Django Auth用户与用户组详述

在第一章的《Django auth应用模块》我们简单的介绍了 auth 应用，它能够帮助开发者快速构建用户模块的基本功能，包括用户与用户组的实现以及定义用户与用户组权限等，例如，社交平台需要现有用户才可以发布动态话题；管理后台 admin 需要有用户才能登陆等。而且对于不同的用户，Web 站点还可以提供不同的服务，这就是权限的概念。

Django 框架内置的用户认证系统实现了上述功能，即身份的验证和权限的管理，与其他的内置的模块类似，这套系统能够很好的支持扩展和自定义功能，本章我们将一起认识 Django 的用户认证系统。用户认证系统中定义了三个 Model 用来标识用户与用户关系，分别是 User（用户）、AnonymousUser（匿名用户）和 Group（用户组），它们都定义在下面的路径文件中

django/contrib/auth/models.py

1. User用户模型

在《Django Admin数据表可视化》一节，我们使用 createsuperuser 命令创建了超级用户，在视图函数中我们可以通过 HttpRequest 的 user 属性获取当前的登录用户。这里的用户其实就是 Django 框架中内置的 User Model（即auth_user表）因为它被定义在 auth 应用下所以表名是auth_user，可用如下方式引入 User 模型：

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

在《Django auth应用模块》中我们提到过 auth_user 表，在执行完毕 migrate 后，它的表结构如下所示：

```
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int(11)      | NO   | PRI | NULL    | auto_increment |
| password     | varchar(128) | NO   |     | NULL    |                |
| last_login   | datetime(6)  | YES  |     | NULL    |                |
| is_superuser | tinyint(1)   | NO   |     | NULL    |                |
| username     | varchar(150) | NO   | UNI | NULL    |                |
| first_name   | varchar(30)  | NO   |     | NULL    |                |
| last_name    | varchar(150) | NO   |     | NULL    |                |
| email        | varchar(254) | NO   |     | NULL    |                |
| is_staff     | tinyint(1)   | NO   |     | NULL    |                |
| is_active    | tinyint(1)   | NO   |     | NULL    |                |
| date_joined  | datetime(6)  | NO   |     | NULL    |                |
```

```
+--------------+-------------+------+-----+---------+----------------+
11 rows in set (0.02 sec)
```

对于上表前面介绍是我们只是一笔带过，在这里有必讲解一下需要重点理解的属性。如下所示：

- is_superuser：布尔值，默认值是 False。标识是否是超级用户，代表用户拥有所有权限。
- username：用户名，具有唯一性限制，最大长度为150个字符，只可以包含字母、数字、@、.、+、-、_这些字符。
- password：密码，Django 并不会存储原始密码，其存储的实际是原始密码经过 Hash 散列处理之后的值。
- is_staff：布尔值，默认为 False。标识用户是否可以访问管理后台。
- is_active：布尔值，默认值是True。标识当前用户是否处于激活状态。

除了基础属性之外，User 中还定义了与 Group 和 Permission（权限）之间的关联关系：

```
1. class PermissionsMixin(models.Model):
2. group=models.ManyToManyField(Group,...)
3. user_permission=models.ManyToManyField(Permission,...)
```

User 关联表，即 auth_user_groups 和 auth_user_user_permissions，其表分别结构如下图所示：

```
mysql> desc auth_user_groups;
+----------+---------+------+-----+---------+----------------+
| Field    | Type    | Null | Key | Default | Extra          |
+----------+---------+------+-----+---------+----------------+
| id       | int(11) | NO   | PRI | NULL    | auto_increment |
| user_id  | int(11) | NO   | MUL | NULL    |                |
| group_id | int(11) | NO   | MUL | NULL    |                |
+----------+---------+------+-----+---------+----------------+
3 rows in set (0.02 sec)

mysql> desc auth_user_user_permissions;
+---------------+---------+------+-----+---------+----------------+
| Field         | Type    | Null | Key | Default | Extra          |
+---------------+---------+------+-----+---------+----------------+
| id            | int(11) | NO   | PRI | NULL    | auto_increment |
| user_id       | int(11) | NO   | MUL | NULL    |                |
| permission_id | int(11) | NO   | MUL | NULL    |                |
+---------------+---------+------+-----+---------+----------------+
3 rows in set (0.02 sec)
```

1) User模型创建用户与超级用户

我们可以使用 User 模型的 create_user 和 create_superuser 分别创建用户或者是超级用户，在创建超级用户的时候需要注意 is_staff 与 is_superuserd 的 bool 值，需要都设置为 Ture 才可以。下面看一下查看其中一个方法即 create_user 的源代码，通过查看，加深对

Django 的学习，如下所示：

```
class UserManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, username, email, password, **extra_fields):
        """
        创建并保存具有给定用户名、电子邮件和密码的用户。
        """
        if not username:
            raise ValueError('The given username must be set')
        email = self.normalize_email(email)
        #使用户名规范化调用normalize_username
        username = self.model.normalize_username(username)
        #新建user实例
        user = self.model(username=username, email=email, **extra_fields)
        #设置密码的方法
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, username, email=None, password=None, **extra_fields):
        #普通用户的is_staff和is_superuser都为False
        extra_fields.setdefault('is_staff', False)
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(username, email, password, **extra_fields)
```

使用该方法的实例如下所示：

```
from django.contrib.auth.models import User
user=User.objects.create_user('bookstore','123@163.com','python_django')
```

同样我们可以 set_password() 方法修改密码，最后记得调用 save() 方法保存即可。

2. AnonymousUser匿名用户模型

对于 AnonymousUser，它的常见用法是对视图的请求，对于未登陆的用户，request 的 user 属性即指向了 AnonymousUser 表示匿名用户，我们看一下这个类是如何实现的：

```
class AnonymousUser:
    id = None
    pk = None
    username = ''
    is_staff = False
    is_active = False
```

```
    is_superuser = False
    _groups = EmptyManager(Group)
    _user_permissions = EmptyManager(Permission)
```

从源码分析可以看出 AnonymousUser 定义匿名用户的主要属性， 可以看到它的 is_staff 和 is_active 以及 is_superuser 都设置成为了 False，它还定义了一些方法如下所示：

```
def save(self):
    raise NotImplementedError("Django doesn't provide a DB representation for AnonymousUser.")

def delete(self):
    raise NotImplementedError("Django doesn't provide a DB representation for AnonymousUser.")

def set_password(self, raw_password):
    raise NotImplementedError("Django doesn't provide a DB representation for AnonymousUser.")

def check_password(self, raw_password):
    raise NotImplementedError("Django doesn't provide a DB representation for AnonymousUser.")
```

从上述代码可以看出 AnonymousUser 匿名用户定义的方法都抛出了 NotImplementedError 异常，所以它并没实现任何方法。

3. Group用户组模型

上面讲解了用户模型与匿名用户模型，最后一个模型就是用户组 Group，首先我们来理解一下用户组的概念。

1) Group用户组概念

组是对用户进行分类的通用方法，以便将权限或其他标签应用到这些用户。用户可以属于任意数量的组。组中的用户自动拥有授予该组的所有权限。例如，如果“网站编辑”组有权限 can_edit_home_page，该组中的任何用户都将拥有该权限。除了权限之外，组还可以方便地对用户进行分类，以便对他们应用一些标签或扩展功能。例如，您可以创建一个“特殊用户”组，并且您可以编写相应的代码对这个特殊用户组，让组内用户做一些特殊的事情——比如让他们访问您站点的成员权限部分，或者给他们发送成员权限的电子邮件消息。

然后我们可以查看一下 Django 实现 Group 编写的源码部分。如下所示：

```
class Group(models.Model):
    name = models.CharField(_('name'), max_length=150, unique=True)
    permissions = models.ManyToManyField(
        Permission,
        verbose_name=_('permissions'),
        blank=True,
```

```
    )

    objects = GroupManager()

    class Meta:
        verbose_name = _('group')
        verbose_name_plural = _('groups')

    def __str__(self):
        return self.name

    def natural_key(self):
        return (self.name,)
```

从源码解析来看，Group 用户组之定义了一个字段 name，代表用户组的名称而且必须具有唯一性，其最大字符长度为 150，它还定义与 Permission 模型之间多对多关联关系，那么它们之间就有有一张中间表即 auth_group_permissions，通过数据库查看一下它的表结构，如下所示：

```
mysql> desc auth_group_permissions;
+---------------+---------+------+-----+---------+----------------+
| Field         | Type    | Null | Key | Default | Extra          |
+---------------+---------+------+-----+---------+----------------+
| id            | int(11) | NO   | PRI | NULL    | auto_increment |
| group_id      | int(11) | NO   | MUL | NULL    |                |
| permission_id | int(11) | NO   | MUL | NULL    |                |
+---------------+---------+------+-----+---------+----------------+
```

2) Group用户组实例应用

下面我们创建一个用户组名称为 reader，然后将 user 加入到该组当中：

```
In [1]: from django.contrib.auth.models import User,Group
In [2]: group=Group.objects.create(name="reader")
In [3]: user=User.objects.get(username="bookstore")
In [4]: user.groups.add(group)
In [5]: user.groups.all()
Out[5]: <QuerySet [<Group: reader>]>
```

通过上述的代码就将用户 user 加入到了组 reader 中，我们可以通过用户组权限再给这个组设置相应的权限，查看 auth_user_groups 表可得如下结果：

```
mysql> select * from auth_user_groups;
+----+---------+----------+
| id | user_id | group_id |
+----+---------+----------+
|  1 |       2 |        1 |
+----+---------+----------+
```

本节我们详细介绍了 Django 用户认证系统中的用户与用户组，从源码的角度出发对它们之间的关联关系进行了深度的剖析，通过本节的讲解大家对用户与用户组的概念不在感到陌生，在下一节我们将讲解如何进行用户的身份认证。