

《Django Web框架教学笔记》

目录

《Django Web框架教学笔记》

- 目录
- 文件上传
- Django中的用户认证 (使用Django认证系统)
 - auth基本模型操作:
 - auth扩展字段
- 电子邮件发送
- 项目部署
 - uWSGI 网关接口配置 (ubuntu 18.04 配置)
 - nginx 及反向代理配置
 - nginx 配置静态文件路径

文件上传

- 文件上传必须为POST提交方式
- 表单 `<form>` 中文件上传时必须带有 `enctype="multipart/form-data"` 时才会包含文件内容数据。
- 表单中用 `<input type="file" name="xxx">` 标签上传文件
 - 名字 `xxx` 对应 `request.FILES['xxx']` 对应的内存缓冲文件流对象。可通过 `request.FILES['xxx']` 返回的对象获取上传文件数据
 - `file=request.FILES['xxx']` `file` 绑定文件流对象，可以通过文件流对象的如下信息获取文件数据
 - `file.name` 文件名
 - `file.file` 文件的字节流数据
- 上传文件的表单书写方式

```
<!-- file: index/templates/index/upload.html -->
<html>
<head>
  <meta charset="utf-8">
  <title>文件上传</title>
</head>
<body>
  <h3>上传文件</h3>
  <form method="post" action="/test_upload" enctype="multipart/form-data">
    <input type="file" name="myfile"/><br>
    <input type="submit" value="上传">
  </form>
</body>
</html>
```

- 在 `setting.py` 中设置MEDIA相关配置；Django把用户上传的文件，统称为media资源

```
# file : settings.py
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

- 在当前项目文件夹下创建 `media` 文件夹

```
$ mkdir media
```

- 上传文件的视图处理函数 方案1 传统写入

```
# file views.py
from django.http import HttpResponse
from django.conf import settings
from django.views.decorators.csrf import csrf_exempt
import os

@csrf_exempt
def upload_view(request):
    if request.method == 'GET':
        return render(request, 'test_upload.html')
    elif request.method == "POST":
        a_file = request.FILES['myfile']
        print("上传文件名是:", a_file.name)
        filename = os.path.join(settings.MEDIA_ROOT, a_file.name)
        with open(filename, 'wb') as f:
            data = a_file.file.read()
            f.write(data)
        return HttpResponse("接收文件:" + a_file.name + "成功")
```

- 上传文件的视图处理函数 方案2 借助orm

```
#test_upload/models.py
from django.db import models

# Create your models here.
class Content(models.Model):

    desc = models.CharField(max_length=100)
    myfile = models.FileField(upload_to='myfiles')

#test_upload/views.py
from test_upload.models import *
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def upload_view_dj(request):
    if request.method == 'GET':
        return render(request, 'test_upload.html')
    elif request.method == 'POST':
        title = request.POST['title']
        a_file = request.FILES['myfile']
        Content.objects.create(desc=title, myfile=a_file)
        return HttpResponse('----upload is ok----')
```

- 若要在浏览器中访问 上传的资源，runserver环境下，需要在项目得主路由下添加media路由的绑定

```
from django.conf import settings
from django.conf.urls.static import static
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

浏览器可以访问 <http://127.0.0.1:8000/media/xxxx>

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/2.2/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	

auth基本模型操作:

- 创建用户
 - 创建普通用户create_user

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

- 删除用户

```
from django.contrib.auth.models import User
try:
    user = 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")
```

- 修改密码set_password

```
from django.contrib.auth.models import User
try:
    user = 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.models import User
try:
    user = 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SMTP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环境
2. django 项目迁移

\$ 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:
```

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

uWSGI 网关接口配置 (ubuntu 18.04 配置)

- 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
- 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WSGI
- uWSGI是WSGI的一种, 它实现了 http协议 WSGI协议 以及 uwsgi协议
- 安装uWSGI

- 终端输入如下命令

```
sudo pip3 install uwsgi==2.0.18 -i
https://pypi.tuna.tsinghua.edu.cn/simple/
```

- 检查是否安装成功

```
sudo pip3 freeze|grep -i 'uwsgi'
#如果成功安装，则会输出
uWSGI==2.0.18
```

- 配置uWSGI

- 添加配置文件 项目同名文件夹/uwsgi.ini
 - 如: mysite1/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
# 服务的日志文件位置
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

```

$ 进入到项目同名文件夹下 【即settings.py所在目录】
$ sudo uwsgi --ini uwsgi.ini

```

- 停止 uwsgi

```

$ 进入到项目同名文件夹下 【即settings.py所在目录】
$ 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 ?        S1   10:02   0:00
uwsgi --ini uwsgi.ini

ps -ef | grep 'uwsgi' | grep -v grep | awk '{print $2}' | xargs
sudo 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 及反向代理配置

- 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 将请求转发 uWSGI 运行的django
- 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、操作

- 修改uWSGI配置
 - **修改** 项目同名文件夹/`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**

```
进入到 项目同名文件夹下
$ sudo uwsgi --stop uwsgi.pid
$ sudo uwsgi --ini 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;
    }
    ...
}
```