25-用户认证-拓展模型

0、必要的配置

Django 在新建工程时已经为使用用户认证系统做好了全部必要的配置，因此**使用默认配置即可**。不过有可能你并非使用 django-admin 命令新建的工程，或者你使用的是一个正在开发中的项目，因此最好再检查一下 settings.py 文件中是否已经做好了全部必要配置。

首先检查一下必要的应用是否已经在 `INSTALLED_APPS` 配置里列出：

【blogproject/settings.py】

```
INSTALLED_APPS = [
    # 其它应用列表...
    'django.contrib.auth',
    'django.contrib.contenttypes',
]
```

为了使用用户认证系统，必须安装以下两个应用：

- django.contrib.auth
- django.contrib.contenttypes

django.contrib.contenttypes 是 auth 模块的用户权限处理部分依赖的应用。

其次需要在中间件 `MIDDLEWARE` 配置里列出以下两个中间件：

- SessionMiddleware 用户处理用户会话。
- AuthenticationMiddleware 绑定一个 User 对象到请求中（具体将在后面介绍）。

即像下面这样的配置：

【blogproject/settings.py】

```
MIDDLEWARE = [
    # 其它中间列表...
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
]
```

如果以上配置没问题的话就可以正常地使用用户认证系统了。

1、新建一个应用

推荐的做法是新建一个应用，用于存放和用户功能相关的代码。把应用命名为 users：

```
$ python manage.py startapp users
```

新建的应用一定要记得在 settings.py 里注册，否则 Django 无法得知你新建了应用。

【blogproject/settings.py】

```
INSTALLED_APPS = [
    # 其它应用列表...
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'users', # 注册新建的应用 users
]
```

项目的基本准备工作到这里就结束了，接下来让我们开始使用 Django 用户系统为我们提供的功能。

2、拓展 User 模型

Django 用户认证系统提供了一个内置的 User 对象，用于记录用户的用户名，密码等个人信息。对于 Django 内置的 User 模型， 仅包含以下一些主要的属性：

- username，即用户名
- password，密码
- email，邮箱
- first_name，名
- last_name，姓

对于一些网站来说，用户可能还包含有昵称、头像、个性签名等等其它属性，因此仅仅使用 Django 内置的 User 模型是不够。好在 Django 用户系统遵循可拓展的设计原则，我们可以方便地拓展 User 模型。

```
abs:1)absolute  2)abstract
```

（1）继承 AbstractUser 拓展用户模型

这是推荐做法。事实上，查看 User 模型的源码就知道，以下就是 **User 的源码**：

```
class User(AbstractUser):
    """
```

```
    Users within the Django authentication system are represented by this
    model.

    Username, password and email are required. Other fields are optional.
    """
    class Meta(AbstractUser.Meta):
        swappable = 'AUTH_USER_MODEL'
```

User 也是继承自 AbstractUser 抽象基类，而且仅仅就是继承了 AbstractUser，没有对 AbstractUser 做任何的拓展。所以，如果我们继承 AbstractUser，将获得 User 的全部特性，而且还可以根据自己的需求进行拓展。

我们之前新建了一个 users 应用，通常我们把和数据库模型相关的代码写在 models.py 文件里。打开 users/models.py 文件，写上我们自定义的用户模型代码：

【users/models.py】

```
from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    nickname = models.CharField(max_length=50, blank=True)
    class Meta(AbstractUser.Meta):
        pass
```

其实User也是继承于AbstractUser，又是继承于AbstractBaseUser

新创建的昵称字段

包含了5个已有字段

我们给自定义的用户模型新增了一个 nickname（昵称）属性，用来记录用户的昵称信息，设置 blank=True 的目的是让用户在注册时无需填写昵称。根据你的需求可以自己进一步拓展，例如增加用户头像、个性签名等等，添加多少属性字段没有任何限制。

同时，我们继承了 AbstractUser 的内部类属性 Meta ，不过目前什么也没做。**在这里继承 Meta 的原因是在你的项目中可能需要设置一些 Meta 类的属性值，不要忘记继承 AbstractUser.Meta 中已有的属性。**

注意：一定要继承 AbstractUser，而不是继承 auth.User。尽管 auth.User 继承自 AbstractUser 且并没有对其进行任何额外拓展，但 AbstractUser 是一个抽象类，而 auth.User 不是。如果你继承了 auth.User 类，这会变成多表继承，在目前的情况下这种继承方式是不被推荐的。关于 Django 的抽象模型类和多表继承，请查阅 Django 的官方文档 模型继承。

此外，AbstractUser 类又继承自 AbstractBaseUser，前者在后者的基础上拓展了一套**用户权限**（Permission）系统。因此如非特殊需要，尽量不要从 AbstractBaseUser 拓展，否则你需要做更多的额外工作。

为了让 Django 用户认证系统使用我们自定义的用户模型，必须在 settings.py 里通过 AUTH_USER_MODEL 指定自定义用户模型所在的位置，即需要如下设置：

【blogproject/settings.py】

users是一个app

```
# 其它设置...
AUTH_USER_MODEL = 'users.User'
```

不需要users.models.User

即告诉 Django，使用 users 应用下的 User 用户模型。

然后在 blog/models.py 中修改 Post 的作者外键为新的 User 类：

【blog/models.py】

```
-from django.contrib.auth.models import User

+from users.models import User
```

设置好自定义用户模型后，生成数据库迁移文件，并且迁移数据库以生成各个应用必要的数据库表。即运行如下两条命令：

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

OK，现在 Django 用户系统使用的用户模型就是自定义的 User 模型了。

注意：一定要在设置好 AUTH_USER_MODEL = 'users.User' 后再第一次迁移数据库，即指定好自定义的用户模型后再执行数据库迁移命令。

（2）使用 Profile 模式拓展用户模型

如果想为一个已使用了 Django 内置 User 模型的项目拓展用户模型，上述继承 AbstractUser 的拓展方式会变得有点麻烦。**Django 没有提供一套自动化的方式将内置的 User 迁移到自定义的用户模型，因为 Django 已经为内置的 User 模型生成了相关数据库迁移文件和数据库表。如果非要这么做的话，需要手工修改迁移文件和数据库表，并且移动数据库中相关的用户数据。**

所以我们采用另一种不改动数据库表的方式来拓展用户模型，具体来说，我们在创建一个模型（通常命名为 Profile）来记录用户相关的数据，然后使用一对一的方式将这个 Profile 模型和 User 关联起来，就好像每个用户都关联着一张记录个人资料的表一样。代码如下：

【users/models.py]

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

class Profile(models.Model):
    nickname = models.CharField(max_length=50, blank=True)
    user = models.OneToOneField(User)
```

注意：必须是一对一的关系

这种方式和 `AbstractUser` 的区别是，继承 `AbstractUser` 的用户模型只有一张数据库表。而 Profile 这种模式有两张表，一张是 User 模型对应的表，一张是 Profile 模型对应的表，两张表通过一对一的关系关联。可见，当要查询某个用户的 Profile 时，需要执行额外的跨表查询操作，所以这种方式比起直接继承 AbstractUser 效率更低一点。因此对于新项目来说，优先推荐使用继承 AbstractUser 的方式来拓展用户模型。（如果是老项目，则只能使用 profile 模式）

PS：如果你使用了 Profile 模式，你可能希望在创建 User 对象的时候同时也创建与之关联的 Profile 对象。你可以使用 Django 的 Signal 实现这个需求。由于 Profile 模式不是我们要介绍的重点内容，因此具体的实现细节请参照相关的文档，这里不再赘述。

OK，自定义的 User 模型已经建立好了，接下来就是如何创建用户，即用户注册流程了。