报障系统是基于博客系统升级而来的。

# 需求分析

## 报障

### 用户：

提交报障单

自己报障记录

### 处理者：

查看所有人报障单

处理报账单

## 知识库（博客）

### 主页：

展示最新文章

展示最热文章

展示评论最多文章

分类查看

### 个人博客：

个人博客主页

个人博客文章详细：赞，踩，评论

个人博客分类：标签、分类、时间

个人博客主题定制：后台修改

### 后台管理：

个人信息管理

个人标签

个人分类

个人文章

# 数据库设计

用户表： uid,username,pwd,email,img,

博客表：bid,surfix(不同的人有不同的后缀),theme,title,summary, FK(用户表,unique)=OneToOne(用户表)

互粉表： id 明星ID（用户表） 粉丝ID（用户表）

2 1

1 2

1 3

5 3

报障单：UUID(一大堆字符串) title detail creator2user（FK to用户表,提交故障的人） processor2user（FK to用户表 可以为null，处理故障的人） status(待处理，处理中，已处理) 创建时间 处理时间

分类表：caption(标题) Fk (博客bid)

标签表：caption Fk(博客bid)

文章：id,title,summary,ctime(创建时间),FK(个人分类表，一对多),主站分类（choices）

文章详细：detail OneToOne(文章)

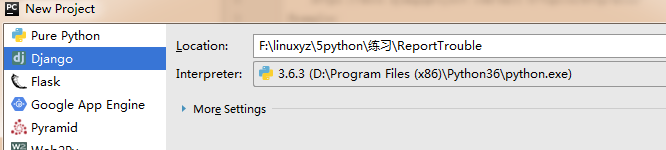
文章标签关系： 文章ID 标签ID （文章和标签的多对多的第三张表）

赞踩文章关系： 文章ID 用户ID 赞或踩（True，False） 联合唯一索引：（文章ID 用户ID 。如果赞了就不能踩了，如果踩了就不能赞了）

评论表：id,content,FK(文章),FK(user),ctime,parent\_comment\_id（上一次评论的id）

# 程序目录结构

创建django工程“保障系统”



创建APP web，做主站的主页管理和个人的主站管理

F:\linuxyz\5python\练习\ReportTrouble>python manage.py startapp web

创建APP backend，做后台管理

F:\linuxyz\5python\练习\ReportTrouble>python manage.py startapp backend

创建APP repository，只操作数据库，不做业务请求。

F:\linuxyz\5python\练习\ReportTrouble>python manage.py startapp repository

web和backend只做业务请求，不存放数据库，可以把他们俩的models.py删掉

repository，只做数据库，不做业务请求，可以把respository里的views.py删掉。

创建目录utils，存放公共功能模块

保障系统

- backend # 后台管理

-repository # 数据仓库（操作数据Model）

-static #静态文件目录

-css #存放自己写的css文件

-extra #存放第三方的css/js，例如bootstrap，jquery, jquery-cookie等

-js #存放自己写的js文件

-templates #存放html文件

- utils # 工具包（公共模块）

- web # 首页，个人博客

-报障系统 #项目目录

db.sqlite3 #数据库

manage.py #管理程序

# 工作安排：

1. 需求分析

2. 数据库设计思路

3. 实现数据库设计（Admin添加数据）

4. 主站：分类(主站)查看+分页

- 标题菜单：母版

- 登录成功：把用户名写到 session['username'] = 'root'

- 主页html：

判断是否有用户：显示用户名

否则 ： 登录，注册

<div class="pg-header">

{% if request.session.username %}

<a>{{ request.session.username }}</a>

{% else %}

<a>登录</a><a>注册</a>

{% endif %}

</div>

- 博文内容布局：

div div -> float => 图片下方空白

<a></a>asdflkjasdfkj => 文字环绕

- URL分类筛选：

url(r'^all/(?P<article\_type\_id>\d+).html$', home.index, name='index'),

url(r'^', home.index),

index根据kwargs判断是否分类查询？

a. 生成分类菜单

b. 考虑是否选中

c. 根据条件model.xxx.objects.filter(\*\*kwargs)

- URL:

from django.urls import reverse

url(r'^all/(?<article\_type\_id>\d+).html$', home.index, name='index'),

在HTML中：{% url "index" article\_type\_id=1 %} => all/1.html

在views中：reverse('index',kwargs={"article\_type\_id":1}) =>all/1.html

url(r'^all/(\d+).html$', home.index, name='index'),

在HTML中：{% url "index" 1 %} =>all/1.html

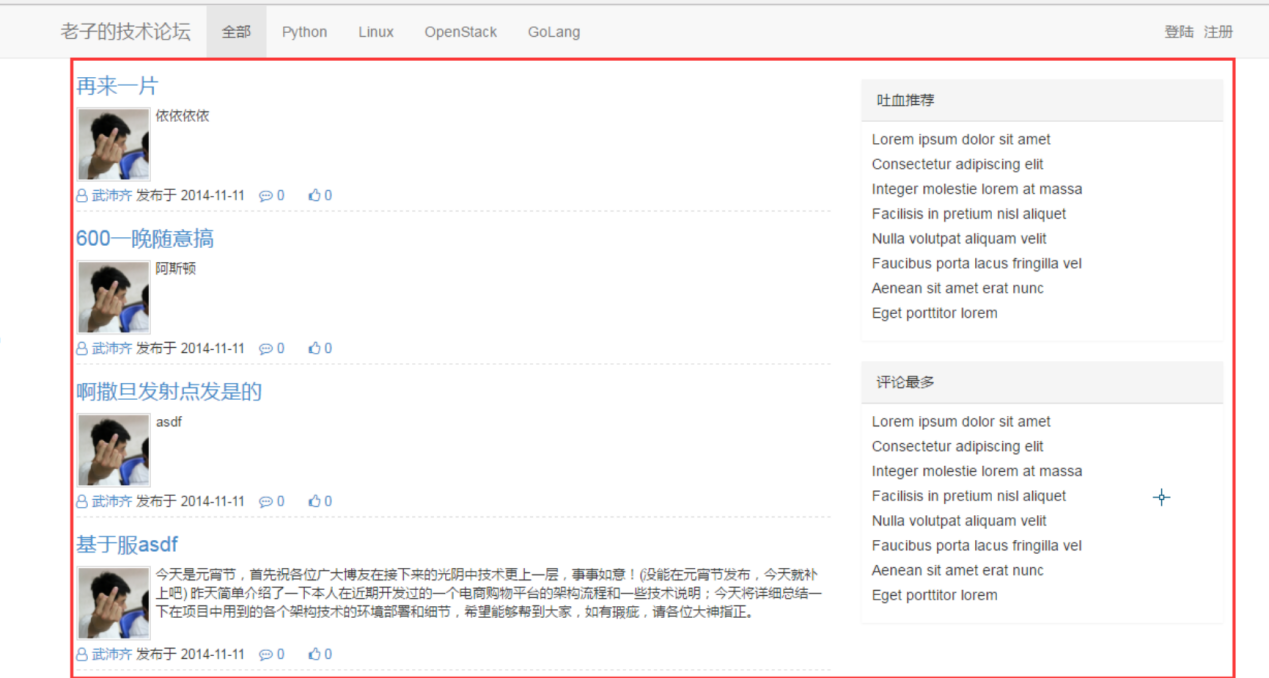
在views中：reverse('index',args=(1,)) =>all/1.html

- 利用reverse（获取当前url）+分页组件完成：分类查看+分页

5. 登录，注册

- 密码：数字，字母，特殊字符，不用ajax，用form表单

- 密码两次输入一致





自定义密码验证规则

注册时确认2次输入的密码是否一样，在def clean（）方法中验证 上面代码是在python中先把整体错误提示从errors中取了出来，再单独传到html里。

如果不取出来，html里变量不能是\_\_all\_\_，但可以写成 {{obj.non\_field\_errors}}表示整体错误信息

看看参考里注册和登录用的是不是一个form表单

个人博客：

个人主页：http://127.0.0.1:8000/wupeiqi.html

文章最终：http://127.0.0.1:8000/wupeiqi/5.html

个人筛选：

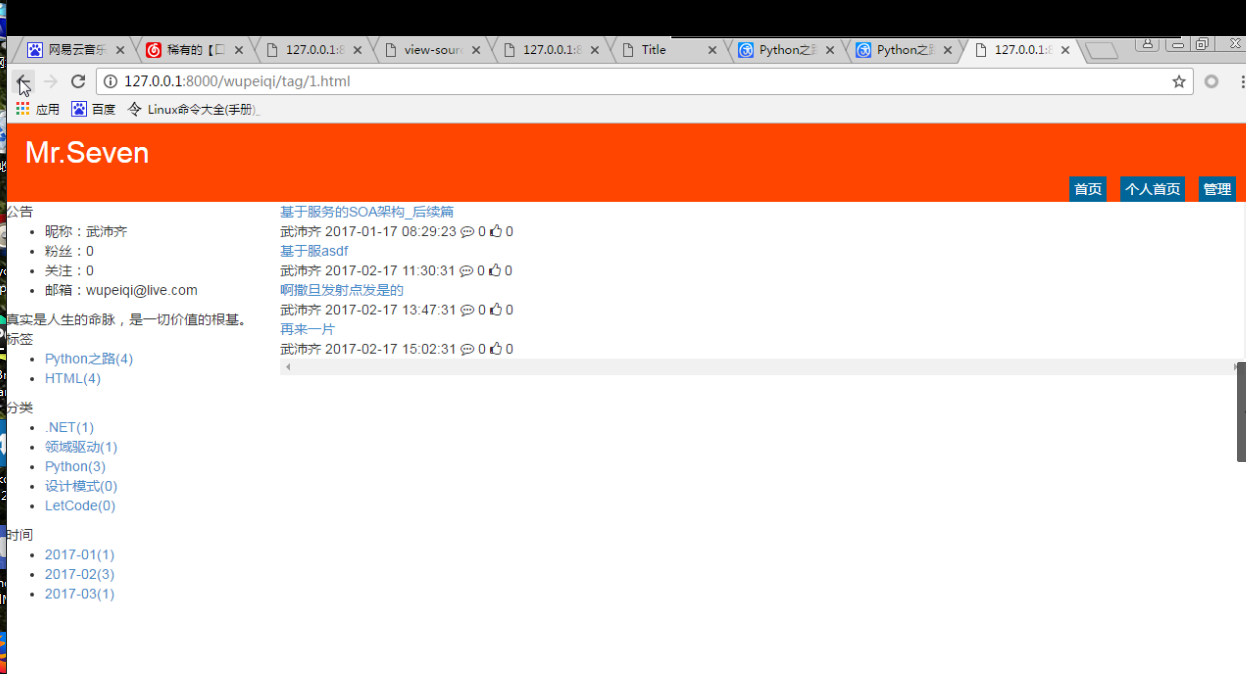
http://127.0.0.1:8000/wupeiqi/tag/2.html

http://127.0.0.1:8000/wupeiqi/category/3.html

http://127.0.0.1:8000/wupeiqi/date/2017-02.html

注意：个人主题定制

点击‘头像’，进入个人博客

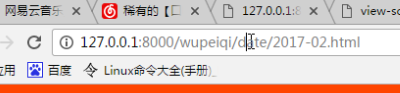


筛选：

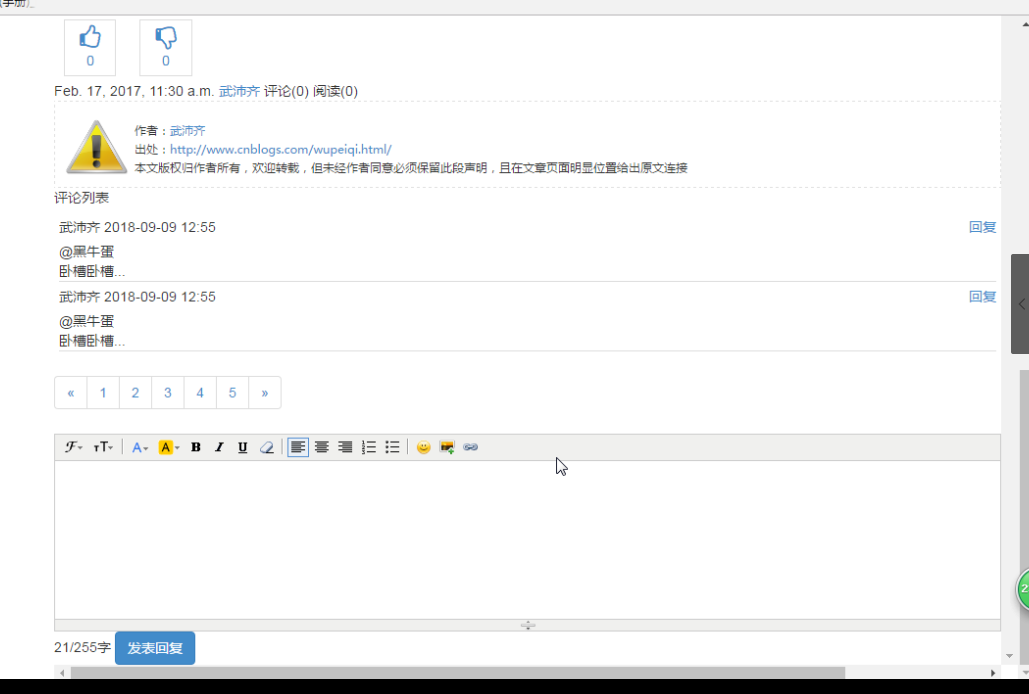
url：用户名/标签id 标签分类



根据时间分类



文章详细



# 路由系统

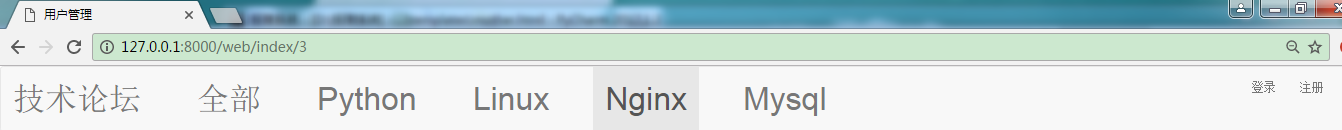
采用多级路由方式

一级路由：’保障系统’ ->urls.py

二级路由：APP -> urls.py

用bootstrap实现页面装饰

# 母板 mother.html



## 实现功能

1. 表头用母板实现
2. 主页内容可分类查看，可显示‘全部’/‘Python’/‘Linux’/‘Nginx’/Mysql/
3. 用户选中的类型背景颜色要加深显示。
4. 判断用户是否已登录，如果登录显示用户名，否则显示 ’登录 注册’

## 表头的文章类型

1、文章类型Python/Linux/Nginx...，点Linux，url中是2； 点Nginx,url中是3，

2、考虑当前标签是否被选中。过程：用户点Nginx标签，把article\_id=3传到后台，后台筛选出数据后，再把article\_id=3传给html，html中循环显示类型时判断，如果article\_id=3，那么加深显示。

3、如果点了’Linux’，把这个id(2)传到后台，后台去数据库中查出home\_choice\_id=1的数据，显示出来。如果点‘全部’，id是0

例如：用户点了Linux标签，url为http://127.0.0.1:8000/web/index/2，末尾的数字是Article表中home\_choice中的id，传给urls.py的路由中：

urls.py中：

urlpatterns = [

url(r'index$',views.index,name='index'), #要加上$，以dex结尾，否则会和下面的有正则的url冲突

url(r'index/(?P<article\_id>\d+)',views.index,name='index'), #2传给article\_id，传到views.py中

views.py中:

def index(request,\*args,\*\*kwargs):

if request.method == 'GET':

article\_id = int(kwargs.get('article\_id')) #article\_id是用户点的主站的文章类型标签的id，article\_id转成整数，方便html中和item.0比较

...

return render(request, 'index.html',{'article\_home\_choice':article\_home\_choice,'article\_id':article\_id,})#把article\_id传给html，告诉html用户点的哪个标签，把这个标签class=active

mother.html中：

{% for item in article\_home\_choice%}

{% if article\_id == item.0 %} <!--如果用户点了这个标签-->

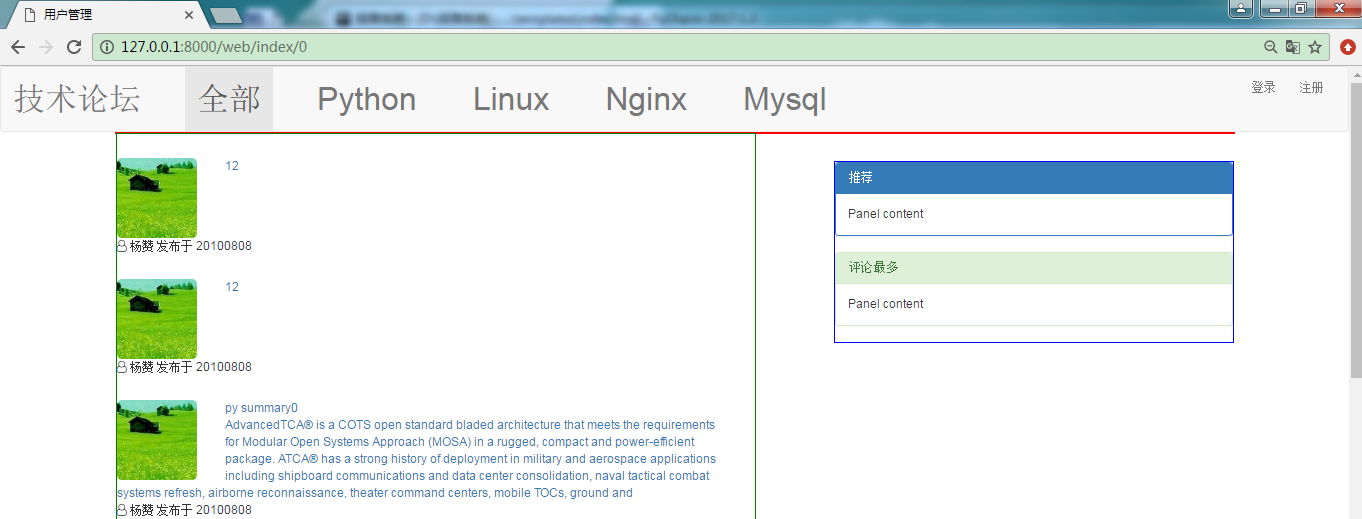
<li class="active"><a href="{% url "web:index" item.0 %}" class="tab">{{ item.1 }}</a></li> <!--给这个标签设为active，url中用item.0来区分不同的文章类型-->

{% else %}

<li><a href="{% url "web:index" item.0 %}" class="tab">{{ item.1 }}</a></li> <!--url中用item.0来区分不同的文章类型-->

{% endif %}

# 主页 index.html





## 实现功能

1. 主页的内容可分页显示，

2、可选择每页显示多少行内容，

3、可点击’上一页’/’下一页’

4、可跳转到第几页

5、可直接跳转到’首页’或’末页’

5、页面左侧显示文章内容，右侧显示推荐内容

# 登录页面 login

用户名或密码输入错误要有错误提示。

通过session技术进行登录验证。登录成功后，把登录状态写入session中。

用装饰器给所有需要认证登录的页面增加认证功能

10天内免登陆功能(从第一次正确登录开始计算时间）

注销登录功能

用django的form组件进行用户输入内容的格式验证（输入长度）。

格式验证通过后再去数据库验证内容是否正确

html中跳转url用namespace和name方式。<a href="{% url 'web:register' %}">注册</a>

# 注册页面 register

## 注册页面相关文件

templates - register.html、

web- forms- registerForm.py ：定义Form表单类，定义验证规则

web- views里的register方法

## 实现功能

1、用django的form组件进行用户输入内容的格式验证

-所有字段内容不能为空

-用自定义def clean\_username方法对username字段进行验证： 用户名是否已经被注册，如果用户名已经被注册，进行错误提示，不能用相同的用户名注册

-用django Form组件自带的邮箱字段EmailField进行验证： 邮箱格式是否正确，如果错误要有错误提示。

-用自定义def clean()方法对2次输入的密码进行联合验证：用户2次输入密码如果不同，要有错误提示

- 用pwd字段的参数validators，自定义正则表达式验证：密码必须是数字+字母组合，如果不是，要有错误提示

2、单独的数据库表存放用户名密码

3、注册成功后跳转到登录页面

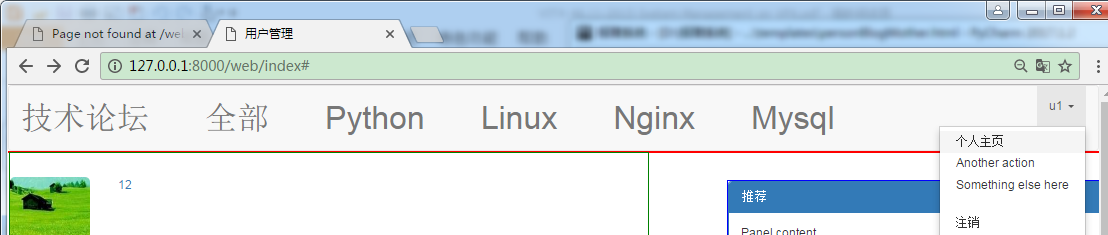
# 个人主页blog

## 从网站主页跳转到个人blog主页思路

html数据 -> urls.py正则匹配 -> 后端视图函数里 ->处理后传给html，在html中显示

通过正则匹配url里的数据，进行数据传递。

1、在主站页面，点击“个人主页”



跳转的href=中，带上这个“个人”的姓名（从session中取出用户名）

<li><a href="{% url "web:userBlog" blogName=request.session.username %}">个人主页</a></li>

跳转的url是；http://127.0.0.1:8000/web/userBlog/u1/

2、在urls.py中，用正则匹配出“个人”的姓名，传到后台视图函数

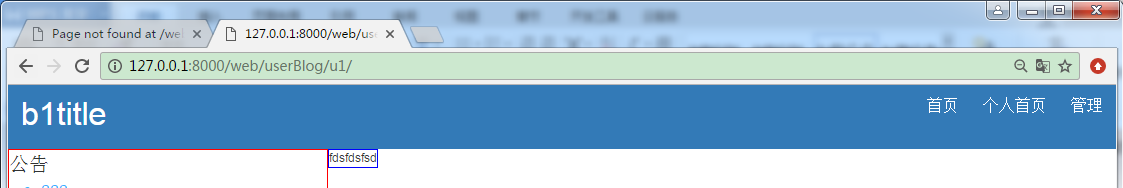
urlpatterns = [  
 url(**r'userBlog/(?P<blogUser>\w+)/'**,personalBlogApp.userBlog,name=**'userBlog'**)  
]

1. 在视图函数中，通过kwargs取出“个人”的姓名，在数据库中找出这个user的数据，找出他的blog内容，再render传到个人主页的html里。

**def** userBlog(request,\*args,\*\*kwargs):  
 blogUser = kwargs.get(**"blogUser"**)  
 obj\_blog = models.UserInfo.objects.filter(username=blogUser).first().userinfo2blog#取出这个人的blog  
 print(**"bloguser is"**,blogUser)  
 print(obj\_blog)  
 **return** render(request,**"personBlogMother.html"**,{**"obj\_blog"**:obj\_blog,})#传到html里

1. 在个人的blog主页里显示这个人blog的内容

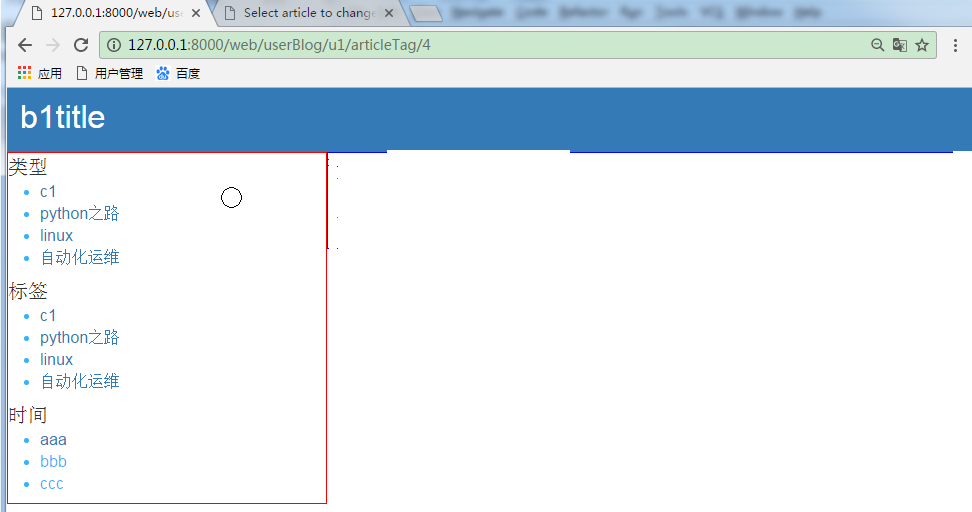
<**div class="navbar-header"**>  
 <**a class="navbar-brand tab blogName" href="#"**>{{ **obj\_blog**.**title** }}</**a**> <!--显示blog的title-->  
</**div**>



## 个人主页模板

模板写表头和左侧选择栏

personBlogMother.html



## 左侧筛选思路

访问博客的url：

<http://127.0.0.1:8000/web/userBlog/u1/>

根据类型筛选的url：articleType表示是根据文章类型筛选，2表示articleType的nid

<http://127.0.0.1:8000/web/userBlog/u1/articleType/2>

根据文章标签筛选的url：articleTag表示是根据文章类型筛选，2表示articleType的nid

<http://127.0.0.1:8000/web/userBlog/u1/articleTag/2>

personBlogMother.html里，拼凑出url来：

<div class="title1">标签</div>

<ul>

{% for item in obj\_blog.blog2articletype.all %}

<li class="title2">

<a href="{% url "web:userBlog" blogUser=request.session.username choiceTitle="articleTag" choiceSubTitle=item.nid %}">

{{ item.caption }}

</a>

</li>

{% endfor %}

</ul>

传给urls.py,进行正则匹配：

urlpatterns = [

url(r'^index$',views.index,name='index'), #要加上$，以dex结尾，否则会和下面的有正则的url冲突

url(r'index/(?P<article\_id>\d+)',views.index,name='index'),

url(r'login',login.login,name='login'), #执行web/viewApps/login.py中的login函数

url(r'register',views.register,name='register'),

url(r'logout',views.logout,name='logout'),

#要有多个匹配，分别来匹配不同长度的url，带一个参数/带2个参数/带3个参数的url(r'userBlog/(?P<blogUser>\w+)/$',personalBlogApp.articleList,name='userBlog'),

url(r'userBlog/(?P<blogUser>\w+)/(?P<choiceTitle>\w+)$',personalBlogApp.articleList,name='userBlog'),

url(r'userBlog/(?P<blogUser>\w+)/(?P<choiceTitle>\w+)/(?P<choiceSubTitle>\d+)$',personalBlogApp.articleList,name='userBlog'),

]

在views.py中，根据不同的参数从数据库筛选不同的数据，再返给html

if userChoice == "articleType":

article\_show\_obj = obj\_blog.blog2article.filter(article2articletype\_id=choiceSubTitle) #按类型 取出要在个人博客主页展示的文章的对象

elif userChoice == "articleTag":

article\_show\_obj = obj\_blog.blog2article.filter(article2articletag\_\_nid=choiceSubTitle) #按标签 取出要在个人博客主页展示的文章的对象

else:

article\_show\_obj = obj\_blog.blog2article.all() #如果没有筛选，取出所有的文章

return render(request, "personBlog/articleList.html", {"obj\_blog":obj\_blog, "article\_show\_obj":article\_show\_obj})