register（Question）行替换为：

polls/admin.py¶

```
from django.contrib import admin
from .models import Question

class QuestionAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question_text']

admin.site.register(Question, QuestionAdmin)
```

将遵循这个模式——创建一个模型管理类，然后将它作为第二个参数传递给 admin.site.register()—任何时候您需要更改模型的管理选项。

上面这个特殊的变化使得“发布日期”出现在“问题”字段之前：


对于只有两个字段的管理表单来说，这并不令人印象深刻，但是对于有几十个字段的管理表单来说，择直观的顺序是一个重要的可用性细节。

说到几十个字段的表单，你可能想把表单分成字段集：

```
from django.contrib import admin
from .models import Question
```

```
class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date']}),
    ]

admin.site.register(Question, QuestionAdmin)
```

字段集中每个元组的第一个元素是字段集的标题。这是我们现在的表格：

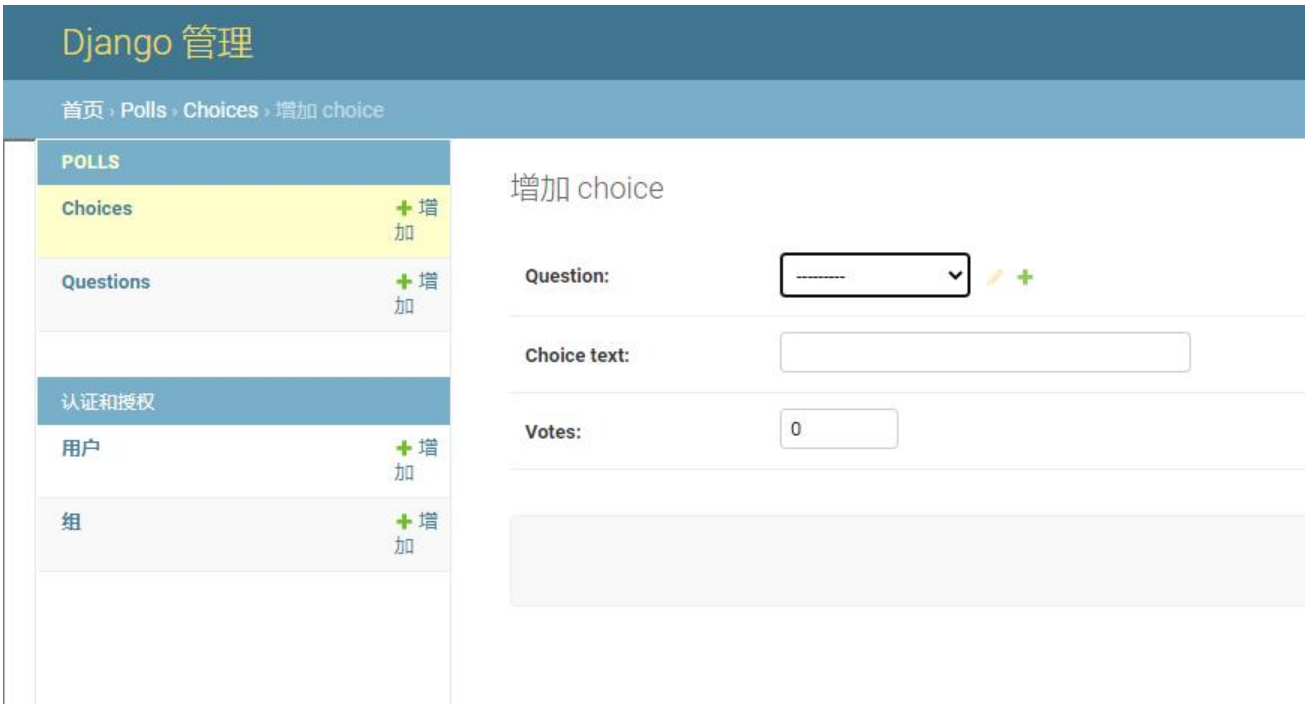

## 添加相关对象

我们有问题管理页面，但是问题有多个选择，并且管理页面不显示选择。 然而。
有两种方法可以解决此问题。首先是像在 Question 上一样向管理员注册 Choice：

polls/admin.py¶

```
from django.contrib import admin
from .models import Choice, Question
# ...
admin.site.register(Choice)
```

现在， “选择”是 Django 管理员中的可用选项。 “添加选择”表单如下所示：




在该表单中，“Question”字段是一个选择框，包含数据库中的每个问题。Django 知道一个外键应在管理中表示为一个框。在我们的例子中，目前只存在一个问题。

还要注意在“问题”旁边的“添加另一个”链接。每一个与其他对象具有 ForeignKey 关系的对象都以免费得到这个。当你点击“添加另一个”，你会得到一个弹出窗口的“添加问题”形式。如果在该口中添加一个问题并单击“Save”，Django 会将该问题保存到数据库中，并在您正在查看的“add choice”表单中动态地将其添加为选中的选项。

但是，实际上，这是向系统添加 Choice 对象的一种低效方式。最好在创建 Question 对象时直接添一堆 Choices。让我们做到这一点。

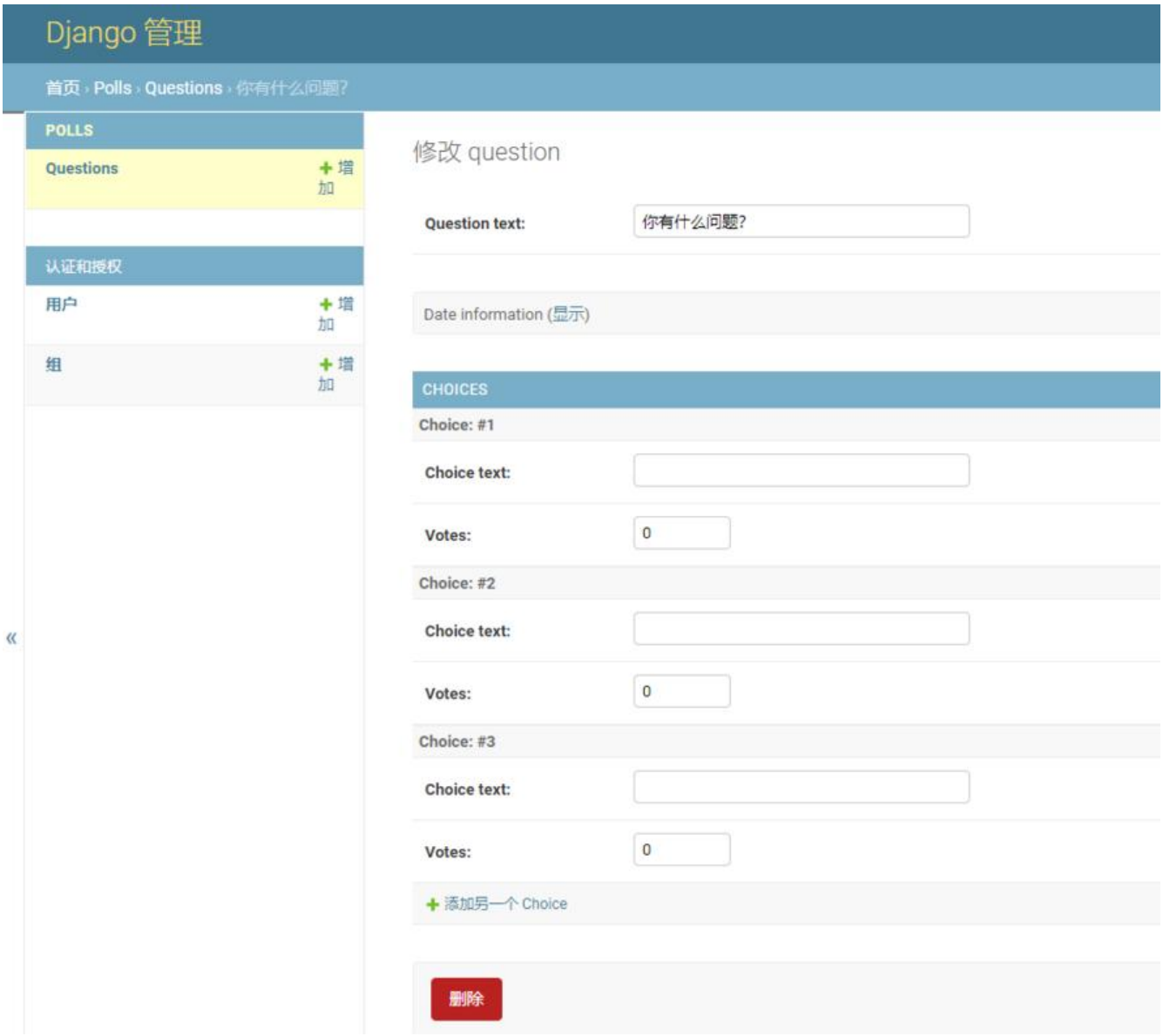
删除对 Choice 模型的 register()调用。然后，编辑问题注册码以读取：

polls/admin.py¶

```
from django.contrib import admin
from .models import Choice, Question

class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3

class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)
```

这告诉 Django：“选择对象在问题管理页面上编辑。默认情况下，为 3 个选项提供足够的字段。”

加载“添加问题”页面，看看是什么样子：




不过有一个小问题。它需要大量的屏幕空间来显示用于输入相关选择对象的所有字段。因此，Django 提供了一种表格方式来显示内联相关的对象。要使用它，将 ChoiceInline 声明更改为：

polls/admin.py¶

```
class ChoiceInline(admin.TabularInline):
    #...
```

使用表格内联(而不是 StackedInline)，相关对象将以更紧凑的、基于表格的格式显示：


原文链接：Django 3.1 官网学习路线

# 自定义管理员更改列表

现在问题管理页面看起来不错了，让我们对“更改列表”页面做一些调整——这个页面显示系统中的有问题。

这是它现在的样子：


默认情况下，Django 显示每个对象的 str()。但有时如果我们能显示单独的字段会更有帮助。要做到一点，使用 list_display 管理选项，它是一个字段名的元组，以列的形式显示在对象的更改列表页面：

polls/admin.py¶

```
class QuestionAdmin(admin.ModelAdmin):
    # ...
    list_display = ('question_text', 'pub_date')
```

现在，问题更改列表页面如下所示：


您可以单击列标题按这些值进行排序——was_published_recent 标题除外，因为不支持按任意方法输出进行排序。还要注意，was_published_recent 的列标题在默认情况下是方法的名称(下划线替换空格)，并且每行包含输出的字符串表示。

您可以通过为该方法（在 polls / models.py 中）提供一些属性来改进该属性，如下所示：

polls/models.py¶

```
class Question(models.Model):
    # ...
```

```
    def was_published_recently(self):
        now = timezone.now()
        return now - datetime.timedelta(days=1) <= self.pub_date <= now
    was_published_recently.admin_order_field = 'pub_date'
    was_published_recently.boolean = True
    was_published_recently.short_description = 'Published recently?'
```

更多关于list_display 的信息看这里

再次编辑您的 poll /admin.py 文件，并向问题更改列表页面添加一个改进：使用 list_filter 的过滤器在 QuestionAdmin 类里面添加以下代码：

```
list_filter = ['pub_date']
```

这会添加一个“过滤器”侧边栏，可以通过 pub_date 字段过滤更改列表：


显示的过滤器类型取决于您要过滤的字段类型。由于 pub_date 是 DateTimeField，因此 Django 知提供适当的过滤器选项：“任何日期”，“今天”，“过去 7 天”，“本月”，“今年”。

这很好。让我们添加一些搜索功能：

```
search_fields = ['question_text']
```


现在还需要注意的是，更改列表提供了免费的分页。默认值是每个页面显示 100 个条目。更改列表分、搜索框、过滤器、日期层次结构和列标题排序都像您认为的那样协同工作。

成果：


# Python改如何学习?

- 看书
- 好好学
- 睡觉
- 玩游戏

Vote