
Table of Contents

简介	1.1
准备工作	1.2
创建和使用虚拟环境	1.2.1
创建并配置项目	1.2.2
最最基本的博客	1.3
运行项目	1.3.1
模板语言和学习方法	1.4
博客的类型和标签	1.5
自定义模板标签	1.6
模板拓展方法	1.7
写博客项目的技巧	1.8

简介

本书是关于使用Django2.1搭建博客的流程。

什么是Django?

Python的一个web框架。

好了说完了。

与其说Django是什么，不如说成是**Django**可以做什么？

Django可以开发API接口、网站的管理后台、也可以展示前端页面。

其实讲那么多也都是废话。

动手才是最实际的。

不过动手前，可以先了解一下到底什么是博客。

如果你有python3基础的话，可以把这个[最基础的博客](#)。git下来先尝试一下。

再决定是否继续阅读后面的内容。启动方法和安装方法都放到了项目的README.md里面。

没有python3基础的话，先去学习变量、函数、字典、包的引用、类继承、for、if后，再阅读本系列教程。

博客

博客简单来说，就类似于记事本、日记、长篇微博、QQ空间之类的网站。

不过，它的后台和前端页面都是我们自己手写的。

如果你有打开上面的项目的源码进行浏览，会发现templates目录下的blog_list.html和blog_detail.html里面的代码也就几句话，直接使用浏览器打开的话只是几个 {{ xxxx }}

而我们运行项目并打开<http://localhost:8000/django-1>,却可以看到本篇博文。

这是为什么呢？？

继续浏览后面并动手的话，你就会知道为什么了。

准备工作

请确保项目是在虚拟环境中运行。这里使用的是virtualenv来创建python虚拟环境
如果你会使用和创建虚拟环境，可以跳过本节内容。

安装virtualenv

如果安装了virtualenv可以跳过。

```
sudo apt-get install python-virtualenv
```

上面代码不需要指定python版本。

创建虚拟环境

请找到你的python3安装路径的路径。

linux的python3路径一般在**/usr/bin/python3**

最好是专门创建一个目录来存放虚拟环境的文件。

这里我们选择和django项目同级吧，方便我们调用。

首先，确定你的项目要放在哪里。例如我要放在桌面。

在桌面新建一个目录，例如：work_space。

然后进入该目录。

```
在终端输入 virtualenv -p /usr/bin/python3 venv
```

注意，终端的操作路径是在**work_space**目录下，不是在根目录下。

这时，可以看到**work_space**目录中多了一个**venv**的目录。

这就是我们的虚拟环境文件目录。

进入和退出虚拟环境

```
在终端输入 source /home/username/Desktop/work_space/venv/bin/activate
```

请把上面的路径替换成你虚拟环境目录的路径。

不知道的可以手动点开**venv**然后点开**bin**后，按下 **Ctrl** 和 **L** 来查看虚拟环境的路径

记得在后面加上**activate**。

成功后会发现你的终端命令提示符会多了(**venv**)

如果需要退出虚拟环境，在终端输入 **deactivate** 即可。

安装Django2.1

请确保在虚拟环境中进行以下的操作。

```
pip install django==2.1
```

安装完成后把终端的操作路径切换到你想创建项目的地方。

如果是之前看了创建和使用虚拟环境

就在work_space目录下创建。

创建Django项目

```
django-admin startproject mysite
```

上面的mysite是项目名称，你也可以换成你喜欢的名字。

```
cd mysite && ls
```

 进入项目并查看我们的项目内容

发现里面只有一个名为mysite的目录和一个名为manage.py的文件。

网上很多教程都是说在终端输入 `python manage.py runserver`

把项目运行起来，看到一个Debug页面，然后创建应用什么的，然后在页面打印个**hello world**。

之前我就是看这些教程，把我的思维搞混乱了。

所以现在我不打算讲这些内容，直接去做一个博客。

这系列教程就不讲这些基础，这些基础在下一章节搭建一个最简单博客结合项目再进行讲解。

接下来进行的操作很重要，这是项目的最基础配置。后面的教程都会用到。

项目的基礎配置

如无意外，你跟着上面的操作，你现在终端操作路径是在项目的根目录下。

项目的根目录就是有**manage.py**的目录下。

先跟着下面的操作，到最后会有个树结构进行项目内容讲解。

```
mkdir apps templates static
```

使用编辑器打开mysite中的**settings.py**。

根据下方内容给出的提示，进行添加必要的内容。

```
"""
Django settings for mysite project.

Generated by 'django-admin startproject' using Django 2.1.

For more information on this file, see
https://docs.djangoproject.com/en/2.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.1/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
```

```
# 在项目中创建你的路径，像右边这样 os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# 快速启动项目设置不适合于生产，更多设置可浏览下方网页
# See https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
# 安全密钥，不要让别人人知道
SECRET_KEY = '04-gadagh_)f*yoacxwog9d#!zb9e8ugj06%pahg4!tdxo)*^%'

# SECURITY WARNING: don't run with debug turned on in production!
# 不要在生产环境打开Debug模式
DEBUG = True

# 允许的端口，里面填写的内容，在部署的时候会用到
ALLOWED_HOSTS = []

# Application definition
# 定义应用的地方
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

# 中间件
MIDDLEWARE = [
    # 安全中间件
    'django.middleware.security.SecurityMiddleware',
    # 会话中间件
    'django.contrib.sessions.middleware.SessionMiddleware',
    # 共同的中间件
    'django.middleware.common.CommonMiddleware',
    # 防止跨域请求攻击的中间件
    'django.middleware.csrf.CsrfViewMiddleware',
    # 验证的中间件
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    # 消息的中间件
    'django.contrib.messages.middleware.MessageMiddleware',
    # 防止点击劫持的中间件
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

# 主要路由配置，这里默认是在mysite中的urls.py文件
ROOT_URLCONF = 'mysite.urls'
```

```

# 模板 注意这里有更改的地方
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        # 下面的DIR添加 os.path.join(BASE_DIR, 'templates')
        # 这里是项目寻找模板文件目录的配置很重要
        'DIRS': [os.path.join(BASE_DIR, 'templates')], 
        # APP_DIRS这里是 是否到应用目录中寻找模板目录，后期会用到，必须开启
        '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',
            ],
        },
    },
],
# 接口规范应用
WSGI_APPLICATION = 'mysite.wsgi.application'

# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

# 数据库 开发环境使用sqlite3即可,
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Password validation
# 密码验证器
# https://docs.djangoproject.com/en/2.1/ref/settings/#auth-password-validators
#
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
]

```

```

{
    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
},
]

# Internationalization
# 国际化 这里有很多修改，可以直接复制粘贴即可
# https://docs.djangoproject.com/en/2.1/topics/i18n/

LANGUAGE_CODE = 'zh-hans' # 有修改，zh-hans 简体中文

TIME_ZONE = 'Asia/Shanghai' # 有修改 设置时区 亚洲/上海

USE_I18N = True

USE_L10N = True

USE_TZ = False          #关闭国际时间

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.1/howto/static-files/

STATIC_URL = '/static/'
# 静态文件配置路径 有修改 后期部署环节需要删掉
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static'),]

```

项目文件讲解

```

├── apps           # 该目录存放后期创建的目录
├── manage.py      # 用于启动、检查、创建应用等一系列操作的文件，后面会进行讲解
├── mysite         # 存放项目的主要配置的目录
│   ├── __init__.py # 声明mysite目录是一个python包，使用mysql数据库会用到
│   ├── settings.py # 项目设置配置
│   ├── urls.py     # 路由器，分发网址后缀等配置
│   └── wsgi.py     # 规范接口文件，部署到服务器会用到
└── static         # 静态文件目录
└── templates      # 模板文件目录

```

下一章节，可以看到怎么搭建一个最最基本的博客，也就是一开始让大家体验的一个小项目。

说明

大家最好把之前做的项目基本配置复制一份到另外的目录下。

也可以每次新建项目都重新配置一次。

因为后续会分几种类型的博客进行搭建。

有以分类、标签、书结构等博客类型。

最后还会有动态网页型的博客。

这次教程完成后，我的博客也会进行升级。

希望大家忘了之前的例子，现在重新做一个。

因为可以让你了解Django的**MVC**是什么。

相信大家都做好了项目的基本配置。

开始吧

创建你的第一个应用

什么是应用？你现在打开手机，有微信、淘宝、支付宝、邮箱，他们都是手机上的应用，而手机系统就相当于项目。

如果你现在打开微信，可以发现底下有四个区域，每一个区域也是一个小的应用，而微信就相当于项目。

那么博客有几个应用呢？？

我们先把一个博客应该有的内容先列出来：

文章、标题、作者、创建时间、标签、分类、评论、阅读数、用户、图片、友情链接……一大堆东西。

其实你想有什么，塞进去就好了。

不过最好每次动手前，先想清楚那些功能是共通的和相似，整合到一块去，公共的也分出来。

例如阅读数，这个功能放到哪里比较好呢？？

这个功能最好单独拿出来，因为可以拓展成总来访人数、今天来访人数。

讲那么多，动手吧。

把终端切换到项目的根目录下（确保在虚拟环境中）

```
python manage.py startapp blog
```

上面这句代码的意思是创建一个名为blog的应用。

```
mv blog apps 把blog移动到apps目录下。
```

移动到apps目录下是为了方便管理应用，全部堆到根目录下很难看。

```
touch apps/blog/urls.py , 在应用中创建一个名为urls.py的文件。
```

models.py

这个文件主要是用来创建数据库键名和内容的索引的。

结合这个应用的功能来说：这个应用是我们博客的主体，博文方面的。

我们一篇博文中的基本信息有：标题、作者、文章、创建时间、简介、图片还有一个索引。

索引这个字段是关乎路由方面的知识，后面再讲。

```

from django.db import models
import markdown

# Create your models here.
# 生成一个基类，让其它类拥有该方法
# 这是用来把markdown格式的文本转化成html格式的方法
class BlogBase(models.Model):
    def body_markdown(self):
        mark = markdown.Markdown(extensions=[
            'markdown.extensions.extra',
            'markdown.extensions.codehilite',
            'markdown.extensions.toc',
        ])
        self.body = mark.convert(self.body)
        self.toc = mark.toc
        return self.body
    ...

讲解一下，下面的models.xxxx方法
models.CharField 这个字段是用来表示短语字段，必须带有最大长度的参数
models.TextField 这个字段是文本格式，也就是文章
models.DateTimeField 这个字段是用来获取时间的
models.SlugField 这个字段有点难翻译，由于Django是新闻方面公司放出来的一个开源项目
    属于新闻方面的术语。我们平时看新闻，控制新闻顺序的就是用slug了
    它是把新闻标题的文章转换为两到四个字或者单词来确定播放顺序的。
    香港澳门那边好像都是用这种。
    我们这里把它用作网址的后缀。
参数方面就不介绍了，看文档和复制粘贴翻译就好了
    ...

class Blog(BlogBase):
    title = models.CharField(verbose_name="标题", max_length=50)
    body = models.TextField(verbose_name="内容")
    author = models.CharField(verbose_name="作者", max_length=50, default='sing')
    created_time = models.DateTimeField(auto_now_add=True, verbose_name='创建时间')
    update_time = models.DateTimeField(auto_now=True, verbose_name='修改时间')
    slug = models.SlugField(verbose_name="索引后缀", unique=True)

    class Meta:
        verbose_name = '文章'
        verbose_name_plural = verbose_name
        ordering = ['-created_time']

    # 使对象在后台显示更友善
    def __str__(self):
        return "<Blog:%s>" % self.title

```

views.py

这个文件的作用是视图逻辑。

也就是我们打开<http://127.0.0.1:8000>之类的效果。

就是靠这个文件的方法呈现的，比如说打开上面的链接是看到首页，而看到文章详情页又是另外一种页面。

就是靠这里的逻辑实现的。原理看代码的注释。

```
# 从Django的快捷方法中导入 渲染、要么获取对象要么404方法
from django.shortcuts import render, get_object_or_404
# 从当前目录下的models.py中导入Blog类
from .models import Blog

# 所有博客列表方法 首页的处理方法
def blog_list(request):
    context = {} # 生成一个空字典
    context['blogs'] = Blog.objects.all() # 获取Blog中的所有对象

    # request 请求 意思是当我们在浏览器中访问 http://127.0.0.1:8000
    # 就相当于发送了一个请求给Django，当请求成功后
    # django就把context的内容在blog_list.html中渲染并返回给这个请求。
    # blog_list.html在后面环节再创建
    return render(request, 'blog_list.html', context)

# 文章详情页内容的处理方法
def blog_detail(request, slug): # 接收请求和slug参数
    context = {} # 生成一个空字典

    # 在Blog也就是我们的数据库中寻找有没有slug字段和传进来的slug字段匹配的
    # 没有，则返回404，有则返回该对象的内容
    context['blog'] = get_object_or_404(Blog, slug=slug)
    # blog_detail.html未创建
    return render(request, 'blog_detail.html', context)
```

admin.py

这是后台的字段参数，先复制吧，后面运行项目时，才能看到效果

```
from django.contrib import admin
from .models import Blog
# Register your models here.
@admin.register(Blog)
class BlogAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'slug', 'author', 'created_time', 'update_time')
```

urls.py

这是路由文件，现在项目中有两个urls.py文件。

注意区分开来，一个是在mysite目录下，另一个在应用里面。

讲之前先说明一下路由的作用是什么？

路由的作用，用来配发地址的。我们把网站比作是一个小区，小区中的门牌号就是网站的网址后缀。

我们回去自己家，总有个门牌号吧，我们要通过门牌号来区分

比如说我的域名是porksuimai.com这是小区

而后面的django-1相当于门牌号。

门牌号是唯一的，如果出现相同的门牌号，就会出现冲突。

总路由

我们项目的总路由在根目录下的mysite目录中的urls.py文件

内容如下注意里面代码的注释：

```
"""mysite URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/2.1/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
# 从django的核心方法导入管理员方法
from django.contrib import admin
# 从django的路由方法导入 路径、包括方法
from django.urls import path,include

urlpatterns = [
    # 根路径 从apps中的blog中导入urls.py文件,
    # 这里的意思就是包括了apps中的blog中urls.py的内容
    # 如果有多台路由器进行拼接经验的话，不难理解，这里相当与桥接路由器
    path(' ',include('apps.blog.urls')),
    # 下面的这个路径是后台管理路径，后面我们写博文都在这个网址写
    # http://127.0.0.1:8000/admin
    path('admin/', admin.site.urls),
]
```

blog的路由

看代码注释吧

```
# 从django的路由方法导入 路径、正则路径方法
from django.urls import path,re_path
# 从当前目录下导入views.py的内容
from . import views

urlpatterns = [
    # 下面方法中第一个参数是路径
    # 总路由也是为空，所以我们的博客列表就是首页
    # 当我们在浏览器打开 http://127.0.0.1:8000
    # 就会发送一个请求给django
    # django就会从路由器中寻找是否有这路径的，没有就会返回404
    # 有就寻找该路径对应的方法，也就是下面的第二个参数，
    # 第三个参数，相当于是变量名，也就是说 blog_list = path('', views.blog_list)
    # 后面使用模板时会用到，现在知道一下就好了
    path('', views.blog_list, name='blog_list'), 

    # re_path不难理解是正则匹配路径
    # 大家可以回头看一下views.py中的blog_detail函数
    # 它除了request参数还有一个slug参数
    # 解释一下这里的正则表达式吧：
    # 这是正则表达式匹配路径， /(?P<slug>[\w-]+)/
    # 以/开头 以/结束 括号()是括起来的表达式作为一个分组
    # [\w-]是匹配任意字母数字和减号
    # (?P<slug>[\w-]+)就是把slug进行分组，匹配里面的[\w-]
    re_path('detail/(?P<slug>[\w-]+)/', views.blog_detail, name='detail'),
]
```

运行项目前的必要操作

我们的应用写好了，我们需要在项目中应用我们的应用。

打开settings.py，找到INSTALLED_APPS进行添加

```
# Application definition
# 定义应用的地方
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'apps.blog', # 加上左边的这句内容，养成良好习惯在后面加上逗号
]
```

在templates目录下生成我们的模板文件blog_list.html和blog_detail.html文件
先不要复制之前的体验项目的文件，接下来要讲模板语法。

blog_list.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>测试</title>
    </head>
    <body>
        {% for blog in blogs %}
            {{ blog.title }}
        {% endfor %}
    </body>
</html>
```

blog_detail.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>{{ blog.title }}</title>
    </head>
    <body>
```

```
    {{ blog.body_markdown|safe }}  
  </body>  
</html>
```

新建应用或修改**models.py**后的必要操作

所有操作都在根目录下, 请保证你在虚拟环境中生成数据库迁移文件

```
python manage.py makemigrations
```

生成数据库

```
python manage.py migrate
```

创建超级管理员 (这个只需要一次就好了, 不用每次都创建)

```
python manage.py createsuperuser
```

根据提示依次输入信息即可。

运行项目

```
python manage.py runserver
```

浏览器打开 <http://127.0.0.1:8000/> 即可看到效果。

一个空白页对吧? ? ? 空白页说明运行成功。

空白的原因是因为我们的数据库还没有内容,

打开<http://127.0.0.1:8000/admin>

会有一个登录界面, 输入你刚才设置的帐号密码 (忘记了, 重新设定一个) 。

登录完成后会看到一个后台管理页面。

点击文章后面的增加写你的第一篇博文吧。

没有**markdown**类型的文章可以复制下面的内容哦。

注意, 索引的内容必须为英文、数字和-号, 其他字符不可以设置, 建议刚开始就先设置成 111 吧。

方便我继续讲解。

```
### 新建应用或修改models.py后的必要操作
```

```
所有操作都在根目录下, 请保证你在**虚拟环境**中
```

```
生成数据库迁移文件
```

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

```
生成数据库
```

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

```
创建超级管理员 (这个只需要一次就好了, 不用每次都创建)
```

```
`python manage.py createsuperuser`
```

```
根据提示依次输入信息即可。
```

```
## 运行项目
```

```
`python manage.py runserver`
```

```
浏览器打开`http://127.0.0.1:8000/`即可看到效果。
```

```
一个空白页对吧? ? ? 空白页说明运行成功。
```

```
空白的原因是因为我们的数据库还没有内容,
```

```
打开[http://127.0.0.1:8000/admin](http://127.0.0.1:8000/admin)
```

```
会有一个登录界面, 输入你刚才设置的帐号密码 (忘记了, 重新设定一个) 。
```

```
登录完成后会看到一个后台管理页面。
```

```
点击**文章**后面的**增加**写你的第一篇博文吧。
```

```
没有markdown类型的文章可以复制下面的内容哦。
```

保存后，回到<http://127.0.0.1:8000/>看一下效果。

出现了刚才设定的文章标题。

再打开<http://127.0.0.1:8000/detail/111/>可以看到刚才写的文章内容。

为什么会出现标题、文章的内容呢？

这个涉及到Django的模板语言，下一章节专门讲模板语言和我学习前端的方法。

大家可以尝试在blog_detail.html加入以下的内容试试看。

```
<a href="{% url 'blog_list'%}">博客列表</a>
{{ blog.body }}
{{ blog.title }}
{{ blog.author }}
{{ blog.created_time }}
#{ 先不要在blog_list.html中尝试，因为涉及到列表的关系，很容易报错的哦 #}
```

接下来，讲一下前端也就是浏览器中的内容是怎么来的？。

回顾

不知道大家有没有发现，我们的根目录下多了一个叫**db.sqlite3**的文件。

这个就是我们的数据库。用编辑器打开它试试，可以看到一大堆乱码和一丢丢我们可以阅读的信息。

不要修改里面的内容哦，我们只是看看。

它是用来存放我们的数据的，我们在<http://127.0.0.1:8000/admin>添加的内容都会存放到db.sqlite3。

那么它是怎么产生的呢？

首先sqlite3是属于关系型数据库，而它的键名都是在我们的models.py中定义的。我们在后台写入内容后，内容存储到了数据库后。

结合这个博客来说，我们在浏览器打开 <http://127.0.0.1:8000/>，就发送了一个请求给Django。

Django就会从总路由中（urls.py）寻找匹配 <http://127.0.0.1:8000/> 的方法。

注意：总路由（mysite/urls.py）是包括/包含了blog中的urls.py的内容的。

然后该路径匹配的处理方法是在views.py中的**blog_list**函数。

该函数返回的结果在**blog_list.html**上渲染**Blog**的全部对象。

Blog可以看成是关系型数据库的一个表，或者说是非关系型数据库的一个集合。

所以blog_list函数返回的是一个列表内容，在blog_list.html中，我们需要一个for循环来把里面的内容遍历出来方可使用。

其实上面的内容涉及到设计模式的**MVC**

MVC

models.py，就是M，也就是数据模型

views.py，视图，也可以说是视图逻辑

模板语言，是我们在html文件中写的大括号，就是C。控制命令。

我这里就不复制网上的说法，给个链接让大家看看**MVC**，我的习惯是结合项目来讲解。

如果还不明白数据是如何来到网页上的话，可以看一下以下的顺序，并结合项目中的代码观察一下。

```
# python manage.py runserver 运行项目
# 浏览器中输入http://127.0.0.1:8000 打开首页
# django接收到请求，匹配地址，成功
# 寻找该地址的处理方法，views.py中blog_list函数
# 该方法是生成一个空字典，
# 把数据库中名为Blog的表单的所有对象，
# 以键值对的形式赋值给blogs这个键名，
# 并以blog_list.html作为容器，渲染blogs的内容到相应中
# 通过模板中的控制语句，控制我们想要呈现的内容
# 最后我们就可以在浏览器中看到该页面了
```

后台

之前说过admin.py文件的内容，运行起来才能看到效果。

大家先打开 <http://127.0.0.1:8000/admin/>

点击文章，可以看到一个表单。

再打开admin.py，有没有发现，有些地方很相似啊？？

让我们把admin.py中的**id**和**title**的位置替换一下并保存吧。

在回头看一下后台页面？？

相信这个顺序比较符合我们的查看习惯。

Django的模板语言

最基础的三句：

```
{% xxxx.oooo %}      {# 变量语句 #}

{% for xxxx in xxxx %}      {# 开头除了用for 还可以用if 等控制语句 #}
    {{ xxxx }}              {# 这里是使用变量语句 #}
{% endfor %}                  {# 如果开头用了if 结尾要改成endif #}
                                {# 大多数控制语句都需要end和语句形式结尾 #}
                                {# 也有例外，例如url，还有些语句等到后面在进行讲解 #}

{# 我就不难猜了吧，我是注释 #}
```

html和Django基础配合写法

下面展示的例子是最基础的知识，后续还有更高级的方法来减少我们的代码量。

a标签

下面例子中**blog_list**是我们在urls.py中定义的方法的变量名

```
<a href="{% url 'blog_list' %}">博客列表</a>
```

传参：

```
<a href="{% url 'detail' each.slug %}">{{ each.title }}</a>
```

注意：控制语句是可以传参，由于文章详情页的处理方法是根据slug的值决定返回哪一篇博文，所以需要取里面的slug数据。

H1

比如说想要标题变得很醒目。

```
<h1>{{ blog.title }}</h1>
```

其他标签同理。

li标签

```
{% for blog in blogs %}
    <li><a href="{% url 'detail' blog.slug %}">{{ blog.title }}</a></li>
{% endfor %}
{# 通过这个方法就可以取出所有博文的链接 #}
```

关于前端页面的设计

也就是我们的博客界面。希望大家都可以自己写出自己心中想要的博客界面。

当然，也可以直接使用我提供的模板界面。

不过我自己都嫌弃我现在的界面，因为实在是太刺眼了，文章主体这里颜色选得不好，太亮太刺眼了。

我后续也会把这个界面用Vue.js重新设计，并把博客结构转化为书结构形式，有点类似于gitbook吧。

关于前端学习的心得

提供的模板是我花了一天左右才写出来了。

我当时没有任何前端知识，还是硬生生被我写出来了。技巧在于：首先，你需要一只笔和一张纸。把你想要的博客界面画出来，例如首页要怎么样的，详情页是怎么样的。画出来。

然后，把这个页面分为三部分，头、躯干、脚。

每个部位再细分一下。不懂的地方，谷歌一下，写一个。一个一个的写出来。CSS也是一样，
a标签不要下划线、li标签不要点.....

最后，把写好的代码像乐高积木拼接起来就好了。

你们可以打开我提供的模板，页面也就三个部分就搞定了。

写完以后，你就有了前端的基础知识了。

先不要使用其他例如Vue的框架。他们的语法使用会和python的web框架起冲突。Tornado、Flask我都试过，会出现错误的。

关于vue的解决方案有的，如果你会vue.js，也请先了解一下后端方面的知识。

后期的话，我也会讲Django和Vue配合搭建博客的教程。

初学者可以先利用Bootstrap来辅助布局方面的工作，响应式可以在以后学了前端后再添加。

大家可以先练习怎么用模板语言配置好模板的页面后，再进行后面的内容。

CSS和JS导入

如下：

```
<link rel="stylesheet" href="/static/css/base.css">
<link rel="stylesheet" href="/static/css/blog.css">
<link rel="stylesheet" href="/static/css/bootstrap.min.css">
<script type="application/javascript" src="/static/js/jquery.min.js"></script>
<script type="application/javascript" src="/static/js/bootstrap.min.js"></script>
```

如果部署上线的话需要修改为下面的这种形式，这也是模板控制语句的。

```
<link rel="stylesheet" href="#">{% static 'css/blog.css' %}>
<script type="application/javascript" src="#">{% static 'js/jquery.min.js' %}></script>
>
```


博客的标签和类型

本章节开始后，先不要动手，先过一遍大概内容后，再决定你的博客要设计那一种类型。而且会介绍Django的自定义模板标签、模板拓展方法、可以很大程度上减少你的代码量。一开始，会讲解博客的标签，分类的数据库设计方面。再讲如何使用模板拓展方法，减少代码。讲解完，自定义模板标签后，会让你的博客项目像乐高积木一样，就可以拼凑起来。

开始

回头阅读如何创建和配置Django项目，创建一个新的mysite项目。和新建一个名为blog的应用。由于之前的项目是生成了数据库，删除它比较麻烦，可以删但现在是文字讲解，讲解起来比较麻烦。当练习一下吧。如果你要偷懒的话也可以到我的github中下载一个[Django的基础配置文件](#)下载名为mysite的目录即可。下载完后自行创建templates、static、apps目录。并依次执行下面的命令

```
python manage.py startapp blog  
mv blog apps  
touch apps/blog/urls.py
```

models.py

打开我们在apps/blog/models.py文件，写入以下内容：这个版本是博客标签和分类同时存在的版本。如果你是只要分类不要标签的话，把关于标签的内容去掉即可，只要标签的同理。

```
from django.db import models  
import markdown  
# Create your models here.  
# 生成基类，减少代码量  
class BlogBase(models.Model):  
    def body_markdown(self):  
        mark = markdown.Markdown(extensions=[  
            'markdown.extensions.extra',  
            'markdown.extensions.codehilite',  
            'markdown.extensions.toc',  
        ])  
        self.body = mark.convert(self.body)  
        self.toc = mark.toc  
        return self.body  
  
    # 标签  
    class BlogTag(BlogBase):  
        name = models.CharField(max_length=64, verbose_name='标签名')
```

```

slug = models.SlugField(unique=True, verbose_name='标签后缀')
body = models.TextField(verbose_name='标签简介')
class Meta:
    verbose_name = '标签'
    verbose_name_plural = '标签列表'
    ordering = ['id']          # 排序, 按id排序
# 使对象在后台显示更友善

# 标签
class BlogType(BlogBase):
    name = models.CharField(max_length=64, verbose_name='类型名')
    slug = models.SlugField(unique=True, verbose_name='类型后缀')
    body = models.TextField(verbose_name='类型简介')
    class Meta:
        verbose_name = '标签'
        verbose_name_plural = '标签列表'
        ordering = ['id']          # 排序, 按id排序
# 使对象在后台显示更友善

class Blog(BlogBase):
    title = models.CharField(verbose_name='标题', max_length=50)
    body = models.TextField(verbose_name='文章')
    author = models.CharField(verbose_name='作者', default='sing', max_length=50)
    created_time = models.DateTimeField(verbose_name='创建时间', auto_now_add=True)
    update_time = models.DateTimeField(verbose_name='修改时间', auto_now=True)
    slug = models.SlugField(verbose_name='后缀', unique=True)

    blog_type = models.ForeignKey(BlogType, verbose_name='博客类型', on_delete=models.CASCADE)

    blog_tag = models.ManyToManyField(BlogTag, verbose_name='博客标签')

    ...

```

有没有发现上面的BlogType和BlogTag只是名字不一样
 他们的内容都是一样的, 不同的地方在于他们在Blog这个类中的关系
 blog_type是外键关联,
 blog_tag是多对多关系

区别在于一篇博文只有一个 类型,
 但可以有多个 标签
 现在的项目是两种类型同时存在,
 注意, 使用多对多关系也就是标签的话,
 返回的内容是一个列表, 需要用到for循环方可取出里面的内容

blog_type里面的on_delete=models.CASCADE参数的意思是
 如果删除了该分类, 该分类的内容也就是博文也会被删除掉。

...

```

class Meta:
    verbose_name = '文章'
    verbose_name_plural = verbose_name
    ordering = ['-created_time']

    # 使对象在后台显示更友善
    def __str__(self):
        return "<Blog:%s>" % self.title

    # 分页
    def get_pre(self):# 上一页
        return Blog.objects.filter(id__lt=self.id).order_by('-id').first()

    def get_next(self):#下一页
        return Blog.objects.filter(id__gt=self.id).order_by('id').first()

```

先不要敲代码，想一下，你的博客还需要什么内容，需要展示什么内容？？

阅读数、图片和文章列表分页这部分后面会讲到，可以先不考虑。

其实上面的代码都是根据前端方面去设计的，可以说成是前端决定后端

拿起你的纸和笔，画一下你的博客的首页、博客列表页、文章详情页。

按标签归档页和按分类归档页，其实和博客列表页可以说基本上一模一样，就名字不一样。

可以先不画，后面暂时也不用写，等介绍到模板拓展方法再写。

views.py

视图部分就相对于之前多了一个分页的通用函数，其它函数都很简单的啦。

```

from django.shortcuts import render, get_object_or_404
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from .models import Blog, BlogTag, BlogType
# Create your views here.

# 分页通用函数，第一个参数是内容列表，第二个参数是页码
def page_gen(contact_list, page):
    # 分页规则，第一个参数，内容列表，第二个参数，每页多少内容。第三个参数，孤儿参数
    # 孤儿参数：假如现在总共有12篇博文，正常来说每页有10篇博文，那么总页码应该有两页。
    # 但采取了孤儿参数后，第一页会有12篇博文，没有了第二页。
    # 除非有13篇博文，才会出现第二页
    paginator = Paginator(contact_list, 10, 2)
    try:
        contacts = paginator.page(page)
    except PageNotAnInteger:
        # 如果传入的参数不是整数类型，返回第一页内容
        # 假如用户输入page=abcd或者其他字符，返回第一页内容
        contacts = paginator.page(1)
    except EmptyPage:
        # 如果传入的页码大于最大的页码数，返回最后一页的内容
        # 现在总共有5页，如果用户输入第6页则返回第5页的内容

```

```
    contacts = paginator.page(paginator.num_pages)

    return contacts

# 首页
def home(request):
    return render(request, 'home.html')

# 所有博客列表
def blog_list(request):
    context = {}

    contact_list = Blog.objects.all()
    # 从请求中查找是否有page参数，如果有则取page的参数
    # 没有则取1
    page = request.GET.get('page', 1)

    context['blogs'] = page_gen(contact_list, page)

    return render(request, 'blog_list.html', context)

# 按标签分类的博客列表
def blog_tag(request, slug):
    context = {}
    # 在BlogTag中也就是我们的数据库中寻找有没有slug字段和传进来的slug字段匹配的
    blog_tag = get_object_or_404(BlogTag, slug=slug)

    # 从Blog中寻找是在该标签下的文章列表
    contact_list = Blog.objects.filter(blog_tag=blog_tag)

    page = request.GET.get('page', 1)
    # 从请求中查找是否有page参数，如果有则取page的参数
    # 没有则取1
    context['blogs'] = page_gen(contact_list, page)

    return render(request, 'blog_tag.html', context)

# 按类型分类页
def blog_type(request, slug):
    context = {}
    # 在BlogTag中也就是我们的数据库中寻找有没有slug字段和传进来的slug字段匹配的
    blog_type = get_object_or_404(BlogType, slug=slug)

    # 从Blog中寻找是在该标签下的文章列表
    contact_list = Blog.objects.filter(blog_type=blog_type)

    page = request.GET.get('page', 1)
    # 从请求中查找是否有page参数，如果有则取page的参数
```

```
# 没有则取1
context['blogs'] = page_gen(contact_list, page)

return render(request, 'blog_type.html', context)

def blog_detail(request, slug): # 接收请求和slug参数
    context = {} # 生成一个空字典

    # 在Blog也就是我们的数据库中寻找有没有slug字段和传进来的slug字段匹配的
    # 没有，则返回404，有则返回该对象的内容
    context['blog'] = get_object_or_404(Blog, slug=slug)
    # blog_detail.html未创建
    return render(request, 'blog_detail.html', context)
```

上面的代码基本上，没几个地方值得优化了。

关于页码的使用留到模板方面再讲。

urls.py

相信大家看下面的内容都没什么问题，有问题回头看最基础的博客

```
from django.urls import path, re_path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('blog_list/', views.blog_list, name='blog_list'),
    re_path('type/(?P<slug>[\w-]+)/', views.blog_tag, name='blog_type'),
    re_path('tag/(?P<slug>[\w-]+)/', views.blog_tag, name='blog_tag'),
    re_path('detail/(?P<slug>[\w-]+)/', views.blog_detail, name='detail'),
]
```

admin.py

后台管理，这里给个装*的机会大家，留意最后的注释，运行项目并进入后台，有个小彩蛋哦。

```
from django.contrib import admin
from .models import Blog, BlogTag, BlogType
# Register your models here.

@admin.register(BlogType)
class BlogTypeAdmin(admin.ModelAdmin):
    list_display = ('name', 'id', 'slug')

@admin.register(BlogTag)
```

```
class BlogTagAdmin(admin.ModelAdmin):
    list_display = ('name', 'id', 'slug',)

@admin.register(Blog)
class BlogAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'slug', 'author', 'created_time', 'update_time')

admin.site.site_header = '恭喜发财'
admin.site.site_title = '最帅的管理员'
```

我们的项目的后端就基本上完成了，但请大家先不要敲代码。

接下来的内容可以让你的项目开发过程少了很多坑。

下一篇提前讲**Django**的自定义模板标签

通常这个内容都是放到后面，这里提前讲是为了拓展大家的开发思路。

因为就我个人来讲，开发思路比写代码重要。

思路理清了。做什么都比别人好。

什么是自定义模板标签?

其实问你什么是xxxxx?

把问题转化成为什么要使用自定义模板标签?

学习的动力就来了。

结合这个项目来说，现在有个需求。

需要在博客列表页的右半部分显示我们的所有标签的列表。

按照我们前面的写法，做不到，显示不出来（利用模板语言在所有博客列表页可以做到，但很麻烦，其他页面肯定不行）

如果要显示的话，需要把views.py中的blog_list函数改写成下面的样式。

```
def blog_list(request):
    context = {}

    contact_list = Blog.objects.all()
    # 从请求中查找是否有page参数，如果有则取page的参数
    # 没有则取1
    page = request.GET.get('page', 1)

    context['blogs'] = page_gen(contact_list, page)

    context['blog_tag'] = BlogTag.objects.all()

    return render(request, 'blog_list.html', context)
```

现在再来个需求、要在首页显示所有的标签和类型又要改写home函数，如下：

```
def home(request):
    context = {}
    context['blog_tag'] = BlogTag.objects.all()
    context['blog_type'] = BlogType.objects.all()
    return render(request, 'home.html', context)
```

需求又双叒叕来了。

那样就会导致我们的views.py的代码变得超样衰。

自定义模板标签可以帮我们解决这个问题。

现在有了学习的动力了吧？？

创建你的第一个自定义模板标签

把终端切换到项目根目录下，依次输入以下代码

```
mkdir apps/blog/templatetags
touch apps/blog/templatetags/__init__.py
```

```
touch apps/blog/templatetags/diy_tags.py
```

`templatetags` 这个目录名不能修改，系统默认的。
`__init__.py` 这个文件是声明`templatetags`是一个python包
`diy_tags.py` 是我们编写自定义模板标签的文件。
 刚才上面说了要在博客列表页显示我们所有的标签列表对吧？？
 要在要在首页显示所有的标签和类型列表对吧？？？

diy_tags.py

```
# pork_suimai/apps/blog/templatetags/diy_tags
from django import template      # 从Django中导入模板方法
from ..models import BlogTag, Blog, BlogType
from django.db.models.aggregates import Count    # 从django的数据库模型总数中导入计数方法
register = template.Library()    # 注册 模板的库 Library 图书馆，可以理解为放书进去，

@register.simple_tag
def test(): # 测试
    return 'hello world'

@register.simple_tag      # 注册简单标签，这是python关于装饰器的方法
def get_blog_list():# 获取所有博客列表
    return Blog.objects.all()

@register.simple_tag
def get_blog_tag_list(): # 获取博客标签列表
    return BlogTag.objects.annotate(total_num=Count('blog')).filter(total_num__gt=0)

@register.simple_tag
def get_blog_type_list(): # 获取博客标签列表
    # 下面的代码意思是
    # 通常很多人把博客部署上线后，都会列一大堆标签、分类等（例如我）
    # 心中总有个大计划，要写Django、Tornado、Requests、
    # Scrapy、Flask、Linux、MongoDB、MySQL、Vue
    # 但又迟迟不更新。很多标签或类型下是没有文章的
    # 简单来说，如果该标签或者分类下没有文章，则不返回该标签或分类
    return BlogType.objects.annotate(total_num=Count('blog')).filter(total_num__gt=0)

@register.simple_tag
def get_detail_tags(each):
    tags = Blog.blog_tag
```

```
return tags
```

好了写完了自定义模板标签。

该怎么用呢？？ 例如说要在首页显示该内容。

首页的模板文件是home.html。

注意阅读里面的注释如下：

```
<!DOCTYPE html>
{% load diy_tags %}  {# 导入我们的自定义模板标签 #}
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>首页</title>
    </head>
    <body>
        <div class="site-me-widget-first">
            <h4 class="title">标签</h4>
            <hr>
            <div class="content community">
                {% get_blog_tag_list as tag_list %}  {# 把函数实例化为变量名 #}
                {% for tag in tag_list %}      {# 通过for循环遍历出内容 #}
                    <li class="blog-type-tag"><a target="_blank" href="{% url 'blog
_tag' tag.slug %}">{{ tag.name }}</a></li>
                {% empty %}  {# 这个empty是说，假如没有标签的话，显示下面p标签的内容 #}
                    <p>暂无标签 敬请期待</p>
                {% endfor %}
            </div>
            <div class="content community">
                {% get_blog_type_list as type_list %}  {# 把函数实例化为变量名 #}
                {% for type in type_list %}      {# 通过for循环遍历出内容 #}
                    <li class="blog-type-tag"><a target="_blank" href="{% url 'blog
_type' type.slug %}">{{ type.name }}</a></li>
                {% empty %}  {# 这个empty是说，假如没有标签的话，显示下面p标签的内容 #}
                    <p>暂无分类 敬请期待</p>
                {% endfor %}
            </div>
        </body>
    </html>
```

那么需求不管它怎么来，我们都可以从容应对，这就是自定义模板标签。

其实自定义模板还可以应用到搜索方面去哦。大家自己写个函数试试。

因为它可以做到在模板上传参。

下一篇介绍模板拓展方法。

还是那句话，先不要敲代码。

最后会总结开发的思路。回头再研究代码。

模板拓展

注意，进行下面操作，务必把之前写好的模板文件保存一份到其它目录中，防止操作失误，导致前功尽废。

还是那句话，为什么需要模板拓展方法？

我现在写好了首页、博客列表页、文章详情页的内容。

三个文件的内容如下：

首页

```
{% load diy_tags %}      {# 导入自定义标签 #}
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>首页</title>{# 这里有不同 #}
    <link rel="stylesheet" href="/static/css/base.css">
    {% block css_file %}{% endblock %}
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <script type="application/javascript" src="{% static 'js/jquery.min.js' %}"></script>
    <script type="application/javascript" src="{% static 'js/bootstrap.min.js' %}">
</script>
</head>
<body>
    <header class="site-header">
        <div class="site-branding">
            <h1 class="site-title"><a href="{% url 'home' %}">花若盛开 蝴蝶自来</a></h1>
        <br>
            <div class="site-introduction">这是我用Django2.0和bootstrap搭建的博客</div>
        </div>
        <nav class="navbar-default site-navigation" role="navigation">
            <div class="container-fluid" >
                <div class="navbar-header" >
                    <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#navbar-collapse"><span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>
                </div>
                <div id="navbar-collapse" class="collapse navbar-collapse">
                    <ul class="menu">
                        <li><a href="{% url 'home' %}">首页</a></li>{# 这里有不同 #}
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="content">
        <div class="row" style="margin-top: 20px;">
            <div class="col-md-8" style="border: 1px solid #ccc; padding: 10px; border-radius: 5px; background-color: #f9f9f9;">
                <h2>最新文章</h2>
                <ul style="list-style-type: none; padding-left: 0;">
                    <li><a href="#">标题一</a></li>
                    <li><a href="#">标题二</a></li>
                    <li><a href="#">标题三</a></li>
                </ul>
            </div>
            <div class="col-md-4" style="border: 1px solid #ccc; padding: 10px; border-radius: 5px; background-color: #f9f9f9;">
                <h2>分类别</h2>
                <ul style="list-style-type: none; padding-left: 0;">
                    <li><a href="#">分类一</a></li>
                    <li><a href="#">分类二</a></li>
                    <li><a href="#">分类三</a></li>
                </ul>
            </div>
        </div>
    </div>
</body>
```

```

        <li><a href="{% url 'blog_list' %}">博客列表</a></li>{# 这里有不同
    #}
        <li><a href="#">文档</a></li>
        <li><a href="#">关于</a></li>
    </ul>
</div>
</div>
</nav>
</header>

<div class="site-me-widget-first">
    <h4 class="title">标签</h4>
    <hr>
    <div class="content community">
        {% get_blog_tag_list as tag_list %} {# 把函数实例化为变量名 #-}
        {% for tag in tag_list %} {# 通过for循环遍历出内容 #-}
            <li class="blog-type-tag"><a target="_blank" href="{% url 'blog_tag
' tag.slug %}">{{ tag.name }}</a></li>
            {% empty %} {# 这个empty是说，假如没有标签的话，显示下面p标签的内容 #-}
                <p>暂无标签 敬请期待</p>
            {% endfor %}
        </div>
        <div class="content community">
            {% get_blog_type_list as type_list %} {# 把函数实例化为变量名 #-}
            {% for type in type_list %} {# 通过for循环遍历出内容 #-}
                <li class="blog-type-tag"><a target="_blank" href="{% url 'blog_typ
e' type.slug %}">{{ type.name }}</a></li>
                {% empty %} {# 这个empty是说，假如没有分类的话，显示下面p标签的内容 #-}
                    <p>暂无分类 敬请期待</p>
                {% endfor %}
            </div>
        </div>

        <footer>
            <div class="diy-card">
                <p>Copyright © 2018 Sing. Powered by Django.</p>
                <p>粤ICP备18079962号</p>
            </div>
        </footer>
    </body>
</html>

```

博客列表页

```

{% load diy_tags %}      {# 导入自定义标签 #-}
{% load static %}
<!DOCTYPE html>
<html lang="en">

```

```

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>博客列表</title>{# 这里有不同 #-}
    <link rel="stylesheet" href="/static/css/base.css">
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <script type="application/javascript" src="{% static 'js/jquery.min.js' %}"></s
cript>
    <script type="application/javascript" src="{% static 'js/bootstrap.min.js' %}">
</script>
</head>
<body>
    <header class="site-header">
        <div class="site-branding">
            <h1 class="site-title"><a href="{% url 'home' %}">花若盛開 蝴蝶自來</a></h1
>
            <div class="site-introduction">这是我用Django2.0和bootstrap搭建的博客</div>
        </div>
        <nav class="navbar-default site-navigation" role="navigation">
            <div class="container-fluid" >
                <div class="navbar-header" >
                    <button type="button" class="navbar-toggle" data-toggle="colla
pse" data-target="#navbar-collapse"><span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>
                </div>
                <div id="navbar-collapse" class="collapse navbar-collapse">
                    <ul class="menu">
                        <li><a href="{% url 'home' %}">首页</a></li>{# 这里有不同 #-}
                        <li><a href="{% url 'blog_list' %}">博客列表</a></li>{# 这里有不同
#}
                        <li><a href="#">文档</a></li>
                        <li><a href="#">关于</a></li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>

    <div class="container"><!--容器-->
        <div class="row"><!--列-->
            <div class="col-xs-12 col-md-10"><!--文章列表-->
                <div class="panel panel-default">
                    <div class="panel-heading">
                        <h3 class="panel-title" style="text-align: center; font-
size: 40px">文章列表</h3>
                    </div>
                    <div class="panel-body">

```

```

        {% for each in blogs %}
            <div class="blog">
                <h3><a href="{% url 'detail' each.slug %}">{{ each.title }}</a></h3>

                    <p class="blog-info">阅读数: ({{ read_count_tag each.slug }})&ampnbsp&ampnbsp分类: <a href="{% url 'blog_type' each.blog_type.slug %}">{{ each.blog_type }}</a>&ampnbsp&ampnbsp;
                        创建时间: {{ each.created_time|date:"Y-m-d" }}
                    &ampnbsp&ampnbsp标签: {% for i in each.blog_tag.values %}
                        <a href="{% url 'blog_tag' i.slug %}">{{ i.name }}</a>&ampnbsp
                    {% endfor %}
                </p>
                <p>{{ each.body_markdown|safe|truncatechars:120 }}</p>
            <hr>
        </div>
        {% empty %}
            <div class="blog">
                <h3>暂无博客，敬请期待</h3>
            </div>
        {% endfor %}
    </div>
</div><!--./col-md-10-->
<div class="hidden-xs col-md-2"><!--文章分类-->
    {%# 分类 %}
    <div class="panel panel-default">
        <div class="panel-heading">全部分类</div>
        <div class="panel-body blog-tags">
            {% get_blog_type_list as type_list %}      {%# 实例化函数名 #
}
            {% for type in type_list %}
                <li class="blog-type-tag"><a href="{% url 'blog_type' type.slug %}">{{ type.name }}</a></li>
            {% empty %}
                <p>暂无分类 敬请期待</p>
            {% endfor %}
        </div>
    </div>
    {%# 标签 %}
    <div class="panel panel-default">
        <div class="panel-heading">标签列表</div>
        <div class="panel-body blog-tags">
            {% get_blog_tag_list as tag_list %}
            {% for tag in tag_list %}
                <li class="blog-type-tag"><a href="{% url 'blog_tag' tag.slug %}">{{ tag.name }}</a></li>
            {% empty %}
                <p>暂无分类 敬请期待</p>
            
```

```

        {% endfor %}
    </div>
</div>
</div><!--col-md-2-->
</div><!--row-->
</div><!--container-->

<footer>
<div class="diy-card">
    <p>Copyright © 2018 Sing. Powered by Django.</p>
    <p>粤ICP备18079962号</p>
</div>
</footer>
</body>
</html>

```

文章详情页

```

{% load diy_tags %}      #{ 导入自定义标签 #}
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>首页</title>#{ 这里有不同 #}
    <link rel="stylesheet" href="/static/css/base.css">
    {% block css_file %}{% endblock %}
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <script type="application/javascript" src="{% static 'js/jquery.min.js' %}"></script>
    <script type="application/javascript" src="{% static 'js/bootstrap.min.js' %}">
</script>
</head>
<body>
    <header class="site-header">
        <div class="site-branding">
            <h1 class="site-title"><a href="{% url 'home' %}">花若盛開 蝴蝶自來</a></h1>
        <div class="site-introduction">这是我用Django2.0和bootstrap搭建的博客</div>
        </div>
        <nav class="navbar-default site-navigation" role="navigation">
            <div class="container-fluid" >
                <div class="navbar-header" >
                    <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#navbar-collapse"><span class="icon-bar"></span>

```

```

                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
        </div>
    <div id="navbar-collapse" class="collapse navbar-collapse">
        <ul class="menu">
            <li><a href="{% url 'home' %}">首页</a></li>{# 这里有不同 #-}
            <li><a href="{% url 'blog_list' %}">博客列表</a></li>{# 这里有不同
}
        <li><a href="#">文档</a></li>
        <li><a href="#">关于</a></li>
        </ul>
    </div>
</div>
</nav>
</header>

<div class="container">
    <div class="row">
        <div class="col-xs-12 col-sm-9">
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h3 class="panel-title" style="text-align: center;font-size
: 40px">{{ blog.title }}</h3>
                </div>
                <div class="panel-body">
                    <p class="detail-nav"> &ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp创建时间: {{ blog.created_time }}&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp作者: {{ blog.auth
or }}&ampnbsp&ampnbsp&ampnbsp&ampnbsp标签: {% for i in blog.blog_tag.values %}
                    <a href="{% url 'blog_tag' i.slug %}">{{ i.name }}</a>&ampnb
sp;
                    {% endfor %}</p>

                    {{ blog.body_markdown|safe }}
                </div>
            </div>
        <div class="hidden-xs col-sm-3">
            <div class="diy-toc">
                {{ blog.toc|safe }}
            </div>
        </div>
    </div>
</div>

<footer>
    <div class="diy-card">
        <p>Copyright © 2018 Sing. Powered by Django.</p>
        <p>粤ICP备18079962号</p>
    </div>
</footer>

```

```
</body>  
</html>
```

有没有发现他们又臭又长啊？而且发现哪里有问题的话，想修改也很麻烦啊。
上面这份代码是用不了的，这是我以前写过的一个项目复制过来的，有些功能，不存在。不能直接使用。

现在我想修改上面的代码，想找到错误的地方找不到。

有没有更简洁的写法呢？这个时候就要用到我们的模板拓展方法了。

首先利用 `{# 注释 #}`，把三个页面的不同的地方标注一下。大概是有5个到6个不同的地方。

你自己的模板可能数量不一样。

头部和底部的title标签不一样。

我的头部导航部分的链接是有选中效果的，样式内容不一样。

页面的主体内容不一样。

那么就把一样的地方做成一个容器来接收不一样数据。

基础的拓展语法

```
{% extends 'base.html' %}    {# 拓展 装载内容的容器文件 #}  
  
{% block title %}    {# block是块的意思 title是变量名 #}  
    首页          {# 首页 是我们需要传递的内容 空格不计算 #}  
{% endblock %}        {# 结束拓展 #}  
  
{% include 'base_right.html' %} {# 导入一个html的文件内容，后面会介绍 #}
```

现在主要围绕着上面的第一条和第二段语句的使用进行讲解。

前面说到需要一个容器。新建一个名为**base.html**的文件作为我们的容器。

```
touch templates/base.html
```

注意，进行以下的操作前请把之前做好的模板备份一份到其它目录中，以防不测（之前试过遇到停电）。

base.html

首先复制你认为代码量最多的一份文件的内容到**base.html**中。

在**base.html**中，把之前用注释标注的地方，也就是三个文件不同之处，把他们的内容统统删掉。但是要保留注释。

然后用 `{% block title %}{% endblock %}` 放到之前删掉的内容中。

注意，上面的**title**是一个变量名，根据内容的含义来起变量名。我这里的例子是因为在模板文件的title标签的内容不同，所以给他命名为**title**。

其它部位根据其内容含义，起一些见名知义的变量名。

看例子

```
{# base.html #}
```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>{% block title %}{% endblock %}</title>{# 这里有不同 #-}
    <link rel="stylesheet" href="/static/css/base.css">
    {% block css_file %}{% endblock %} {# 需要导入的文件不同 #-}
    <link rel="stylesheet" href="/static/css/bootstrap.min.css">
    <script type="application/javascript" src="/static/js/jquery.min.js"></script>
    <script type="application/javascript" src="/static/js/bootstrap.min.js"></script>
>
</head>
<body>
    <header class="site-header">
        <div class="site-branding">
            <h1 class="site-title"><a href="{% url 'home' %}">花若盛開 蝴蝶自來</a></h1>
        >
            <div class="site-introduction">这是我用Django2.0和bootstrap搭建的博客</div>

        </div>
        <nav class="navbar-default site-navigation" role="navigation">
            <div class="container-fluid" >
                <div class="navbar-header" >
                    <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#navbar-collapse"><span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div id="navbar-collapse" class="collapse navbar-collapse">
                <ul class="menu">
                    <li {% block home_active %}{% endblock %}><a href="{% url 'home' %}">首页</a></li>{# 这里有不同 #}{# 选中效果 #-}
                        <li {% block blog_active %}{% endblock %}><a href="{% url 'blog_list' %}">博客列表</a></li>{# 这里有不同 #}
                            <li><a href="#">行情分析</a></li>
                            <li><a href="#">关于</a></li>
                </ul>
            </div>
        </div>
    </header>
    {% block content %}{% endblock %} {# 页面的大体内容不同 #}
</body>
</html>

```

当完成后，进行下一步

blog_list.html

保留内容不相同的地方，也就是我们注释标注的地方。

把其他相同的部分的内容全部去掉。

并用在base.html定义的 `{% block xxxx %}{% endblock %}` 包裹住对应的内容。

最后在文件的第一行写上 `{% extends 'base.html' %}`

最终效果如下：

```
{% extends 'base.html' %}      {%# 继承的模板文件 #-}

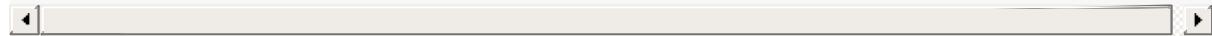
{% block title %}      {%# 这里是页面窗口的标题名 %}
    博客列表
{% endblock %}

{% block css_file %}          {%# 这里是需要导入的css文件 %}
    <link rel="stylesheet" href="/static/css/blog.css">
{% endblock %}

{% block blog_active %}  {%# 这里是博客的点击效果 %}
    class="nav-active"
{% endblock %}

{% block content %}      {%# 这里是内容 %}
<div class="container"><!-- 容器-->
    <div class="row"><!-- 列-->
        <div class="col-xs-12 col-md-10"><!-- 文章列表-->
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h3 class="panel-title" style="text-align: center;font-size: 40px">文章列表</h3>
                </div>
                <div class="panel-body">
                    {% for each in blogs %}
                        <div class="blog">
                            <h3><a href="{% url 'detail' each.slug %}">{{ each.title }}</a></h3>
                            <p class="blog-info">
                                <span class="glyphicon glyphicon-tag"></span><a href="{% url 'blog_type' each.blog_type.slug %}">{{ each.blog_type }}</a>&ampnbsp&nbs
p;
                                <span class="glyphicon glyphicon-time"></span>
                            {{ each.created_time|date:"Y-m-d" }}&ampnbsp&ampnbsp
                            </p>
                            <p>{{ each.body_markdown|safe|truncatechars:120}}</p
                        >
                            <hr>
                        </div>
                    {% empty %}
                        <div class="blog">
```

```
        <h3>暂无博客，敬请期待</h3>
    </div>
    {% endfor %}
    </div>
    </div><!-- ./col-md-10 -->
<div class="hidden-xs col-md-2"><!-- 文章分类 -->
    <p>留空，占位</p>
</div><!-- col-md-2 -->
</div><!-- row -->
</div><!-- container -->
{% endblock %}
```



先完成blog_list.html文件。尝试运行项目看一下效果。

和未修改前无误的话，继续改写home.html文件，blog_detail.html文件。

由于本篇文章的代码量太多了，大家可以到[本链接](#)中查看其他文件的优化效果。下一章节讲解，总结Django写博客项目的技巧。

项目的规划思路

- 博客主体设计
- 阅读数（可拓展成今天来访人数、总来访人数、阁下是今天第几位访客，总阅读数）
- 用户登录验证（没有注册是因为采取第三方登录，用户只要填写昵称即可）
- 评论或留言板（代码一模一样，前端代码不一样）
- 各种小工具（友情链接、个人社交网站链接、学习书籍链接等等）

上面的这些功能，会被写成五个应用，如果你有更多的想法，写十几个应用都可以。

第一步，也是最重要的一步，博客主体的设计是决定了你的后面功能的实现。

如果前期规划不好的话，需要修改博客的数据库，迁移数据库这个步骤很麻烦，因为是关系型数据库。

困难点比较多。

数据库设计方面可以参考前面给的**models.py**

也可以到我的github上面找关于Django的项目。只要README.md 上写了使用方法和介绍的都是可用的。

后续也会根据本次教程，分章节生成对应的项目文件。

博客这个应用的代码写作流程

- 用笔和纸画出你的博客界面和内容位置
- 收藏Django的文档，时常阅读文档
- 根据你的图纸设计博客主体的数据库
- views.py中，每个方法只输出出一种数据（例如文章详情页就只输出文章详情的数据，其他需要的额外内容交给自定义模板标签负责输出）
- 配置好你的路由器，也就是urls.py
- 根据你的图纸写模板文件，并把数据传输到上面去，如果两个页面几乎一模一样的（例如博客列表和标签类型列表，写好博客列表页即可。）
- 采取模板拓展方法减少代码量，提高可阅读性，方便后续维护。

技巧

常见的博客设计的写作技巧

通常博客的右边会是这个博主的个人空间、博客标签云或者分类列表、友情链接、时间归档、推荐的书籍等信息。

这个部分可以写成一个html文件，例如base_right.html

注意，不是规范的html文件。

只是写上一部分的代码，如下：

```
{% load diy_tags %}  {# 导入我们的自定义模板标签 #}
<div class="site-me-widget-first">
    <h4 class="title">标签</h4>
    <hr>
```

```

<div class="content community">
    {% get_blog_tag_list as tag_list %}  {# 把函数实例化为变量名 #}
    {% for tag in tag_list %}          {# 通过for循环遍历出内容 #}
        <li class="blog-type-tag"><a target="_blank" href="{% url 'blog_tag' ta
g.slug %}">{{ tag.name }}</a></li>
        {% empty %}      {# 这个empty是说，假如没有标签的话，显示下面p标签的内容 #}
        <p>暂无标签 敬请期待</p>
    {% endfor %}
</div>
<h4 class="title">类型</h4>
<hr>
<div class="content community">
    {% get_blog_type_list as type_list %}  {# 把函数实例化为变量名 #}
    {% for type in type_list %}          {# 通过for循环遍历出内容 #}
        <li class="blog-type-tag"><a target="_blank" href="{% url 'blog_type' t
ype.slug %}">{{ type.name }}</a></li>
        {% empty %}      {# 这个empty是说，假如没有标签的话，显示下面p标签的内容 #}
        <p>暂无分类 敬请期待</p>
    {% endfor %}
</div>
</div>

```

更多的内容自行添加上去。

当上面的文件完成后，在需要的位置导入即可，例如在blog_list.html中需要用到上面的内容 在需要的地方写入 `{% include 'base_right.html' %}` 即可。

代码优化

之前说到，按标签分类的博客列表和按类型分类的博客列表可以先不用写。

是因为，只要你在views.py中所有博客列表、标签分类的博客列表、按类型分类的博客列表返回的键名都设定为 `blogs` 。即可使用下面的方法。

只要写好blog_list.html。

在blog_type.html和blog_tag.html中写入一句代码即可，如下：

```
{% extends 'blog_list.html' %}
```

未完待续

还有更多方法等待整理

经常遇到的错误信息解答

下面是我整理的经常遇到的报错信息

自定义模板标签不生效，且报错

答：自定义标签的函数写错了，或者是写完自定义模板标签后未重启项目。

注意：每次写完自定义模板标签后，必须重启项目。

文章的内容变成了**html**代码

这是因为你没有使用管道（过滤器），如果在模板上索引的数据是html格式的，需要加上 `|safe`。

例如我们的文章 `{{ blog.body_markdown|safe }}`

还有我们的目录 `{{ blog.toc|safe }}`

这是表示该html是安全的代码，允许嵌入到html代码当中。

文章列表中出现**css**代码溢出（不知道这个说法对不对）。

通常我们的博客列表页之类的。会设计成显示一小部分正文，后面加上阅读全文。

限制正文长度的过滤器是下面代码中的`truncatechars`。

```
{{ blog.body_markdown|safe|truncatechars:120}}
```

在此基础上还要加上 `|striptags`，这是去除css样式的代码。

写成下面的例子即可

```
{% for tag in blog.blog_tag %}
```

```
    <a href="{% url 'blog_tag' tag.slug %}">{{ tag.name }}</a>
```

```
{% endfor %}
```

无法显示博客的标签，且报错。

通常情况下，在博客列表和博客详情页中，我们需要在文章介绍中显示博客的标签。

博客的标签可以有一个或者多个。

但不管它是一个还是多个，它都是一个列表。

需要用到`for`来遍历出内容。

如下面的例子：