12-评论

1、创建评论应用

相对来说，评论其实是另外一个比较独立的功能。Django 提倡，如果功能相对比较独立的话，最好是创建一个应用（**相当于 Flask 里面的蓝本**），把相应的功能代码写到这个应用里。我们的第一个应用叫 blog，它里面放了展示博客文章列表和细节等相关功能的代码。而这里我们再创建一个应用，名为 comments，这里面将存放和评论功能相关的代码。首先激活虚拟环境，然后输入如下命令创建一个新的应用：

```
python manage.py startapp comments
```

我们可以看到生成的 comments 应用目录结构和 blog 应用的目录是类似的。创建新的应用后一定要记得在 settings.py 里注册这个应用（相当于在工厂函数中注册蓝本），Django 才知道这是一个应用。

【blogproject/settings.py】

```
...
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
    'comments', # 注册新创建的 comments 应用
]
...
```

2、设计评论的数据库模型

用户评论的数据必须被存储到数据库里，以便其他用户访问时 Django 能从数据库取回这些数据然后展示给访问的用户，因此我们需要为评论设计数据库模型，这和设计文章、分类、标签的数据库模型是一样的。我们的评论模型设计如下（评论模型的代码写在 comment\models.py 里）：

【comments/models.py】

```
from django.db import models
```

```
class Comment(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField(max_length=255)
    url = models.URLField(blank=True)
    text = models.TextField()
    created_time = models.DateTimeField(auto_now_add=True)

    post = models.ForeignKey('blog.Post')

    def __str__(self):
        return self.text[:20]
```

这里我们会保存评论用户的 name（名字）、email（邮箱）、url（个人网站），用户发表的内容将存放在 text 字段里，created_time 记录评论时间。最后，这个评论是关联到某篇文章（Post）的，由于一个评论只能属于一篇文章，一篇文章可以有多个评论，是一对多的关系，因此这里我们使用了 ForeignKey。关于 ForeignKey 我们前面已有介绍，这里不再赘述。

同时注意我们为 `DateTimeField` 传递了一个 `auto_now_add=True` 的参数值。`auto_now_add` 的作用是，当评论数据保存到数据库时，自动把 `created_time` 的值指定为当前时间。`created_time` 记录用户发表评论的时间，我们肯定不希望用户在发表评论时还得自己手动填写评论发表时间，这个时间应该自动生成。

创建了数据库模型就要迁移数据库，迁移数据库的命令也在前面讲过。在虚拟环境下分别运行下面两条命令：

```
python manage.py makemigrations
python manage.py migrate
```

3、评论表单设计

在 HTML 文档中这样的代码表示一个表单：

```
<form action="" method="post">
  <input type="text" name="username" />
  <input type="password" name="password" />
  <input type="submit" value="login" />
</form>
```

为什么需要表单呢？表单是用来收集并向服务器提交用户输入的数据的。考虑用户在我们博客网站上发表评论的过程。当用户想要发表评论时，他找到我们给他展示的一个评论表单（我们已经看到在文章详情页的底部就有一个评论表单，你将看到表单呈现给我们的样子），然后根据表单的要求填写相应的数据。之后用户点击评论按钮，这些数据就会发送给某个 URL。我们知道每一个 URL 对应着一个 Django 的视图函数，于是 Django 调用这个视图函数，我们在视图函数中写上处理用户通过表单提交上来的数据的代码，比如验证数据的合法性并且保存数据到数据库中，那么用户的评论就被 Django 后台处理了。如果通过表单提交的数据存在错误，那么我们把错误信

息返回给用户，并在前端重新渲染，并要求用户根据错误信息修正表单中不符合格式的数据，再重新提交。

下面开始编写评论表单代码。在 comments\ 目录下（和 models.py 同级）新建一个 forms.py 文件，用来存放表单代码，我们的表单代码如下：

【comments/forms.py】

```
from django import forms
from .models import Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ['name', 'email', 'url', 'text']
```

要使用 Django 的表单功能，我们首先导入 forms 模块。Django 的表单类必须继承自 **forms.Form** 类或者 **forms.ModelForm** 类。如果表单对应有一个数据库模型（例如这里的评论表单对应着评论模型），那么使用 ModelForm 类会简单很多，这是 Django 为我们提供的方便。之后我们在表单的内部类 Meta 里指定一些和表单相关的东西。model = Comment 表明这个表单对应的数据库模型是 Comment 类。fields = ['name', 'email', 'url', 'text'] 指定了表单需要显示的字段，这里我们指定了 name、email、url、text 需要显示。

关于表单进一步的解释

Django 为什么要给我们提供一个表单类呢？为了便于理解，我们可以把表单和前面讲过的 Django ORM 系统做类比。回想一下，我们使用数据库保存我们创建的博客文章，但是我们从头到尾没有写过任何和数据库有关的代码（要知道数据库自身也有一门数据库语言），这是因为 Django 的 ORM 系统内部帮我们做了一些事情。我们遵循 Django 的规范写的一些 Python 代码，例如创建 Post、Category 类，然后通过运行数据库迁移命令将这些代码反应到数据库。

Django 的表单和这个思想类似（**相当于 ORM 思想，不需要手写前端代码**），正常的前端表单代码应该是和本文开头所提及的那样，但是我们目前并没有写这些代码，而是写了一个 CommentForm 这个 Python 类。通过调用这个类的一些方法和属性，Django 将自动为我们创建常规的表单代码，接下来的教程我们就会看到具体是怎么做的。

4、评论视图函数

当用户提交表单中的数据后，Django 需要调用相应的视图函数来处理这些数据，下面开始写我们视图函数处理逻辑：

【comments/views.py】

```
from django.shortcuts import render, get_object_or_404, redirect
from blog.models import Post

from .models import Comment
from .forms import CommentForm

def post_comment(request, post_pk):
    # 先获取被评论的文章，因为后面需要把评论和被评论的文章关联起来。
    # 这里我们使用了 Django 提供的一个快捷函数 get_object_or_404，
    # 这个函数的作用是当获取的文章（Post）存在时，则获取；否则返回 404 页面给用户。
    post = get_object_or_404(Post, pk=post_pk)

    # HTTP 请求有 get 和 post 两种，一般用户通过表单提交数据都是通过 post 请求，
    # 因此只有当用户的请求为 post 时才需要处理表单数据。
    if request.method == 'POST':
        # 用户提交的数据存在 request.POST 中，这是一个类字典对象。

        # 我们利用这些数据构造了 CommentForm 的实例，这样 Django 的表单就生成了。
        form = CommentForm(request.POST)

        # 当调用 form.is_valid() 方法时，Django 自动帮我们检查表单的数据是否符合格式要求。
        if form.is_valid():
            # 检查到数据是合法的，调用表单的 save 方法保存数据到数据库，
            # commit=False 的作用是仅仅利用表单的数据生成 Comment 模型类的实例，但还不保存评论数据到数据库。因为还要和被评论文章关联起来
            comment = form.save(commit=False)

            # 将评论和被评论的文章关联起来。
            comment.post = post

            # 最终将评论数据保存进数据库，调用模型实例的 save 方法
            comment.save()

            # 重定向到 post 的详情页，实际上当 redirect 函数接收一个模型的实例时，它会调用这个模型实例的 get_absolute_url 方法，
            # 然后重定向到 get_absolute_url 方法返回的 URL。
            return redirect(post)

        else:
            # 检查到数据不合法，重新渲染详情页，并且渲染表单的错误。
            # 因此我们传了三个模板变量给 detail.html，
            # 一个是文章（Post），一个是评论列表，一个是表单 form
            # 注意这里我们用到了 post.comment_set.all() 方法，
            # 这个用法有点类似于 Post.objects.all()
```

```
        # 其作用是获取这篇 post 下的的全部评论，
        # 因为 Post 和 Comment 是 ForeignKey 关联的，
        # 因此使用 post.comment_set.all() 反向查询全部评论。
        # 具体请看下面的讲解。
        comment_list = post.comment_set.all()
        context = {'post': post,
                'form': form,
                'comment_list': comment_list
                }
        return render(request, 'blog/detail.html', context=context)
    # 不是 post 请求，说明用户没有提交数据，重定向到文章详情页。
    return redirect(post)
```

这个评论视图相比之前的一些视图复杂了很多，主要是处理评论的过程更加复杂。具体过程在代码中已有详细注释，这里仅就视图中出现了一些新的知识点进行讲解。

首先我们使用了 redirect 函数。这个函数位于 django.shortcuts 模块中，它的作用是对 HTTP 请求进行重定向（即用户访问的是某个 URL，但由于某些原因，服务器会将用户重定向到另外的 URL）。redirect 既可以接收一个 URL 作为参数，也可以接收一个模型的实例作为参数（例如这里的 post）。如果接收一个模型的实例，那么这个实例必须实现了 **get_absolute_url 方法**，这样 redirect 会根据 get_absolute_url 方法返回的 URL 值进行重定向。

另外我们使用了 post.comment_set.all() 来获取 post 对应的全部评论。 Comment 和 Post 是通过 ForeignKey 关联的，回顾一下我们当初获取某个分类 cate 下的全部文章时的代码：Post.objects.filter(category=cate)。这里 post.comment_set.all() 也等价于 Comment.objects.filter(post=post)，即根据 post 来过滤该 post 下的全部评论。但既然我们已经有了一个 Post 模型的实例 post（它对应的是 Post 在数据库中的一条记录），那么获取和 post 关联的评论列表有一个简单方法，即**调用它的 xxx_set 属性来获取一个类似于 objects 的模型管理器**，然后调用其 all 方法来返回这个 post 关联的全部评论。 其中 xxx_set 中的 xxx 为**关联模型的类名（小写）**。例如 Post.objects.filter(category=cate) 也可以等价写为 cate.post_set.all()。

5、绑定 URL

视图函数需要和 URL 绑定，这里我们在 comment 应用中再建一个 urls.py 文件，写上 URL 模式：

comments/urls.py

```
from django.conf.urls import url

from . import views
```

```
app_name = 'comments'
urlpatterns = [
    url(r'^comment/post/(?P<post_pk>[0-9]+)/$', views.post_comment, name='post_comment'),
]
```

别忘了给这个评论的 URL 模式规定命名空间，即 app_name = 'comments'。

最后要在项目的 blogproject 目录的 urls.py 里包含 comments\urls.py 这个文件：
blogproject/urls.py

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'', include('blog.urls')),
    + url(r'', include('comments.urls')),
]
```

6、更新文章详情页面的视图函数

我们可以看到评论表单和评论列表是位于文章详情页面的，处理文章详情页面的视图函数是 detail，相应地需要更新 detail，让它生成表单和从数据库获取文章对应的评论列表数据，然后传递给模板显示：
blog/views.py

```
import markdown

from django.shortcuts import render, get_object_or_404

+ from comments.forms import CommentForm
from .models import Post, Category

def detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    post.body = markdown.markdown(post.body,
                                  extensions=[
                                     'markdown.extensions.extra',
                                     'markdown.extensions.codehilite',
                                     'markdown.extensions.toc',
                                  ])
    # 记得在顶部导入 CommentForm，这里实例化没有 request.POST 作为参数，因此表单都是空的
    form = CommentForm()
    # 获取这篇 post 下的全部评论
    comment_list = post.comment_set.all()

# 将文章、表单、以及文章下的评论列表作为模板变量传给 detail.html 模板，以便渲染相应数据。
    context = {'post': post,
               'form': form,
               'comment_list': comment_list
               }
    return render(request, 'blog/detail.html', context=context)
```

7、在前端渲染表单

使用 Django 表单的一个好处就是 Django 能帮我们自动渲染表单。我们在表单的视图函数里传递了一个 form 变量给模板，这个变量就包含了自动生成 HTML 表单的全部数据。在 detail.html 中通过 form 来自动生成表单。删掉原来用于占位的 HTML 评论表单代码，即下面这段代码：

```
<form action="#" method="post" class="comment-form">
 <div class="row">
  <div class="col-md-4">
   <label for="id_name">名字：</label>
   <input type="text" id="id_name" name="name" required>
  </div>
  ...
 </div>   <!-- row -->
</form>
```

替换成如下的代码：

```
<form action="{% url 'comments:post_comment' post.pk %}" method="post" class="comment-form">
 {% csrf_token %}
 <div class="row">
  <div class="col-md-4">
   <label for="{{ form.name.id_for_label }}">名字：</label>
   {{ form.name }}
   {{ form.name.errors }}
  </div>
  <div class="col-md-4">
   <label for="{{ form.email.id_for_label }}">邮箱：</label>
   {{ form.email }}
   {{ form.email.errors }}
  </div>
  <div class="col-md-4">
   <label for="{{ form.url.id_for_label }}">URL：</label>
   {{ form.url }}
   {{ form.url.errors }}
  </div>
  <div class="col-md-12">
   <label for="{{ form.text.id_for_label }}">评论：</label>
   {{ form.text }}
   {{ form.text.errors }}
   <button type="submit" class="comment-btn">发表</button>
  </div>
 </div>   <!-- row -->
</form>
```

{{ form.name }}、{{ form.email }}、{{ form.url }} 等将自动渲染成**表单控件**，例如 **<input>** 控件。

{{ form.name.errors }}、{{ form.email.errors }} 等将渲染表单对应字段的错误（如果有的话），例如用户 email 格式填错了，那么 Django 会检查用户提交的 email 的格式，然后将格式错误信息保存到 errors 中，模板便将错误信息渲染显示。

8、显示评论内容

在 detail 视图函数我们获取了全部评论数据，并通过 comment_list 传递给了模板。和处理 index 页面的文章列表方式是一样的，我们在模板中通过 {% for %} 模板标签来循环显示文章对应的全部评论内容。

删掉占位用的评论内容的 HTML 代码，即如下的代码：

```
<ul class="comment-list list-unstyled">
 <li class="comment-item">
  <span class="nickname">追梦人物</span>
  <time class="submit-date">2017 年 3 月 12 日 14:56</time>
  <div class="text">
   文章观点又有道理又符合人性，这才是真正为了表达观点而写，不是为了迎合某某知名人士粉丝而写。我觉得如果琼瑶是前妻，生了三孩子后被一不知名的女人挖了墙角，我不信谁会说那个女人是追求真爱，说同情琼瑶骂小三的女人都是弱者。
  </div>
 </li>
 ...
</ul>
```

替换成如下的代码：

```
<ul class="comment-list list-unstyled">
 {% for comment in comment_list %}
 <li class="comment-item">
  <span class="nickname">{{ comment.name }}</span>
  <time class="submit-date">{{ comment.created_time }}</time>
  <div class="text">
   {{ comment.text }}
  </div>
 </li>
 {% empty %}
 暂无评论
 {% endfor %}
</ul>
```

接下来尝试在详情页下的评论表单提交一些评论数据，可以看到详情页的评论列表处渲染了你提交的评论数据。

发表评论

RSS 订阅

名字： 杨学光

邮箱： test@test.com

URL： http://127.0.0.1:8000/

评论：

A field for storing periods of time - modeled in Python by timedelta. When used on PostgreSQL, the data type used is an interval and on Oracle the data type is INTERVAL DAY(9) TO SECOND(6). Otherwise a bigint of microseconds is used.

发表

评论列表，共 4 条评论

管理员 · 2017年5月13日 21:20

发表评论测试...

追梦人物 · 2017年5月13日 21:20

评论功能已经可以了！

---

zmrenwu · 2017年5月13日 21:22

The auto_now and auto_now_add options will always use the date in the default timezone at the moment of creation or update. If you need something different, you may want to consider simply using your own callable default or overriding save() instead of using auto_now or auto_now_add; or using a DateTimeField instead of a DateField and deciding how to handle the conversion from datetime to date at display time.

>