Django完全教程

从一个例子开始

我们需要理解，一个web应用的本质是处理浏览器request请求返回html页面的容器。开始浏览器产生一个http请求，服务器接收到请求后，生成html文档作为返回数据的body，返回给浏览器，浏览器提取数据进行显示。常用的web容器有apache，tomcat等。如果要动态生成html内容，就需要手动处理上述过程，wsgi定义了一组接口，它要求开发者写出符合该标准的应用程序，wsgi来处理底层http处理的细节。使用原生wsgi编写程序依然十分繁琐，所以出现了功能不断完善的web框架，让程序员免于在底层信息处理中迷失。python最著名的网络框架有两个，一个是flask，小巧轻便，灵活，另一个是Django，框架较为庞大，功能扩展多。在搭建web应用的时候，经常碰到不同项目python包冲突的情况，为此我们使用虚拟环境。虚拟环境pyenv可以，官方的virtualenv亦可，都可以达到项目与系统分离的目的。

django的运行流程

python web框架一般运行于web容器之中，各个容器都可以配置wsgi标准应用，与web框架进行耦合。apache安装mod_wsgi模块后，可以将浏览器发送的httprequest请求以wsgi标准的形式发送给web后端，web后端处理该请求，返回response，其中的操作包括url映射、通过ORM技术从数据库中取出数据、模板渲染，最后将生成的response内容返回给web容器，由web容器返回给浏览器

django的安装

在虚拟环境中使用pip即可，如果我们要安装特定版本的

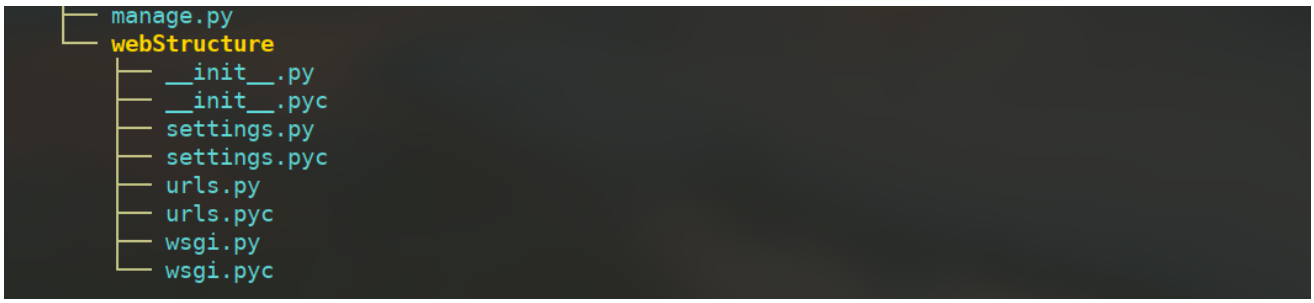
```
pip install django==1.8.3
```

django项目结构与基本概念

项目是django中最为宏观的概念，即web项目。一个web应用可能需要许多不同的功能模块儿，这些模块儿叫做application。在处理request和response时如果我们希望可以自己对它们做出一些更改，需要通过中间件来完成，使用loader从模板中加载内容。使用如下命令创建一个web应用，名称为webStructure

```
django-admin startproject webStructure
```

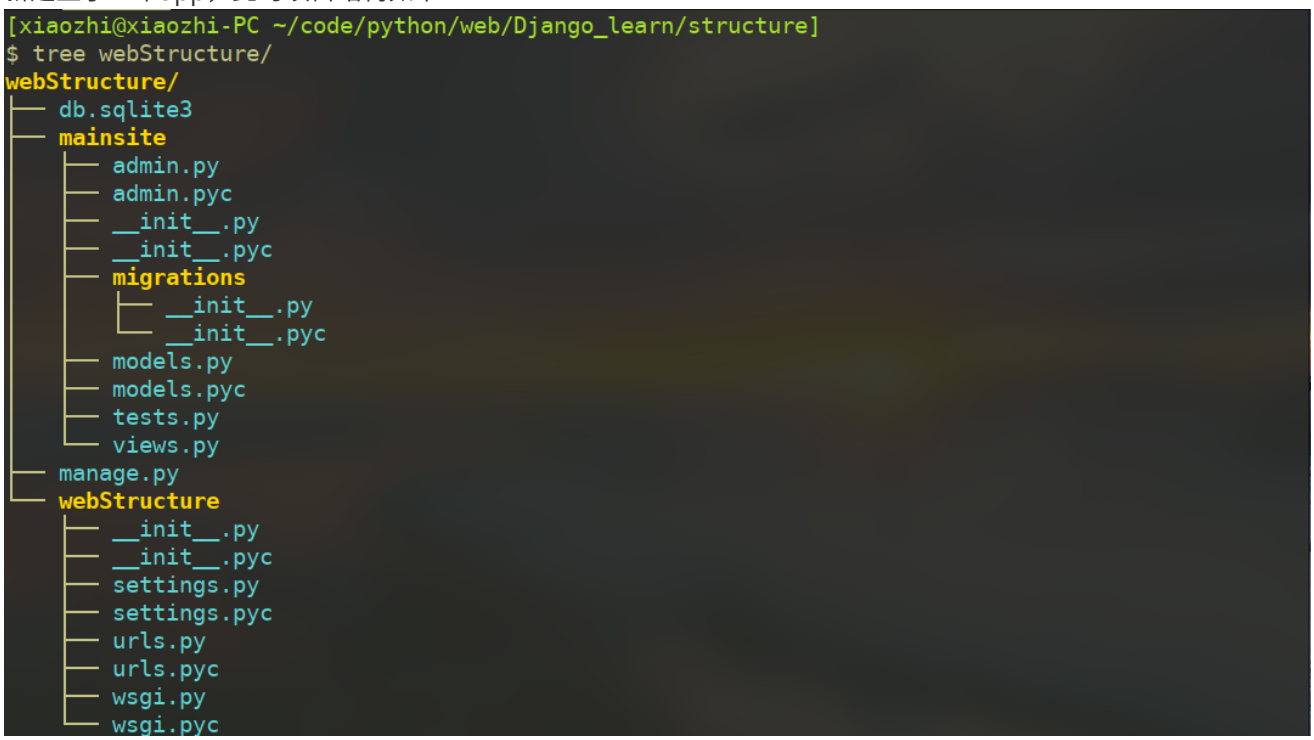
该操作会在当前目录下生成一个名为webStructure的文件夹，其目录结构如下



manager.py是django用来管理项目的工具程序，可以接收命令行参数，在项目内的所有命令都需要该文件的支持。二层目录webStructure下，urls.py文件定义了url与处理url的视图函数之间的映射关系，settings定义了项目配置信息，wsgi则是djangoweb应用的wsgi接口，该文件是web项目的启动入口。在执行以上操作后，我们只是有了一个大体的框架，没有加入任何的功能，进入webStructure目录，使用如下命令新建一个application

```
django-admin startapp mainsite
```

新建立了一个app，此时项目结构如下



views.py定义了一系列处理url请求的函数，称为视图，models.py定义了网站要使用的数据结构，这里需要说明一下，django使用ORM技术存取数据库信息，这样做的好处是我们无需关心数据库类型，无需关心连接数据库的语句是那些，只要我们按照一定的标准定义数据结构，django自动帮我们完成数据库的操作。mainsite下的migrations文件夹下存放的文件是网站migrate的历史记录，什么是migrate呢，我们定义了models数据结构，要将models文件中的数据结构与数据库关联起来，需要执行migrate操作。在setting文件中，加入我们刚才创建的app

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mainsite',
)
```

设置时区与语言

```
LANGUAGE_CODE = 'zh-hans'

TIME_ZONE = 'Asia/Shanghai'
```

使用下面的命令执行migrate操作，创建对应的数据库与数据表

```
python manage.py migrate
```

看到如下输出

```
Synchronize unmigrated apps: staticfiles, messages
Apply all migrations: admin, contenttypes, auth, sessions
Synchronizing apps without migrations:
Creating tables...
  Running deferred SQL...
Installing custom SQL...
Running migrations:
Rendering model states... DONE
Applying contenttypes.0001_initial... OK
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying sessions.0001_initial... OK
```

每次对models文件做出更改时，都要执行该操作以将更改同步至数据库中。另外，在初始的时候，执行makemigrations命令以创建django与数据库之间的中间文件。该操作将在app目录下创建migrations目录，并在该目录下记录我们对models文件的所有改动（未同步至数据库），执行migrate后，该操作都会同步至数据库中。

django中的models

models文件中定义了网站使用的数据结构，我们在models中定义数据，并通过migrate同步至数据库，随后我们对该类型数据的操作，可以反映至数据库中，这是使用ORM的便捷之处。django默认使用sqlite数据库，如果需要使用其他的数据库类型，需要在settings文件中配置相关选项，并且安装相应的python驱动。使用如下代码在models中定义一个类型

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
from django.utils import timezone
from django.db import models

# Create your models here.
class Post(models.Model):
    title = models.CharField(max_length=200)
    slug = models.CharField(max_length=200)
    body = models.TextField()
    pub_date = models.DateTimeField(default=timezone.now)

```

```
    def __unicode__(self):
        return self.title

    class Meta:
        ordering = ('-pub_date',)
```

同步至数据库中

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

启用admin管理界面

django项目创建之时，有web页面管理功能的支持，在使用这个功能之前，需要新建一个管理员用户

```
python manage.py createsuperuser
```

输入用户名与密码创建超级账户

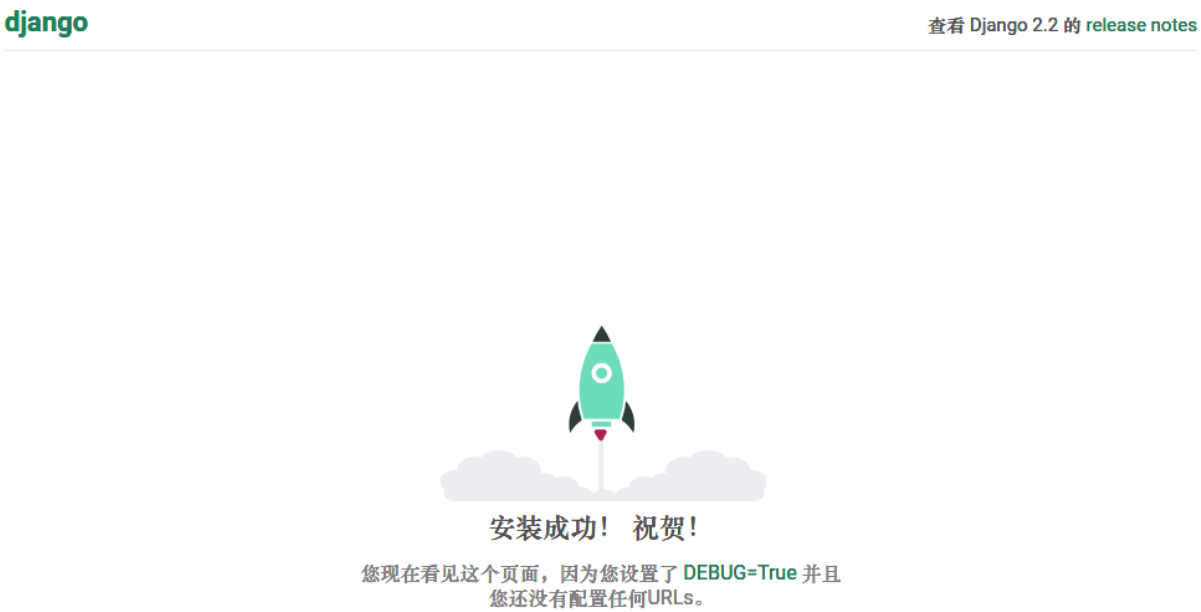
```
[xiaozhi@xiaozhi-PC ~/code/python/web/Django_learn/structure/webStructure]
$ python manage.py createsuperuser
用户名 (leave blank to use 'xiaozhi'): xiaozhi
电子邮件地址: test@gmail.com
Password:
Password (again):
Superuser created successfully.
```

在admin文件中，注册我们刚在创建的Post类，让我们可以在web控制台操作该对象

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.contrib import admin
from .models import Post
# Register your models here.
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'pub_date')
admin.site.register(Post, PostAdmin)
```

其中PostAdmin类为我们为了更改默认的显示方式而创建的，使用migrate命令将内容同步至数据库中，随后使用如下命令启动测试模式 `python manage.py runserver 127.0.0.1:8000` 访问根网页，我没没有为根目录设置任何内容，下面的结果只是告诉我们服务器运行成功了



访问管理页面`http://127.0.0.1:8000/admin`,看到如下图所示的登录窗体，使用我们刚才创建的管理员账户登录即可

登录以后看到我们刚才创建的Post类

可以在web控制台手动添加Post实例，这些更改都会被同步到数据库中。

django中的视图

一般来说，web应用包含两部分，前端页面和后端逻辑。前台通过post或get请求与后台交互，后台根据传递的参数进行系列动作，返回需要的内容。在Django中，这些后台逻辑，就是视图需要完成的工作。表面上看，视图是定义在views.py文件中的一系列函数，当用户请求某个网址时，根据urls.py文件中的动态网址映射到某个视图函数，执行某个动作，并返回HTTPResponse对象，返回给客户端

django中的模板

html页面的内容应该是可重用的，页面是动态的、可渲染的，才有可能建立动态的网站，python中的jinja2模板引擎很好的应用在了web开发中，将原始html页面看做模板，模板中有一些变量，变量的内容会根据运行时视图函数传递过来的参数而变化，从而达到动态渲染的目的。这样方便了后端程序传值，但是网页解析与特定模板引擎绑定，并且最近几年逐渐出现了前后端分离的模式，尽量采用前端原生网页的形式。

django中的模型

模型是网站要使用的数据结构，比如用户模型，定义了用户这个类别的数据结构，可能包含用户名、密码、邮箱地址等等信息。本来这些信息需要自己组织在数据库的表中，用户认证时，我们从数据库表中取出对应内容进行比对。Django使用ORM技术为我们简化了上述过程，我们仅需创建一个类，包含必要的字段，django会自动为我们在数据库中创建该类对应的表，而且为我们简化数据查询操作，这些类，定义自models.py文件中

创建我们的视图函数

在mainsite下的views.py文件中，创建一个函数，处理http请求

```
from .models import Post
from datetime import datetime
from django.http import HttpResponse
from django.template.loader import get_template
def homepage(request):
    # 查询对应的内容
    posts = Post.objects.all()
    now = datetime.now()
    # get_template是Django中一个加载模板的函数，使用之前需先导入
    template = get_template('mainsite/index.html')
    # render函数用来渲染html页面，将必要的数据传过去，locals（）是python的一个全局
    # 函数，表示所有本地变量字典
    html = template.render(locals())
    return HttpResponse(html)
```

函数必须有一个固定参数request，表示请求上下文， 从中可以获取http请求的所有信息

get_template是django提供的快捷函数，获取模板文件，其render函数将变量渲染进模板中，最后返回HttpResponse对象。

配置模板

在使用模板之前，需要配置django模板文件目录，当收到读取模板请求，django默认会去各个app目录下寻找templates目录，从该目录中找对应名称的模板文件，我们有多个app的时候，为了放置模板名称冲突，最好在模板目录下新建app名称目录，将模板文件放在该目录中，在此例子中，配置项目根目录下的templates目录为模板目录

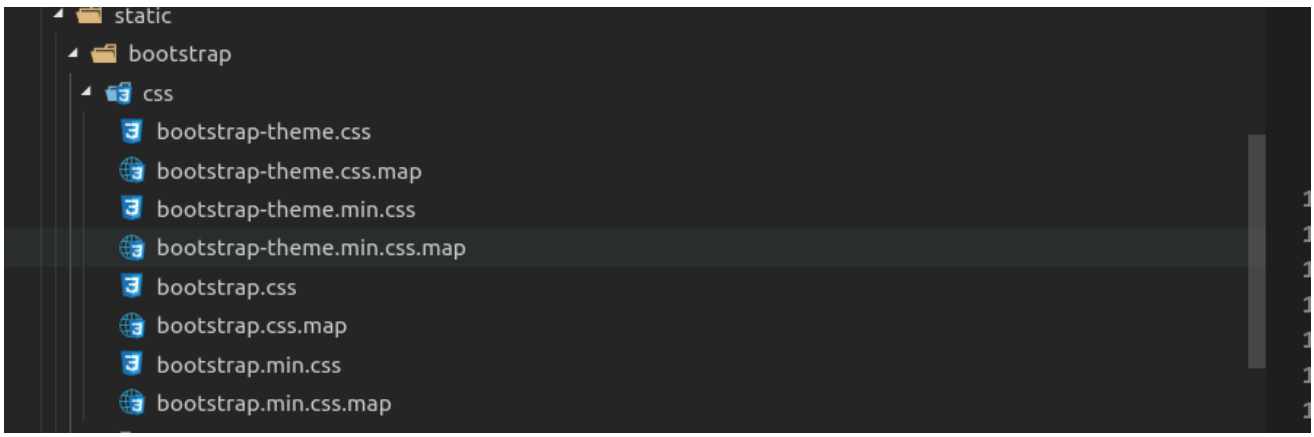
```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

静态文件目录

另外，在django的模板文件中使用静态文件比如css，js，图片资源时，不能简单的使用相对路径引入，而要用static请求请求资源。配置静态文件路径如下

```
STATICFILES_DIRS=[
    os.path.join(BASE_DIR, 'static'),
]
```

在静态目录static中建立相应文件夹，并放入静态资源文件，供模板使用



编写模板文件

使用jinjia2模板的继承功能，base.html文件内容

```
<!DOCTYPE html>
<html>
{% load staticfiles %}

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap.css' %}">
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap-grid.css' %}">
```

```
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap-grid.min.css' %}">
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap-reboot.css' %}">
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap-reboot.min.css' %}">
    <title>
        {% block title %} {% endblock %}
    </title>
</head>

<body>
    <div class="container-fluid">
        {% include "mainsite/header.html" %}
        <div class="row">
            <div class="col-sm-4">
                <div class="card">
                    <div class="card-header">
                        <h3>MENU</h3>
                    </div>
                    <div class="card-body">
                        <div class="list-group">
                            <a href='/' class="list-group-item">HOME</a>
                            <a href='#' class="list-group-item">实时新闻</a>
                            <a href='#' class="list-group-item">电视新闻</a>
                        </div>
                    </div>
                </div>
            </div>
            <div class="col-sm-8">
                <div class="card">
                    <div class="card-header">
                        {% block headmessage %}{% endblock %}
                    </div>
                    <div class="card-body">
                        {% block content %}{% endblock %}
                    </div>
                    <div class="card-footer">
                        {% include "mainsite/footer.html" %}
                    </div>
                </div>
            </div>
        </div>

    </div>
    <script src="https://cdn.bootcss.com/jquery/3.2.1/jquery.min.js"></script>
    <script src="{% static 'bootstrap/js/bootstrap.bundle.js' %}"></script>
    <script src="{% static 'bootstrap/js/bootstrap.bundle.min.js' %}"></script>
    <script src="{% static 'bootstrap/js/bootstrap.js' %}"></script>
    <script src="{% static 'bootstrap/js/bootstrap.min.js' %}"></script>
</body>

</html>
```

index.html文件如下

```
{% extends "mainsite/base.html" %}
{% block title %} 欢迎光临我的博客 {% endblock %}
{% block headmessage %}
<h3>本站文章列表</h3>
{% endblock %}
{% block content %}
    {% for post in posts %}
        <div class="card">
            <div class="card-header">
                <p>
                    <a href="/post/{{post.slug}}">{{post.title}}</a>
                </p>
            </div>
            <div class="card-body">
                <p>
                    {{ post.body | truncatechars:40 }}
                </p>
            </div>
            <div class="card-footer">
                <p>
                    发布时间:{{ post.pub_date | date:"Y M d, h:m:s"}}
                </p>
            </div>
        </div>

{% endfor %}
{% endblock %}
```

header.html

```
<h1 style="font-family: 'Microsoft Sans Serif'">欢迎光临，文学天地</h1>
```

footer.html

```
{% block footer %}
{% if now %}
<p style="font-family: 'Microsoft Sans Serif'">现在时刻: {{ now }}</p>
{% else %}
<p style="font-family: 'Microsoft Sans Serif'">内容来自网络，如有侵权请通知下架。。。</p>
{% endif %}
{% endblock %}
```

使用之前在admin管理界面为django添加几条数据

配置url映射

在webStructure目录下的urlspy添加url映射

```
from django.contrib import admin
from django.urls import path
from django.conf.urls import url, include
from mainsite.views import homepage
urlpatterns = [
    url('^$', homepage),
    path('admin/', admin.site.urls),
]
```

django网站部署

django 的manage.py的runserver参数可以在本地启动单线程的调试服务器，但正常项目上线的时候都使用web容器+wsgi+django的方式，web容器可以选择apache2或者nginx，作为解析http请求的应用，wsgi处理web容器转发给django的请求，最后django应用响应http请求

在centos中配置django+apache

python环境配置

```
unzip webStructure.zip
cd webStructure
```

使用yum安装的python没有sqlite支持，并且为了python版本的管理，我们使用pyenv工具

```
https://github.com/pyenv/pyenv-installer.git
```

使用该工具安装pyenv

```
curl https://pyenv.run | bash
```

编辑.bashrc

```
export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

重新登录shell激活pyenv工具，pyenv install 3.7.0会从源码镜像中下载3.7.0源码包并编译安装，可以先下载该文件，放入pyenv缓存中

```
[root@python- ~]
$ pyenv install 3.7.0
Downloading Python-3.7.0.tar.xz...
-> https://www.python.org/ftp/python/3.7.0/Python-3.7.0.tar.xz
```

如

```
makedir ~/.pyenv/cache
cd ~/.pyenv/cache
wget  https://www.python.org/ftp/python/3.7.0/Python-3.7.0.tar.xz
```

随后再安装3.7.0版本python时不需要再重新下载，另外编译安装python的时候需要一些系统依赖，比如C编译环境，以及一些库的源码包

```
yum install sqlite-devel sqlite readline readline-devel readline-static openssl openssl-devel
openssl-static bzip2-devel bzip2-libs mysql-devel libffi-devel make gcc xz-devel
```

编译可能要花费一点时间，需要耐心等待，安装完成后切换回项目目录，创建虚拟环境，安装依赖，并且运行django进行测试。安装django时，最好使用2.1版本，否则会因为sqlite版本不满足要求服务无法运行

```
cd /var/www/webStructure
pyenv virtualenv 3.7.0 django_deps
pyenv activate django_deps
pip install -r requirements -i https://pypi.tuna.tsinghua.edu.cn/simple
python manage.py runserver
```

修改**settings.py**

关闭调试模式，放通主机访问

```
DEBUG = False

ALLOWED_HOSTS = ['*']

```

设置static_root

```
STATIC_URL = '/static/'
STATICFILES_DIRS=[
    os.path.join(BASE_DIR, 'static'),
]
STATIC_ROOT = '/var/www/static'
```

apache服务配置

安装apache插件

```
mod_wsgi mod_proxy_uwsgi
```

随后在httpd主配置文件中加载该模块

```
# LoadModule foo_module modules/mod_foo.so
LoadModule wsgi_module modules/mod_wsgi.so
"conf/httpd.conf" 354L, 11797C written
```

在conf.d目录中新建django网站配置

```
<VirtualHost *:80>
```

```
2
3  WSGIScriptAlias / /var/www/webStructure/webStructure/wsgi.py
4  Alias /static/ /var/www/static/
5
6  ServerName 192.168.56.103
7  #ServerName example.com
8  #ServerAlias www.example.com
9
10 <Directory /var/www/static>
11     Options Indexes  FollowSymLinks
12     AllowOverride None
13     Require all granted
14 </Directory>
15
16 <Directory /var/www/webStructure/>
17     Require all granted
18 </Directory>
19     ErrorLog   /etc/httpd/logs/django.error.log
20     LogLevel warn
21 </VirtualHost>
22
```

在django项目目录下收集静态文件

```
1  python manage.py collectstatic
```

```
(django_deps) [root@python- /var/www/webStructure]
$ python manage.py collectstatic

136 static files copied to '/var/www/static'.
(django_deps) [root@python- /var/www/webStructure]
$
```

在centos中配置django + uwsgi + nginx

创建python虚拟环境此处就不再赘述了，需要安装pyenv，建立虚拟python环境，安装相应的django依赖，保证python manage.py runserver可以正常启动

安装uwsgi

```
1  pip install uwsgi
```

编辑wsgi配置文件，在项目目录中（项目名称为mysite，使用socket方式）

```
1  [uwsgi]
2  # Django-related settings
3  # the base directory (full path)
4  chdir           = /root/django_projs/mysite
5  # Django's wsgi file  django项目中的wsgi文件
6  module          = mysite.wsgi
7  # the virtualenv (full path)python虚拟环境的目录
```

```
home            = /root/.pyenv/versions/django_deps
# process-related settings
# master
master          = true
# maximum number of worker processes
processes       = 1
# the socket (use the full path to be safe
socket          = :8000
# ... with appropriate permissions - may be needed
# chmod-socket    = 664
# clear environment on exit
vacuum          = true
```

保证uwsgi --ini wsgi.ini可以正常启动并接收请求

配置nginx

```
yum install nginx
```

在nginx的默认配置文件中，加入如下选项

```
    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
        include uwsgi_params;
        uwsgi_pass 127.0.0.1:8000;
    }
```

表示加载uwsgi参数，并将/的请求转发给uwsgi，nginx中也常采用upstream的方式。此时使用nginx可以访问uwsgi接口