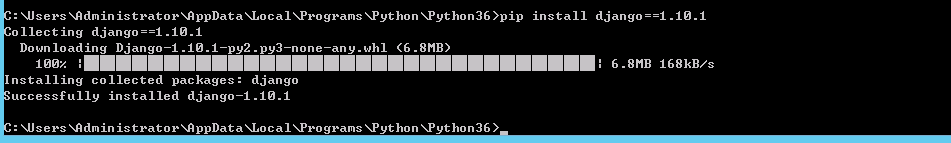
本地环境：Windows Server 2012 R2 Standard 64bit

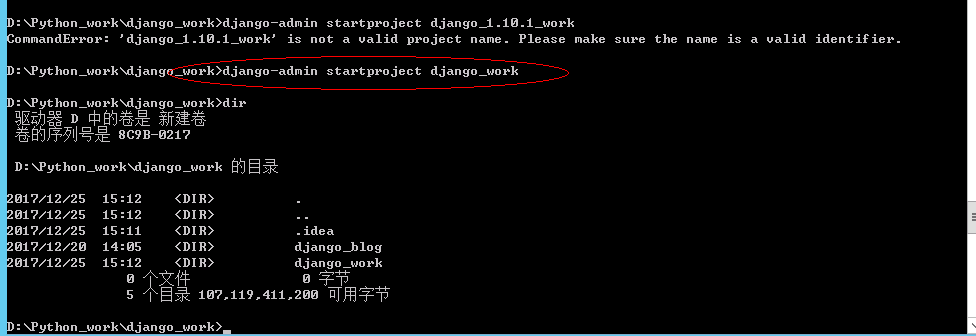
Python 版本 3.6

Django 1.10.1

>pip install django==1.10.1



>django-admin startproject django\_work



>cd django\_work

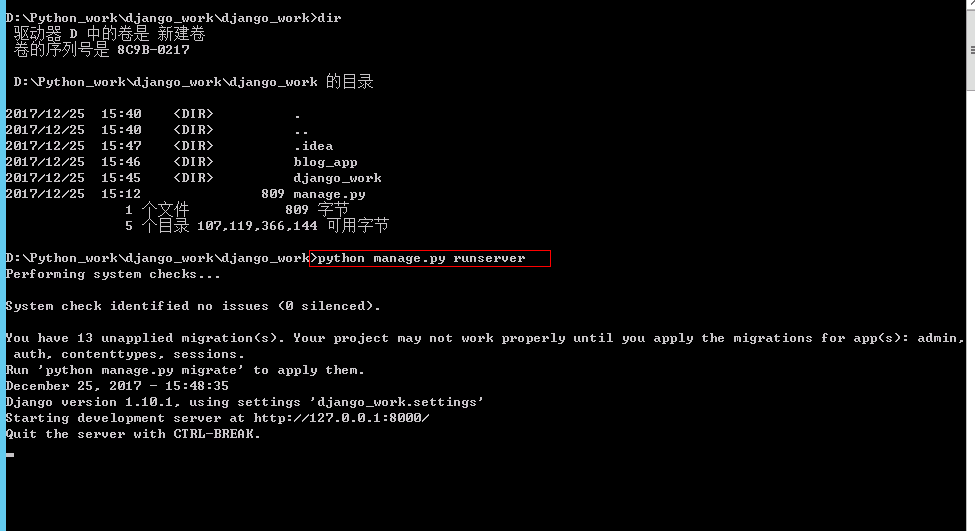
>python manage.py startapp blog\_app

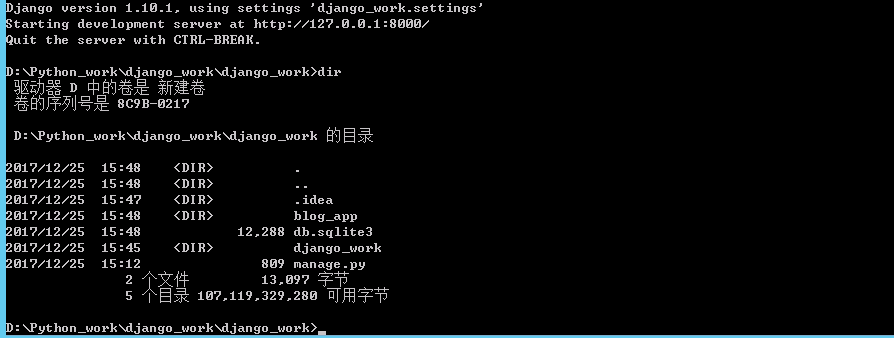


编辑 setting.py 文件：

ALLOWED\_HOSTS = [**'\*'**]  
  
*# Application definition*INSTALLED\_APPS = [  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'blog\_app'**,  
]

>python manage.py runserver

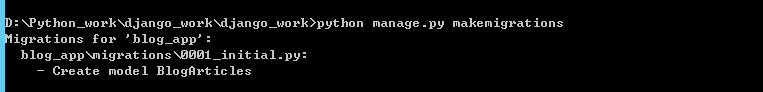




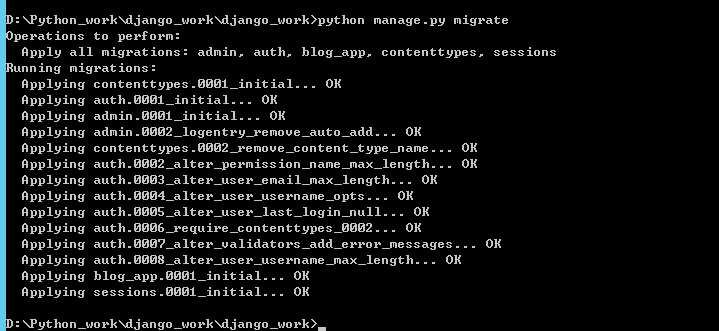
Vim blog\_app/models.py

**from** django.db **import** models  
**from** django.utils **import** timezone  
**from** django.contrib.auth.models **import** User  
  
**class** BlogArticles(models.Model):  
 title = models.CharField(max\_length=300)  
 author = models.ForeignKey(User, related\_name=**'blog\_posts'**)  
 *# author = models.ForeignKey(User, related\_name='mycomment', verbose\_name = u'评论人')  
 # author = models.ForeignKey(User)* body = models.TextField()  
 publish = models.DateTimeField(default=timezone.now)  
  
 **class** Mata:  
 ordering = (**"-publish"**, )  
  
 **def** \_\_str\_\_(self):  
 **return** self.title

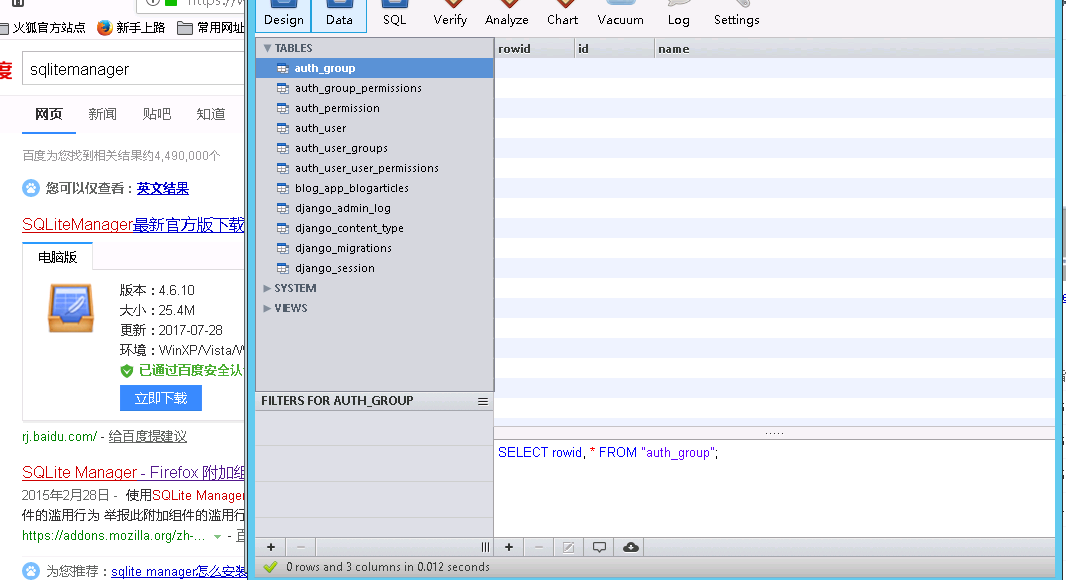
>python manage.py makemigrations

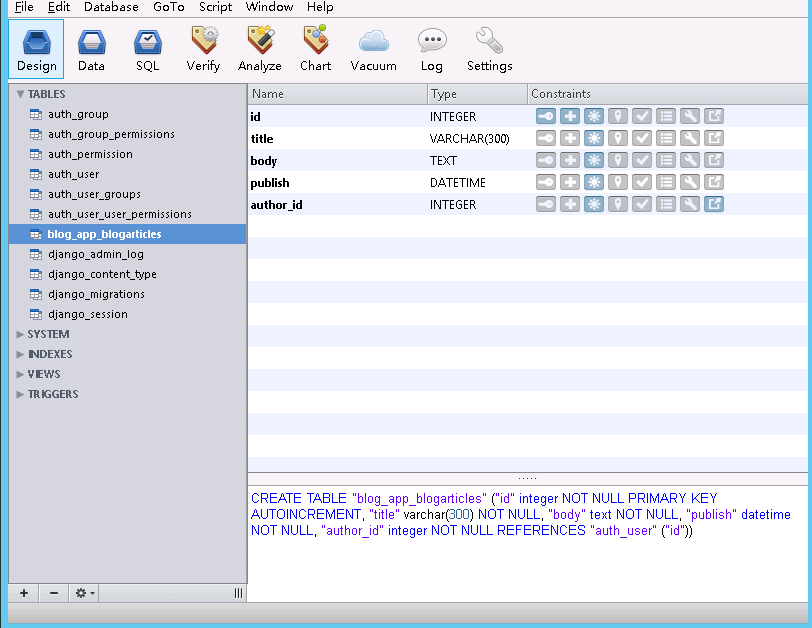


>python manage.py migrate









>

D:\Python\_work\django\_work\django\_work>python manage.py createsuperuser

Username (leave blank to use 'administrator'): admin

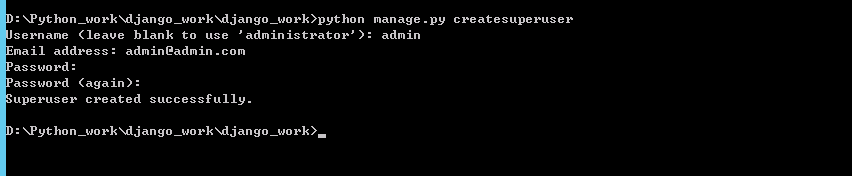
Email address: admin@admin.com

Password:Jfz!955988?

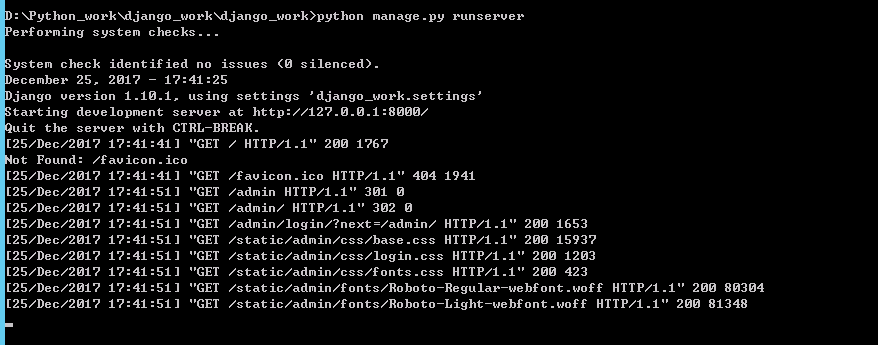
Password (again):

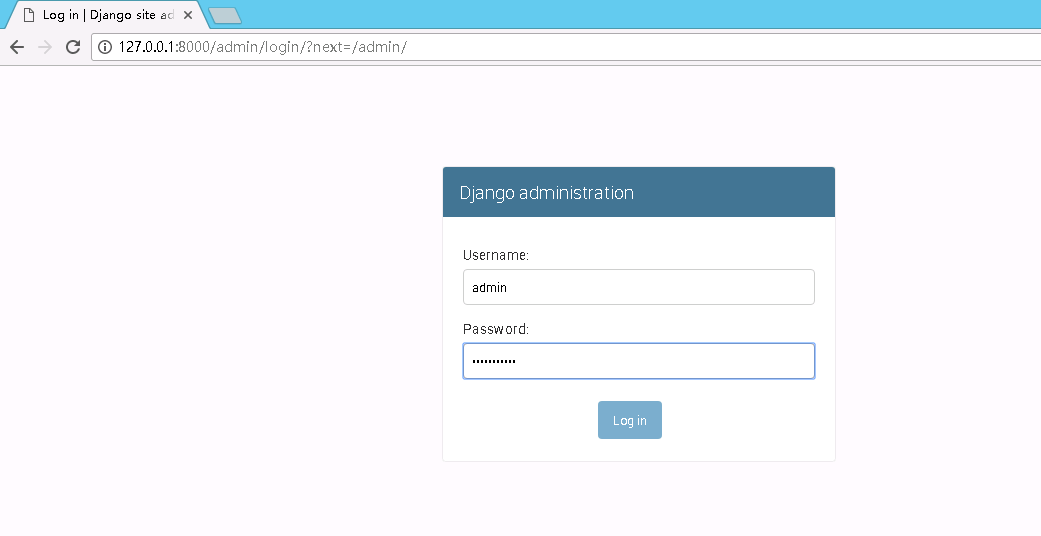
Superuser created successfully.

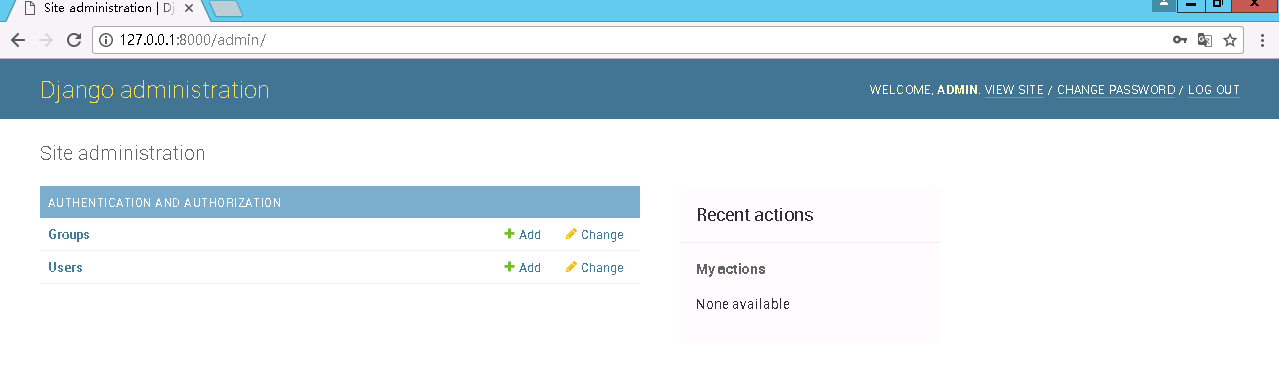
D:\Python\_work\django\_work\django\_work>



启动服务后runserver 然后访问：http://127.0.0.1:8000/admin

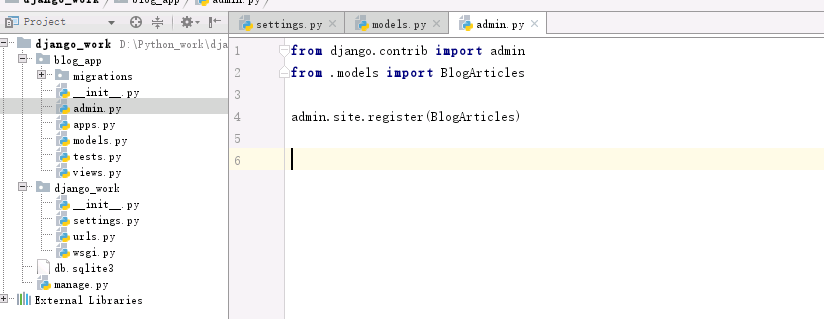




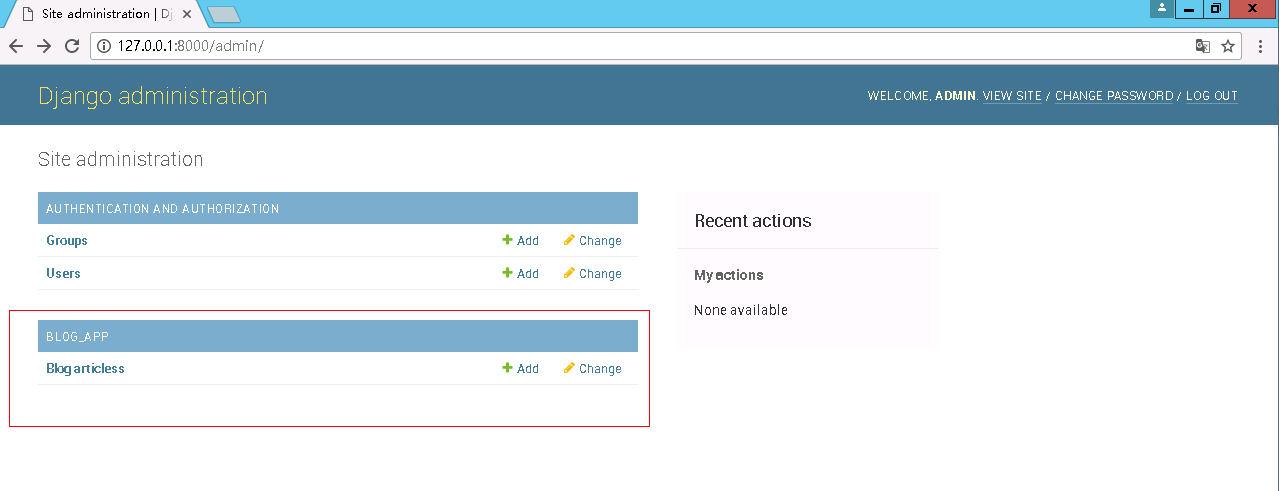


Vim ./blog\_app/admin.py

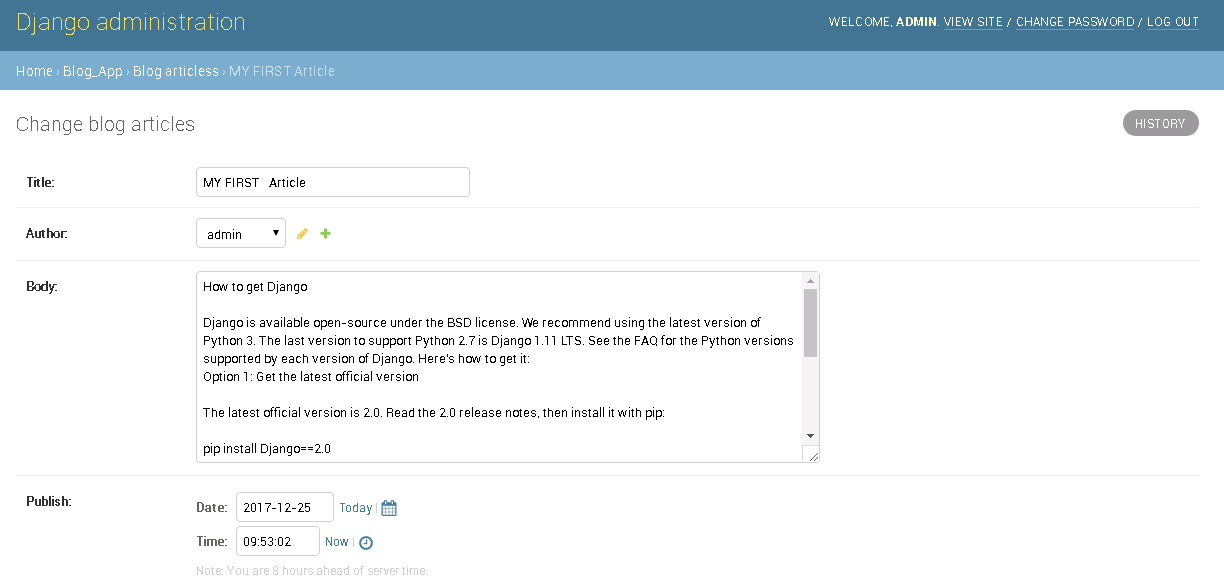
**from** django.contrib **import** admin  
**from** .models **import** BlogArticles  
  
admin.site.register(BlogArticles)



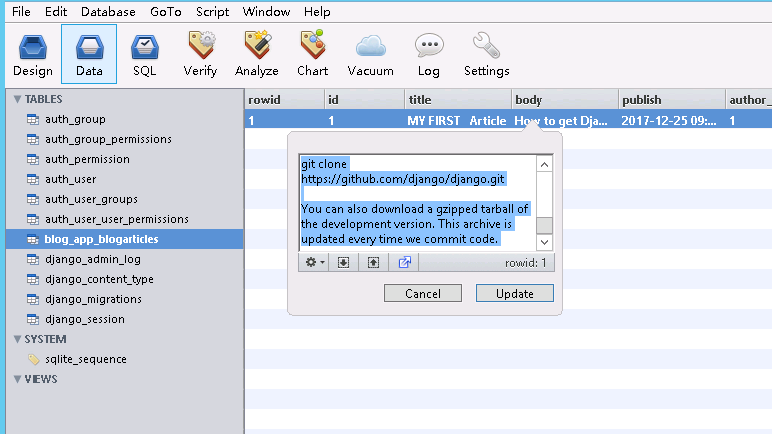
重启服务，刷新页面http://127.0.0.1:8000/admin/

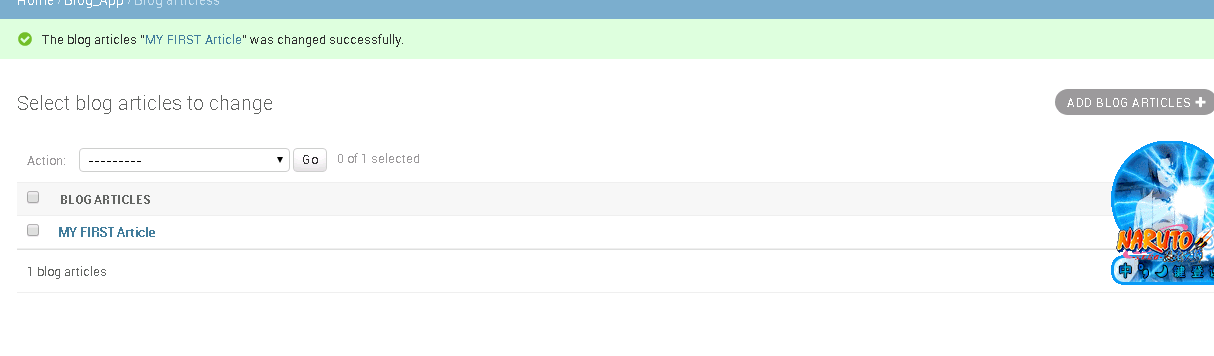


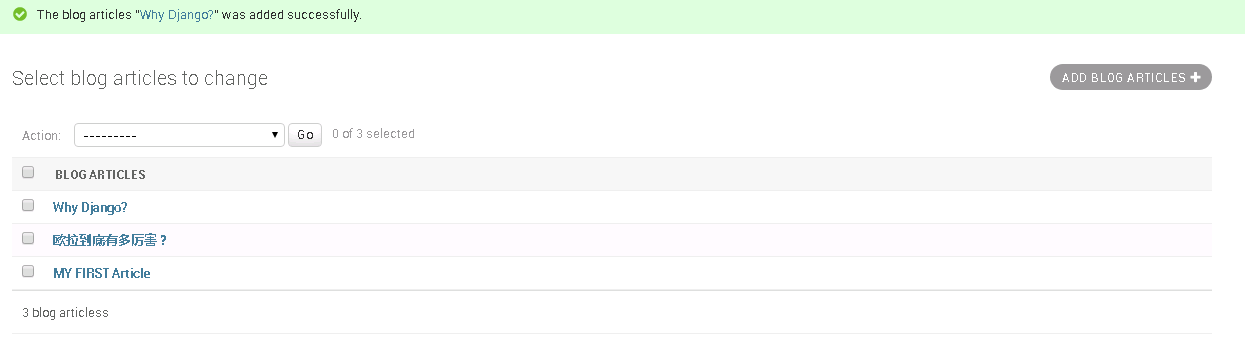
点击 Blog articles 右侧的 Add 按钮可以添加博客文章：



点击SAVE 后就可以在数据库中查看到加入的数据。

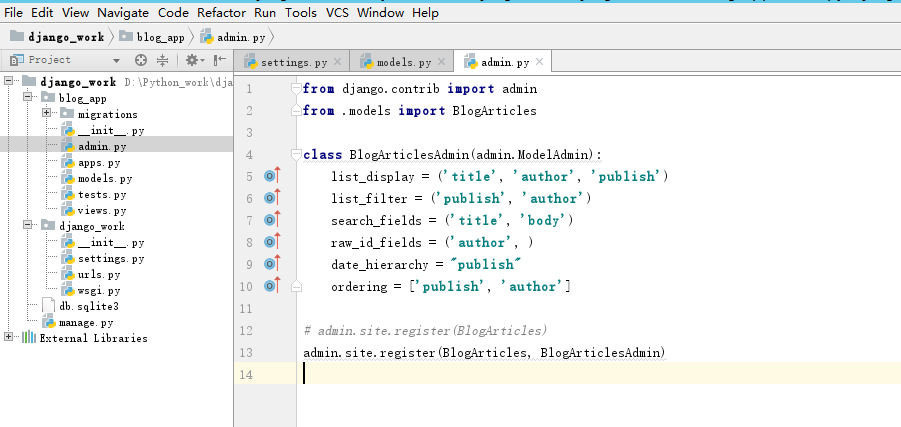






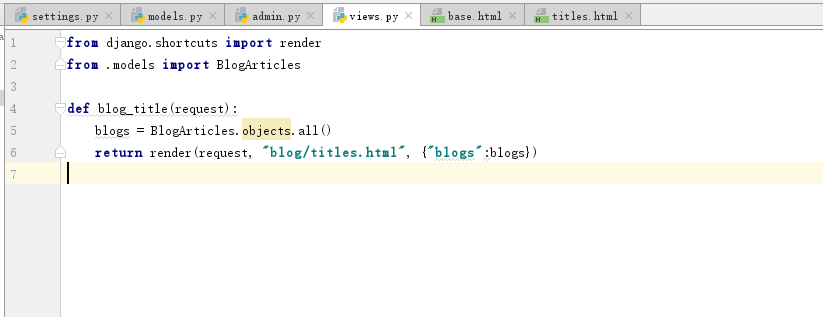
为了让列表页的信息丰富，还可以在./blog\_app/admin.py中多写些东西，进行补充：

**from** django.contrib **import** admin  
**from** .models **import** BlogArticles  
  
**class** BlogArticlesAdmin(admin.ModelAdmin):  
 list\_display = (**'title'**, **'author'**, **'publish'**)  
 list\_filter = (**'publish'**, **'author'**)  
 search\_fields = (**'title'**, **'body'**)  
 raw\_id\_fields = (**'author'**, )  
 date\_hierarchy = **"publish"** ordering = [**'publish'**, **'author'**]  
  
*# admin.site.register(BlogArticles)*admin.site.register(BlogArticles, BlogArticlesAdmin)

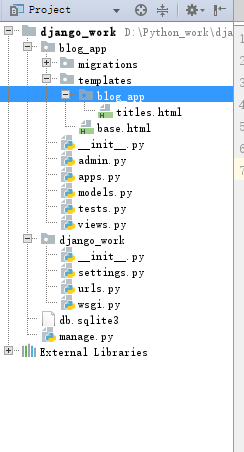


vim ./blog\_app/views.py

**from** django.shortcuts **import** render  
**from** .models **import** BlogArticles  
  
**def** blog\_title(request):  
 blogs = BlogArticles.objects.all()  
 **return** render(request, **"blog/titles.html"**, {**"blogs"**:blogs})



然后进行创建模板存放目录template，该目录要存放在应用（blog\_app）目录下。



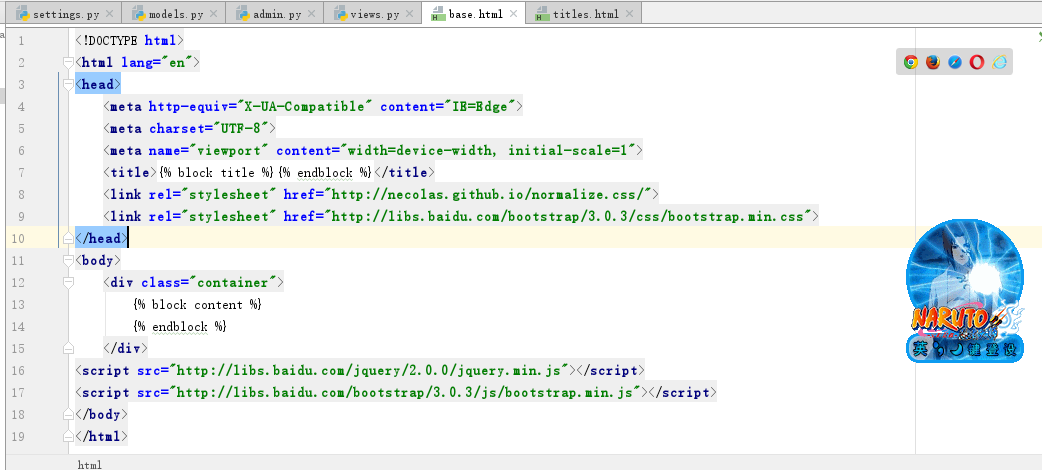
编写base.html和title.html文件

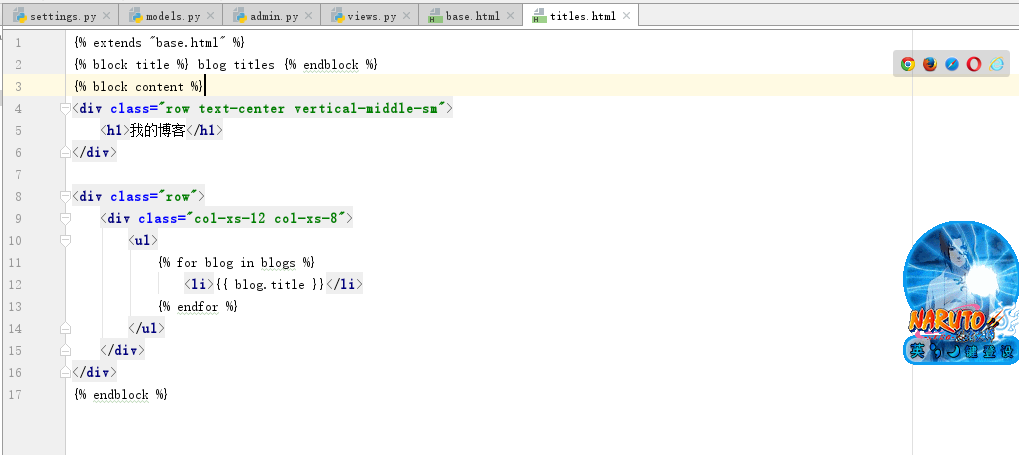
vim base.html

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta http-equiv="X-UA-Compatible" content="IE=Edge"**>  
 <**meta charset="UTF-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**title**>{% block title %}{% endblock %}</**title**>  
 <**link rel="stylesheet" href="http://necolas.github.io/normalize.css/"**>  
 <**link rel="stylesheet" href="http://libs.baidu.com/bootstrap/3.0.3/css/bootstrap.min.css"**>  
</**head**>  
<**body**>  
 <**div class="container"**>  
 {% block content %}  
 {% endblock %}  
 </**div**>  
<**script src="http://libs.baidu.com/jquery/2.0.0/jquery.min.js"**></**script**>  
<**script src="http://libs.baidu.com/bootstrap/3.0.3/js/bootstrap.min.js"**></**script**>  
</**body**>  
</**html**>

vim title.html

{% extends "base.html" %}  
  
{% block title %} blog titles {% endblock %}  
  
{% block content %}  
<**div class="row text-center vertical-middle-sm"**>  
 <**h1**>我的博客</**h1**>  
</**div**>  
  
<**div class="row"**>  
 <**div class="col-xs-12 col-xs-8"**>  
 <**ul**>  
 {% for blog in blogs %}  
 <**li**>{{ blog.title }}</**li**>  
 {% endfor %}  
 </**ul**>  
 </**div**>  
</**div**>  
{% endblock %}

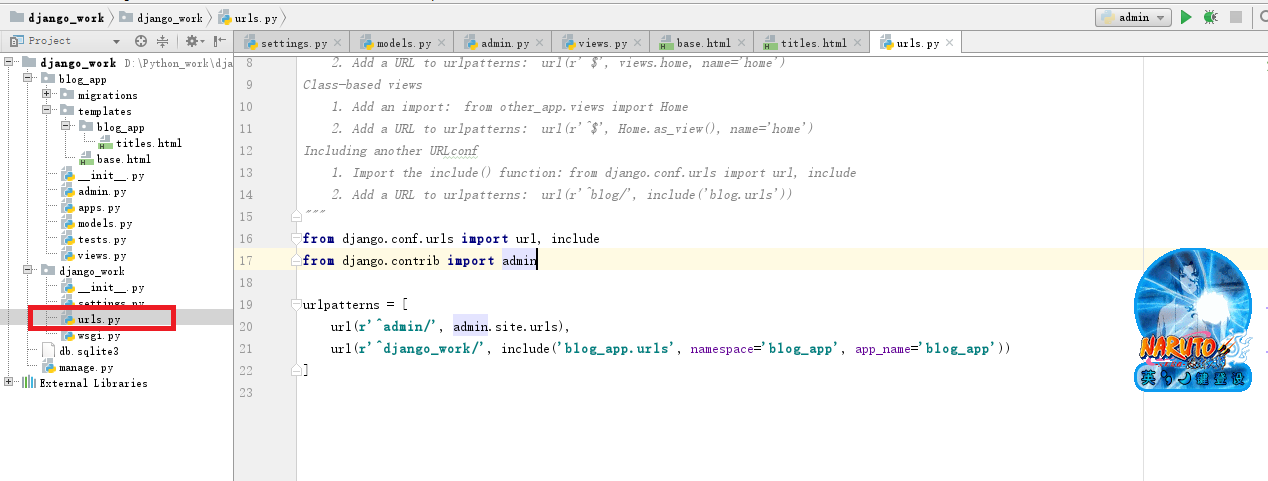




配置url。首先要配置./django\_work/urls.py

vim ./django\_work/urls.py

**from** django.conf.urls **import** url, include  
**from** django.contrib **import** admin  
  
urlpatterns = [  
 url(**r'^admin/'**, admin.site.urls),  
 url(**r'^blog\_app/'**, include(**'blog\_app.urls'**, namespace=**'blog\_app'**, app\_name=**'blog\_app'**))  
]

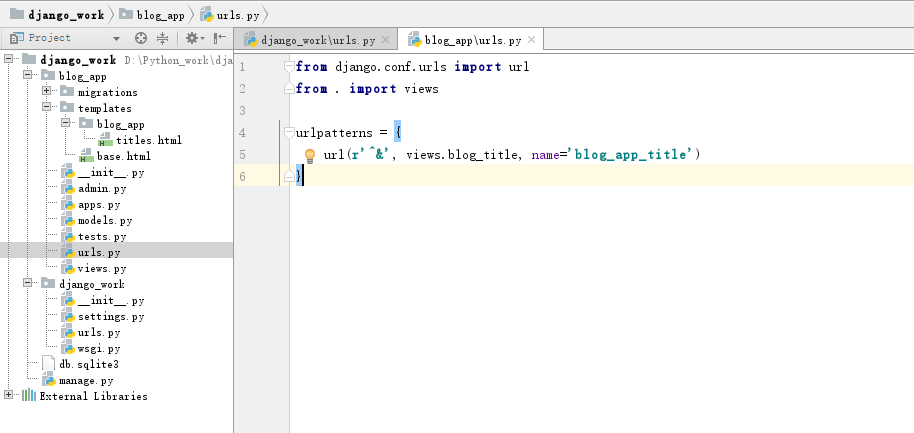


然后需要配置应用内的urls.py文件，该文件需要创建才会有，即：

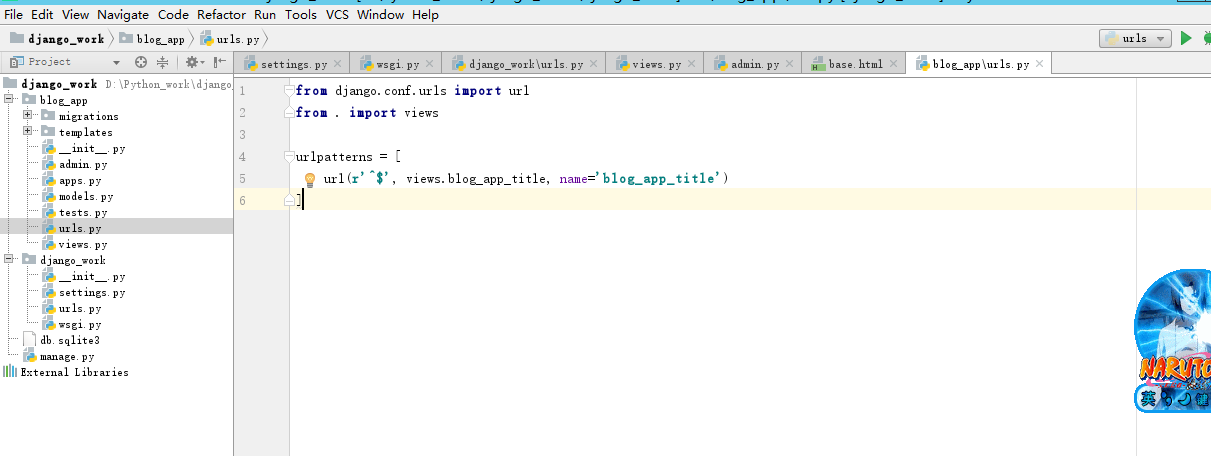
vim ./blog\_app/urls.py

**from** django.conf.urls **import** url  
**from** . **import** views  
  
urlpatterns = [  
 url(**r'^&'**, views.blog\_title, name=**'blog\_app\_title'**)  
]

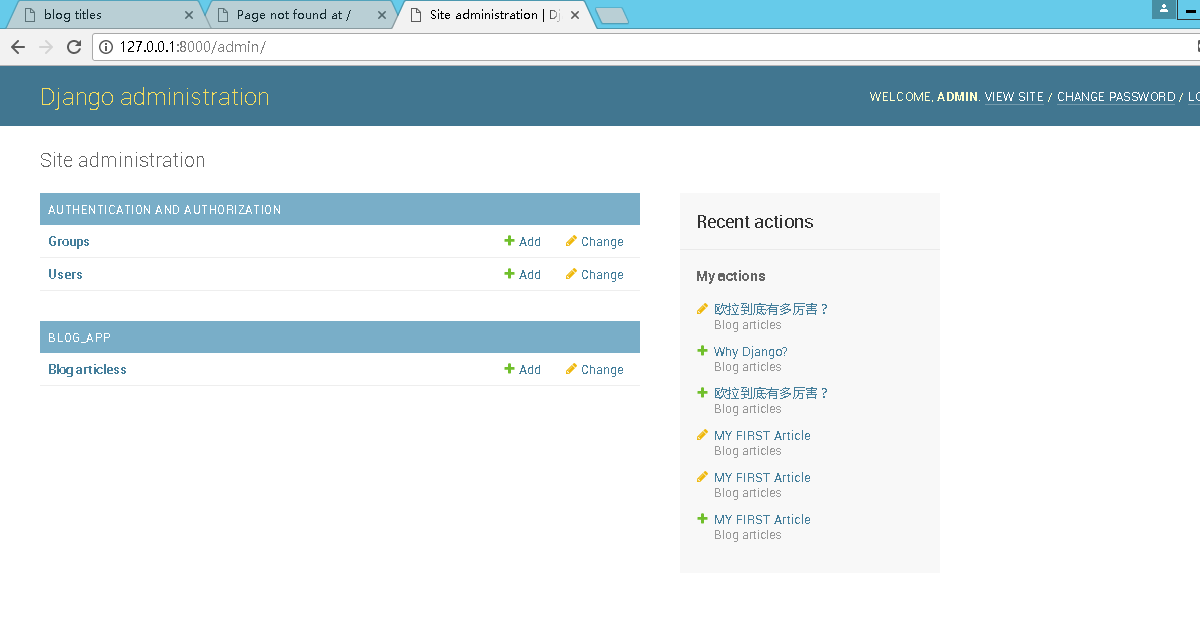
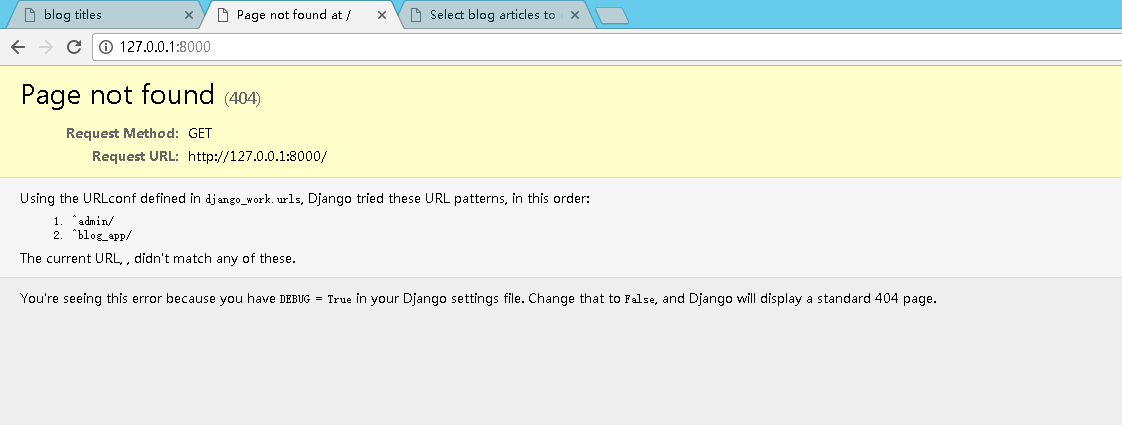
error:错误示范如下，写成如下会导致出现TYPEERROR错误。



正确的为：



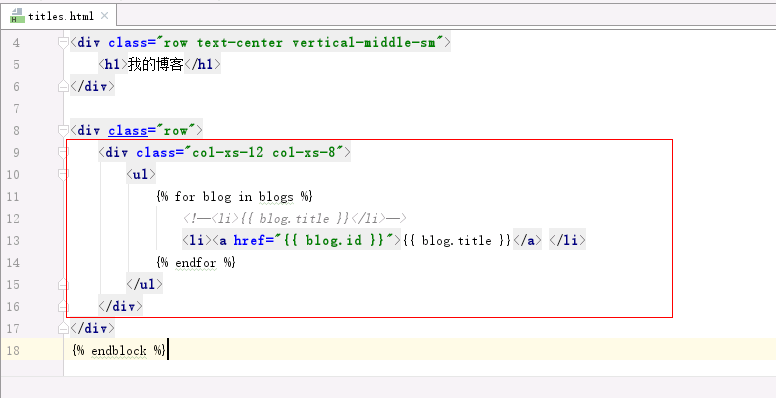
运行项目验证blog\_app显示效果：



为配置可以通过文章标题查看博客文档内容：

需要修改 ./blog\_app/template/blog\_app/titles.html

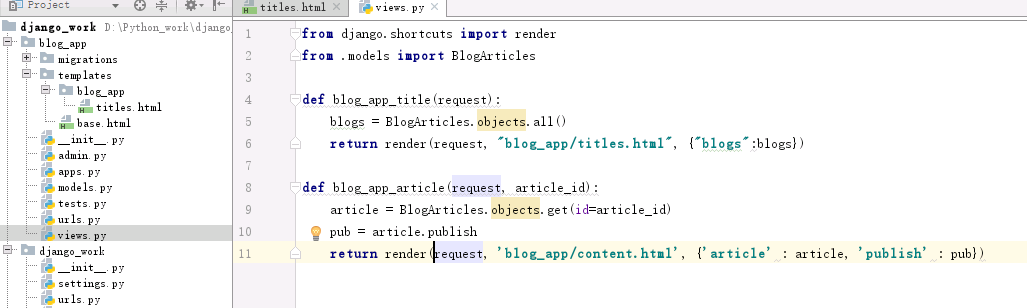
修改代码部分为：



<**ul**>  
 {% for blog in blogs %}  
 *<!--<li>{{ blog.title }}</li>-->* <**li**><**a href="{{ blog.id }}"**>{{ blog.title }}</**a**> </**li**>  
 {% endfor %}  
</**ul**>

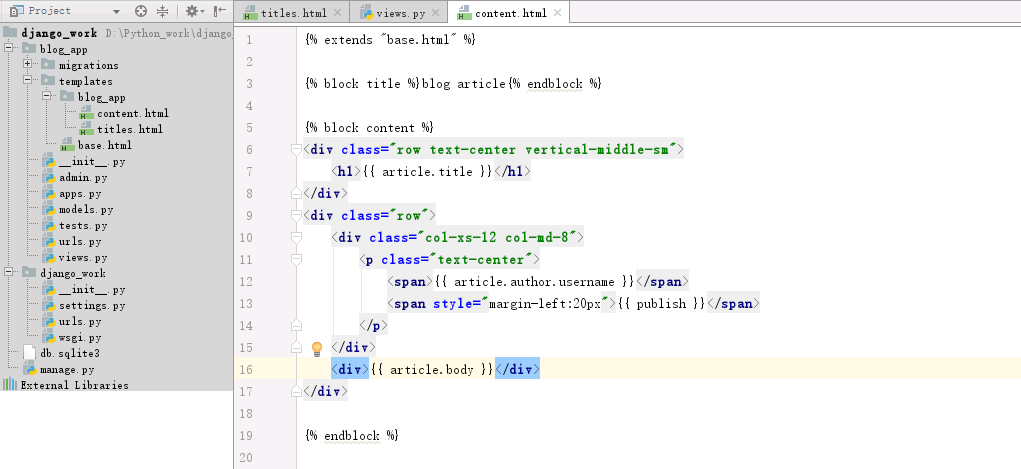
编辑 ./blog\_app/views.py文件，增加响应查看文章请求的函数 blog\_app\_article()

**def** blog\_app\_article(request, article\_id):  
 article = BlogArticles.objects.get(id=article\_id)  
 pub = article.publish  
 **return** render(request, **'blog\_app/content.html'**, {**'article'** : article, **'publish'** : pub})



然后进行编写对于的模板，创建./templates/blog\_app/content.html 文件

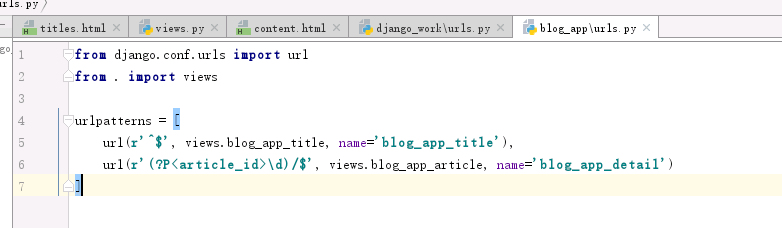
{% extends "base.html" %}  
  
{% block title %}blog article{% endblock %}  
  
{% block content %}  
<**div class="row text-center vertical-middle-sm"**>  
 <**h1**>{{ article.title }}</**h1**>  
</**div**>  
<**div class="row"**>  
 <**div class="col-xs-12 col-md-8"**>  
 <**p class="text-center"**>  
 <**span**>{{ article.author.username }}</**span**>  
 <**span style="**margin-left:20px**"**>{{ publish }}</**span**>  
 </**p**>  
 </**div**>  
 <**div**>{{ article.body }}</**div**>  
</**div**>  
  
{% endblock %}



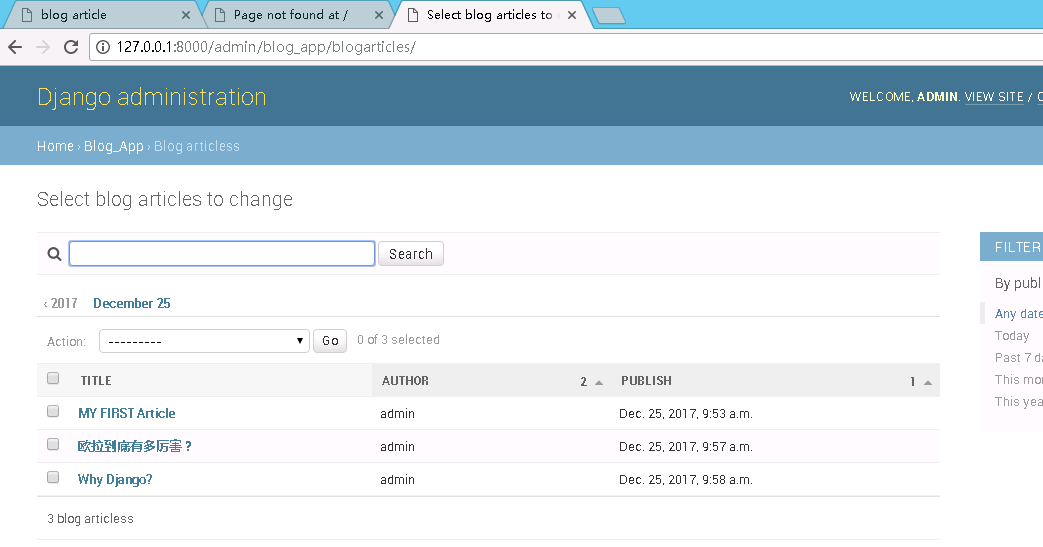
接着需要配置url，由于还是该应用blog\_app 内，所以不需要修改 ./django\_work/urls.py

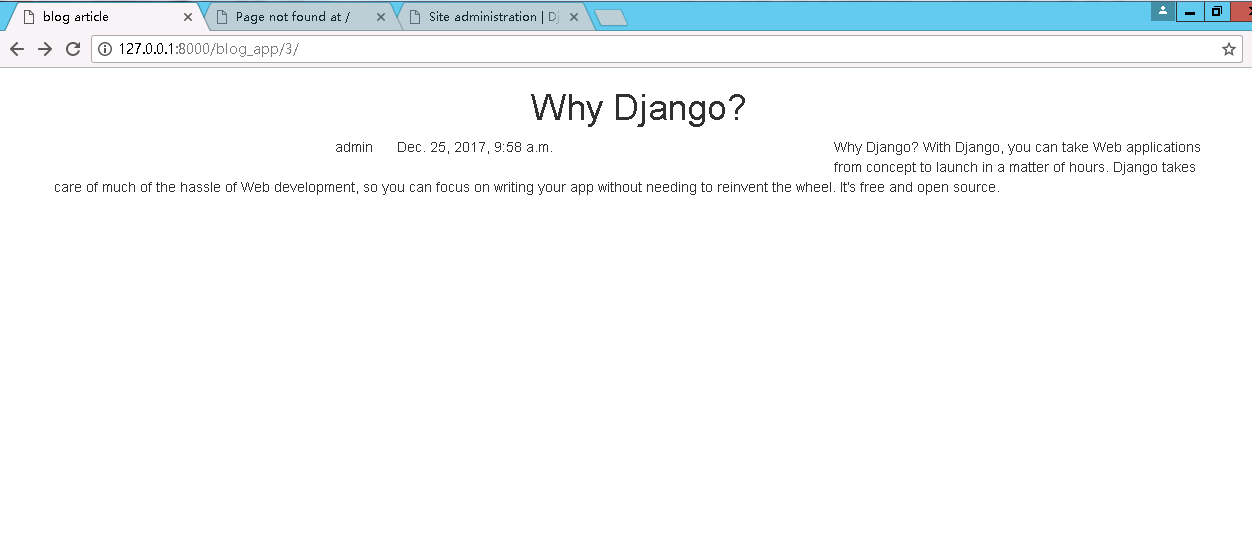
在blog\_app/urls.py 文件中新增URL路径就可以了

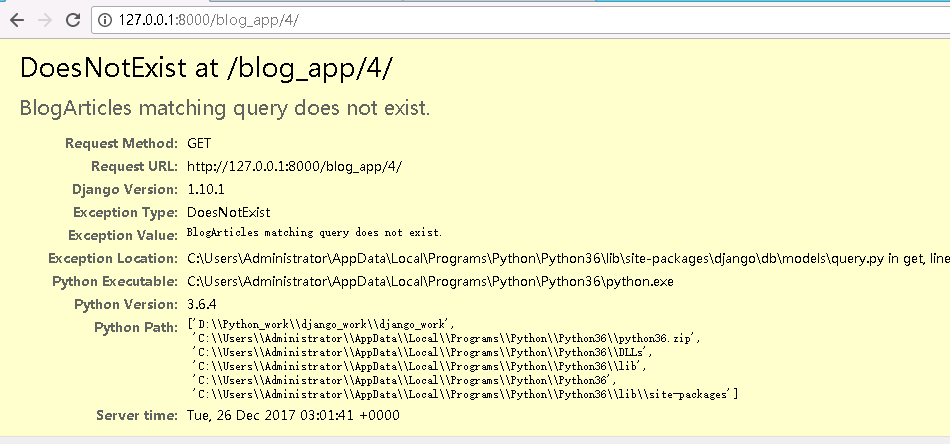
**from** django.conf.urls **import** url  
**from** . **import** views  
  
urlpatterns = [  
 url(**r'^$'**, views.blog\_app\_title, name=**'blog\_app\_title'**),  
 url(**r'(?P<article\_id>\d)/$'**, views.blog\_app\_article, name=**'blog\_app\_detail'**)  
]



配置完成后，重启服务访问进行测试，当前后台只有3片文档，在访问url中为4时，就出现报错了，需要进行异常的处理。



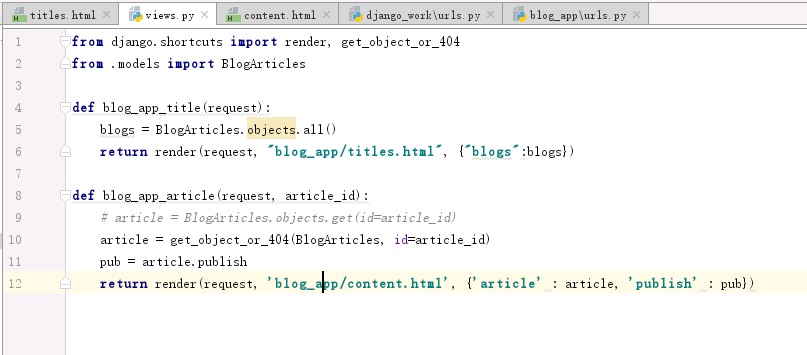


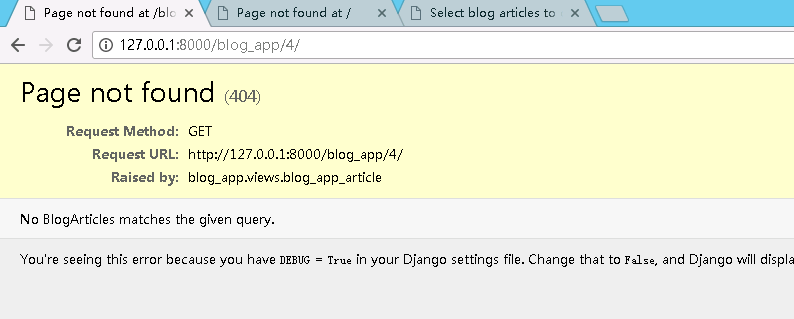


处理该场景异常问题，需要修改 ./blog\_app/views.py文件中的blog\_app\_atricle()函数：

修改后为：

**from** django.shortcuts **import** render, get\_object\_or\_404  
**from** .models **import** BlogArticles  
  
**def** blog\_app\_title(request):  
 blogs = BlogArticles.objects.all()  
 **return** render(request, **"blog\_app/titles.html"**, {**"blogs"**:blogs})  
  
**def** blog\_app\_article(request, article\_id):  
 *# article = BlogArticles.objects.get(id=article\_id)* article = get\_object\_or\_404(BlogArticles, id=article\_id)  
 pub = article.publish  
 **return** render(request, **'blog\_app/content.html'**, {**'article'** : article, **'publish'** : pub})





get\_object\_or\_404()方法，是简化对请求网页不存在的异常捕获。

Django 目前采用的是MTV模式。

M model 数据存取层。

T templte 表现层，处理与表现有关的事务

V view 视图，业务逻辑层。包含存取模型及调取相应模板的相关逻辑，是M 和 T 之间的桥梁。

额外：

windows 安装的 django ，Django自带的模板和配置文件在：

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Lib\site-packages\django\contrib\admin\templates

**第二章 用户管理**

自定义模板位置：

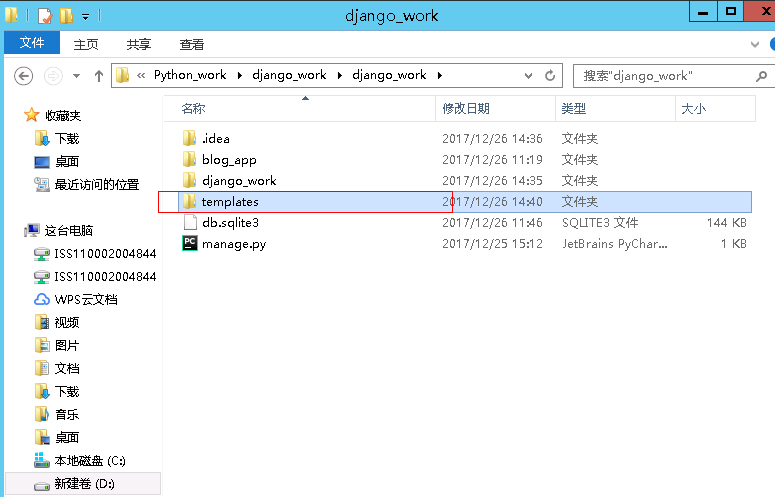
编辑 ./django\_work/setting.py 文件，修改如下：

TEMPLATES = [  
 {  
 **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  
 **'DIRS'**: [os.path.join(BASE\_DIR, **'templates'**), ],  
 **'APP\_DIRS'**: **False**,  
 **'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'**,  
 ],  
 },  
 },  
]

BASE\_DIR 是表示本项目的根目录，

APP\_DIRS 设置为False 是表示，不再允许Django安装默认方式寻找模板文件。

如配置文件所言，在项目根目录创建templates 文件夹。

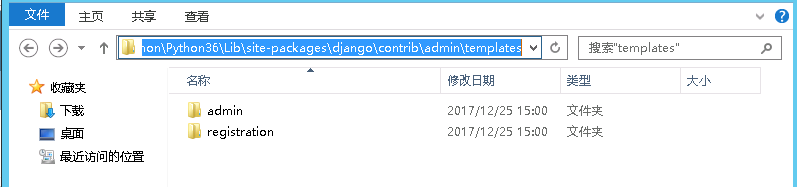


并将blog\_app/templates 下的文件拷贝到 根目录下 的 templates.

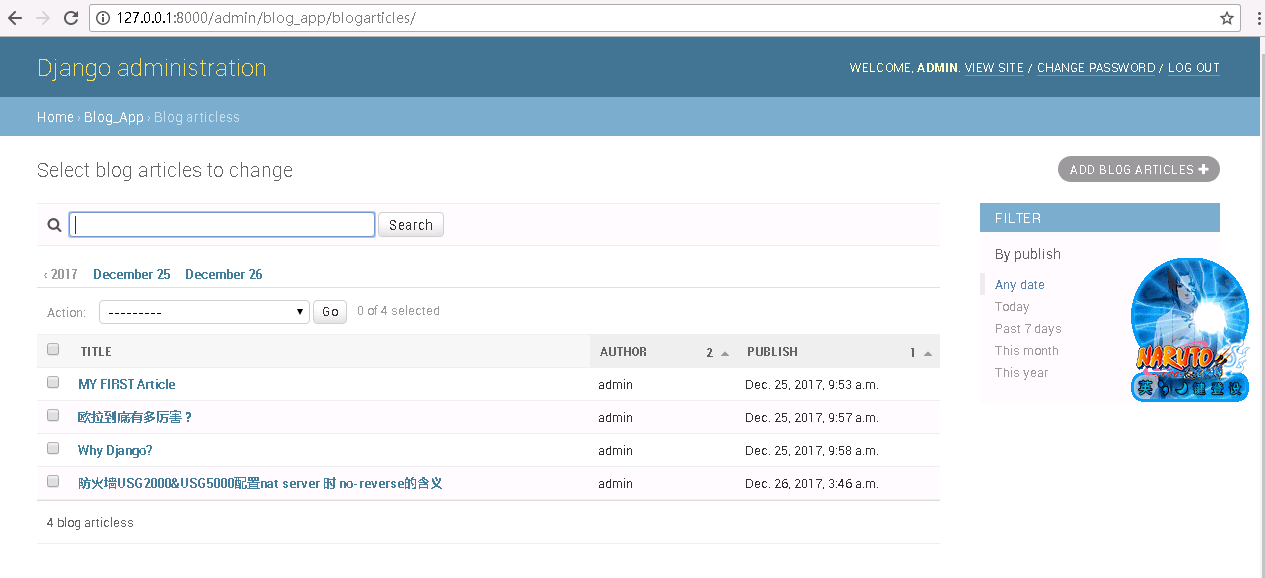
以及admin 对于的模板文件也需要拷贝到根目录下的 templates 下，

django 安装默认路径是在：

C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Lib\site-packages\django\contrib\admin\templates



重启服务进行验证：





自定义静态文件位置：

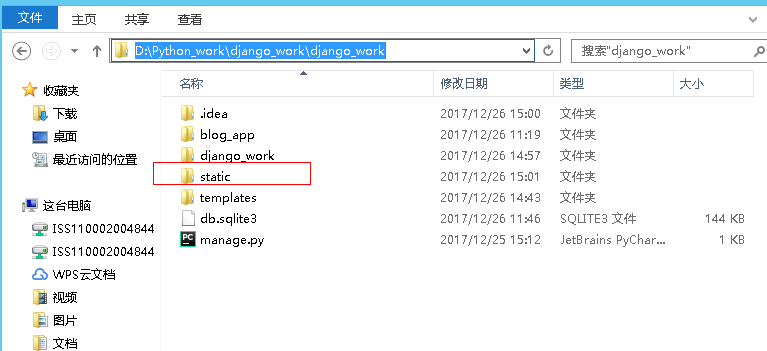
css、javasrcipt、image文件称为静态文件。

配置settings.py 文件，配置指定静态文件存在目录。

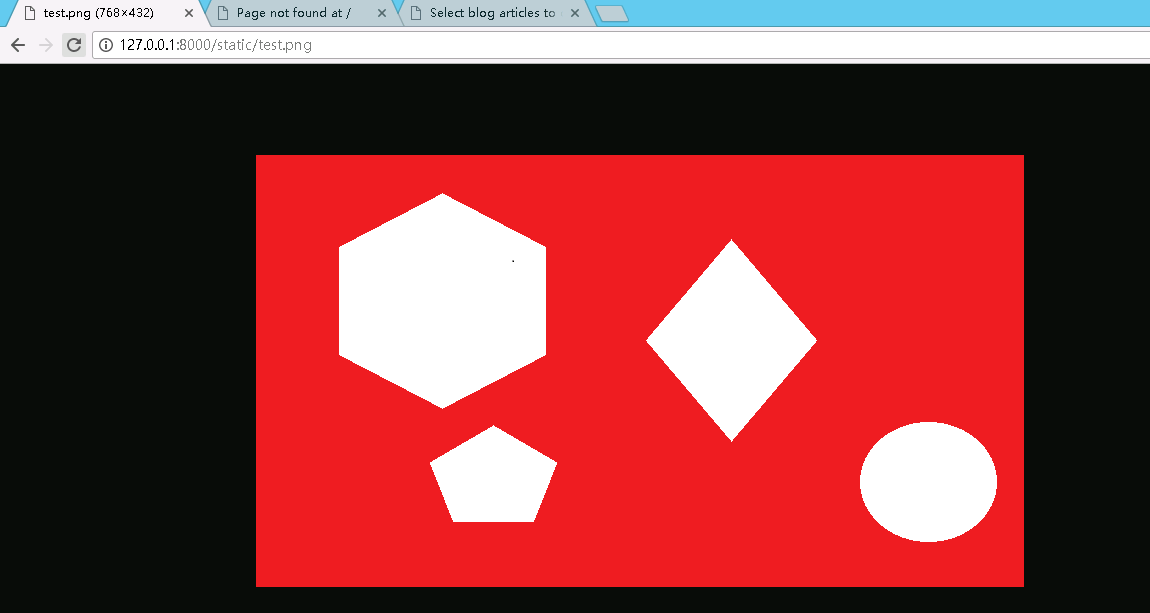
*# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/1.10/howto/static-files/*STATIC\_URL = **'/static/'**STATICFILES\_DIRS = (  
 os.path.join(BASE\_DIR, **"static"**)  
)



在项目根目录下，创建目录：static



创建目录后，可以在该目录下放置一个图片文件，进行测试：

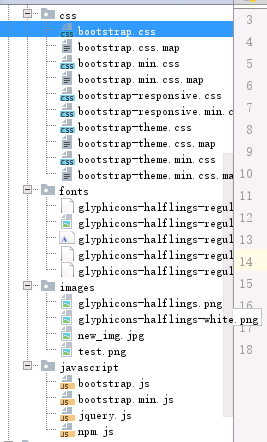


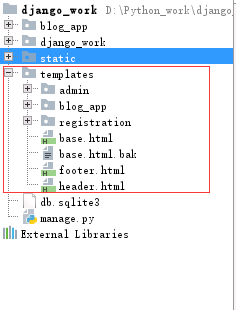
在static 中进行分类，可以在该目录下，新建存放 css 、image、js 、fonts 的子目录

相关 css 、 js 、fonts 可以在连接：

<https://v3.bootcss.com/getting-started/#download>

进行下载使用。

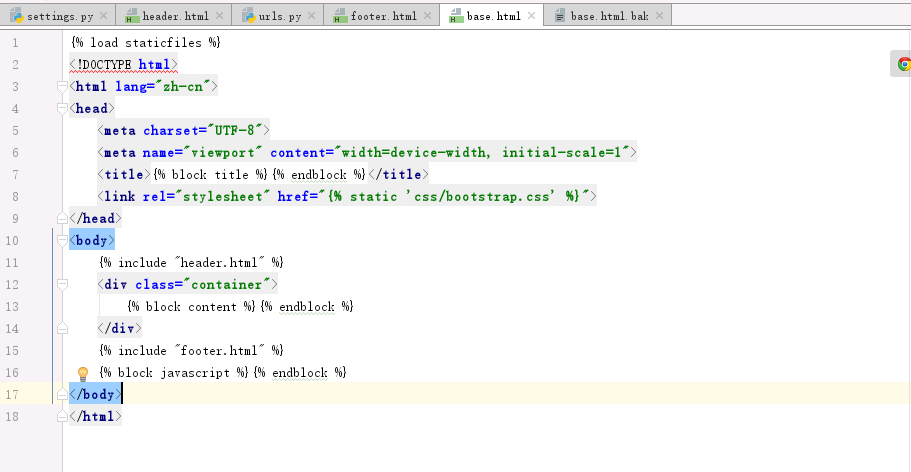


重新编写templates模板文件，

创建头部header.html 、footer.html 并修改重写 base.html文件。

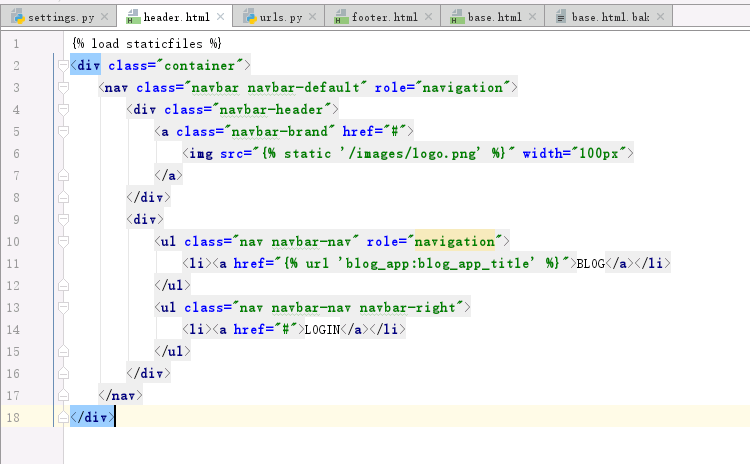
vim base.html

{% load staticfiles %}  
<!DOCTYPE **html**>  
<**html lang="zh-cn"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**title**>{% block title %}{% endblock %}</**title**>  
 <**link rel="stylesheet" href="{% static 'css/bootstrap.css' %}"**>  
</**head**>  
<**body**>  
 {% include "header.html" %}  
 <**div class="container"**>  
 {% block content %}{% endblock %}  
 </**div**>  
 {% include "footer.html" %}  
 {% block javascript %}{% endblock %}  
</**body**>  
</**html**>



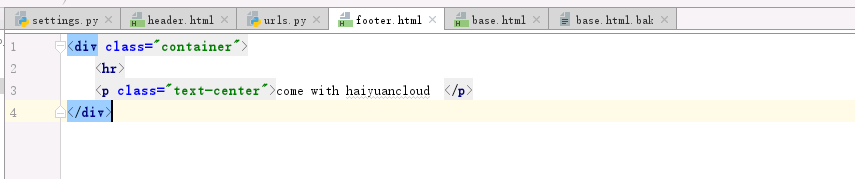
vim header.html

{% load staticfiles %}  
<**div class="container"**>  
 <**nav class="navbar navbar-default" role="navigation"**>  
 <**div class="navbar-header"**>  
 <**a class="navbar-brand" href="#"**>  
 <**img src="{% static '/images/logo.png' %}" width="100px"**>  
 </**a**>  
 </**div**>  
 <**div**>  
 <**ul class="nav navbar-nav" role="navigation"**>  
 <**li**><**a href="{% url 'blog\_app:blog\_app\_title' %}"**>BLOG</**a**></**li**>  
 </**ul**>  
 <**ul class="nav navbar-nav navbar-right"**>  
 <**li**><**a href="#"**>LOGIN</**a**></**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
</**div**>



vim foooter.html

<**div class="container"**>  
 <**hr**>  
 <**p class="text-center"**>come with haiyuancloud </**p**>  
</**div**>



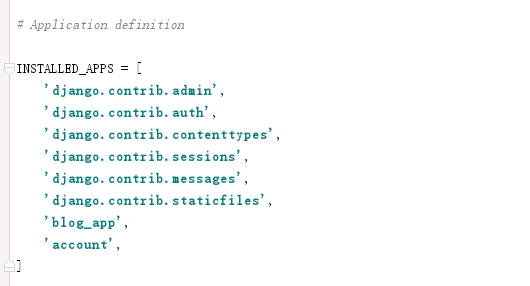
用户管理：

创建应用：

python manage.py startapp account

配置setting 文件：

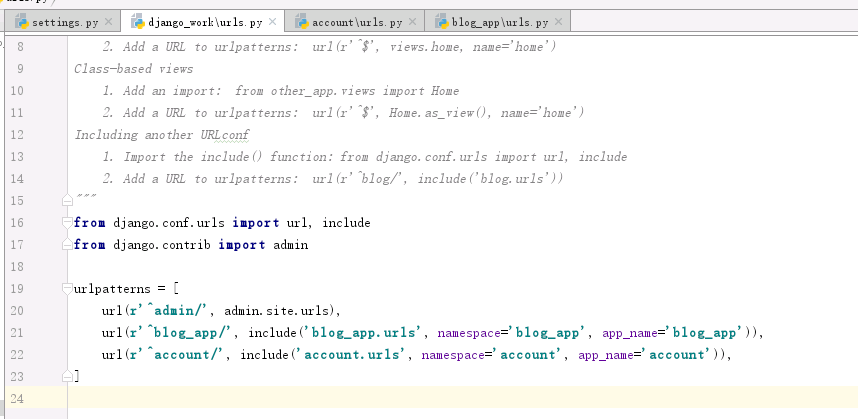
*# Application definition*INSTALLED\_APPS = [  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'blog\_app'**,  
 **'account'**,  
]



配置URL：

django\_work 目录下 urls.py ：

**from** django.conf.urls **import** url, include  
**from** django.contrib **import** admin  
  
urlpatterns = [  
 url(**r'^admin/'**, admin.site.urls),  
 url(**r'^blog\_app/'**, include(**'blog\_app.urls'**, namespace=**'blog\_app'**, app\_name=**'blog\_app'**)),  
 url(**r'^account/'**, include(**'account.urls'**, namespace=**'account'**, app\_name=**'account'**)),  
]



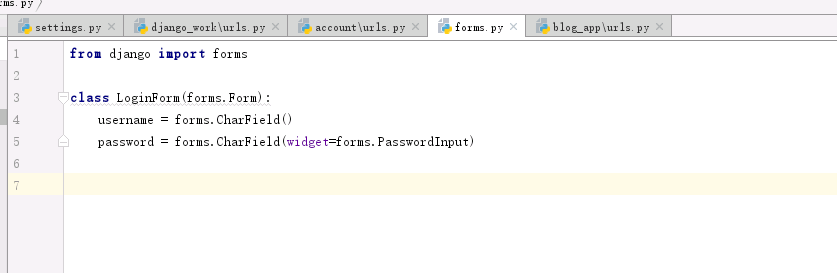
最后在 ./account 目前创建urls.py 文件：

**from** django.conf.urls **import** url  
**from** . **import** views  
**from** django.conf **import** settings  
  
urlpatterns = [  
 url(**r'^login/$'**, views.user\_login, name=**'user\_login'**)  
]

views.user\_login 意味着需要在views.py中创建一个名为user\_login 的函数来响应请求。

mkdir ./account/forms.py ## 专门存放 各种表单有关的类：

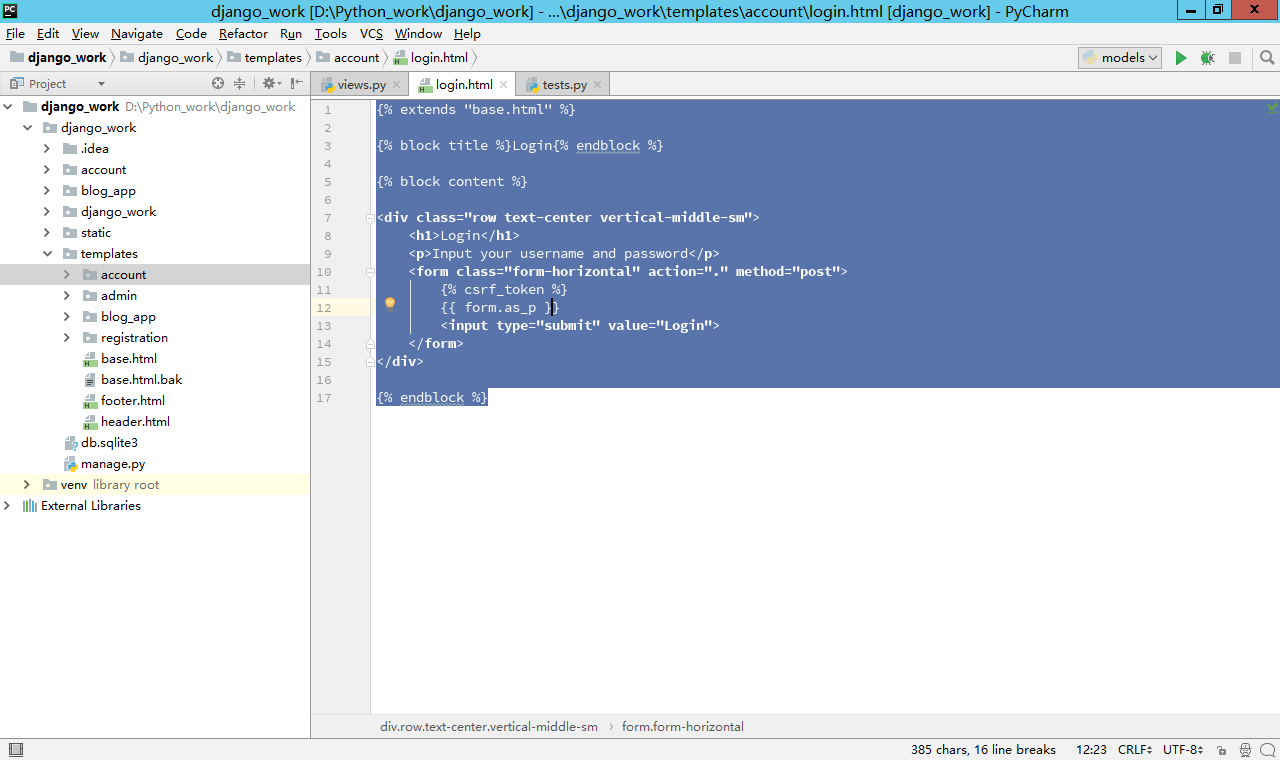
**from** django **import** forms  
  
**class** LoginForm(forms.Form):  
 username = forms.CharField()  
 password = forms.CharField(widget=forms.PasswordInput)



登陆前端页面

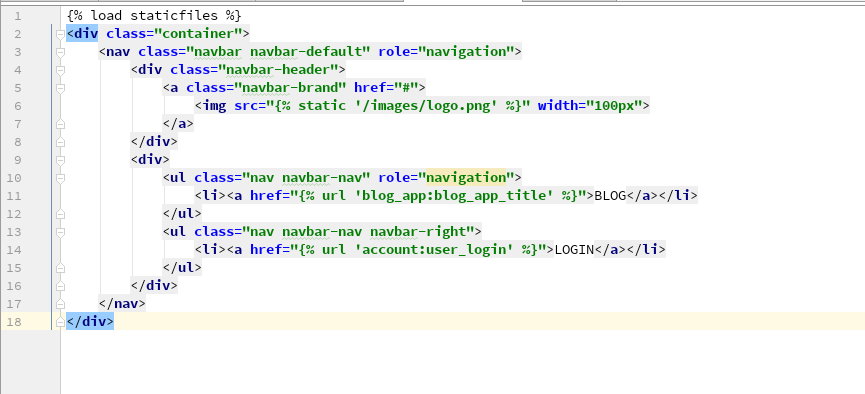
Vim ./tempaltes/account/login.html

{% extends "base.html" %}  
  
{% block title %}Login{% endblock %}  
  
{% block content %}  
  
<**div class="row text-center vertical-middle-sm"**>  
 <**h1**>Login</**h1**>  
 <**p**>Input your username and password</**p**>  
 <**form class="form-horizontal" action="." method="post"**>  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <**input type="submit" value="Login"**>  
 </**form**>  
</**div**>  
  
{% endblock %}



修改./templates/header.html

{% load staticfiles %}  
<**div class="container"**>  
 <**nav class="navbar navbar-default" role="navigation"**>  
 <**div class="navbar-header"**>  
 <**a class="navbar-brand" href="#"**>  
 <**img src="{% static '/images/logo.png' %}" width="100px"**>  
 </**a**>  
 </**div**>  
 <**div**>  
 <**ul class="nav navbar-nav" role="navigation"**>  
 <**li**><**a href="{% url 'blog\_app:blog\_app\_title' %}"**>BLOG</**a**></**li**>  
 </**ul**>  
 <**ul class="nav navbar-nav navbar-right"**>  
 <**li**><**a href="{% url 'account:user\_login' %}"**>LOGIN</**a**></**li**>  
 </**ul**>  
 </**div**>  
 </**nav**>  
</**div**>

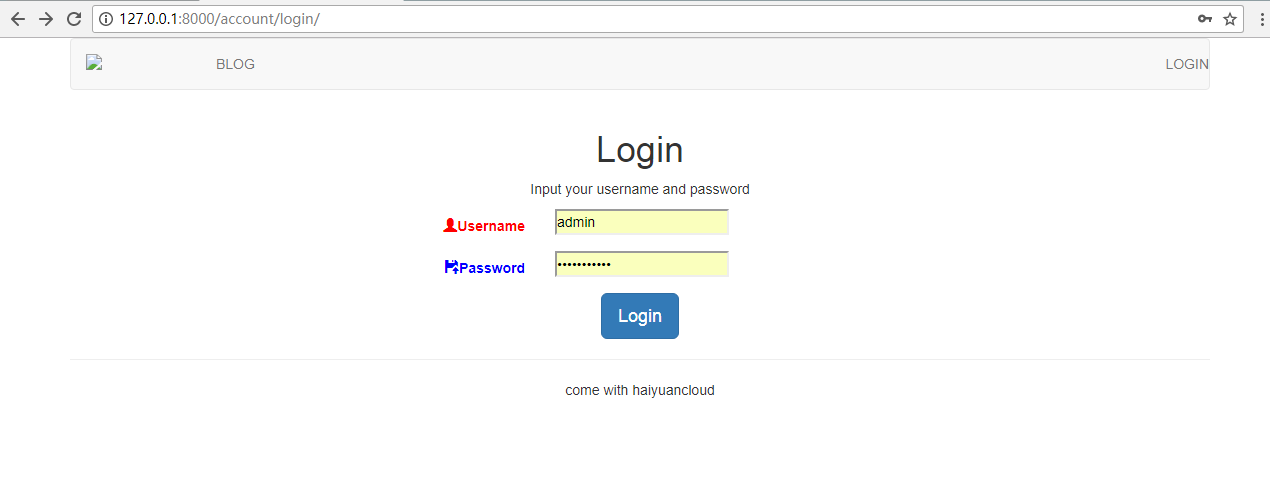


主要是将 “Login”的a标签链接调用进行修改。

考虑到login.html页面上一些样式比较死板，再进行后续优化。

{% extends "base.html" %}  
  
{% load staticfiles %}  
  
{% block title %}Login{% endblock %}  
  
{% block content %}  
  
<**div class="row text-center vertical-middle-sm"**>  
 <**h1**>Login</**h1**>  
 <**p**>Input your username and password</**p**>  
 <**form class="form-horizontal" action="." method="post"**>  
 {% csrf\_token %}  
 *<!--{{ form.as\_p }}-->* <**div class="form-group"**>  
 <**label for="{{ form.username.id\_for\_label }}" class="col-md-5 control-label" style="**color:red**"**>  
 <**span class="glyphicon glyphicon-user"**></**span**>Username  
 </**label**>  
 <**div class="col-md-6 text-left"**>{{ form.username }}</**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label for="{{ form.password.id\_for\_label }}" class="col-md-5 control-label" style="**color:blue**"**>  
 <**span class="glyphicon glyphicon-floppy-open"**></**span**>Password  
 </**label**>  
 <**div class="col-md-6 text-left"**>{{ form.password }}</**div**>  
 </**div**>  
 <**input class="btn btn-primary btn-lg" type="submit" value="Login"**>  
 </**form**>  
</**div**>  
  
{% endblock %}





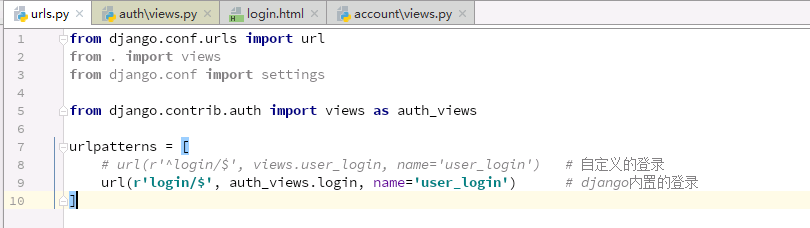
用内置方法实现登录和退出

进入django 目录中，找到写有内置登录函数的文件。

即C:\Users\Administrator\AppData\Local\Programs\Python\Python36\Lib\site-packages\django\contrib\auth\views.py

Vim ./account/urls.py

**from** django.conf.urls **import** url  
**from** . **import** views  
**from** django.conf **import** settings  
  
**from** django.contrib.auth **import** views **as** auth\_views  
  
urlpatterns = [  
 *# url(r'^login/$', views.user\_login, name='user\_login') # 自定义的登录* url(**r'login/$'**, auth\_views.login, name=**'user\_login'**) *# django内置的登录*]



需要在./templates/registration 中建立 login.html 模板。

可以直接将./templates/account/login.html 复制过来。

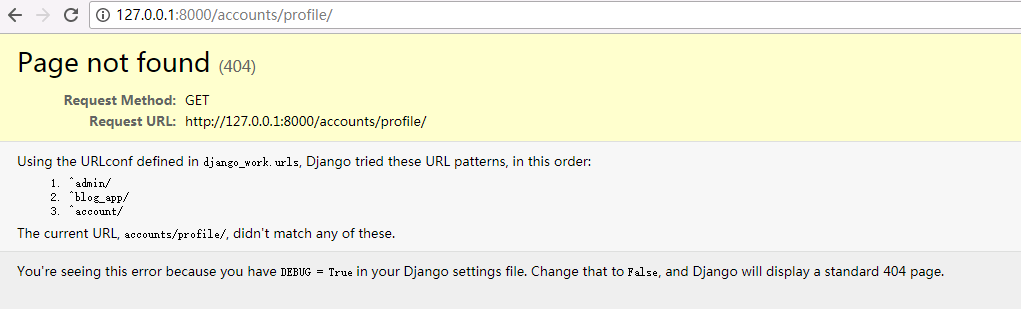
{% extends "base.html" %}  
  
{% load staticfiles %}  
  
{% block title %}Login{% endblock %}  
  
{% block content %}  
  
<**div class="row text-center vertical-middle-sm"**>  
 <**h1**>Login</**h1**>  
 <**p**>Input your username and password</**p**>  
 <**form class="form-horizontal" action="{% url 'account:user\_login' %}" method="post"**>  
 {% csrf\_token %}  
 *<!--{{ form.as\_p }}-->* <**div class="form-group"**>  
 <**label for="{{ form.username.id\_for\_label }}" class="col-md-5 control-label" style="**color:red**"**>  
 <**span class="glyphicon glyphicon-user"**></**span**>Username  
 </**label**>  
 <**div class="col-md-6 text-left"**>{{ form.username }}</**div**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label for="{{ form.password.id\_for\_label }}" class="col-md-5 control-label" style="**color:blue**"**>  
 <**span class="glyphicon glyphicon-floppy-open"**></**span**>Password  
 </**label**>  
 <**div class="col-md-6 text-left"**>{{ form.password }}</**div**>  
 </**div**>  
 <**input class="btn btn-primary btn-lg" type="submit" value="Login"**>  
 </**form**>  
</**div**>  
  
{% endblock %}

主要修改了一句为：

<**form class="form-horizontal" action="{% url 'account:user\_login' %}" method="post"**>

修改完后，可以正常打开登录页面。点击登录后，会进入一个404页面报错。

报错页面为：

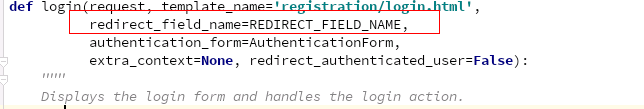


从上图可以看到点击“登录”，发现页面被跳转到了：

<http://127.0.0.1:8000/accounts/profile/>

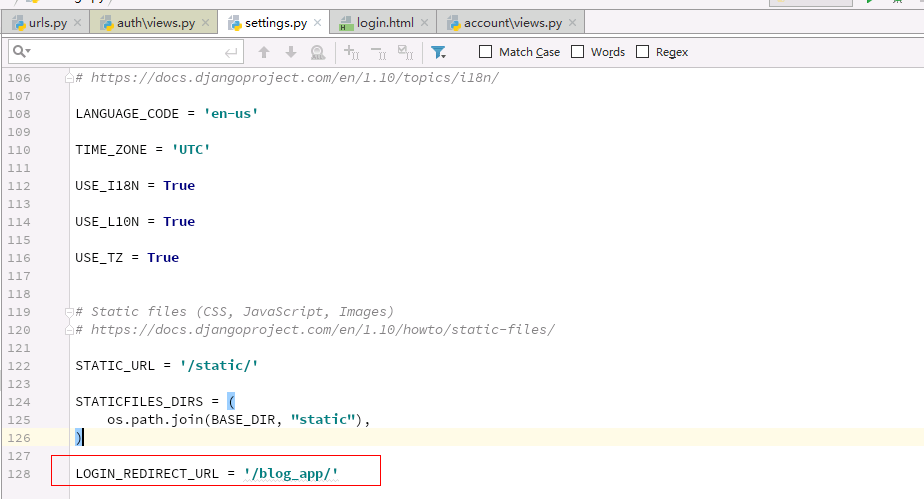
该跳转是由于内置函数中的参数：redirect\_field\_name=REDIRECT\_FIELD\_NAME,

决定的。



如果需要修改重定向的值，需要修改 ./python\_work/settings.py 中设置LOGIN\_REDIRECT\_URL = **'/blog\_app/'**

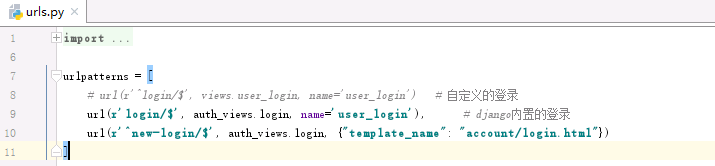
该配置可以手写加至最后。



在观察内置函数login的源码后，发现可以传参数（template\_name）传值，就是传一个新的模板文件。

Vim ./account/urls.py

**from** django.conf.urls **import** url  
**from** . **import** views  
**from** django.conf **import** settings  
  
**from** django.contrib.auth **import** views **as** auth\_views  
  
urlpatterns = [  
 *# url(r'^login/$', views.user\_login, name='user\_login') # 自定义的登录* url(**r'login/$'**, auth\_views.login, name=**'user\_login'**), *# django内置的登录* url(**r'^new-login/$'**, auth\_views.login, {**"template\_name"**: **"account/login.html"**})  
]



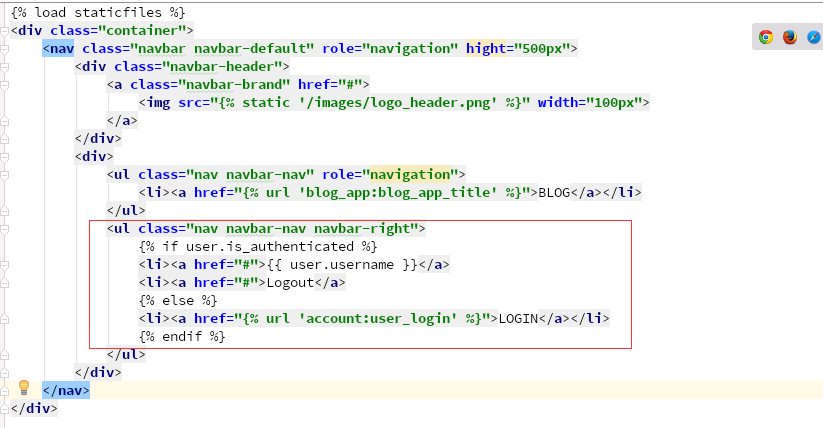
为进行对比，原来的URL不变，新增加一个URL进行配置，以一个字典类型，向默认的auth\_views.login 函数对象传template\_name的值，account/login.html就是此前已经写好的登录模板文件。

**判断用户是否登录**

需要在页面根据用户当前是否登录来进行不用的显示策略等。

修改 ./templates/header.html 文件内容，添加策略：当前有用户登录时，在header部分显示 用户名和logout ，无用户登录时，显示login。

{% load staticfiles %}  
<**div class="container"**>  
 <**nav class="navbar navbar-default" role="navigation" hight="500px"**>  
 <**div class="navbar-header"**>  
 <**a class="navbar-brand" href="#"**>  
 <**img src="{% static '/images/logo\_header.png' %}" width="100px"**>  
 </**a**>  
 </**div**>  
 <**div**>  
 <**ul class="nav navbar-nav" role="navigation"**>  
 <**li**><**a href="{% url 'blog\_app:blog\_app\_title' %}"**>BLOG</**a**></**li**>  
 </**ul**>  
 <**ul class="nav navbar-nav navbar-right"**>  
 {% if user.is\_authenticated %}  
 <**li**><**a href="#"**>{{ user.username }}</**a**>  
 <**li**><**a href="#"**>Logout</**a**>  
 {% else %}  
 <**li**><**a href="{% url 'account:user\_login' %}"**>LOGIN</**a**></**li**>  
 {% endif %}  
 </**ul**>  
 </**div**>  
 </**nav**>  
</**div**>

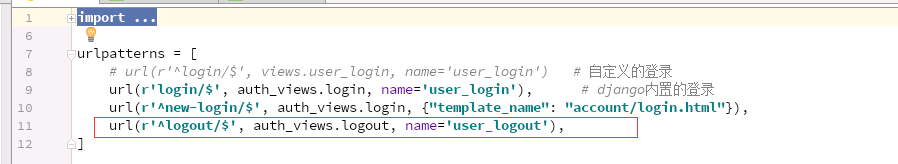


**内置的退出方法**

首先配置URL

Vim ./account/urls.py 文件，增加内容：

urlpatterns = [  
 *。。。*  
 url(**r'^logout/$'**, auth\_views.logout, name=**'user\_logout'**),  
]

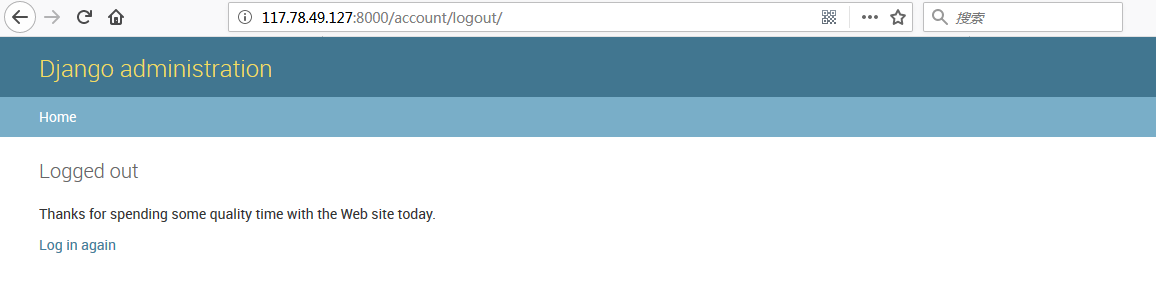


此处配置好后，修改./templates/header.html 中 Logout的登出链接。

修改为：

<**li**><**a href="{% url 'account:user\_logout' %}"**>Logout</**a**>

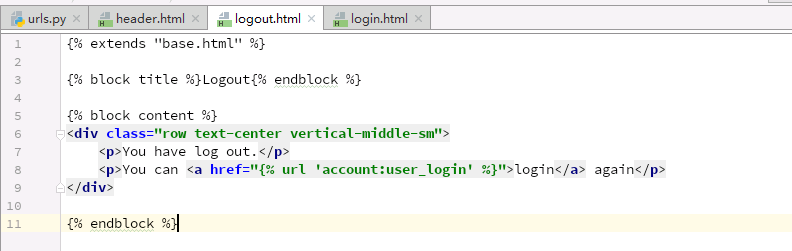
刷新页面后，登录状态下，点击logout按钮，跳转页面。



表示已经成功logout。

该页面时Django默认的模板文件，在./templates/account/中创建一个logout.html文件。

{% extends "base.html" %}  
  
{% block title %}Logout{% endblock %}  
  
{% block content %}  
<**div class="row text-center vertical-middle-sm"**>  
 <**p**>You have log out.</**p**>  
 <**p**>You can <**a href="{% url 'account:user\_login' %}"**>login</**a**> again</**p**>  
</**div**>  
  
{% endblock %}



进行配置 退出的URL配置。

Vim ./account/urls.py

url(**r'^logout/$'**, auth\_views.logout, {**'template\_name'**: **'account/logout.html'**}, name=**'user\_logout'**),

