

# 《Django 教程》

---

- 讲师: 魏明择
- 时间: 2019

## 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:
  - Web请求中的认证
- 项目部署
  - WSGI Django工作环境部署
  - uWSGI 网关接口配置
  - nginx 反射代理配置
  - nginx 配置静态文件路径
  - 404 界面

## 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。可以通过`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
- 如下上传文件为图片类型, 可以用模块类属性定义成models.ImageField类型
  - `image_file = models.ImageField(upload_to='images/')`
  - 注意: 如果属性类型为ImageField需要安装包Pillow
  - `pip install Pillow==3.4.1`
  - 图片存储路径
- 练习: 在项目根目录下创建static文件夹 图片上传后, 会被保存到"/static/files/"下 打开settings.py文件, 增加MEDIA\_ROOT项 `MEDIA_ROOT=os.path.join(BASE_DIR,"static/files")` 使用django后台管理, 遇到ImageField类型的属性会出现一个file框, 完成文件上传 手动上传的模板代码
- 上传文件的表单书写方式

```
<!-- file:static/upload.html -->
<html>
<head>
```

```

<title>文件上传</title>
<meta charset="utf-8">
</head>
<body>
  <h1>上传文件</h1>
  <form method="post" action="/upload" enctype="multipart/form-data">
    <input type="file" name="myfile"/><br>
    <input type="submit" value="上传">
  </form>
</body>
</html>

```

- 在setting.py 中设置一个变量MEDIA\_ROOT 用来记录上传文件的位置

```

# file : settings.py
...
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]
MEDIA_ROOT = os.path.join(BASE_DIR, 'static/files')
...

```

- 在当前项目文件夹下创建 `static/files` 文件夹

```
$ mkdir -p static/files
```

- 添加路由及对应的处理函数

```

# file urls.py
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^upload', views.upload_file)
]

```

- 上传文件的视图处理函数

```

# file views.py
from django.http import HttpResponse, Http404
from django.conf import settings
import os

# from django.views.decorators.http import require_POST
# @require_POST
def upload_file(request):

```

```
if request.method == "POST":
    file = request.FILES['myfile']
    print("上传文件名是:", file.name)

    filename = os.path.join(settings.MEDIA_ROOT, file.name)
    with open(filename, 'wb') as f:
        f.write(file.file.read())
    return HttpResponse("接收文件成功")
raise Http404
```

- 访问地址: <http://127.0.0.1:8000/static/upload.html>

## 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
  3. 登陆和退出管理
  4. 查看已登陆用户
- 文档参见
  - [https://yiyibooks.cn/xx/Django\\_1.11.6/topics/auth/default.html](https://yiyibooks.cn/xx/Django_1.11.6/topics/auth/default.html)
- 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	是否必须存在
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	

属性名	类型	是否必须存在
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',...)
...
user.save()
```

- 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',...)
...
user.save()
```

- 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='laowei')
    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='laowei')
    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! ")
```

## Web请求中的认证

- 相关函数
  - from django.contrib.auth import authenticate, login, logout
- 认证用户
  - 函数 authenticate(username='用户名', password='原始密码')
  - 作用验证用户名和密码是否存在于用户中.如果存在返回User对象, 如果不合法返回None
  - 示例

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # 通过认证
else:
    # 没有通过认证
```

- 登陆用户

- 函数 login(request, user)
- 参数request 为一个HttpRequest对象
- 参数 user 为一个User对象
- login()使用Django的session框架来将用户的ID保存在session中。
- 示例

```
from django.contrib.auth import authenticate, login

def login_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username,
                        password=password)
    if user is not None:
        login(request, user)
        # 重新指定登陆成功页
        ...
    else:
        # 返回一个无效的登陆信息
        ...
```

- 登出用户

- 函数 logout(request)
- 通过已经通过django.contrib.auth.login()登入的用户，可以使用django.contrib.auth.logout()退出登陆
- 示例：

```
from django.contrib.auth import logout

def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

§ #### 自定义User模型

- 当django自带的 `django.contrib.auth.models.User` 的属性不能满足我们现有的需求时，可以自定义User模型
- 如，向用户中加入性别和家庭住址字段等
- 自定义User 类需要继承自 `django.contrib.auth.models.AbstractUser`, 如:

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

class User(AbstractUser):
    address = models.CharField("手机号", max_length=100)
    ...
```

- 自定义User类后需要再次做迁移操作
- 文档参见[https://yiyibooks.cn/xx/Django\\_1.11.6/topics/auth/customizing.html](https://yiyibooks.cn/xx/Django_1.11.6/topics/auth/customizing.html)

## 项目部署

- 项目部署在软件开发完毕后，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版本及依赖的包)
  3. 用 uwsgi 替代 `python3 manage.py runserver` 方法启动服务器
  4. 配置 nginx 反射代理服务器
  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 -a 当前项目源代码 远程主机地址和文件夹`
- 3.

## 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

- 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 `$ sudo pip3 install uwsgi`
- 配置uWSGI

file: 项目文件夹/uwsgi.ini # 如: mywebsite1/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/weimz/my\_django\_project ... 这里需要换为项目地址

项目中wsgi.py文件的目录，相对于当前工作目录

---

wsgi-file=my\_django\_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

- 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。

- 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 反射代理配置

- 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
  - 反射代理
- 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
- nginx 配置
  - 修改nginx 的配置文件 /etc/nginx/sites-available/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
    }
    ...
}
```

- 启动 nginx

- `$ sudo /etc/init.d/nginx start`
- 或
- `$ sudo service nginx restart`

- 查看nginx进程

- `$ ps aux | grep nginx`
- 或
- `$ sudo /etc/init.d/nginx status`
- 或
- `$ sudo service nginx status`

- 停止nginx

- `$ sudo /etc/init.d/nginx stop`
- 或
- `$ sudo service nginx stop`

- 重启nginx

- `$ sudo /etc/init.d/nginx restart`
- 或
- `$ sudo service nginx restart`

- 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,如:

```
[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, 此时端口号为80(nginx默认值)

## nginx 配置静态文件路径

- 解决静态路径问题

```
# file : /etc/nginx/sites-available/default
# 新添加location /static 路由配置, 重定向到指定的绝对路径
server {
    ...
    location /static {
        # root static文件夹所在的绝对路径, 如:
        root /home/weimz/my_django_project; # 重定向/static请求的路径, 这里改为你项目的文件夹
    }
    ...
}
```

- 修改配置文件后需要重新启动 nginx 服务

## 404 界面

- 在模板文件夹内添加 404.html 模版, 当响应返回HttpResponseNotFound 或 raise Http404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
- 当响应处理函数触发Http404异常时就会跳转到404界面

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