

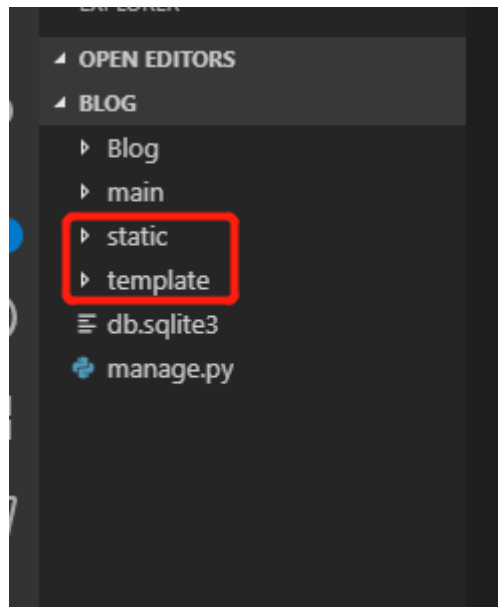
# Python + Django搭建个人博客 Dya02

## 替换默认模板

### 1. 创建static文件夹和template文件夹

- static存放静态文件，比如css, js, 图片等
- template文件夹存放HTML页面

创建好后的目录结构：



### 2. 在settings中添加配置

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, 'template')],  
        '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',  
            ],  
        },  
    ],  
]
```

修改的内容为

```
'DIRS': [os.path.join(BASE_DIR, 'template')],
```

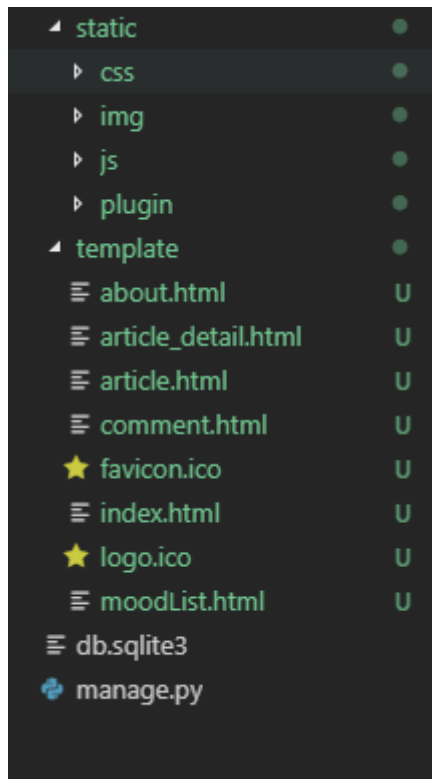
这段代码是添加template文件夹到模板目录中。并且在该文件前面添加了项目路径。

### 3. 将静态文件夹添加到项目目录中

```
STATICFILES_DIRS = (  
    os.path.join(BASE_DIR, 'static'), #记得这里一定要加逗号, 因为这是一个元组类型的数据  
)
```

接下来就是将我们自己的模板文件, 放到相应的文件夹中。

修改后的目录结构:



## 业务逻辑

### 1. 创建我们的第一个方法, 用来显示首页

views.py

```
def index(request):  
    '''显示首页'''  
    template_name = "index.html" # 要显示的模板名称  
    return render(request, template_name) #将我们定义好的页面返回到页面
```

这里template\_name是自己定义的一个变量, 用来指定要返回页面的HTML文件

urls.py

```

from django.contrib import admin
from django.urls import path

from main.views import index # 将views中定义个函数引入进来

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index, name='index'), #定义首页的路由规则
]

```

添加的代码为：

```

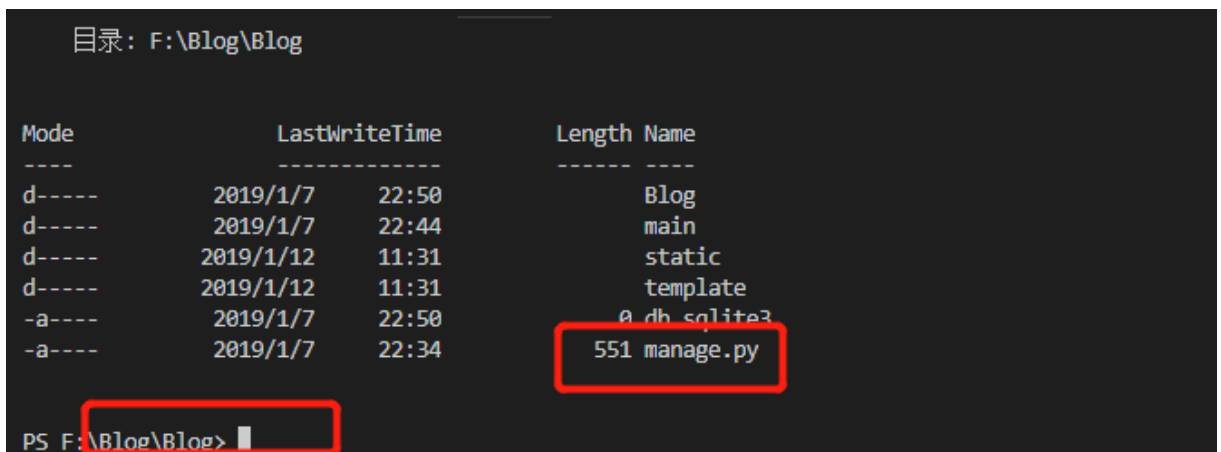
from main.views import index # 将views中定义个函数引入进来
...
path('', index, name='index'), #定义首页的路由规则

```

因为首页一般都是直接输入的网址：比如baidu.com后面是没有路由地址的。所以path的第一个参数这里为空，第二个参数是views中的index函数，用来处理指定的业务逻辑，name='index'，是为这个路由规则起一个别名，方便在其他地方调用（后面会看到）

第一次调试项目：

- 命令窗口进入到项目根目录下，如下所示：



```

目录： F:\Blog\Blog

Mode                LastWriteTime         Length Name
----                -
d-----          2019/1/7         22:50      Blog
d-----          2019/1/7         22:44      main
d-----          2019/1/12        11:31     static
d-----          2019/1/12        11:31     template
-a----          2019/1/7         22:50     db.sqlite3
-a----          2019/1/7         22:34     551 manage.py

PS F:\Blog\Blog>

```

- 如果有虚拟环境要激活虚拟环境，我这里用的是conda创建的虚拟环境，激活方法如下：

```

PS F:\Blog\Blog> activate MyblogPro
(MyblogPro) PS F:\Blog\Blog>

```

激活成功后，在路径前面会多一个括号，里面就是你的虚拟环境名称

- 运行项目：

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

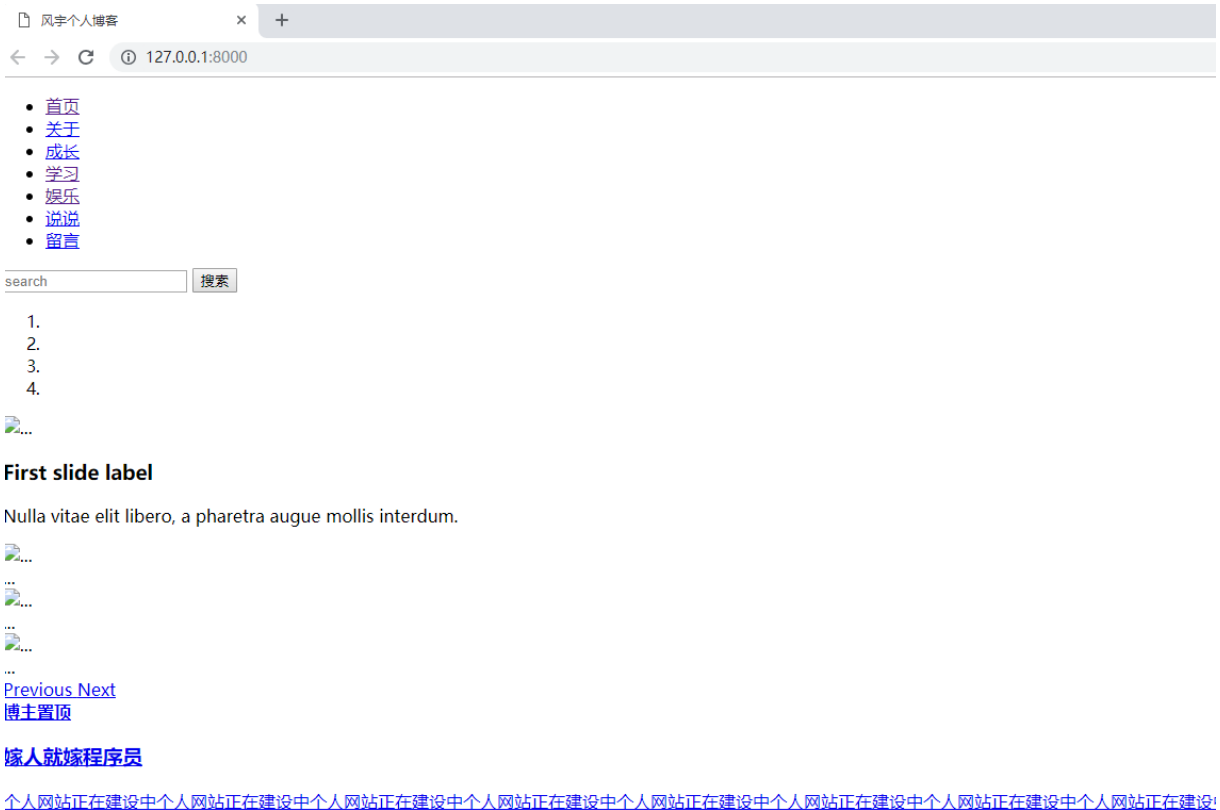
启动成功如下：

```
(MyblogPro) PS F:\Blog\Blog> python .\manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
January 12, 2019 - 11:49:06
Django version 2.1.5, using settings 'Blog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

访问网站查看项目效果：



可以看到，已经不是小火箭的默认模板了，但是这里的样式是乱的。接下来我们来修改一下。

## 修改页面样式

1. 在index.html页面中引入static文件目录

```
{% load static %}
```

2. 修改样式文件等的写法

■ 修改前

```
<link href="plugin/bootstrap/css/bootstrap.min.css" rel="stylesheet">
```

■ 修改后

```
<link href="{% static 'plugin/bootstrap/css/bootstrap.min.css' %}"
rel="stylesheet">
```

所有的静态文件都要这样修改一下。

```
<link href="{% static 'plugin/bootstrap/css/bootstrap.min.css' %}"
rel="stylesheet">
<link rel="stylesheet" type="text/css" href="{% static 'css/common.css'
%}" />
<link href="{% static 'logo.ico' %}" rel="shortcut icon" />
<script src="{% static 'plugin/jquery.min.js' %}"></script>
<script src="{% static 'plugin/bootstrap/js/bootstrap.min.js' %}">
</script>
<!--<script type="text/javascript" src="{% static 'plugin/jquery.page.js'
%}"></script-->
<!--<script src="{% static 'js/common.js' %}"></script-->
<!--<script src="{% static 'js/snowy.js' %}"></script-->
```

再刷新一下页面，看看效果

## TemplateSyntaxError at /

'staitc' is not a registered tag library. Must be one of:

- admin\_list
- admin\_modify
- admin\_static
- admin\_urls
- cache
- i18n
- l10n
- log
- static
- staticfiles
- tz

这里由于输入错误，导致了一个错误信息，修改一下



可以看出，基本样式已经有了，出个个别图片没有显示出。

## 拆分模板

网站模板内容可以发现：



包括头部导航，右侧导航分类，以及中间的正文内容，还有一个在下面的底部内容



- 头部

- 右侧
- 中间
- 底部

2. 复制一个index.html页面，命名为base.html

这个是我们的基础模板，接下来修改一下里面的内容。

```
<body>
  <div class="w_header"> ...
</div>

  <div class="w_container"> ...
</div>
  <div class="w_foot"> ...
</div>
</body>
<script type="text/javascript"> ...
</script>

</html>
```

上面红框对应的就是头部

```
<div class="w_container">
  <div class="container">
    <div class="row w_main_row">
      <div class="col-lg-9 col-md-9 w_main_left"> ...
      </div>
      <!-- 右侧开始 -->
      <div class="col-lg-3 col-md-3 w_main_right"> ...
      </div>
    </div>
  </div>
</div>
```

这里的红框对应的就是右侧的内容。

我们把

```

<div class="w_container">
  <div class="container">
    <div class="row w_main_row">
      <div class="col-lg-9 col-md-9 w_main_left">...
    </div>

    <!--右侧开始-->
    <div class="col-lg-3 col-md-3 w_main_right">...
    </div>
  </div>
</div>

```

这部分去掉。因为他是文章列表，只有在首页会显示，其他页面不需要。

用下面的内容替换掉

```

<div class="w_container">
  <div class="container">
    <div class="row w_main_row">
      {% block blogList %}
      {% endblock %}

      <!--右侧开始-->
      <div class="col-lg-3 col-md-3 w_main_right">...
      </div>
    </div>
  </div>
</div>

```

```

{% block blogList %}
{% endblock %}

```

这是Django框架的模板语言。后面会遇到其他的。那时我们再讲。

这个里的blogList是自己定义的变量，目的是接收其他页面传过来的内容。

### 3. 修改index.html

去掉公共部分的内容，也就是base.html页面的内容，保留只属于index页面的内容内容。

删除后的页面如下：



```
4
5
6
7 <div class="col-lg-9 col-md-9 w_main_left">...
191 </div>
192 |
193
194
195
196
197
198
```

多余的标签和内容全部删除掉了。现在我们来预览一下页面效果



发现只有文章的内容了，头部右侧还有底部的内容都没有了，但是那些我们也是要显示出来的，因此需要将其引用进来。

在头部最上面添加一段代码

```
{% include 'base.html' %}
```

```

1  {% include 'base.html' %}
2  {% load static %}
3
4
5
6
7
8  <div class="col-lg-9 col-md-9 w_main_left">...
192 </div>
193
194
195
196

```

预览效果如下：



还不是一个正常的首页，原因是，我们没有把文章列表的内容传过去。修改如下：

```

6
7   {% block blogList %}
8   <div class="col-lg-9 col-md-9 w_main_left">..
9   </div>
10  {% endblock %}
11
12
13
14
15
16
17
18

```

用我们在base.html页面中的定义的块，将它包含起来。现在我们再看一下页面效果。

上面的写错了，不是include 而实extends

```

1   {% extends 'base.html' %}
2   {% load static %}
3
4
5
6

```

他们两个是有区别的，后面会说。也可以自行百度一下。



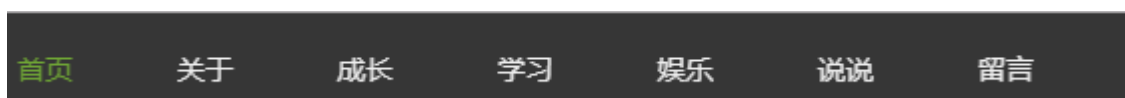
可以看出页面显示正常了，接下来就最重要的业务逻辑的内容了。

## 创建数据库信息

观察页面，暂时可以创建如下的几个表：

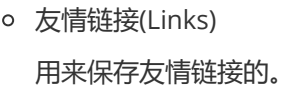
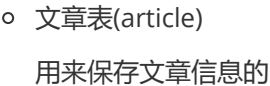
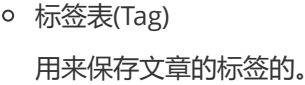
- 头部导航表(category)

用来保存网站分类的信息



- 站点公告(webInfo)

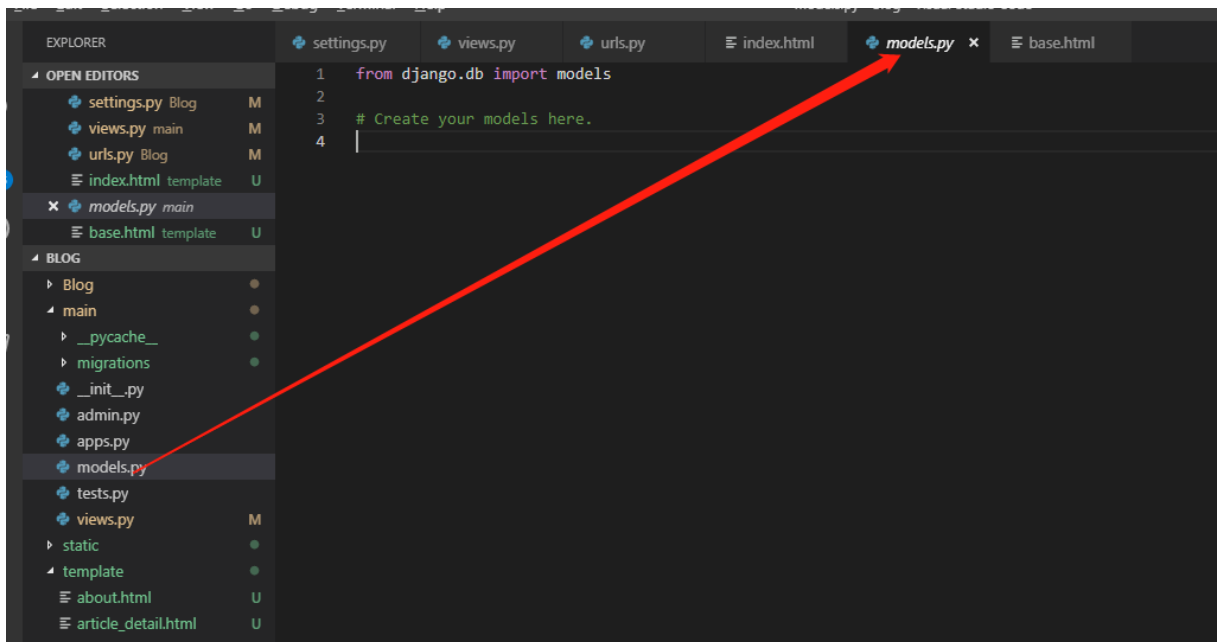
用来保存网站通知的。





暂时先建这个几个表，后面有需要再添加。

## 1. 创建表信息



在Django里面，表的信息是建在models.py中。

## 2. Category表

```
class Category(models.Model):  
    '''导航类'''  
    name = models.CharField(verbose_name='导航名称', max_length=50)  
    add_date = models.DateField(verbose_name='添加时间', auto_now=True,  
                                auto_now_add=True)  
  
    def __str__(self):  
        return self.name
```

models.CharField 表示的是文字列字段

models.DateField表示时间类型的字段

verbose\_name是为该字段起一个别名。

def \_\_str\_\_(self): 方法是返回类型，当在后台预览时看到的内容，后面会看到。

用同样的方法把其他的表也建好。

```
from django.db import models

# Create your models here.

class Category(models.Model):
    '''导航类'''
    name = models.CharField(verbose_name='导航名称', max_length=50)
    add_date = models.DateField(verbose_name='添加时间', auto_now=True)

    def __str__(self):
        return self.name

class WebInfo(models.Model):
    '''站点公告'''
    indof = models.CharField(verbose_name='公告内容', max_length=50)
    add_date = models.DateField(verbose_name='添加时间', auto_now=True)

    def __str__(self):
        return self.indof

class Tag(models.Model):
    '''标签'''
    name = models.CharField(verbose_name='标签名称', max_length=50)
    add_date = models.DateField(verbose_name='添加时间', auto_now=True)

    def __str__(self):
        return self.name

class Article(models.Model):
    '''博客内容'''
    title = models.CharField(verbose_name='标题', max_length=50)
    tags = models.ManyToManyField("Tag", verbose_name="标签") #一个文章可以有多个标
    签, 所以选择多对多
    content = models.CharField(verbose_name='文章内容', max_length=50) # 这里暂时用这
    个, 后面要用markdow格式, 这里需要修改
    add_date = models.DateField(verbose_name='添加时间', auto_now=True)
    view_count = models.IntegerField(verbose_name='阅读量')
    comments = models.IntegerField(verbose_name='留言量')

    def __str__(self):
        return self.title

class Links(models.Model):
    '''友情链接'''
    name = models.CharField(verbose_name='网站名', max_length=50)
    url = models.URLField(verbose_name='友情链接地址', max_length=200)
    add_date = models.DateField(verbose_name='添加时间', auto_now=True)
```

```
def __str__(self):  
    return self.name
```

3. 创建好表后，需要将它同步到数据库中，这里暂时用的是sqlite，最后发布时会换成mysql的。

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

```
Migrations for 'main':  
  main\migrations\0001_initial.py  
    - Create model Article  
    - Create model Category  
    - Create model Links  
    - Create model Tag  
    - Create model WebInfo  
    - Add field tags to article
```

在生成同步时，发现有几个地方写的不对，因此修改了一下。

看到上图的内容后，再执行如下代码：

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

```
(MyblogPro) PS F:\Blog\Blog> python .\manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, main, sessions  
Running migrations:  
  Applying contenttypes.0001_initial... OK  
  Applying auth.0001_initial... OK  
  Applying admin.0001_initial... OK  
  Applying admin.0002_logentry_remove_auto_add... OK  
  Applying admin.0003_logentry_add_action_flag_choices... 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 auth.0007_alter_validators_add_error_messages... OK  
  Applying auth.0008_alter_user_username_max_length... OK  
  Applying auth.0009_alter_user_last_name_max_length... OK  
  Applying main.0001_initial... OK  
  Applying sessions.0001_initial... OK  
(MyblogPro) PS F:\Blog\Blog> █
```

看到这样的信息说明创建成功了。

我们发现，除了我们自己的APP表之外，还有一些auth表被创建了，它是Django自带的。

接下来我们就创建一个管理员账号，他保存的就是再auth表中。

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

createsuperuser是由 create super user 三个单词组成的，很好记。

运行后，会有提示，按照提示输入内容就可以了。

```
(MyblogPro) PS F:\Blog\Blog> python .\manage.py createsuperuser
Username (leave blank to use 'think'): Tango
Email address (required): 1234567890.com
Password:
Password (again):
Superuser created successfully.
(MyblogPro) PS F:\Blog\Blog> |
```

如上图所示，创建成功了。

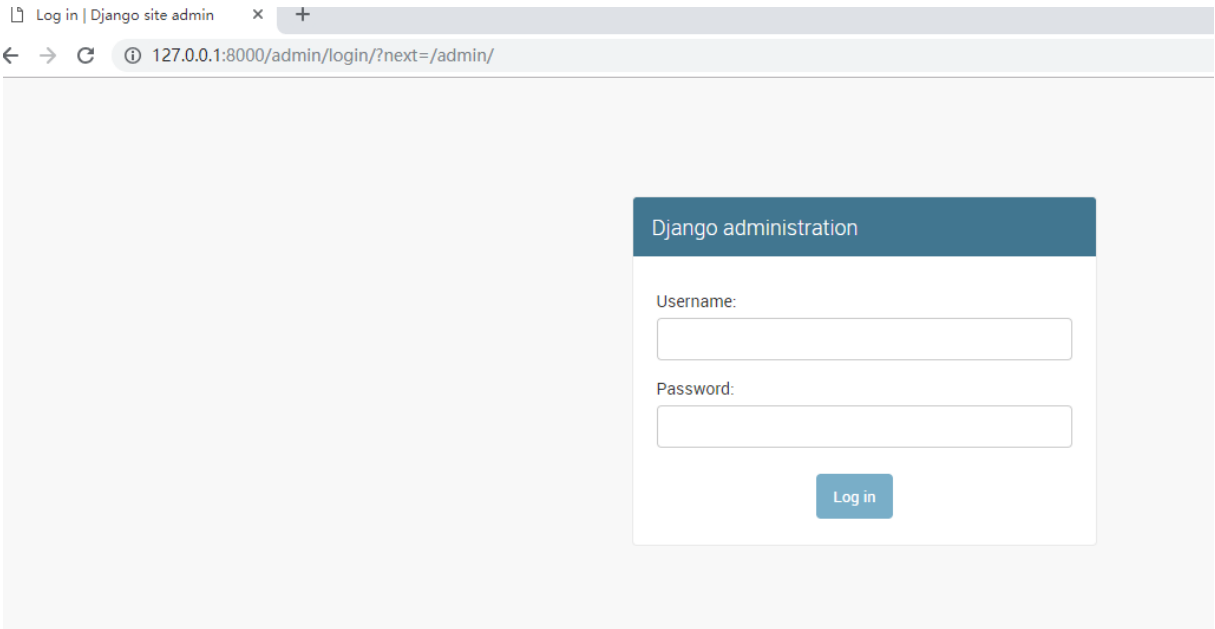
（休息中。。。顺便帮朋友调一个BUG）

继续....

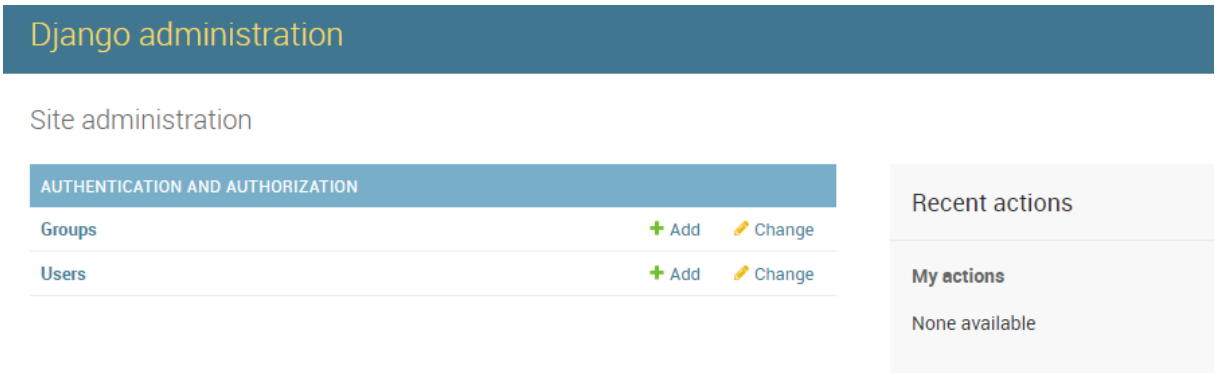
运行项目，看一下效果

然后输入：<http://127.0.0.1:8000/admin>

这个是Django为我们自带的后台页面。



输入账号密码后就能进去了，账号密码就是刚才创建的那个。





这里有几个问题要解决一下。

- 界面是英文，如何修改成中文  
修改settings.py文件

```
LANGUAGE_CODE = 'zh-hans'

TIME_ZONE = 'Asia/Shanghai'

USE_I18N = True

USE_L10N = True

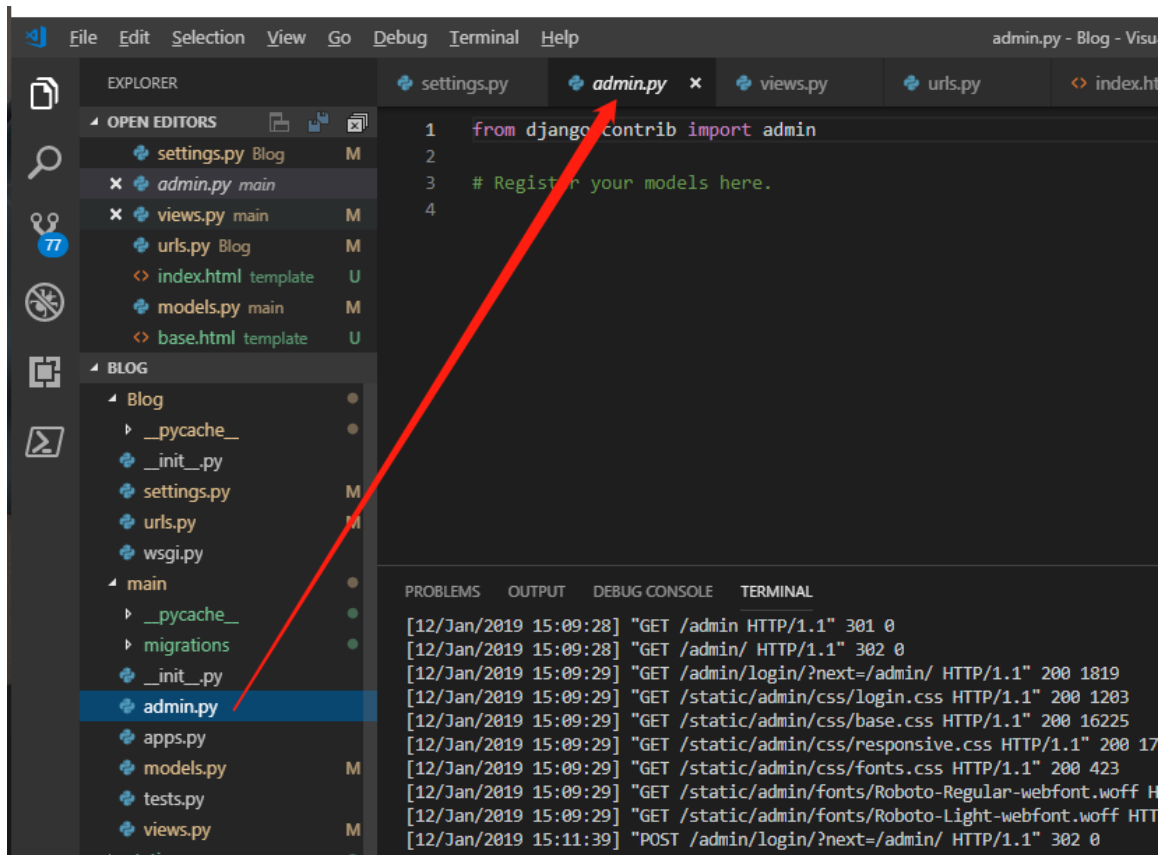
USE_TZ = True
```

再看一下后台样式：



已经变成我们熟悉的中文了。

- 我们创建的表没有显示出来  
修改admin.py



```
from django.contrib import admin
```

```
from .models import Category, WebInfo, Tag, Article, Links # 将我们定义好的表导入进来
```

```
# Register your models here.
```

```
# 将表注册到后台中
```

```
admin.site.register(Category)
```

```
admin.site.register(WebInfo)
```

```
admin.site.register(Tag)
```

```
admin.site.register(Article)
```

```
admin.site.register(Links)
```

MAIN		
Articles	+ 增加	✎ 修改
Categorys	+ 增加	✎ 修改
Linkss	+ 增加	✎ 修改
Tags	+ 增加	✎ 修改
Web infos	+ 增加	✎ 修改
认证和授权		
用户	+ 增加	✎ 修改
组	+ 增加	✎ 修改

#### 最近动作

我的动作

无可用的

刷新页面可以看到如图效果，但是表的名字不是中文，接下来在models.py文件中修改一下。  
在之前创建好的每个类下面再添加如下代码：

```
class Meta:
    verbose_name = '友情链接'
    verbose_name_plural = verbose_name
```

例如修改后的友情链接：

```
class Links(models.Model):
    '''友情链接'''
    name = models.CharField(verbose_name='网站名', max_length=50)
    url = models.URLField(verbose_name='友情链接地址', max_length=200)
    add_date = models.DateField(verbose_name='添加时间', auto_now=True)

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = '友情链接'
        verbose_name_plural = verbose_name
```

这时再刷新一下页面看看。这里有一个问题，我错把Meta协程Meda了。所以没有效果  
站点管理

MAIN		
博客	+ 增加	✎ 修改
友情链接	+ 增加	✎ 修改
导航	+ 增加	✎ 修改
标签	+ 增加	✎ 修改
站点公告	+ 增加	✎ 修改

认证和授权		
用户	+ 增加	✎ 修改
组	+ 增加	✎ 修改

最近动作

我的动作

无可用的

这是表就有了。大家可以多填写数据在里面，为接下来的内容做准备。