[笔记][跟老齐学 Python Django 实战 第2版][5-6]

5. 收集和展示图片

本书专门安排一章讲解图片的收集和展示功能。收集就是将网上看到的图片保存下来，展示就是在页面中向浏览者展示自己收集到的图片。

5.1 收集网络图片

网上有很多用户喜欢的图片，人总有一种癖好，自己喜欢的就想纳为己有。现在，数字化的东西可以让我们与他人分享，同时自己也不失去。

本节就完成一个功能，允许用户通过图片地址将图片收藏到自己的账户中，并保存在服务器中。

将本章的应用与前面发表文章的应用区分开来，新建一个应用。进入本项目的根目录中，执行如下命令。

```
python manage.py startapp image
```

上述代码新建了一个名为 `image` 的应用。本章的所有开发代码都放在这个应用中。

新建的应用要在项目中声明，即在 `./mysite/settings.py` 文件的 `INSTALLED_APPS` 的值列表中增加 `image` 应用，代码如下。

```
INSTALLED_APPS = [
    # ...,
    'image',
]
```

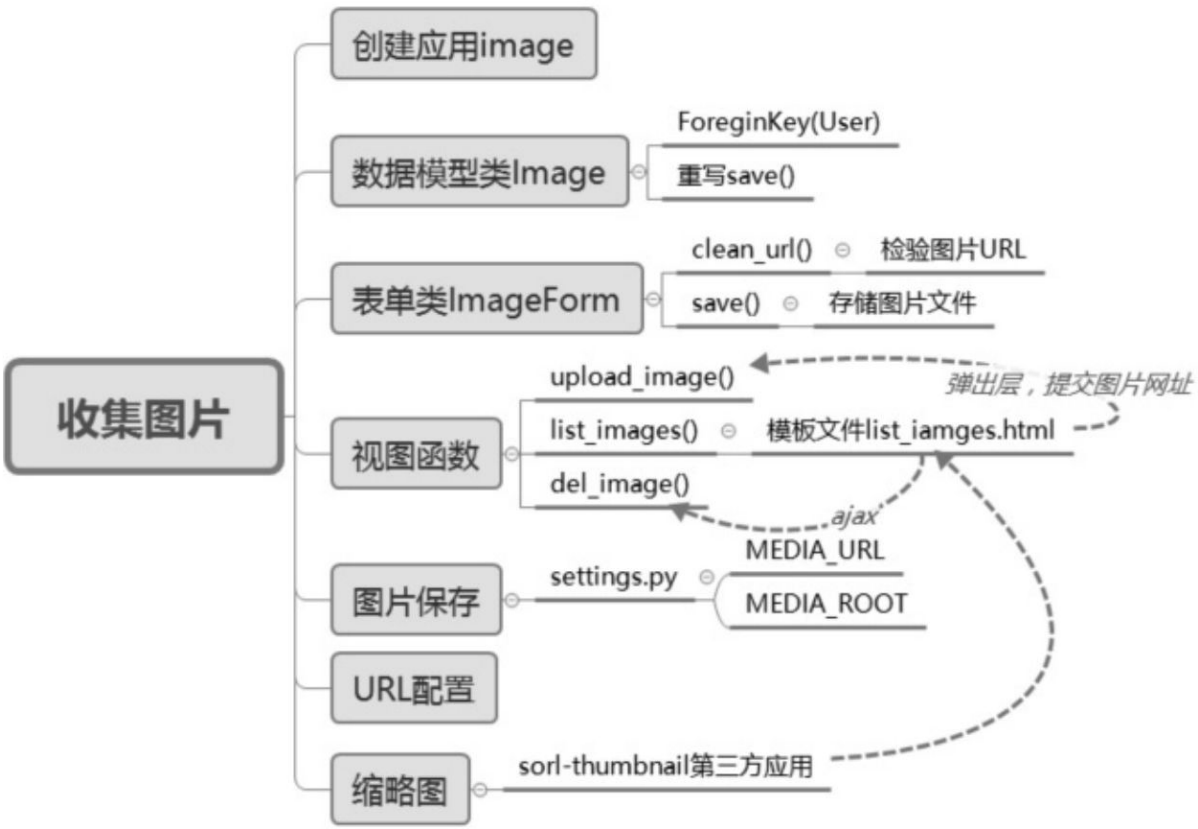
在正式开发之前，还要做一项准备——安装 `Pillow`，它是 `Python` 中的一个图片支持模块（详细内容可以访问 http://pillow.readthedocs.io/en/latest/index.html）。

> 团子注：使用 `ImageField` 必须安装 `Pillow`。

```
pip install pillow
```

> 团子注：每安装一个库，导出一下 `requirements.txt`。

准备工作已经完成，下面就开始编写实现功能的代码。



创建图片相关类

为了能够实现图片的上传和保存，必须要建立数据模型类和表单类。数据模型类用于保存图片的相关信息，对应着数据库表；表单类用于图片上传的数据检验和保存等。

编辑 `./image/models.py` 文件，输入如下代码，创建 `Image` 类。

```
from django.contrib.auth.models import User
from django.db import models
from slugify import slugify


class Image(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCA
DE, related_name='images')
    title = models.CharField(max_length=300)
    url = models.URLField()
    slug = models.SlugField(max_length=500, blank=True)
    created = models.DateField(auto_now_add=True, db_inde
x=True)
    image =
models.ImageField(upload_to='images/%Y/%m/%d')

    def __str__(self):
        return self.title

    def save(self, *args, **kwargs):
        self.slug = slugify(self.title)
        super(Image, self).save(*args, **kwargs)
```

`Image` 类和 `User` 类属于多对一（一对多）的关系，它们通过外键实现对应关系，从而确定某张图片的拥有者是这个用户。

`URLField` 类，这个类继承了 `CharField` 类。它们的区别在于 `URLField` 中如果不规定最大长度，则**默认是 `200`**，即为存储 `URL` 做了专门的属性设置。

`slug` 字段，与本书前面提到的一样，用在 `URL` 显示上。

`created` 是 `Image` 实例的创建时间（保存该图片的时间），`auto_now_add=True` 表示当这个类的实例被创建时，时间值会自动创建并写入数据库；`db_index=True` 意味着用数据库的此字段作为索引。

`image` 规定了图片上传到服务器的物理存储地址。`ImageField` 类继承了 `FileField` 类，所以能够接收上传的图片，其中参数 `upload_to` 规定了所上传的图片文件的存储路径。

然后重写了父类的 `save` 方法，当通过 `Image` 实例调用此方法时，自动实现 `slug` 的生成（`slugify(self.title)`）和存储。

`Image` 类写好之后，就要迁移数据库，以生成相应的数据库表。

```
    python manage.py makemigrations
Migrations for 'image':
  image/migrations/0001_initial.py
    - Create model Image

    python manage.py migrate
Operations to perform:
  Apply all migrations: account, admin, article, auth, bl
og, contenttypes, image, sessions
Running migrations:
  Applying image.0001_initial... OK
```

用如下方式查看数据库表结构：

```
sqlite> pragma table_info(image_image);
cid         name        type        notnull     dflt_value  pk
----------  ----------  ----------  ----------  ----------  ----------
0           id          integer     1                       1
1           title       varchar(30  1                       0
2           url         varchar(20  1                       0
3           slug        varchar(50  1                       0
4           descriptio  text        1                       0
5           created     date        1                       0
6           image       varchar(10  1                       0
7           user_id     integer     1                       0
```

图片的数据模型创建完成后，就要编写表单类了。本项目的功能需求是用户提交图片网址，然后由程序将该图片下载并保存到指定的位置。

创建 `./image/forms.py` 文件，并在其中增加如下代码。

```
from urllib import request

from django import forms
from django.core.files.base import ContentFile
from slugify import slugify

from .models import Image


class ImageForm(forms.ModelForm):
    class Meta:
        model = Image
        fields = ('title', 'url', 'description')

    def clean_url(self):
        url = self.cleaned_data['url']
        valid_extensions = ['jpg', 'jpeg', 'png']
        extension = url.rsplit('.', 1)[1].lower()
        if extension not in valid_extensions:
            raise forms.ValidationError(
                'The given url does not match valid image
 extension.'
            )
        return url

    def save(self, force_insert=False, force_update=Fals
e, commit=True):
        image = super(ImageForm, self).save(commit=False)
        image_url = self.cleaned_data['url']
        image_name = '{0}.{1}'.format(
            slugify(image.title),
            image_url.rsplit('.', 1)[1].lower())
        response = request.urlopen(image_url)
        image.image.save(image_name, ContentFile(respons
e.read()), save=False)
        if commit:
            image.save()

        return image
```

函数 `clean_<fieldname>` 的作用是处理某个字段值，其中 `fieldname` 就是数据模型类中的字段名称，例如函数名称 `clean_url` 中的“url”就是 `Image` 类中的字段“url”。这个函数除 `self` 外，不传入其他参数，更不直接得到表单提交的内容，而是通过 `self.cleaned_data` 获取相应字段的值。

> 团子注：可以理解为**清洗**某个字段值。

根据 `url` 里面的扩展名判断对象是否为图片的方法比较简单，因为本书在这里的终极目标不是判别对象是否是图片，所以暂时用这个方法。**在真实项目中，建议用专门的方法判断一个对象是否为图片，而不是仅凭其扩展名。**

然后，把经过验证后的字段值返回。

> 团子注：千万不要忘了把清洗后的数据返回！

如读者所知，`ModelForm` 类中有一个 `save()` 方法，其作用就是将表单提交的数据保存到数据库。`ImageForm` 类是 `ModelForm` 类的子类，当建立了 `ImageForm` 类的实例后，该实例也拥有了 `save()` 方法，即实现实例数据的保存，这是我们在前面已经应用过的。

我们重写了 `save()` 这个方法，当通过 `ImageForm` 类实例调用 `save()` 方法时，调用的是我们在这里重写的这个方法。

> 团子注：之所以重写方法，是因为原始的 `ModelForm` 的 `save` 方法不能满足我们的需求，我们需要额外扩展它，让它可以保存图片。

一开始依然执行父类 `ModelForm` 的 `save()` 方法，`commit=False` 意味着实例虽然被建立了，但并没有保存数据。

> 团子注：也就是说，实例在内存中，并没有持久化到数据库。

导入进来的 `request` 不是视图函数中的参数 `request`，而是 `Python` 标准库 `Urllib` 中的一部分，是一个很好的爬虫工

具。`request.urlopen(image_url)` 的作用是以 `GET` 方式访问该图片地址，就是模拟用户把网址输入到浏览器的地址栏中，之后在浏览器中返回相应的内容，这里同样返回一个 `Request` 对象。通过该对象得到所访问 `URL` 的数据（图片的 `ASCII` 字符串），后面用 `response.read()` 得到此数据。

`image.image.save` 将返回的结果保存到本地，并按照约定的名称给该图片文件命名。`ContentFile` 类是 `Django` 中 `File` 类的子类，它接收字符串为参数。这里直接使用了 `image` 属性或字段（第二个 image）的 `save()` 方法。在 `Image` 类中有 `image=models.ImageField(upload_to='images/%Y/%m/%d')`，属性是 `ImageField` 类型。`ImageField` 继承了 `FileField`，而 `FileField` 具有一个 `save()` 方法，其官方文档中显示的样式是 `FieldFile.save(name, content, save=True)`。因此，本句采用这种方式保存图片。

> 团子注：2.2.7 文档的 1031 页说：`car.photo.save('myphoto.jpg', content, save=False);car.save()` 等价于 `car.photo.save('myphoto.jpg', content, save=True)`。
> 所以这里实际上是延缓了数据的保存，然后让最后一句根据 `commit` 的取值判断是否进行保存。

收集和管理图片

要收集网上的图片，就必须提交图片的地址，所以要有视图函数接收所提交的图片 `URL` 等信息。此外，仅保存了还不够，用户还要浏览和管理，比如删除图片。所以，要在 `./image/views.py` 文件中编写两个视图函数，专门用来处理上述问题，代码如下。

```
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
from django.shortcuts import render
from django.views.decorators.csrf import csrf_exempt
from django.views.decorators.http import require_POST

from .forms import ImageForm
from .models import Image
```

```
login_required(login_url='/account/login/')
csrf_exempt
require_POST
def upload_image(request):
    form = ImageForm(data=request.POST)
    if form.is_valid():
        try:
            new_item = form.save(commit=False)
            new_item.user = request.user
            new_item.save()
            return JsonResponse({'status': '1'})
        except:
            return JsonResponse({'status': '0'})
```

上述视图函数的作用是处理前端表单提交的图片 `URL` 及其相关信息。

首先，利用提交的数据建立了表单类的实例，然后用该实例的 `save()` 方法，但这里没有保存数据（`commit=False`）。当执行 `new_item.save()` 时，该图片被保存到本地指定目录中。

最后返回的是 `JSON` 数据，这与前面返回字符串稍有不同，当然也可以返回字符串，**只不过这里给读者展示另外一种返回的数据类型。**

> 团子注：之前都是 `return HttpResponse('1')`

`upload_image()` 是接收并处理前端提交的数据的函数，下面还要继续编写展示图片的函数 `list_images()`，代码如下。

```
@login_required(login_url='/account/login/')
def list_images(request):
    images = Image.objects.filter(user=request.user)
    return render(request, 'image/list_images.html', {'im
ages': images})
```

上面两个视图函数编写完之后，为了能够让本应用被项目接收，还要在 `./mysite/urls.py` 文件中增加应用的 `URL` 配置，代码如下。

```
urlpatterns = [
    path('image/', include('image.urls', namespace='imag
e')),
]
```

然后在本应用中创建 `./image/urls.py` 文件，进行 `image` 应用相关的 `URL` 配置，代码如下。

```
from django.urls import path

from . import views

app_name = 'image'

urlpatterns = [
    path('list-images/', views.list_images, name='list_im
ages'),
    path('upload-image/', views.upload_image, name='uploa
d_image'),
]
```

下面做一些零碎的准备工作。

前面对文章管理的部分已经建立了一个后台，此处依然采用类似的方式，用户在后台管理中能够添加、删除图片。

依然使用原来的后台导航和侧边栏功能的模板文件。

将图片列表的功能添加到后台导航模板文件中，即修改 `./templates/article/header.html` 文件，添加如下代码。

```
<ul class="nav navbar-nav" role="tablist">
  <li><a href="{% url 'article:article_column' %}">文章管
理</a></li>
  <!-- 新增 -->
  <li><a href="{% url 'image:list_images' %}">图片管理</a>
</li>
```

```
</ul>
```

> 团子注：这个是放到导航栏的东西。

然后修改 `./templates/article/leftslider.html` 文件，将前面已有的功能命名为“文章管理”，现在新增加的部分命名为“图片管理”。以下代码放在原有代码的后面。

```
<div class="bg-info">
  <div class="text-center" style="margin-top: 5px;">
    <p><a href="{% url 'article:article_column' %}">栏目管
理</a></p>
    <p><a href="{% url 'article:article_post' %}">发布文章
</a></p>
    <p><a href="{% url 'article:article_list' %}">文章列
表</a></p>
    <p><a href="{% url 'article:article_tag' %}">文章标签
</a></p>
  </div>
  <!-- 新增 -->
  <hr>
  <div class="text-center" style="margin-top: 5px;">
    <p>
      <h4>图片管理</h4>
    </p>
    <p><a href="{% url 'image:list_images' %}">图片管理</a>
</p>
  </div>
</div>
```

准备工作完毕，开始创建 `./templates/image` 目录，将本应用的模板文件都放到这个目录中。

在此目录中创建模板文件 `list_images.html`，其代码如下。

```
{% extends "article/base.html" %}
{% load static %}
```

```
{% block title %}
images
{% endblock title %}

{% block content %}
<div>
  <button type="button" class="btn btn-primary btn-lg btn
-block" onclick="addImage()">
    添加图片
  </button>
  <div style="margin-top: 10px;"></div>
  <table class="table table-hover">
    <tr>
      <td>序号</td>
      <td>标题</td>
      <td>图片</td>
      <td>操作</td>
    </tr>
    {% for image in images %}
    <tr>
      <td>{{ forloop.counter }}</td>
      <td>{{ image.title }}</td>
      <td>{{ image.image }}</td>
      <td>
        <a href="javascript:;" onclick="del_image(this,
 {{ image.id }})" name="delete">
          <span class="glyphicon glyphicon-trash"
style="margin-left: 20px;"></span>
        </a>
      </td>
    </tr>
    {% empty %}
    <p>还没有图片，请点击上面的按钮添加图片</p>
    {% endfor %}
  </table>
</div>
{% endblock content %}
```

运行 `Django` 服务，在浏览器中访问http://localhost:8000/image/list-images/（用户如果没有登录，请先登录）

接下来，就应该完成单击“添加图片”按钮实现提交图片地址及其标题和描述的功能。还是使用 `JavaScript` 的方式完成这个功能，读者在刚刚创建的模板文件（`./templates/image/list_images.html`）中已经看到了，在 `<button>` 中使用了 `onclick="addImage()"`。继续在这个模板中增加如下代码。

```
<script src="{% static 'js/jquery.js' %}"></script>
<script src="{% static 'js/layer.js' %}"></script>
<script>
  function addImage() {
    var index = layer.open({
      type: 1,
      skin: 'layui-layer-demo',
      closeBtn: 0,
      shift: 2,
      shadeClose: true,
      title: 'Add Image',
      area: ['600px', '440px'],
      content: '<div style="padding: 20px">\
                  <p>请新增扩展名是 .jpg 或 .png 的网上照片地
址</p>\
                  <form>\
                    <div class="form-group">\
                      <label for="phototitle" class="col-
sm-2 control-label">标题</label>\
                      <div class="col-sm-10">\
                        <input id="phototitle" type="tex
t" class="form-control" style="margin-bottom: 5px;">\
                      </div>\
                    </div>\
                    <div class="form-group">\
                      <label for="photourl" class="col-sm
-2 control-label">地址</label>\
```

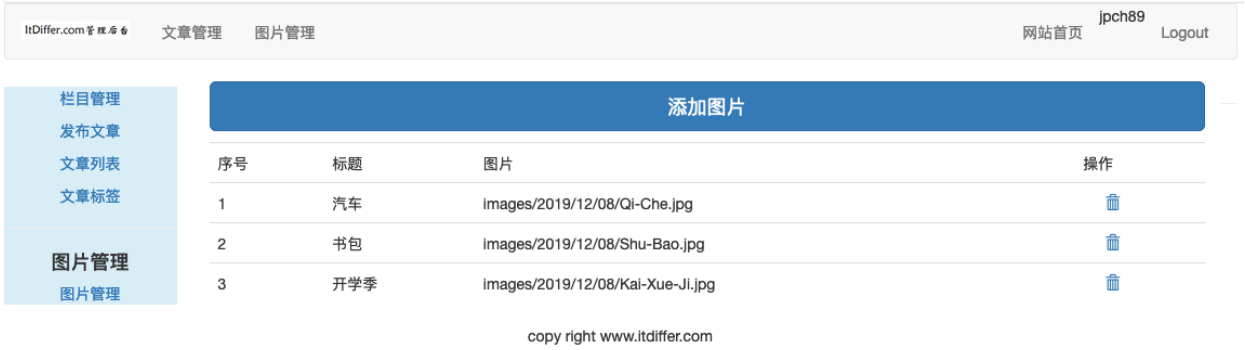
```
                        <div class="col-sm-10">\
                          <input id="photourl" style="margi
n-bottom: 5px;" type="text" class="form-control">\
                        </div>\
                      </div>\
                      <div class="form-group">\
                        <label for="description" class="col
-sm-2 control-label">描述</label>\
                        <div class="col-sm-10">\
                          <textarea class="form-control" st
yle="margin-bottom: 5px;" row="2" id="photodescription">
</textarea>\
                        </div>\
                      </div>\
                      <div class="form-group">\
                        <div class="col-sm-offset-2 col-sm-
10">\
                          <input id="newphoto" type="butto
n" class="btn btn-default" value="Add It">\
                        </div>\
                      </div>\
                    </form>\
                  </div>',
        success: function () {
          $('#newphoto').on('click', function () {
            var title = $('#phototitle').val()
            var url = $('#photourl').val()
            var description = $('#photodescription').val()
            var photo = {'title': title, 'url': url, 'descr
iption': description}
            $.ajax({
              url: '{% url "image:upload_image" %}',
              type: 'POST',
              data: photo,
              success: function (e) {
                var status = e['status']
                if (status === '1') {
                  layer.close(index)
                  window.location.reload()
                } else {
                  layer.msg('图片无法获取，请更换图片')
```

```
                }
              }
            })
          })
        }
      })
    }
  </script>
```

在上面的 `JavaScript` 代码中，再次使用了弹出提示框插件。并且要注意，这些代码要写在 `{% endblock %}` 以内。

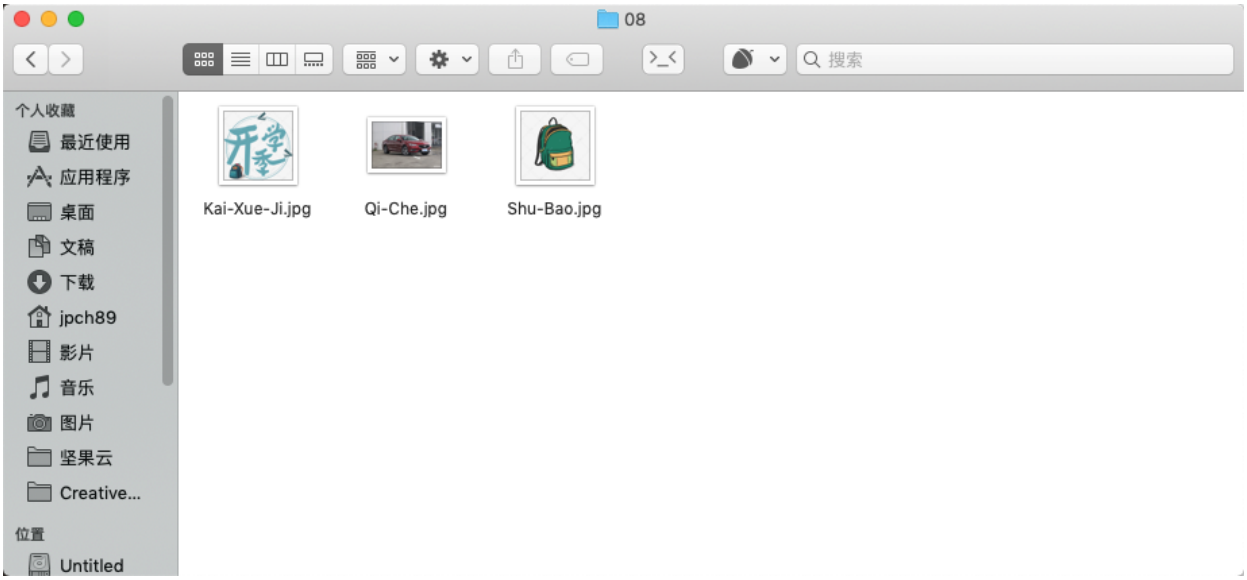
再次刷新页面，单击“添加图片”按钮，在弹出的提示框中输入有关内容。

在网上找一些图片，通过此方式提交给本应用（注意，图片必须是.jpg 或.png 格式）。



当然，这里在显示上还有一些问题，后面需要继续完善，读者重点关注是否把图片提交并保存了。

查看数据模型类 `Image` 中的字段 `image=models.ImageField(upload_to='images/%Y/%m/%d')` 所设置的物理存储位置，是否已经有了刚才上传的图片。



数据库也要检查一下，看看是否按照预设把数据都保存了下来，使用的 `SQL` 命令如下所示。

```
sqlite> select * from image_image;
1|汽车|http://img4.bitautoimg.com/autoalbum/files/20190311/777/20190311133057305721036
0_6542935_16.jpg|Qi-Che|汽车图片|2019-12-08|images/2019/12/08/Qi-Che.jpg|5
2|书包|https://bpic.588ku.com/element_origin_min_pic/19/03/07/919c208cf614cdaf1850c8997
7f3e836.jpg|Shu-Bao|一个书包。。。|2019-12-08|images/2019/12/08/Shu-Bao.jpg|5
3|开学季|https://bpic.588ku.com/art_water_pic/19/03/18/31c840c88985d8240782629d73250892.
jpg|Kai-Xue-Ji|开学季|2019-12-08|images/2019/12/08/Kai-Xue-Ji.jpg|5
```

完善图片管理功能

图片已经实现了上传，如果有图片不喜欢怎么办？删除！所以，列表中的删除功能是必须有的。利用已经学习过的知识就能够完成此功能，编辑 `./image/views.py` 文件，增加如下代码。

```
@login_required(login_url='/account/login/')
@require_POST
@csrf_exempt
def del_image(request):
    image_id = request.POST['image_id']
    try:
        image = Image.objects.get(id=image_id)
        image.delete()
        return JsonResponse({'status': '1'})
    except:
        return JsonResponse({'status': '2'})
```

这个视图函数与前文删除文章的视图函数相差无几。这里再次提醒读者，可以思考并尝试写一个更具有通用性的删除函数（或者类），以实现所有类似的删除功能。

之后不要忘记在 `./image/urls.py` 文件中配置 `URL`，在列表中增加下列值。

```
path('del-image/', views.del_image, name='del_image'),
```

在 `./templates/image/list_images.html` 文件中编写早已经为删除图标设定的 `JavaScript` 函数 `del_image()`，代码如下。

```
  function del_image(the, image_id) {
    var image_title =
$(the).parents('tr').children('td').eq(1).text()
    layer.open({
      type: 1,
      skin: 'layui-layer-rim',
      area: ['400px', '200px'],
```

```
        title: '删除图片',
        content: '<div class="text-center" style="margin-to
p: 20px;">\
                    <p>是否确定删除《' + image_title + '》</p>
\
                </div>',
        btn: ['确定', '取消'],
        yes: function () {
          $.ajax({
            url: '{% url "image:del_image" %}',
            type: 'POST',
            data: {image_id},
            success: function (e) {
              var status = e['status']
              if (status === '1') {
                parent.location.reload()
                layer.msg('has been deleted.')
              } else {
                layer.msg('删除失败')
              }
            }
          })
        }
      })
    }
```

这样就实现了图片的删除功能（请读者自行测试）。

请读者仔细观察，这种删除并不彻底，因为该图片并没有从磁盘上删除，只是从本项目的数据库中删除了。

> 团子注：那你也没说要怎么彻底删除啊？

虽然完成了删除功能，但还有需要改进之处，就是在页面中不显示图片，仅列出图片的标题和地址。因此，要修改 `./templates/image/list_images.html` 文件的代码，将下面的代码：

```
<td>{{ image.image }}</td>
```

改成：（`url of image` 是占位符）

```
<td><img src="url of image" alt=""></td>
```

然后在 `./mysite/settings.py` 文件中增加如下设置。

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

`MEDIA_ROOT` 声明了**媒体文件相对项目根目录的保存路径**。前面我们已经将图片保存到了相对项目根目录的 `images` 中，即 `./images`。考虑到以后还可能上传视频等媒体文件，所以还是设置一个统一的目录较好，于是这里设置了一个名为 `media` 的目录，图片将被保存到 `./media/images` 里面。

> 团子注：上面一段话我略有修改。

`MEDIA_URL` 用于设置 `URL` 映射中的路径，下面就编辑 `./mysite/urls.py` 文件，引入 `settings` 和 `static`，代码如下。

```
from django.conf import settings
from django.conf.urls.static import static
```

在本文件的 `urlpatterns` 之后增加如下代码。

```
urlpatterns += static(settings.MEDIA_URL, document_root=s
ettings.MEDIA_ROOT)
```

这样**就为每个上传的静态图片配置了 `URL` 路径。**做好上面的配置之后，再将上面提到的 `./templates/image/list_images.html` 文件中将要替代的部分用下面的代码替代。

```
<td><img src="{{ image.image.url }}" width="100px"
height="100px" alt=""></td>
```

确保 `Django` 服务运行，访问http://localhost:8000/image/list-images/，原来提交的图片肯定不能显示了，**因为按照上面的设置，图片的访问地址已经发生了变化，需要重新提交**，并且在提交之后，注意观察文件是否被保存到了 `./media/images` 里面。

这样，就实现了对上传图片的可视化管理。

这里展示的图片虽然已经进行了长宽的设置，但仅仅是在显示的时候对图片尺寸进行了更改，实际上仍然要读取原图。说到这里，读者可能也想到了，能不能使用缩略图呢？当然可以！

`sorl-thumbnail` 就是很好的 `Django` 缩略图应用，`github.com` 上的源码地址是 https://github.com/jazzband/sorl-thumbnail。下面简要演示它在本应用的使用方法，读者可以通过阅读官方文档内容，了解更多的使用方式。这里也再次显示了 `Django` 快速开发的特点，“不需要重复造轮子”。

首先要做的总是安装，使用 `pip` 可以快速安装这个应用。

```
pip install sorl-thumbnail
```

> 团子注：然后 `pip freeze requirements.txt`

然后在 `./mysite/settings.py` 的 `INSTALLED_APPS` 中注册名为 `sorl.thumbnail` 的应用。

```
INSTALLED_APPS = [
    'sorl.thumbnail',
]
```

然后修改 `./templates/image/list_images.html` 文件。

在文件中显示缩略图的前面引入 `thumbnail`（通常放在文件的顶部，但不要放在 `{% extends"article/base.html" %}` 之前）。

```
{% load thumbnail %}
```

再将原来显示图片的代码替换为下面的代码。

```
<!-- <td><img src="{{ image.image.url }}" width="100px" h
eight="100px" alt=""></td> -->
{% thumbnail image.image "100x100" crop="center" as im %}
<td><img src="{{ im.url }}" width="{{ im.width }}"
height="{{ im.height }}"></td>
{% endthumbnail %}
```

重新启动 `Django` 服务，然后刷新页面，这时候查看到的图片就是缩略图了。

> 团子注：注意需要先迁移数据库，书上没讲。
> 另外，里面的 `"100x100"` 是 `x` 不是 `*`，如果写成 `*` 根本显示不出来。

缩略图被保存在了 `./media/cache` 目录中。

读者还可以继续创建一个分页功能，因为这个功能和前面文章管理中的分页功能一致，所以笔者就省略了，特别建议读者能够补上。

本节中我们收集了网络图片，并对保存到项目本地的图片进行了管理，还使用了缩略图。因为前面已经奠定了开发基础，所以本节中并没有遇到难点，但并不意味着开发过程会一帆风顺，如果在某些细节上不注意也会有 `Bug` 出现。随着功能越来越多，重复代码也日渐增加，所以读者要充分使用在 `Python` 中已经学会的知识，将重复的功能进行抽象，写成能够被公用的类（或函数），虽然笔者在本书中没有这样做。

知识点

1.模型：ImageField

`ImageField` 专门用来声明某字段是图片类型，其类的完全调用形式如下。

```
class ImageField(upload_to=None, height_field=None, width_field=None, max_length=100,
**options)
```

在字段被设置为 `ImageField` 类型之后，提交图片，`Django` 会自动检测所提交的是否是图片类型，不需要用户写检验方法。

在创建 `ImageField` 实例时，可以指明 `upload_to` 参数，说明图片保存在服务器上的位置，通常还在 `settings.py` 中配置 `MEDIA_ROOT` 和 `MEDIA_URL` 的值，例如在本书的项目中，代码如下。

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
MEDIA_URL = '/media/'
```

其实，`ImageField` 继承了 `FileField`，如果理解了 `ImageField`，后面再遇到 `FileField` 也好解决了。阅读下面这段代码。

```
>>> from image.models import Image
>>> image = Image.objects.all()
>>> image
```

```
<QuerySet [<Image: 发放防雾霾口罩>, <Image: 高楼看雾霾>, <Image: 戴口罩晨练>, <Image:
Truing>, <Image: Hertz>, <Image: cls>, <Image: zhiling>]>
>>> hertz = image[4]    #选一个记录
>>> hertz
<Image: Hertz>
>>> hertz.user    #该记录的用户名
<User: lulaoshi>
>>> hertz.image    #该记录中的图片
<ImageFieldFile: images/2017/01/24/Hertz.jpg>
>>> hertz.image.name    #图片文件名称
'images/2017/01/24/Hertz.jpg'
>>> hertz.image.path    #图片文件保存的物理地址
'/home/qiwsir/DjangoPracticeProject/mysite/media/images/2017/01/24/Hertz.jpg'
>>> hertz.image.url    #图片的 URL（如果要在浏览器中访问，前面加上主域名和端口）
'/media/images/2017/01/24/Hertz.jpg'
```

当然，对这个文件也可以继续按照文件对象的方式进行操作，比如：

```
>>> import os
>>> from django.conf import settings
>>> old_path = hertz.image.path
>>> hertz.image.name = "images/2017/01/24/Hertz_new.jpg"
>>> new_path = settings.MEDIA_ROOT + hertz.image.name
>>> os.rename(old_path, new_path)
>>> hertz.save()
```

只有在执行了实例的 `save()` 方法后，保存在服务器物理空间中的文件名才会改变，否则只是数据库表记录中的文件名被修改了，物理空间中的文件名并没有变化，在调用时会找不到对应的文件。

不要忘记，如果要上传图片，还要依赖一个名为 `Pillow` 的库。

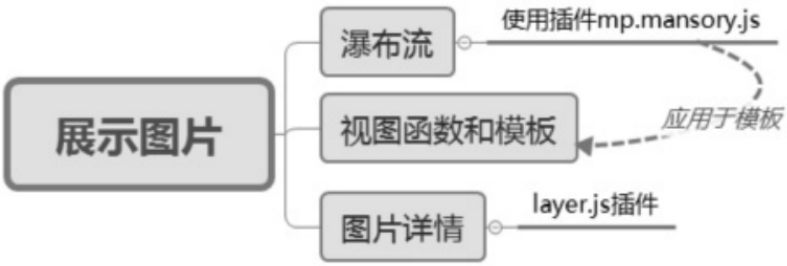
2.文档导读

（1）ImageField source，https://docs.djangoproject.com/en/2.1/ref/models/fields/#imagefield。

（2）Thumbnails for Django， https://github.com/mariocesar/sorl-thumbnail。

（3）Discussion about Thumbnail Contribution for Django，https://code.djangoproject.com/wiki/ThumbNails。

5.2 展示图片



瀑布流方式展示图片

“瀑布流”是一种网站页面布局方式，在视觉上表现为参差不齐的多栏布局，随着页面滚动条向下滚动，这种布局还会不断加载数据块并附加至当前尾部。最早采用此布局的网站是 `Pinterest` ，之后逐渐在我国流行起来。

当然，网页展示是多变的，在这里笔者使用了瀑布流的方式展示图片，并不意味着在日后还流行这种用法，而且“瀑布流”本身的布局样式也有变化，比如图片是参差不齐的还是整齐排列的等。这里仅仅是一个样例罢了。

此处我们使用来自 https://github.com/jmlp-131092/mp-mansory.js 的插件，实现以瀑布流方式展示用户所收集的各种图片。随着需求和技术的变化，已经发展出更多的瀑布流布局方式，读者如果不喜欢此处选择的插件，可以使用其他插件，使用方法都是类似的。

首先下载上述插件，将其中的 `mp.mansory.min.js` 文件复制到 `./static/js` 目录中，然后在 `./static/css` 目录中创建 `mansory-style.css` 文件，并输入如下代码（根据原插件的 `style.css` 代码进行了适当修改）。

```
#my-gallery-container {
    background-color: #ededed;
}

.falls_item {
    border: #c6c6c6 2px solid;
    width: 100%;
    background-color: white;
    min-height: 100px;
    padding: 5px;
    margin: 10px 0px;
    box-shadow: rgba(0, 0, 0, 0.5) 0px 2px 2px;
    border-radius: 4px;,

}

.falls_item:hover {
    opacity: 0.5;
}

.falls_item.h150{
    width: 100%;
    min-height: 150px;
}

.falls_item.h200{
    width: 100%;
    min-height: 200px;

}
```

```
.falls_item.h250{
    width: 100%;
    min-height: 200px;
}
.falls_padding{
    padding: 10px 5px;
}
.falls_item img {
    max-width: 100%;
}
```

之所以用上述代码而不使用原插件中的 `style.css` 文件，是因为原文件中的样式表名称与本项目中已经使用的 `bootstrap.css` 有重复，如果使用就会覆盖原有名称。为此，在这里重写了该文件，并重新命名。读者可以根据自己的实际情况酌情处理。

首先实现瀑布流方式的展示，也是从编写视图函数开始的。

在 `./image/views.py` 文件中增加如下视图函数的代码。

```
def falls_images(request):
    images = Image.objects.all()
    return render(request, 'image/falls_images.html', {'images': images})
```

然后在 `./image/urls.py` 文件中增加一个 `URL` 配置，代码如下。

```
path('images/', views.falls_images, name='falls_images'),
```

在本项目的导航栏中已经有“文章”栏目，现在增加“美图”栏目。编辑 `./templates/header.html` 文件中导航部分的代码。

```
<li>
  <a href="{% url 'image:falls_images' %}">美图</a>
</li>
```

单击“美图”即可看到以瀑布流方式展示的所有用户收集的图片。

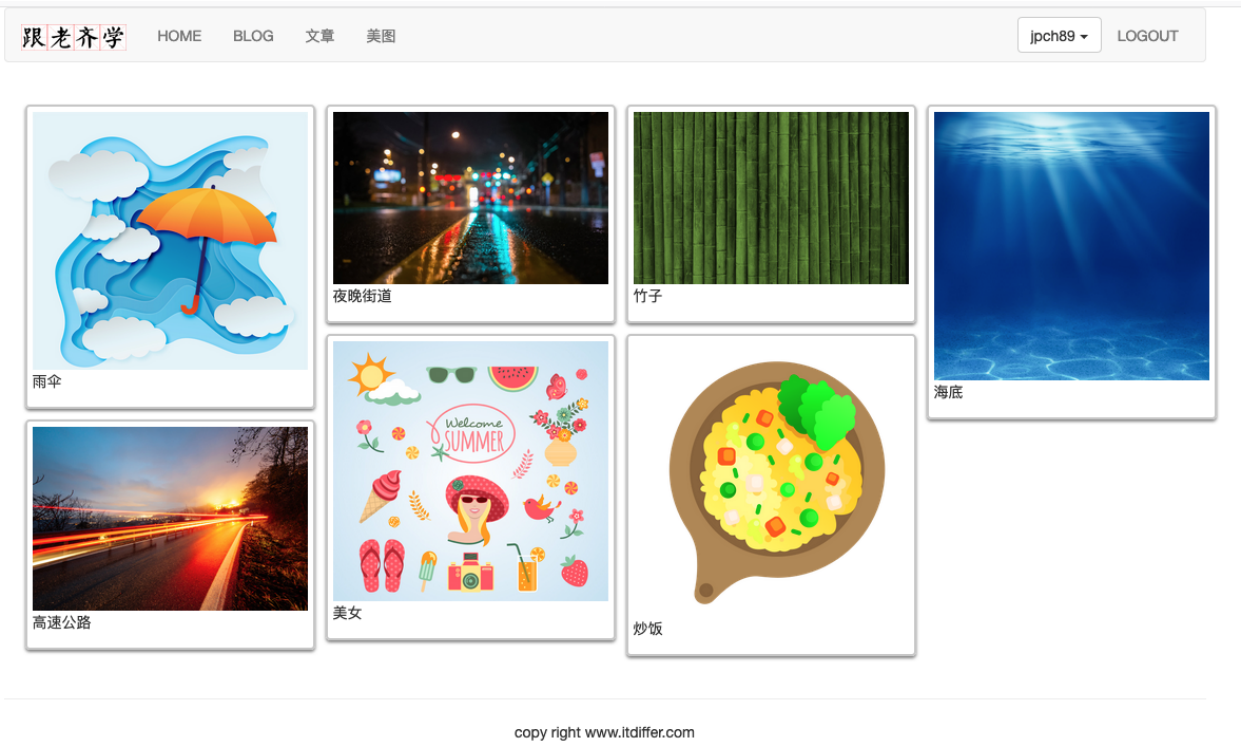
在 `./templates/image` 目录中创建 `falls_images.html` 文件，这就是瀑布流方式展示图片的界面，其代码如下。

```
{% extends "base.html" %}

{% load static %}

{% block title %}
Images
{% endblock title %}

{% block content %}
<div class="container">
  <link rel="stylesheet" href="{% static 'css/mansory-sty
le.css' %}">
  <div id="my-gallery-container">
    {% for image in images %}
      <div class="falls_item h200" data-order="{{ image.i
d }}">
        <img src="{{ image.image.url }}" alt="">
        <p>{{ image.title }}</p>
      </div>
    {% endfor %}
  </div>
</div>
<script src="{% static 'js/jquery.js' %}"></script>
<script src="{% static 'js/mp.mansory.min.js' %}"></scrip
t>
<script>
  jQuery(document).ready(function ($) {
    $('#my-gallery-container').mpmansory({
      childrenClass: 'falls_item',  // default is a div
      columnClasses: 'falls_padding',  // add classes to
 items
      breakpoints: {
        lg: 3,
        md: 4,
        sm: 6,
        xs: 12
```

```
        },
        distributeBy: {
          order: false,
          height: false,
          attr: 'data-order',
          attrOrder: 'desc',
        },  // default distribute by order, options => orde
    r: true/false, height: true/false, attr => 'data-order',
     attrOrder => 'asc'/'desc'
      })
    })
</script>
{% endblock content %}
```

这个界面的写法参考了https://github.com/jmlp-131092/mp-mansory.js提供的样例，读者可以根据自己对此插件的理解，编写所需的瀑布流样式。

完成上述代码之后，检查Django服务是否已经启动，访问http://localhost:8000/image/images/，如果已经收集了若干张图片，就可以看到下面所示的效果。



这就是瀑布流方式展示的图片。

再进一步，还可以查看每张图片的详细信息。

前面以瀑布流方式列出了所有图片，如果要查看每张图片的详细信息，一种方法是用本书前文中所讲述的类似查看文章详细内容的方法，这种方法此处不再重复，请读者自己完成。下面我们使用另外一种纯粹前端的方法来查看图片的完整信息。

在 `./templates/image/falls_images.html` 文件中找到下面这段代码。

```
<div class="falls_item h200" data-order="{{ image.id }}">
  <img src="{{ image.image.url }}" alt=""> <!-- 1 -->
  <p>{{ image.title }}</p>
</div>
```

用下面的代码替换语句 `1`。

```
<a href="javascript:void(0);" onclick="displayImage('{{ i
mage.user }}', '{{ image.title }}', '{{ image.image.url
 }}', '{{ image.description }}', '{{ request.get_host
 }}')">
  <img src="{{ image.image.url }}">
</a>
```

这里主要是在 `<a>` 标签中增加了一个 `onclick` 方法，即 `displayImage()` 函数，并且向这个函数传递了一些值，请读者注意传值的方式，没有直接使用诸如 `{{ image.user }}` 的方式，而是使用了 `'{{image.user}}'` 的方式（外层加一对单引号），其目的在于向 `dispalyImage()` 传递字符串。

然后在本文件中编写 `JavaScritp` 函数 `displayImage()`。

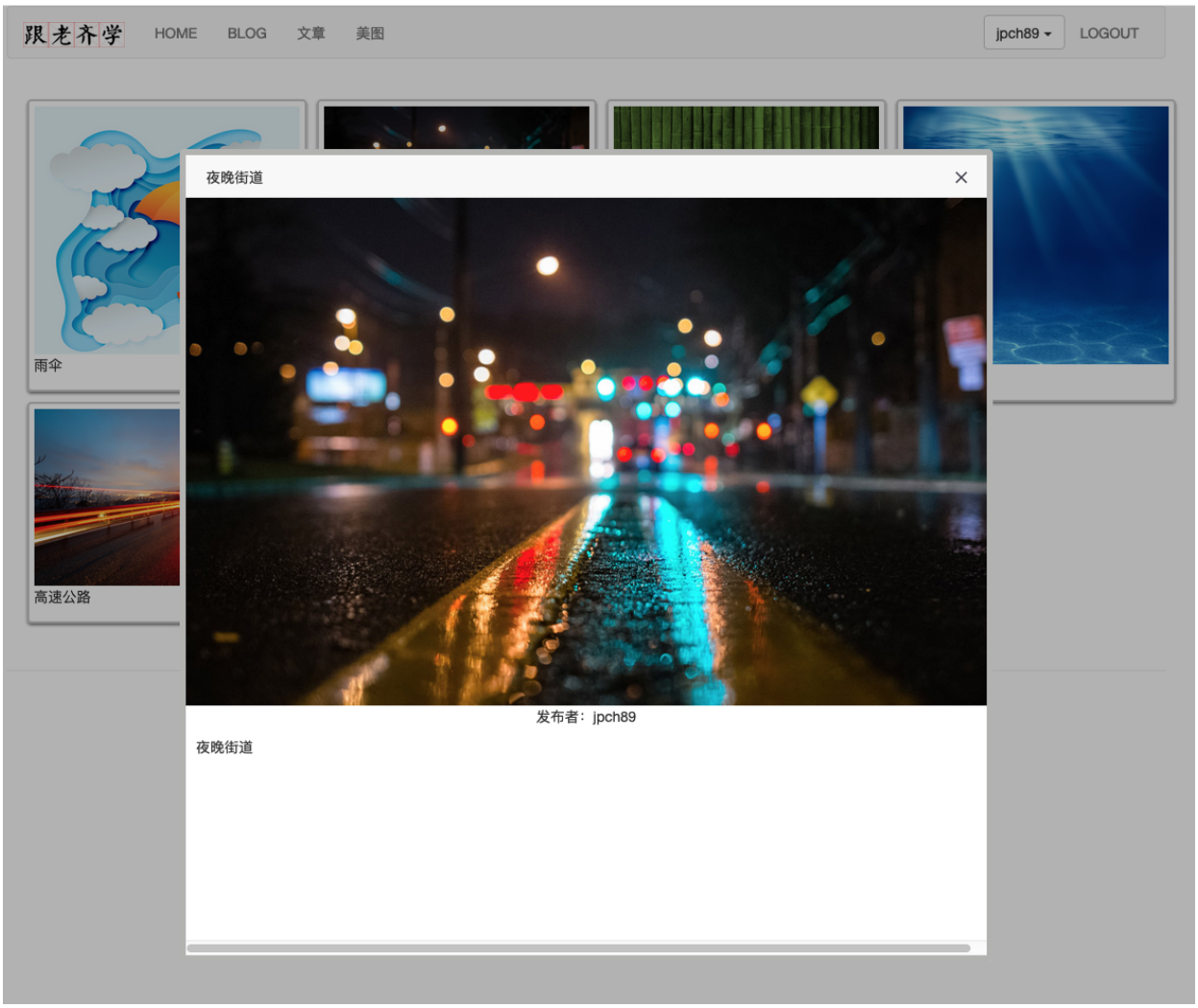
因为要实现单击某图片后在弹出提示框展示该图片详细信息的功能，所以再次使用熟悉的弹出提示框插件，代码如下。

```
<script src="{% static 'js/layer.js' %}"></script>
```

下面是 `dispalyImage()` 的代码，也放在本文件的 `<script></script>` 内。

```
  function displayImage(user, title, url, description, ho
st) {
    layer.open({
      type: 1,
      title: title,
      skin: 'layui-layer-rim',  // 加上边框
      area: ['800px', '800px'],  // 宽高
      content: '<div class="text-center">\
                  <img src="http://' + host + url + '"><p
>发布者: ' + user + '</p></div><div style="margin-left: 10p
x;">' + description + '</div>',
    })
  }
```

检查Django是否运行了，然后访问http://localhost:8000/image/images/，单击某图片，即可查看该图片的详细信息。



要想在图片展示中有非常友好的操作界面，就必须使用 `JavaScript` 组件或者框架。就目前流行的各种 `JavaScript` 组件和框架而言，可谓让人眼花缭乱，读者可以根据自己的喜好和擅长选择使用。

虽然我们处在所谓的“读图年代”，甚至是“读视频”，**但要想成为这个时代中的佼佼者，还必须读书。**

笔者借用宋真宗赵恒写下的流传千古的《励学篇》献给读者。
富家不用买良田，书中自有千钟粟。
安居不用架高堂，书中自有黄金屋。
出门莫恨无人随，书中车马多如簇。
娶妻莫恨无良媒，书中自有颜如玉。
男儿欲遂平生志，六经勤向窗前读。

知识点
1.关于JavaScript

做Web开发，如果不使用JavaScript，一些很酷的效果就难以实现了。首先要非常明确地说明，JavaScript和Java在名字上有点像，在语法上也有相似的地方，但也仅仅是“看上去像”，不要认为JavaScript是从Java 演化来的。事实上，JavaScript语言在设计方面主要受到了Self和Scheme两种语言的影响。

JavaScript诞生于网景公司，虽然现在很多人已经不知道历史上的这家公司了，但是JavaScript 广为传播，所以使用它的时候请在心里默默感谢网景公司和发明者布兰登·艾克（当时就职于网景公司）。

JavaScript被大量用在Web开发中，它不需要进行编译，是一种解释性的脚本语言，可以实现与HTML的交互，并且直接嵌入HTML页面中，可以单独成为一个单元（比如单独的.js文件），且实现了结构和行为的分离。一般情况下，可以用它完成如下工作（源自维基百科）。

- 在HTML页面加入动态文本。
- 浏览器事件做出响应。
- 写HTML元素。
- 数据被提交到服务器之前验证数据。
- 检测访客的浏览器信息。
- 制作cookies，包括创建和修改等。

为了使开发更加便利，聪明的程序员们制作了各种“轮子”，并且开源。因此，对于使用者来说，不仅要了解JavaScript，还要知道各种“轮子”，jQuery就是其中一个。

jQuery是一个JavaScript库，它大大地简化了JavaScript编程，故用户很容易学会。jQuery官方网站是http://jquery.com/。

除jQuery外，还有很多库，可谓多如牛毛，读者要实现某个功能，最好先到网上搜索一下，看看是否有“轮子”可以拿过来使用。这几年有一个名为 `React` 的库（源码在https://github.com/facebook/react），宣称“简单、组件化、学一次各处可用”。不仅如此，它还衍生出了 `React Native`，可用来编写手机上的App代码。

在浏览器端，JavaScript“带领”它的众多库，横扫天下，但它并没有满足，凭借 `Node.js` 进军了服务器端，真的是什么也挡不住了。

`Node.js` 就是运行在服务器端的 `JavaScript`，是 `Chrome JavaScript` 运行时建立的一个平台，还是一个事件驱动I/O服务器端JavaScript环境，基于Google的V8引擎。V8引擎执行JavaScript的速度非常快，性能非常好。运行Node.js的服务器能支持数万个并发连接。读者如果觉得Node.js也值得学，可以参考官网https://nodejs.org/en/。

千里之行始于足下，不管学习哪个JavaScript框架，都要从JavaScript入手。

2.文档导读

（1）JavaScript，https://docs.djangoproject.com/en/2.1/internals/contributing/writing-code/javascript。

（2）django-jquery，https://pypi.python.org/pypi/django-jquery。

（3）Serializing Django objects，https://docs.djangoproject.com/en/2.1/topics/serialization。

---

6. 中场休整

进行到这里，如果读者阅读本书并且将代码都敲过，那么就已经达到了Django开发的入门级水平。前面几章所讲解的技能，对于Django来讲仅仅是一小部分。读者若能够基本理解Django的常规工作方式，通常的项目就能够上手开发了，**并且每节都有“知识点”和“文档导读”，如果读者对这些内容也认真阅读了，而且对照项目反复实践，那么此时已经进阶到较高水平了。**

针对本章的“休整”，请读者做如下工作。

（1）找一个安静的房间，闭上眼睛，把前面学习的功能和使用到的技能尽可能回忆一番。
（2）请读者将前面随着本书已经辛辛苦苦做好的项目全部删除，一点痕迹也不要留下，留下的都是后患。
（3）一切都从头再来。完全独立地完成前几章中所创建的项目和应用，遇到问题时，使用搜索引擎来解决。记住，这是在创建你自己的项目，不必亦步亦趋地按照本书步骤做，所以你的想法是最重要的。

若读者想做一名真正的程序员，那么就请按照第（1）～（3）条执行。

按照上述步骤进行的中场休整，可能不是10分钟或者15分钟，不管多长时间，中场休整之后再进入“下半场”，编程思路不仅会豁然开朗，而且技能会得到飞速提升。

`2019.12.9`