# 4-让 Django 完成翻译:迁移数据库

我们已经编写了博客数据库模型的代码,但那还只是 Python 代码而已,Django 还没有把它翻译成数据库语言,因此实际上这些数据库表还没有真正的在数据库中创建。

# 1、迁移数据库

为了让 Django 完成翻译,创建好这些数据库表,我们再一次请出我的工程管理助手 manage.py。激活虚拟环境,切换到 manage.py 文件所在的目录下,分别运行以下两条命令:

python manage.py makemigrations

python manage.py migrate

当我们执行了 python manage.py makemigrations 命令(相当于 flask-migrate 的 python manage.py init 命令)后,Django 在 blog 应用的 migrations\ 目录下生成了一个 0001\_initial.py 文件,这个文件是 Django 用来记录我们对模型做了哪些修改的文件。目前来说,我们在 models.py 文件里创建了 3 个模型类,Django 把这些变化记录在了 0001\_initial.py 里。

不过此时还只是告诉了 Django 我们做了哪些改变,为了让 Django 真正地为我们创建数据库表,接下来又执行了 python manage.py migrate 命令(相当于 flask-migrate 的 python manage.py migrate 命令)。 Django 通过检测应用中 migrations\目录下的文件,得知我们对数据库做了哪些操作,然后它把这些操作翻译成数据库操作语言,从而把这些操作作用于真正的数据库。

你可以看到命令的输出除了 Applying blog.0001\_initial... OK 外, Django 还对其它文件做了操作。这是因为除了我们自己建立的 blog 应用外, Django 自身还内置了很多应用,这些应用本身也是需要存储数据的。可以在 settings.py 的 INSTALLED\_APP 设置里看到这些应用,当然我们目前不必关心这些。

[blogproject/settings.py]

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

对于了解数据库语言的人,你可以运行下面的命令看看 Django 究竟为我们做了什么:

#### python manage.py sqlmigrate blog 0001

你将看到输出了经 Django 翻译后的数据库表创建语句,这有助于你理解 Django ORM 的工作机制。

## 2、选择数据库版本

我们没有安装任何的数据库软件, Django 就帮我们迁移了数据库。这是因为我们使用了 Python 内置的 SQLite3 数据库。

SQLite3 是一个十分轻巧的数据库,它仅有一个文件。你可以看一到项目根目录下多出了一个 db.sqlite3 的文件,这就是 SQLite3 数据库文件, Django 博客的数据都会保存在这个数据库文件里。

Django 在 settings.py 里为我们做了一些默认的数据库配置:

[blogproject/settings.py]

```
## 其它配置选项...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

## 其它配置选项...
```

可以看到默认的数据库引擎就是使用的 SQLite3。

当然一些人倾向于使用 MySQL 等大型数据库,至于 Django 如何配置 MySQL 这里就不赘述了,你可以自行使用搜索引擎或者查阅 Django 的官方文档解决。对于一个小型博客而言,SQLite3 数据库足以胜任。

# 3、用 Django 的方式操作数据库

数据库最主要的操作就是往里面存入数据、从中取出数据、修改已保存的数据和删除不再需要的数据。和创建数据库表一样,Django为这些操作提供了一整套方法,从而把我们从数据库语言中解放出来。我们不用学习如何利用数据库语言去完成这些操作,只要简单地调用几个Python函数就可以满足我们的需求。

## (1) 存数据

先在命令行中来探索一下这些函数,感受一下如何用 Django 的方式来操作数据库。在 manage.py 所在目录下运行:

python manage.py shell

这打开了一个交互式命令行。

首先我们来创建一个分类和一个标签:

```
>>> from blog.models import Category, Tag, Post
>>> c = Category(name='category test')
>>> <mark>c.save() #注意对比:在 flask 当中是 db.session.add(c),db.session.commit()</mark>
>>> t = Tag(name='tag test')
>>> t.save()
```

我们首先导入 3 个之前写好的模型类,然后实例化了一个 Category 类和一个 Tag 类,为他们的属性 name 赋了值。为了让 Django 把这些数据保存进数据库,调用实例的 save 方法即可。

再创建一篇文章试试,但创建文章之前,我们需要先创建一个 User,用于指定文章的作者。创建 User 的命令 Django 已经帮我们写好了,依然是通过manage.py 来运行。首先按住 Ctrl + c 退出命令交互栏(一次退不出就连续多按几次),运行 python manage.py createsuperuser 命令并根据提示创建用户:

#### \$ python manage.py createsuperuser

Username (leave blank to use 'nanfengpo'): myuser

Email address: a@a.com

Password:

Password (again):

Superuser created successfully.

运行 python manage.py createsuperuser 开始创建用户,之后会提示你输入用户名、 邮箱、密码和确认密码,按照提示输入即可。注意一点的是密码输入过程中不 会有任何字符显示,不要误以为你的键盘出问题了,正常输入即可。最后出现 Superuser created successfully. 说明用户创建成功了。

```
再次运行 python manage.py shell 进入 Python 命令交互栏, 开始创建文章:
>>> from blog.models import Category, Tag, Post
>>> from django.utils import timezone
>>> from django.contrib.auth.models import User
>>> user = User.<mark>objects</mark>.get(username='myuser') # 注意对比:在 flask 当中是
User.guery.filter by(username='myuser')
>>> c = Category.objects.get(name='category test')
>>> p = Post(title='title test', body='body test', created_time=timezone.now(),
modified_time=timezone.now(), category=c, author=user)
>>> p.save()
```

由于我们重启了 shell,因此需要重新导入了 Category、Tag、Post 以及 User。我们 还导入了一个 Diango 提供的辅助模块 timezone , 这是因为我们需要调用它 的 now() 方法为 created\_time 和 modified\_time 指定时间,容易理解 now 方法返回当 前时间。然后我们根据用户名和分类名,通过 get 方法取出了存在数据库中 的 User 和 Category ( 取数据的方法将在下面介绍 )。接着我们为文章指定 了title、body、created\_time、modified\_time 值,并把它和前面创建的 Category 以 及 User 关联了起来。允许为空 excerpt、tags 我们就没有为它们指定值了。

#### 注意:

- 1. 我们这里使用 get 方法根据 Category 的 name 属性的值获取分类的一条记 录。Category.objects.get(name='category test') 的含义是从数据库中取出 name 的 值为 category test 的分类记录。确保数据库中只有一条值为 category test 的记录,否则 get 方法将返回一个 MultipleObjectsReturned 异常。如果你不小 心已经存了多条记录,请删掉多余的记录。如何删除数据请看下文。
- 2. 要对多对多关系赋值,必须使用中括号:p.tag=[t]

### (2)取数据

数据已经存入数据库了,现在要把它们取出来看看: >>> Category.objects.all()

<QuerySet [<Category: Category object>]>

>>> Tag.objects.all()

```
<QuerySet [<Tag: Tag object>]>
>>> Post.objects.all()
<QuerySet [<Post: Post object>]>
>>>
```

objects 是我们的模型管理器(相当于 flask 里的 query),它为我们提供一系列从数据库中取数据方法,这里我们使用了 all 方法,表示我们要把对应的数据全部取出来。可以看到 all 方法都返回了数据,这些数据应该是我们之前存进去的,但是显示的字符串有点奇怪,无法看出究竟是不是我们之前存入的数据。为了让显示出来的数据更加人性化一点,我们为 3 个模型分别增加一

个 str 方法:

[blog/models.py]

return self.title

from django.utils.six import python\_2\_unicode\_compatible

```
# python_2_unicode_compatible 装饰器用于兼容 Python2
@python_2_unicode_compatible
class Category(models.Model):
...

def __str__(self):
    return self.name

@python_2_unicode_compatible
class Tag(models.Model):
...

def __str__(self):
    return self.name

@python_2_unicode_compatible
class Post(models.Model):
...

def __str__(self):
...

def __str__(self):
```

定义好\_str\_方法后,解释器显示的内容将会是\_str\_方法返回的内容。这里 Category 返回分类名 name , Tag 返回标签名 , 而 Post 返回它的 title。

python\_2\_unicode\_compatible 装饰器用于兼容 Python2。如果你使用的 Python3 开发环境,去掉这个装饰器不会有任何影响。如果你使用的 Python2 开发环境,而又不想使用这个装饰器,则将 \_str\_ 方法改为 \_unicode\_ 方法即可。

先按 Ctrl + c 退出 Shell,再重新运行 python manage.py shell 进入 Shell。
>>> **from blog.models import** Category, Tag, Post
>>> Category.objects.all()
<QuerySet [<Category: category test>]>

```
>>> Tag.objects.all()
<QuerySet [<Tag: tag test>]>
>>> Post.objects.all()
<QuerySet [<Post: title test>]>
>>> Post.objects.get(title='title test')
<Post: title test>
```

可以看到返回的是我们之前存入的数据。

此外我们在创建文章时提到了通过 get 方法来获取数据,这里 all 方法和 get 方法的区别是: all 方法返回全部数据,是一个类似于列表的数据结构 (QuerySet);而 get 返回一条记录数据,如有多条记录或者没有记录,get 方法均会抛出相应异常。

## (3)改数据

```
<mark>修改数据和增加数据一样,都是 save()</mark>。尝试修改数据:
>>> c = Category.objects.get(name='category test')
>>> c.name = 'category test new'
>>> c.save()
>>> Category.objects.all()
<QuerySet [<Category: test category new>]>
```

首先通过 get 方法根据分类名 name 获取值为 category test 到分类,修改它的 name 属性为新的值 category test new,然后调用 save 方法把修改保存到数据库,之后可以看到数据库返回的数据已经是修改后的值了。Tag、Post 的修改也一样。

### (4)删数据

#### 删除掉数据:

```
>>> p = Post.objects.get(title='title test')
>>> p
<Post: title test>
>>> p.delete()
(1, {'blog.Post_tags': 0, 'blog.Post': 1})
>>> Post.objects.all()
<QuerySet []>
```

先根据标题 title 的值从数据库中取出 Post,保存在变量 p中,然后调用它的 delete 方法,最后看到 Post.objects.all()返回了一个空的 QuerySet(类似于一个列表),表明数据库中已经没有 Post, Post 已经被删除了。

这就是 Django 对数据库增、删、改、查的操作。除了上述演示的方法外,Django 还为我们提供了大量其它的方法,这些方法有一部分会在教程中使用,用到时我会讲解它们的用法。但以后你开发自己的项目时,你就需要通过阅读 Django 的官方文档来了解有哪些方法可用以及如何使用它们。