《Django Web框架教学笔记》

目录

Django中的用户认证 (使用Django认证系统)

- Django带有一个用户认证系统。 它处理用户账号、组、权限以及基于cookie的用户会话。
- 作用:
  1. 添加普通用户和超级用户
  2. 修改密码
- 文档参见
- https://docs.djangoproject.com/en/1.11/topics/auth/
- User模型类
- 位置: `from django.contrib.auth.models import User`
- 默认user的基本属性有:

属性名	类型	是否必选
username	用户名	是
password	密码	是
email	邮箱	可选
first_name	名	
last_name	姓	
is_superuser	是否是管理员帐号(/admin)	
is_staff	是否可以访问admin管理界面	
is_active	是否是活跃用户,默认True。一般不删除用户，而是将用户的is_active设为False。	
last_login	上一次的登录时间	
date_joined	用户创建的时间	

- 数据库表现形式

```
mysql> use myauth;
mysql> desc auth_user;
+--------------+--------------+------+-----+---------+----------------+
| 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(30)  | 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 (0.00 sec)
```

auth基本模型操作:

- 创建用户
  - 创建普通用户create_user

```
from django.contrib.auth import models
user = models.User.objects.create_user(username='用户名', password='密码', email='邮箱',...)
```

  - 创建超级用户create_superuser

```
from django.contrib.auth import models
user = models.User.objects.create_superuser(username='用户名',
password='密码', email='邮箱',...)
```

- 删除用户

```
from django.contrib.auth import models
try:
    user = models.User.objects.get(username='用户名')
    user.is_active = False  # 记当前用户无效
    user.save()
    print("删除普通用户成功！")
except:
    print("删除普通用户失败")
return HttpResponseRedirect('/user/info')
```

- 修改密码set_password

```
from django.contrib.auth import models
try:
    user = models.User.objects.get(username='xiaonao')
    user.set_password('654321')
    user.save()
    return HttpResponse("修改密码成功！")
except:
    return HttpResponse("修改密码失败！")
```

- 检查密码是否正确check_password

```
from django.contrib.auth import models
try:
    user = models.User.objects.get(username='xiaonao')
    if user.check_password('654321'):  # 成功返回True,失败返回False
        return HttpResponse("密码正确")
    else:
        return HttpResponse("密码错误")
except:
    return HttpResponse("没有此用户！")
```

auth扩展字段

```
如果需要在默认auth表上扩展新的字段，如phone
1，添加新的应用
2，定义模型类 继承 AbstractUser
3，settings.py中 指明 AUTH_USER_MODEL = '应用名.类名'

#models.py案例
from django.db import models
from django.contrib.auth.models import AbstractUser

# Create your models here.
class UserInfo(AbstractUser):

    phone = models.CharField(max_length=11, default='')
```

```
#settings.py添加配置
AUTH_USER_MODEL = 'user.UserInfo'

#添加用户
from user.models import UserInfo
UserInfo.objects.create_user(username='guoxiao', password='123456',
phone='13488871101')
```

电子邮件发送

- 利用QQ邮箱发送电子邮件
- django.core.mail 子包封装了 电子邮件的自动发送SMT协议
- 前其准备:
    1. 申请QQ号
    2. 用QQ号登陆QQ邮箱并修改设置
        - 用申请到的QQ号和密码登陆到 https://mail.qq.com/
        - 修改 `QQ邮箱->设置->帐户->“POP3/IMAP.....服务”`
    3. 设置Django服务器端的，用简单邮件传输协议SMTP(Simple Mail Transfer Protocol) 发送电子邮件
- settings.py 设置

```
# 发送邮件设置
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend' # 固定写法
EMAIL_HOST = 'smtp.qq.com' # 腾讯QQ邮箱 SMTP 服务器地址
EMAIL_PORT = 25  # SMTP服务的端口号
EMAIL_HOST_USER = 'xxxx@qq.com'  # 发送邮件的QQ邮箱
EMAIL_HOST_PASSWORD = '******'  # 在QQ邮箱->设置->帐户->“POP3/IMAP.....服务” 里得到的在第三方登录QQ邮箱授权码
EMAIL_USE_TLS = True  # 与SMTP服务器通信时，是否启动TLS链接(安全链接)默认false
```

视图函数中

```
 from django.core import mail
mail.send_mail(
            subject,  #题目
            message,  # 消息内容
            from_email,  # 发送者[当前配置邮箱]
            recipient_list=['xxx@qq.com'],  # 接收者邮件列表
            )
```

项目部署

- 项目部署是指在软件开发完毕后，将开发机器上运行的开发板软件实际安装到服务器上进行长期运行
- 部署要分以下几个步骤进行

1. 在安装机器上安装和配置同版本的数据库
2. django 项目迁移(在安装机器上配置与开发环境相同的python版本及依懒的包)
3. 用 uwsgi 替代`python3 manage.py runserver`方法启动服务器
4. 配置 nginx 反向代理服务器
5. 用nginx 配置静态文件路径,解决静态路径问题

1. 安装同版本的数据库
    - 安装步骤略
2. django 项目迁移
    1. 安装python
        - `$ sudo apt install python3`
    2. 安装相同版本的包
        - 导出当前模块数据包的信息:
            - `$ pip3 freeze > package_list.txt`
        - 导入到另一台新主机
            - `$ pip3 install -r package_list.txt`
    3. 将当前项目源代码复制到远程主机上(scp 命令)
        - $ sudo scp 当前项目源代码 远程主机地址和文件夹

```
sudo scp /home/tarena/django/mysite1
root@88.77.66.55:/home/root/xxx
请输入root密码:
```

WSGI Django工作环境部署

- WSGI (Web Server Gateway Interface)Web服务器网关接口，是Python应用程序或框架和Web服务器之间的一种接口，被广泛使用
- 它实现了WSGI协议、http等协议。Nginx中HttpUwsgiModule的作用是与uWSGI服务器进行交换。WSGI是一种Web服务器网关接口。

uWSGI 网关接口配置 (ubuntu 18.04 配置)

- 使用`python manage.py runserver`通常只在开发和测试环境中使用。
- 当开发结束后，完善的项目代码需要在一个高效稳定的环境中运行，这时可以使用uWSGI
- uWSGI是WSGI的一种,它可以让Django、Flask等开发的web站点运行其中.
- 安装uWSGI
    - 在线安装 uwsgi

```
$ sudo pip3 install uwsgi
```

    - 离线安装
        1. 在线下载安装包:

```
$ pip3 download uwsgi
```

            - 下载后的文件为`uwsgi-2.0.18.tar.gz`

2. 离线安装

```
$ tar -xzvf uwsgi-2.0.18.tar.gz
$ cd uwsgi-2.0.18
$ sudo python3 setup.py install
```

- 配置uWSGI
  - 添加配置文件 `项目文件夹/uwsgi.ini`
    - 如: mysite1/uwsgi.ini

```
[uwsgi]
# 套接字方式的 IP地址:端口号
# socket=127.0.0.1:8000
# Http通信方式的 IP地址:端口号
http=127.0.0.1:8000
# 项目当前工作目录
chdir=/home/tarena/.../my_project 这里需要换为项目文件夹的绝对路径
# 项目中wsgi.py文件的目录，相对于当前工作目录
wsgi-file=my_project/wsgi.py
# 进程个数
process=4
# 每个进程的线程个数
threads=2
# 服务的pid记录文件
pidfile=uwsgi.pid
# 服务的目志文件位置
daemonize=uwsgi.log
# 开启主进程管理模式
master=true
```

  - 修改settings.py将 DEBUG=True 改为DEBUG=False
  - 修改settings.py 将　ALLOWED_HOSTS = [] 改为　ALLOWED_HOSTS = ['网站域名'] 或者 ['服务监听的ip地址']
- uWSGI的运行管理
  - 启动 uwsgi

```
$ cd 项目文件夹
$ sudo uwsgi --ini 项目文件夹/uwsgi.ini
```

  - 停止 uwsgi

```
$ cd 项目文件夹
$ sudo uwsgi --stop uwsgi.pid
```

  - 说明:
    - 当uwsgi 启动后,当前django项目的程序已变成后台守护进程,在关闭当前终端时此进程也不会停止。
    - 若执行 stop 操作失败，则需要执行如下操作杀死进程

```
ps aux|grep 'uwsgi'  -> 查看uwsgi进程

tarena   103408  0.0  0.9 137172 39984 ?        S    10:02   0:01
uwsgi --ini uwsgi.ini
tarena   103410  0.0  0.9 436200 38552 ?        Sl   10:02   0:00
uwsgi --ini uwsgi.ini

ps -ef | grep 'uwsgi' | grep -v grep | awk '{print $2}' | xargs
kill -9
```

- 测试:
  - 在浏览器端输入http://127.0.0.1:8000 进行测试
  - 注意，此时端口号为8000

nginx 反向代理配置

- Nginx是轻量级的高性能Web服务器，提供了诸如HTTP代理和反向代理、负载均衡、缓存等一系列重要特性，在实践之中使用广泛。
- C语言编写，执行效率高
- nginx 作用
  - 负载均衡， 多台服务器轮流处理请求
  - 反向代理
- 原理:
- 客户端请求nginx,再由nginx 请求 uwsgi, 运行django下的python代码
- ubuntu 下 nginx 安装
  $ sudo apt install nginx

```
vim /etc/apt/sources.list
更改国内源
sudo apt-get update
```

- nginx 配置
  - 修改nginx 的配置文件 /etc/nginx/sites-enabled/default

```
# 在server节点下添加新的location项，指向uwsgi的ip与端口。
server {
    ...
    location / {
        uwsgi_pass 127.0.0.1:8000;  # 重定向到127.0.0.1的8000端口
        include /etc/nginx/uwsgi_params; # 将所有的参数转到uwsgi下
    }
    ...
}
```

- nginx服务控制

```
$ sudo /etc/init.d/nginx start|stop|restart|status
# 或
$ sudo service nginx start|stop|restart|status
```

> 通过 start,stop,restart,status 可能实现nginx服务的启动、停止、重启、查扑克状态等操作

- 修改uWSGI配置
  - 修改`项目文件夹/uwsgi.ini`下的**Http**通信方式改为**socket**通信方式

```
[uwsgi]
# 去掉如下
# http=127.0.0.1:8000
# 改为
socket=127.0.0.1:8000
```

  - 重启**uWSGI**服务

```
$ sudo uwsgi --stop uwsgi.pid
$ sudo uwsgi --ini 项目文件夹/uwsgi.ini
```

- 测试:
  - 在浏览器端输入http://127.0.0.1 进行测试
  - 注意：
    1，此时端口号为80(nginx默认值)
    2，**Django**中有任何修改 需要重启 **uwsgi**，否则修改不生效

nginx 配置静态文件路径

- 创建新路径-主要存放Django所有静态文件 如： /home/tarena/项目名_static/
- 在Django settings.py 中添加新配置

```
STATIC_ROOT = '/home/tarena/项目名_static/static
#注意 此配置路径为 存放所有正式环境中需要的静态文件
```

- 进入项目，执行 **python3 manage.py collectstatic** 。执行该命令后，Django将项目重所有静态文件 复制到 STATIC_ROOT 中 ， 包括Django内建的静态文件【如admin后台的样式】
- Nginx配置中添加新配置

```
# file : /etc/nginx/sites-enabled/default
# 新添加location /static 路由配置，重定向到指定的 第一步创建的路径即可
server {
    ...
    location /static {
        # root 第一步创建文件夹的绝对路径,如:
         root /home/tarena/项目名_static;
    }
    ...
}
```

404/500 界面

- 在模板文件夹内添加 404.html 模版，当视图触发Http404 异常时将会被显示
- 404.html 仅在发布版中(即setting.py 中的 DEBUG=False时) 才起作用
- 当向应处理函数触发Http404异常时就会跳转到404界面

```
from django.http import Http404
def xxx_view( ):
    raise Http404  # 直接返回404
```

邮件告警

- 当正式服务器上代码运行有报错时，可将错误追溯信息发至指定的邮箱
- 配置如下 settings.py中

```
#在基础邮件配置之后 添加如下

#关闭调试模式
DEBUG = False

#错误报告接收方
ADMINS = [('guoxiaonao', 'xxxx@example.com'), ('wanglaoshi',
'xxxx@example.com')]

#发送错误报告方，默认为 root@localhost账户，多数邮件服务器会拒绝
SERVER_EMAIL = 'email配置中的邮箱'
```

- 过滤敏感信息

报错邮件中会显示一些错误的追踪，这些错误追踪中会出现如 password等敏感信息，Django已经将配置文件中的敏感信息 过滤修改为 多个星号，但是用户自定义的视图函数需要用户手动过滤敏感信息

1，视图函数中的局部变量

```
from django.views.decorators.debug import sensitive_variables

@sensitive_variables('user', 'pw', 'cc')
def process_info(user):
    pw = user.pass_word
    cc = user.credit_card_number
    name = user.name
    ...
#注意：
#1 若报错邮件中牵扯到user,pw,cc等局部变量的值，则会将其替换成 *****，而 name 变量还显示其真实值
#2 多个装饰器时，需要将其放在最顶部
#3 若不传参数，则过滤所有局部变量的值
```

2，POST提交中的数据

```
from django.views.decorators.debug import sensitive_post_parameters

@sensitive_post_parameters('password', 'username')
def index(request):
    s = request.POST['username'] + request.POST['abcd']
    #'abcd' 并不存在，此时引发error
#POST中 username 及 password的值会被替换成 ******
```