用户认证系统

一、概要

auth模块是Django提供的标准权限管理系统,可以提供用户身份认证, 用户组和权限管理。

auth可以和admin模块配合使用， 快速建立网站的管理系统。

在INSTALLED_APPS中添加'django.contrib.auth'使用该APP, auth模块默认启用。

主要的操作包括:

1. create_user 创建用户
2. authenticate 验证登录
3. login 记住用户的登录状态
4. logout 退出登录
5. is_authenticated 判断用户是否登录
6. @login_required 判断用户是否登录的装饰器

二、前期配置

1、说明

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。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。

2、配置

1. 在setting.py的INSTALLED_APPS

```
INSTALLED_APPS = [
    'django.contrib.auth',
    # 用户权限处理部分依赖的应用
    'django.contrib.contenttypes',
]
```

2. 在setting.py的MIDDLEWARE

```
MIDDLEWARE = [
# 会话支持中间件
'django.contrib.sessions.middleware.SessionMiddleware',
# 认证支持中间件
'django.contrib.auth.middleware.AuthenticationMiddleware',
]
```

三、User对象

1、user对象

1. 属性

说明	说明	备注
username	少于等于30个字符。 用户名可以包含字母、数字、_、@、+、.和- 字符	必选
password	密码的哈希及元数据。（Django 不保存原始密码）。原始密码可以无限长而且可以包含任意字符。参见密码相关的文档	必选
is_active	布尔值。指示用户的账号是否激活，缺省值为True	必选
first_name	少于等于30个字符	可选
last_name	少于30个字符	可选
email	邮箱地址	可选
groups	与Group 之间的多对多关系	可选
user_permissions	与Permission 之间的多对多关系	可选
is_staff	布尔值。指示用户是否可以访问Admin 站点	可选
is_superuser	布尔值。只是这个用户拥有所有的权限而不需要给他们分配明确的权限。	可选
last_login	用户最后一次登录的时间	默认值
date_joined	账户创建的时间。当账号创建时，默认设置为当前的date/time	默认值

2. 说明

User 对象属性：username， password（必填项）password用哈希算法保存到数据库

is_staff： 用户是否拥有网站的管理权限.

is_active： 是否允许用户登录, 设置为`False`，可以不用删除用户来禁止 用户登录

2、拓展 User 模型

2.1、说明

> 用户可能还包含有头像、昵称、介绍等等其它属性，因此仅仅使用 Django 内置的 User 模型是不够。所有有些时候我们必须使用在系统的User上进行拓展

2.2、继承AbstractUser

1. 说明

   推荐方式、django.contrib.auth.models.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

2. 在app的models.py中

```
class User(AbstractUser):
    phone = models.CharField(
        max_length=12,
        null=True,
        verbose_name="手机号"
    )
    class Meta(AbstractUser.Meta):
        db_table='user'
```

3. 注意

   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

```
AUTH_USER_MODEL = 'app名字.User'
```

4. 迁移

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

3、 常用操作

1、验证登录

1. 说明

   当用户登录的时候用`authenticate(username=username,password=password)`验证用户是否登录，如果数据库中存在用户输入的账号和密码，返回一个user对象，否则返回None。底层将password用hash算法加密后和数据库中password进行对比

2. 示例代码

```
def myauthenticate(request):
    pwd = request.POST.get("pwd", "")
    u_name = request.POST.get("u_name", "")
    if len(pwd) <= 0 or len(u_name) <= 0:
        return HttpResponse("账号或密码不能为空")
    else:
        user = authenticate(username=u_name, password=pwd)
        if user:
            return HttpResponse("验证成功")
        else:
            return HttpResponse("账号或密码错误")
```

2、注册操作

1. 说明

   当用户注册的时候用`create_user(username,password,email)`默认情况下`is_active=True,is_staff=False,is_superuser=False`。

   底层将password用hash算法加密之后存储到数据库中

2. 示例代码

```
def register_view(request):
    if request.method == 'POST':
        try:
            username = request.POST.get('username')
```

```
            password = request.POST.get('password')
            phone = request.POST.get('phone')
            email = request.POST.get('email')
            # 验证用户是否存在
            user = User.objects.filter(username=username).first()
            if user:
                # 用户已经存在
                return render(request, 'register.html', {'msg':
'用户名已存在'})
            else:
                # 保存用户
                User.objects.create_user(username=username,

 password=password,
                                         phone=phone,
                                         email=email)
                return redirect(reverse("App:login"))
        except Exception as e:
            return render(request, 'register.html', {'msg': '注册
失败'})
    else:
        return render(request, 'register.html')
```

3、登录操作

1. 说明

   当用户登录的时候用`login(request,user)`来记住用户的登录状态

   该函数接受一个HttpRequest对象，以及一个认证了的User对象

   此函数使用django的session框架给某个已认证的用户附加上session id等信息。

2. 示例代码

```
def login_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
         # 验证用户是否存在
        user = authenticate(request, username=username,
password=password)
        if user:
```

```
            login(request,user)
             return redirect('/')
        else:
            return render(request, 'test/login.html', {'msg': '用
户密码错误'})
    else:
        return render(request, 'login.html')
```

4、登出操作

1. 说明

   当用户注销的时候用`logout(request)`,只需要一个参数request

2. 示例代码

```
from django.contrib.auth import logout
def logout_view(request):
    logout(request)
```

5、修改密码

1. 说明

2. 示例代码

```
user = auth.authenticate(username=username,
password=old_password)
if user:
    user.set_password(new_password)
    user.save()
```

6、路由保护

1. 说明

   @login_required 修饰器修饰的view函数会先通过session key检查是否登录,已登录用户可以正常的执行操作, 未登录用户将被重定向到login_url指定的位置. 若未指定login_url参数, 则重定向到settings.LOGIN_URL

2. 示例代码

```
# settings 配置
LOGIN_URL = '/day05/login/'
# views
@login_required
def find_user_info(request):
    pass
```

```
@login_required(login_url='/day05/phone/login')
def find_user_info(request):
    pass
```

7、验证登录

1. 说明

   如果是真正的 User 对象，返回值恒为 True 。 用于检查用户是否已经通过了认证。 通过认证并不意味着用户拥有任何权限，甚至也不检查该用户是否处于激活状态，这只是表明用户成功的通过了认证。 这个方法很重要, 在后台用request.user.is_authenticated()判断用户是否已经登录，如果true则可以向前台展示request.user.name

2. 示例代码

```
在后台的视图函数里可以用request.user.is_authenticated()判断用户是否登录
在前端页面中可以用
{% if user.is_authenticated %}
{% endif %}
判断用户是否登录
```