This post was republished to 汪汪家园 at 0:14:21 2017/2/23

Zakkabag

[微信登陆](#_注册登录) | [用户扩展](#_用户扩展) | [图片上传](#_图片上传) | [评论系统](#_评论) | 微信支付 | 积分支付 | [社会分享](#_Social_Share)

# 微信公众号及微网站

## 注册登录

### 普通登录

用户通过手机中微信登录网站

这个指的是当前正在浏览该网站，登陆入口的包含微信选项，可以通过微信选项登陆

**MyUser**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** |  | **Description** |
| username | CharField | unique | 用户名 [如果电话注册= phone] |
| phone | CharField |  | 电话 |
| first\_name | CharField |  | 名 |
| last\_name | CharField |  | 姓 |
| email | EmailField |  | 邮箱 |
| sex | CharField |  | 性别 |
| birthday | DateField |  | 生日 |
| nickname | CharField |  | 昵称 |
| image | ImageField |  | 头像 |
| is\_staff | BooleanField |  | 职员状态  指明用户是否可以登录到这个管理站点。  Boolean. Designates whether this user can access the admin site. |
| Is\_super | BooleanField |  | 超级用户状态  指明该用户缺省拥有所有权限。  Boolean. Designates whether this user account should be considered active.  Set this flag to False instead of deleting accounts. |
| is\_active | BooleanField |  | 有效  指明用户是否被认为活跃的。以反选代替删除帐号。  Boolean. Designates that this user has all permissions without explicitly assigning them. |
| date\_joined | DateTimeField |  | 加入时间 |
| account\_type | CharField | [mail, phone] | 注册类型 |
|  |  |  |  |

**WechatUserProfile**

|  |  |  |  |
| --- | --- | --- | --- |
| openid | CharField | unique |  |
| unionid | CharField |  |  |
| privilege | CharField |  | 权限 |
| headimgurl | CharField |  | 头像 |
| nickname | CharField |  | 昵称 |
| sex | CharField |  | 性别 |
| city | CharField |  | 城市 |
| country | CharField |  | 国家 |
| language | CharField |  | 语言 |
| user | MyUser |  |  |



**配置**

首先登录微信测试号，修改授权

|  |  |
| --- | --- |
| 网页帐号 | [网页授权获取用户基本信息](http://mp.weixin.qq.com/wiki/17/c0f37d5704f0b64713d5d2c37b468d75.html) |

127.0.0.1:8000 本地

zakkabag.applinzi.com 新浪云服务器

确保” 微信web开发者工具”已安装，这个对微信调试非常有帮助

#### 登录入口

首先提供微信登录入口，需要

1. 在导航条上添加微信登录图标，具体图标可以从bootstrap上得到
2. 在URL添加入口

navbar.html

<li><a href=**'{% url "wechatlogin" %}'**><i class=**"fa fa-weixin"** aria-hidden=**"true"**></i></a></li>

zakkabag.urls.py

urlpatterns **=** patterns**(**''**,**

url**(**r'^wechatlogin/$'**,** 'wechat.views.wechatlogin'**,** name**=**'wechatlogin'**),**

url**(**r'^logout/$'**,** 'newsletter.views.logout'**,** name**=**'logout'**),**

url**(**r'^login/$'**,** 'newsletter.views.login'**,** name**=**'login'**),**

#### 微信授权

接下来将在wechatlogin里面实现授权功能

关于微信登录的信息可参考：[**微信公众平台开发者文档**](https://mp.weixin.qq.com/wiki)

具体的实现我们可参考一个python library : [python-weixin](https://pypi.python.org/pypi/python-weixin)

登录成功后，它会重定向到login URL

**from** weixin**.**client **import** WeixinMpAPI

**def** wechatlogin**(**request**):**

REDIRECT\_URI **=** request**.**build\_absolute\_uri**(**'/'**).**strip**(**"/"**)** **+** reverse**(**"login"**,** kwargs**={})**

api **=** WeixinMpAPI**(**appid**=**APP\_ID**,** app\_secret**=**APP\_SECRET**,**redirect\_uri**=**REDIRECT\_URI**)**

redirect\_uri **=** api**.**get\_authorize\_login\_url**(**scope**=(**"snsapi\_userinfo"**,))**

**return** redirect**(**redirect\_uri**)**

#### 继续登录

获得weixin授权之后，接下来再里newsletter.views.login实现登录，中间会用到两个django函数authenticate和auth.login。

如果用户授权失败，则重定向到登录界面

**def** login**(**request**):**

REDIRECT\_URI **=** request**.**POST**.**get**(**'next'**,** request**.**GET**.**get**(**'next'**,** reverse**(**"home"**,** kwargs**={})))** #template指定的重定向地址

**if** request**.**method **==** 'GET'**:**

code **=** request**.**GET**.**get**(**'code'**)**

**if** code**:**

redirect\_to **=** "http://%s%s" **%** **(**request**.**META**[**'HTTP\_HOST'**],** reverse**(**"home"**,** kwargs**={}))**#登录成功后的登录地址

api **=** WeixinMpAPI**(**appid**=**APP\_ID**,**

app\_secret**=**APP\_SECRET**,**

redirect\_uri**=**redirect\_to**)**

auth\_info **=** api**.**exchange\_code\_for\_access\_token**(**code**=**code**)**

api **=** WeixinMpAPI**(**access\_token**=**auth\_info**[**'access\_token'**])**

api\_user **=** api**.**user**(**openid**=**auth\_info**[**'openid'**])**

user **=** authenticate**(**request**=**request**,** user**=**api\_user**)**

**if** user**:** #如果用户授权失败，则重定向到登录界面

auth**.**login**(**request**,** user**)**

**return** redirect**(**redirect\_to**)**

**return** redirect**(**reverse**(**"auth\_login"**,** kwargs**={}))**

**return** auth\_views**.**login**(**request**,** redirect\_field\_name**=**REDIRECT\_URI**,** extra\_context**=None)**

特别注意的是，这边登录用了authenticate函数，参数第二个变成了user，这个其实已经对登录的后台进行了扩展，具体可参考章节“**用户扩展**”

登录成功后需要在templates.navbar.html修改图标为已登录状态

{% if request.user.is\_authenticated %}

{% if request.user.account\_type == 'wechat' %}

<li><a href='{{request.user.get\_absolute\_url}}'}>{{request.user.get\_wechatprofile}}</a></li>

<li><a href="{% url 'logout' %}">Logout</a></li>

{% else %}

<li><a href="{{request.user.get\_absolute\_url}}">{{request.user}}</a></li>

<li><a href="{% url 'logout' %}">Logout</a></li>

{% endif %}

{% elif request.wechat %}

<li><a href="{{request.wechat.get\_absolute\_url}}">{{request.wechat}}</a></li>

<li><a href="{% url 'logout' %}">Logout</a></li>

{% else %}

<li><a href="{% url 'registration\_register' %}">Register</a></li>

<li><a href='{% url "wechatlogin" %}'><i class="fa fa-weixin" aria-hidden="true"></i></a></li>

{% endif %}

#### 退出登录

在newsletter.views.logout实现注销，会调用系统函数auth.logout。

**def** logout**(**request**):**

**try:**

**del** request**.**session**[**'wechat\_id'**]**

**except:**

**pass**

auth**.**logout**(**request**)**

**return** redirect**(**reverse**(**"home"**,** kwargs**={}))**

* [网站微信登录－python 实现](http://www.cnblogs.com/cacique/p/4293348.html)
* <https://open.weixin.qq.com/cgi-bin/showdocument?action=dir_list&t=resource/res_list&verify=1&lang=zh_CN&tab=dev>
* <https://github.com/gusibi/python-weixin>
* <http://gusibi.github.io/post/weixin-python-login/>

### 第三方登录

可以通过微信直接登录微网站，减少登录流程

访问网站时，因为是从微信客户端进入的，所以自动登陆

### 用户注册

用户通过手机注册成为会员，推荐使用手机号注册保证用户的真实性和唯一性，收集准确的会员资料，为后期营销增加有效的用户量

<https://leancloud.cn/>

手机注册/登陆参考

[django-phone-login](http://pypi.python.org/pypi/django-phone-login)

[django\_mobile\_auth](https://pypi.python.org/pypi/django_mobile_auth)

参考[django-registration-redux](https://github.com/macropin/django-registration)实现

login, logout 直接调用django默认的函数，指定template文件

password的修改和完成提示，同样调用django默认的函数，template沿用默认文件名（未特别指定）

password的复位，也是通过django默认的函数，所有这些template都在root/template/registration目录下面

文件registration**/**auth\_urls**.**py

**from** django**.**contrib**.**auth **import** views **as** auth\_views

urlpatterns **=** **[**

url**(**r'^login/$'**,**

auth\_views**.**login**,**

**{**'template\_name'**:** 'registration/login.html'**},**

name**=**'auth\_login'**),**

url**(**r'^logout/$'**,**

auth\_views**.**logout**,**

**{**'template\_name'**:** 'registration/logout.html'**},**

name**=**'auth\_logout'**),**

url**(**r'^password/change/$'**,**

auth\_views**.**password\_change**,**

**{**'post\_change\_redirect'**:** reverse\_lazy**(**'auth\_password\_change\_done'**)},**

name**=**'auth\_password\_change'**),**

url**(**r'^password/change/done/$'**,**

auth\_views**.**password\_change\_done**,**

name**=**'auth\_password\_change\_done'**),**

url**(**r'^password/reset/$'**,**

auth\_views**.**password\_reset**,**

**{**'post\_reset\_redirect'**:** reverse\_lazy**(**'auth\_password\_reset\_done'**)},**

name**=**'auth\_password\_reset'**),**

url**(**r'^password/reset/complete/$'**,**

auth\_views**.**password\_reset\_complete**,**

name**=**'auth\_password\_reset\_complete'**),**

url**(**r'^password/reset/done/$'**,**

auth\_views**.**password\_reset\_done**,**

name**=**'auth\_password\_reset\_done'**),**

**]**

**if** **(**LooseVersion**(**get\_version**())** **>=** LooseVersion**(**'1.6'**)):**

urlpatterns **+=** **[**

url**(**r'^password/reset/confirm/(?P<uidb64>[0-9A-Za-z\_\-]+)/(?P<token>.+)/$'**,**

auth\_views**.**password\_reset\_confirm**,**

**{**'post\_reset\_redirect'**:** reverse\_lazy**(**'auth\_password\_reset\_complete'**)},**

name**=**'auth\_password\_reset\_confirm'**)**

**]**

**else:**

urlpatterns **+=** **[**

url**(**r'^password/reset/confirm/(?P<uidb36>[0-9A-Za-z]{1,13})-(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$'**,**

auth\_views**.**password\_reset\_confirm**,**

**{**'post\_reset\_redirect'**:** reverse\_lazy**(**'auth\_password\_reset\_complete'**)},**

name**=**'auth\_password\_reset\_confirm'**)**

**]**

用户的注册和激活是通过自定义的函数实现的

urlpatterns **=** **[**

url**(**r'^activate/complete/$'**,**

TemplateView**.**as\_view**(**template\_name**=**'registration/activation\_complete.html'**),**

name**=**'registration\_activation\_complete'**),**

# Activation keys get matched by \w+ instead of the more specific

# [a-fA-F0-9]{40} because a bad activation key should still get to the view;

# that way it can return a sensible "invalid key" message instead of a

# confusing 404.

url**(**r'^activate/(?P<activation\_key>\w+)/$'**,**

ActivationView**.**as\_view**(),**

name**=**'registration\_activate'**),**

url**(**r'^register/complete/$'**,**

TemplateView**.**as\_view**(**template\_name**=**'registration/registration\_complete.html'**),**

name**=**'registration\_complete'**),**

url**(**r'^register/closed/$'**,**

TemplateView**.**as\_view**(**template\_name**=**'registration/registration\_closed.html'**),**

name**=**'registration\_disallowed'**),**

**]**

**if** getattr**(**settings**,** 'INCLUDE\_REGISTER\_URL'**,** **True):**

urlpatterns **+=** **[**

url**(**r'^register/$'**,**

RegistrationView**.**as\_view**(),**

name**=**'registration\_register'**),**

**]**

**if** getattr**(**settings**,** 'INCLUDE\_AUTH\_URLS'**,** **True):**

urlpatterns **+=** **[**

url**(**r''**,** include**(**'registration.auth\_urls'**)),**

**]**

Registration-redux仅仅实现了邮箱注册功能，要实现手机注册功能，需要做如下修改

1. 扩展用户，添加phone相关字段
2. 33

### 忘记密码

可通过忘记密码模块可找回帐号，保证用户信息和积分不会丢失，增加用户体验

## 个人中心

### 个人信息展示

展示头像用户名等级及其它用户信息

创建APP : personalcenter

### 上传头像

通过网站可以修改头像

#### 后台上传

细节可参考ecommerce 017 Image Uploads

<https://docs.djangoproject.com/en/1.8/ref/forms/fields/#django.forms.ImageField>

在用户里面添加image域，并通过upload\_to函数指定保存方法

**def** image\_upload\_to**(**instance**,** filename**):**

name **=** instance**.**username

title**,** file\_extension **=** filename**.**split**(**"."**)**

new\_filename **=** "%s-%s.%s" **%(**instance**.**id**,** slugify**(**title**),** file\_extension**)**

**return** "profile/%s/%s" **%(**name**,** new\_filename**)**

#Copy from AbstractUser

**class** **MyUser(**AbstractBaseUser**,** PermissionsMixin**):**

image **=** models**.**ImageField**(**upload\_to**=**image\_upload\_to**)**

MEDIA\_URL **=** '/media/'

MEDIA\_ROOT **=** os**.**path**.**join**(**BASE\_DIR**,** "static\_in\_env"**,** "media\_root"**)**

Admin后台显示：**目前:**[profile/bhe001/1-6-lock.jpg](http://127.0.0.1:8000/media/profile/bhe001/1-6-lock.jpg)

访问网站：<http://127.0.0.1:8000/media/profile/bhe001/1-6-lock.jpg>

本地文件保存位置：d:\virtualenv\zakkabag\zakkabag\static\_in\_env\media\_root\profile\bhe001\1-6-lock.jpg

同样的名字，如果继续覆盖，文件不会被覆盖，而是增加随机数重新拷贝一个

**目前:**[profile/bhe001/1-6-lock\_Ni9llB7.jpg](http://127.0.0.1:8000/media/profile/bhe001/1-6-lock_Ni9llB7.jpg)

#### 前台上传

首先在personalcenter.templates.personcalcenter.profile\_detail.html添加img相关元素

{{form}}会显示img组件，但这个不一定是我想要的格式。可能需要修改。

<img id='img' class= 'img-responsive' src="{{object.get\_image\_url}}"/>会把图片显示出来

在表单中加入[enctype](http://www.w3school.com.cn/tags/att_form_enctype.asp)="multipart/form-data；如果不加enctype="multipart/form-data：得不到图片信息，只有路径

FILES是个字典，它包含每个FileField的键 (或者ImageField，FileField的子类)。这样的话就可以用request.FILES['File']来存放表单中的这些数据了。

注意request.FILES只有在请求方法为POST，并且发送请求的<form>拥有enctype="multipart/form-data属性时，才会包含数据。否则request.FILES为空

{% extends "base.html" %}

{% load crispy\_forms\_tags %}

{% block content %}

<div class='col-sm-6 col-sm-offset-3'>

<h3>{{ object }}</h3>

<form method="POST" enctype="multipart/form-data" action=""> {% csrf\_token %}

{{form|crispy}}

<img id='img' class= 'img-responsive' src="{{object.get\_image\_url}}"/>

<br>

<input type="submit" value='Update' class='btn' />

</form>

</div>

{% endblock %}

在personalcenter.views.py 中添加对form的处理

class ProfileDetailView(FormMixin, DetailView):

model = UserModel

template\_name = "personalcenter/profile\_detail.html"

form\_class = MyUserForm #➊➋➌➍➎➏➐➑➒➓

def get\_context\_data(self, \*args, \*\*kwargs):

context = super(ProfileDetailView, self).get\_context\_data(\*args, \*\*kwargs)

context["form"] = self.form\_class(instance = self.get\_object()) #➋

return context

def get\_object(self, \*args, \*\*kwargs):

user\_pk = self.kwargs.get("id")

usemodel = None

if user\_pk:

usemodel = get\_object\_or\_404(UserModel, pk=user\_pk)

return usemodel

def get\_success\_url(self):

return reverse("personalcenter", kwargs=self.kwargs)

def post(self, request, \*args, \*\*kwargs):

form = self.get\_form()#➌

if form.is\_valid():

usermodel = UserModel.objects.get(id=self.kwargs.get("id"))

usermodel.first\_name = form.cleaned\_data['first\_name']

usermodel.last\_name = form.cleaned\_data['last\_name']

if 'image' in form.cleaned\_data:

usermodel.image = form.cleaned\_data['image'] #➍

usermodel.save()

return self.form\_valid(form)

else:

return self.form\_invalid(form)

➊ Form class

*form\_class = MyUserForm*

定义form\_class，后面通过通过get\_form\_class()或get\_form()获取form相关的内容

➋ *context["form"] = self.form\_class(instance = self.get\_object())*

Profile是DetailView，本来是不带form的，如果要支持image功能，必须支持form，所以我们添加了基类FormMixin。在form\_class指定form类型，并且在context里面将form数据传递给templates。这儿根据object内容对form进行了初始化。

如果是CreateView (例如ProductCreateView)，它的基类有FormMixin，所以form\_class是默认已定义的，不需要额外添加FormMixin。不过，如果要添加额外的form，同样需要在context里面传递

更新form内容除了通过instance，也可以直接更新queryset，例如form.fields['categories'].queryset = Category.objects.all()

➌ *form = self.get\_form()*

get\_form()会根据定义的form\_class获取form类，并根据传递的参数初始化或实例化

在该例子中，我们可以用下面句子代替图片信息获取内容form = self.form\_class(request.POST, request.FILES)

具体可参考class FormMixin(six.with\_metaclass(FormMixinBase, ContextMixin)):

**class** **FormMixin(**six**.**with\_metaclass**(**FormMixinBase**,** ContextMixin**)):**

**def** get\_form\_class**(**self**):**

"""

Returns the form class to use in this view

"""

**return** self**.**form\_class

**def** get\_form**(**self**,** form\_class**=None):**

"""

Returns an instance of the form to be used in this view.

"""

**if** form\_class **is** **None:**

form\_class **=** self**.**get\_form\_class**()**

**return** form\_class**(\*\***self**.**get\_form\_kwargs**())**

**def** get\_form\_kwargs**(**self**):**

"""

Returns the keyword arguments for instantiating the form.

"""

kwargs **=** **{**

'initial'**:** self**.**get\_initial**(),**

'prefix'**:** self**.**get\_prefix**(),**

**}**

**if** self**.**request**.**method **in** **(**'POST'**,** 'PUT'**):**

kwargs**.**update**({**

'data'**:** self**.**request**.**POST**,**

'files'**:** self**.**request**.**FILES**,**

**})**

**return** kwargs

➍ *usermodel.image = form.cleaned\_data['image']*

该语句获取image对象。也可以通过下面文件处理的方式

filename=request.FILES['image']

from PIL import Image

if filename:

img=Image.open(filename)

img.save(os.path.join(settings.MEDIA\_ROOT, photoname))

usermodel.image = photoname

效果图如下：



#### 上传进度条

Tag: cache

具体参考：<https://github.com/Tonetete/Simple-Django-progressbar-upload-file-form-with-ajax>

该文章有相对详细的讲解，但是内容不全[Upload to Django with progress bar using Ajax and jQuery](http://www.laurentluce.com/posts/upload-to-django-with-progress-bar-using-ajax-and-jquery/)

其中用到了

[jQuery Form plugin](http://jquery.malsup.com/form/#download)

ajaxsubmit

<http://malsup.github.io/jquery.form.js>

[jQuery progress bar plugin](http://t.wits.sg/jquery-progress-bar/) 这个当前实现没有，后面可以用这个来代替进度条状态更新的函数

progressbar

<https://code.jquery.com/ui/1.12.1/jquery-ui.js>

<https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css>

文件上传参考函数

<https://docs.djangoproject.com/en/1.8/topics/http/file-uploads/>

首先在personalcenter/forms.py里面新增加一个图片上传的form

+class UploadFileForm(forms.Form):

+ image = forms.ImageField(widget=forms.FileInput(

+ attrs={'required': 'required'})) # required=True is the default, but not show it validation in template

设置required，这样如果不选图片的话，上传会报错

动态上传会用到两个ajax post地址，在personalcenter/urls.py增加两个URL，分别处理文件上传和状态更新功能

urlpatterns **=** **[**

**+** url**(**r'^uploadfile'**,** upload\_file**,** name**=**'upload\_file'**),**

**+** url**(**r'^upload\_status$'**,** upload\_status**,** name**=**'upload\_status'**),**

对应的template如下 profile\_detail.html

{% block content %}

<div class='col-sm-6 col-sm-offset-3'>

<h3>{{ object }}</h3>

<form id="form\_profile" method="POST" enctype="multipart/form-data" action=""> {% csrf\_token %}

{{form|crispy}}

<img id='img' class= 'img-responsive' src="{{object.get\_image\_url}}"/>

<input type="submit" value='Update' class='btn' />

</form>

<br>

<form id="form\_upload" enctype="multipart/form-data" action="/personalcenter/uploadfile " method="POST"> {% csrf\_token %}

{{upload\_form|crispy}}

<div class="progress">

<div class="progress-bar progress-bar-success progress-bar-striped active" role="progressbar" aria-valuenow="0" aria-valuemin="0" aria-valuemax="100" style="width: 0%">

0%

</div>

</div>

<div class="processing-file"></div>

<input id="buttonSubmit" class="btn btn-primary" type="submit" value="Upload" />

</form>

</div>

{% endblock %}

该template里面包含两个form，第二个form专门用于上传图片

**var** options **=** **{**

beforeSubmit**:** showRequest**,** // pre-submit callback

success**:** showResponse**,** // post-submit callback

error**:** showError**,**

// other available options:

url**:** "/personalcenter/uploadfile"**,** // override for form's 'action' attribute

type**:** "post"**,** // 'get' or 'post', override for form's 'method' attribute

dataType**:** "json" // 'xml', 'script', or 'json' (expected server response type)

//clearForm: true // clear all form fields after successful submit

//resetForm: true // reset the form after successful submit

// $.ajax options can be used here too, for example:

//timeout: 3000

**};**

// bind to the form's submit event

$('#form\_upload').submit(function() {

// inside event callbacks 'this' is the DOM element so we first

// wrap it in a jQuery object and then invoke ajaxSubmit

// disable submit button for prevent requests

$("#buttonSubmit").prop("disabled", true);

$(".processing-file").html("");

$(this).ajaxSubmit(options); //处理uploadfile

$(".progress").show();

url = "/personalcenter/upload\_status?key=" + $("input[name=csrfmiddlewaretoken]").val();//处理upload\_status，get方式

progressWorker(url);

// !!! Important !!!

// always return false to prevent standard browser submit and page navigation

return false;

});

**function** progressWorker**(**url**){**

percent **=** 0**;**

$**.**ajax**({**

url**:** url**,**

async**:** **true,**

dataType**:** "json"**,**

contentType**:** "application/json"**,**

success**:** **function** **(**progress**)** **{**

**if(**progress**.**uploaded **&&** progress**.**totalsize**){**

console**.**log**(**"Percent: "**+**percent**+**"%"**);**

percent **=** **(**progress**.**uploaded**/**progress**.**totalsize**)** **\*** 100**;**

percent **=** parseInt**(**percent**,** 10**);**

$**(**'.progress-bar'**).**css**(**'width'**,** percent**+**'%'**).**attr**(**'aria-valuenow'**,** percent**);** //更新进度条

$**(**'.progress-bar'**).**html**(**percent**+**"%"**);** //显示进度内容

/\* Call sleep function before make a new request in order to prevent much

request to server. Use it wisely... \*/

//sleep(1000);

**}**

**},**

complete**:** **function(){**

**if(**percent**<**100**){**

progressWorker**(**url**);**

**}**

**else{**

$**(**".processing-file"**).**show**();**

$**(**".processing-file"**).**append**(**'<span class="glyphicon glyphicon-refresh glyphicon-refresh-animate"></span> Processing file, please wait...'**);**

**}**

**},**

error**:** **function** **(**jqXHR**,** textStatus**,** errorThrown**)** **{**

**if** **(**jqXHR**.**status **==** 500**)** **{**

alert**(**'Internal error: ' **+** jqXHR**.**responseText**);**

**}** **else** **{**

alert**(**'Unexpected error.'**);**

**}**

**}**

**});**

**}**

// post-submit callback

**function** showResponse**(**responseText**,** statusText**,** xhr**,** $form**)** **{**

// for normal html responses, the first argument to the success callback

// is the XMLHttpRequest object's responseText property

// if the ajaxSubmit method was passed an Options Object with the dataType

// property set to 'xml' then the first argument to the success callback

// is the XMLHttpRequest object's responseXML property

// if the ajaxSubmit method was passed an Options Object with the dataType

// property set to 'json' then the first argument to the success callback

// is the json data object returned by the server

**if(**statusText **===** "success"**){**

$**(**".processing-file"**).**html**(**""**);**

$**(**".processing-file"**).**append**(**responseText**.**message**);**

// enable submit button again for make requests

$**(**"#buttonSubmit"**).**prop**(**"disabled"**,** **false);** //重新enable提交按钮

**}**

**}**

上面用到了两个ajax事件，处理上用了不同的实现方法

/personalcenter/upload\_status使用了常用的$.ajax

/personalcenter/uploadfile使用ajaxSubmit，form里指定的action会被options覆盖

jquery中各个事件执行顺序如下：

1.ajaxStart(全局事件)

2.beforeSend

3.ajaxSend(全局事件)

4.success当请求成功时调用函数，即status==200;

5.ajaxSuccess(全局事件)

6.error

7.ajaxError (全局事件)

8.complete当请求完成时调用函数，即status==404、403、302...只要不出错就行

9.ajaxComplete(全局事件)

10.ajaxStop(全局事件)

在personalcenter/views.py增加对两个ajax 的处理

文件上传成功后会调用upload\_file，其中会把文件进行保存，文件名放到cache里保存。这样用户真正保存时再把图片关联到model

**+from** django**.**core**.**cache **import** cache

**+from** django**.**core**.**cache**.**backends**.**memcached **import** MemcachedCache

**+def** upload\_file**(**request**):**

**+** **if** request**.**method **==** 'POST'**:**

**+** upload\_form **=** UploadFileForm**(**request**.**POST**,** request**.**FILES**)** #➊

**+** **if** upload\_form**.**is\_valid**():**

**+** filename**=**request**.**FILES**[**'image'**]**

**+** **from** PIL **import** Image

**+** **if** filename**:**

**+** img **=** Image**.**open**(**filename**)**

**+** **import** os

**+** photopath **=** os**.**path**.**join**(**settings**.**MEDIA\_ROOT**,** 'upload'**)**

**+** **if** **not** os**.**path**.**exists**(**photopath**):**

**+** os**.**makedirs**(**photopath**)**

**+** img**.**save**(**os**.**path**.**join**(**photopath**,** filename**.**name**))**

**+** cache**.**set**(**'cache\_key\_upload'**,**os**.**path**.**join**(**'upload'**,** filename**.**name**)** **,**60**\***15**)** #➋

**+** **return** HttpResponse**(**json**.**dumps**({**'message'**:** 'Upload complete!'**}))**#➌

**+** **else:**

**+** **return** HttpResponse**(**json**.**dumps**({**'message'**:** 'invalid form!'**}))**#➍

**+** **else:**

**+** form **=** UploadFileForm**()**

**+** #return render\_to\_response('index.html', {'form': form}, context\_instance=RequestContext(request))

**+** **return** HttpResponse**(**json**.**dumps**({**'message'**:** 'invalid form!'**}))**

从$('#form\_upload').submit(function()，可以看到，form\_upload submit提交时会首先处理upload\_status，结束之后才会执行执行upload\_file

upload\_status

[03/Jun/2017 15:00:29] "GET /personalcenter/upload\_status?key=6pxOb4jkfnKLUEU28cmIYL6NDVdkYNhm HTTP/1.1" 200 34

upload\_status

[03/Jun/2017 15:00:29] "GET /personalcenter/upload\_status?key=6pxOb4jkfnKLUEU28cmIYL6NDVdkYNhm HTTP/1.1" 200 37

upload\_file

[03/Jun/2017 15:00:31] "POST /personalcenter/uploadfile HTTP/1.1" 200 66

➊ UploadFileForm仅包含图片，且传递时带上request.FILES

➋ 将文件名存到cache里，此时文件已上传完成

➌➍返回成功或失败结果，格式是json

**+def** upload\_status**(**request**):**

**+** **if** request**.**method **==** 'GET'**:**

**+** **if** request**.**GET**[**'key'**]:**

**+** **if** cache**.**get**(**request**.**GET**[**'key'**]):**

**+** value **=** cache**.**get**(**request**.**GET**[**'key'**])** #➊

**+** **return** HttpResponse**(**json**.**dumps**(**value**),** content\_type**=**"application/json"**)**#

**+** **else:**

**+** **return** HttpResponse**(**json**.**dumps**({**'error'**:**"No csrf value in cache"**}),** content\_type**=**"application/json"**)**

**+** **else:**

**+** **return** HttpResponse**(**json**.**dumps**({**'error'**:**'No parameter key in GET request'**}),** content\_type**=**"application/json"**)**

**+** **else:**

**+** **return** HttpResponse**(**json**.**dumps**({**'error'**:**'No GET request'**}),** content\_type**=**"application/json"**)**

➊这儿对返回值得获取包含两步

(1) request**.**GET**[**'key'**]** , 根据url = "/personalcenter/upload\_status?key=" + $("input[name=csrfmiddlewaretoken]")在GET里面包含字典{‘key’: csrfmiddlewaretoken }

(2) cache**.**get[cache\_key], 这个cache的key是csrfmiddlewaretoken值。 根据 UploadProgressCachedHandler.handle\_raw\_input，它会里面会从self.request.META['CSRF\_COOKIE']获取到当前的csrfmiddlewaretoken，并将cache\_key键值设置为这个token，内容为totalsize和uploaded，该函数能根据这个token从UploadProgressCachedHandler的cache里面拿到返回值

具体进度的数据更新在uploadfilehander里面完成

uploadfilehandler

添加配置

**from** django**.**conf **import** global\_settings

FILE\_UPLOAD\_HANDLERS **=** **(**'personalcenter.uploadfilehandler.UploadProgressCachedHandler'**,** **)** \

**+** global\_settings**.**FILE\_UPLOAD\_HANDLERS

uploadfilehandler.py中定义UploadProgressCachedHandler类，具体参考源码

**class** **UploadProgressCachedHandler(**FileUploadHandler**):**

"""

Tracks progress for file uploads.

The http post request must contain a header or query parameter, we gonna use

CSRF token generated by GET request for form in the view but you can modified

according your considerations or parameters.

For more info, redirect to documentation: https://docs.djangoproject.com/en/1.7/ref/files/uploads/#optional-methods

"""

**def** \_\_init\_\_**(**self**,** request**=None):**

super**(**UploadProgressCachedHandler**,** self**).**\_\_init\_\_**(**request**)**

self**.**progress\_id **=** **None**

self**.**cache\_key **=** **None**

**def** handle\_raw\_input**(**self**,** input\_data**,** META**,** content\_length**,** boundary**,** encoding**=None):**

self**.**content\_length **=** content\_length

**if** 'CSRF\_COOKIE' **in** self**.**request**.**GET**:**

self**.**cache\_key **=** self**.**request**.**GET**[**'CSRF\_COOKIE'**]**

**elif** 'CSRF\_COOKIE' **in** self**.**request**.**META**:**

self**.**cache\_key **=** self**.**request**.**META**[**'CSRF\_COOKIE'**]**

**if** self**.**cache\_key**:**

cache**.**set**(**self**.**cache\_key**,** **{**

'totalsize'**:** self**.**content\_length**,**

'uploaded'**:** 0

**})**

**def** new\_file**(**self**,** field\_name**,** file\_name**,** content\_type**,** content\_length**,** charset**=None,** content\_type\_extra**=None):**

**pass**

**def** receive\_data\_chunk**(**self**,** raw\_data**,** start**):**

**if** self**.**cache\_key**:**

data **=** cache**.**get**(**self**.**cache\_key**)**

data**[**'uploaded'**]** **+=** self**.**chunk\_size # data size increase

cache**.**set**(**self**.**cache\_key**,** data**)**

**return** raw\_data

**def** file\_complete**(**self**,** file\_size**):**

"""

Called when a file has finished uploading.

The handler should return an UploadedFile object that will be stored in request.FILES.

Handlers may also return None to indicate that the UploadedFile object should come from subsequent upload handlers.

"""

**if** self**.**cache\_key**:**

data **=** cache**.**get**(**self**.**cache\_key**)**

data**[**'uploaded'**]** **=** self**.**content\_length

cache**.**set**(**self**.**cache\_key**,** data**)**

time**.**sleep**(**2**)** # sleep for give time to receive final response in request progress bar

**def** upload\_complete**(**self**):**

"""

Callback signaling that the entire upload (all files) has completed.

"""

**if** self**.**cache\_key**:**

cache**.**delete**(**self**.**cache\_key**)**

图片上传完成之后，需要将它附着到User对象。这儿通过upload\_file里设置的cache: cache\_key\_upload

DetailView里面将upload\_form传给template，保存时从cache中读取当前的image信息

class ProfileDetailView(FormMixin, DetailView):

def get\_context\_data(self, \*args, \*\*kwargs):

context = super(ProfileDetailView, self).get\_context\_data(\*args, \*\*kwargs)

context["form"] = self.form\_class(instance = self.get\_object())

+ context["upload\_form"] = UploadFileForm()

return context

def post(self, request, \*args, \*\*kwargs):

form = self.get\_form()

if form.is\_valid():

usermodel = UserModel.objects.get(id=self.kwargs.get("id"))

usermodel.first\_name = form.cleaned\_data['first\_name']

usermodel.last\_name = form.cleaned\_data['last\_name']

- if 'image' in form.cleaned\_data:

- usermodel.image = form.cleaned\_data['image']

+ if not cache.get('cache\_key\_upload',None) is None:

+ usermodel.image = cache.get('cache\_key\_upload',None) #➊

+ if cache.has\_key('cache\_key\_upload'):

+ cache.delete('cache\_key\_upload')

usermodel.save()

return self.form\_valid(form)

else:

+ self.object = self.get\_object(\*args, \*\*kwargs)

return self.form\_invalid(form)

➊ 根据key找到image对象，并将它附着到user对象

添加新的js和css来支持ajaxsubmit和progressbar

+ <script src="{% static 'js/jquery-ui.js' %}"></script>

+ <script src="{% static 'js/jquery.form.js' %}"></script>

+ <link href="{% static 'css/jquery-ui.css' %}" rel="stylesheet">



### 我的积分

可以显示用户当前积分，如果需要冻结积分，也可显示冻结积分和可使用积分。还可以显示积分明细，纪录积分赚取花费情况

首先在personalcenterhome.html里面添加tab

<ul class="nav nav-pills ">

<li role="presentation" class="active"><a href='{% url "personalcenter" id=myuser.id %}'>Profile</a></li>

<li role="presentation"><a href="#">Messages</a></li>

<li role="presentation"><a href='{% url "myuser\_profile\_extend" id=myuser.id %}'>Score</a></li>

<li role="presentation"><a href='{% url "myorder" id=myuser.id %}'>Order</a></li>

<li role="presentation"><a href="#">Donate</a></li>

</ul>

URL添加各个tab的链接

url**(**r'^(?P<id>\d+)/$'**,** personalcenterhome**,** name**=**'personalcenter'**),**

url**(**r'^(?P<id>\d+)/myorder$'**,** myorder**,** name**=**'myorder'**),**

url**(**r'^(?P<id>\d+)/myprofileextend$'**,** myuser\_profile\_extend**,** name**=**'myuser\_profile\_extend'**),**

url**(**r'^(?P<id>\d+)/change$'**,** ProfileDetailView**.**as\_view**(),** name**=**'profile\_detail'**),**

以我订单为例

**def** myorder**(**request**,**id**):**

myuser **=** get\_object\_or\_404**(**UserModel**,** id**=**id**)**

**try:**

usercheckout **=** UserCheckout**.**objects**.**get**(**user **=** myuser**)**

**except** UserCheckout**.**DoesNotExist**:**

usercheckout **=** **None**

**except:**

**raise** Http404

order\_objects **=** Order**.**objects**.**filter**(**user **=** usercheckout**)**

template **=** "personalcenter/myorder.html"

context **=** **{**

"myuser"**:** myuser**,**

"order\_objects" **:** order\_objects**,**

**}**

为了支持积分，等级等，添加了新的model-UserProfile

class UserProfile(models.Model):

user = models.OneToOneField(

settings.AUTH\_USER\_MODEL,

on\_delete=models.CASCADE,

)

score = models.DecimalField(max\_digits=10, decimal\_places=0, default=0)

level = models.DecimalField(max\_digits=10, decimal\_places=0, default=0)

def \_\_unicode\_\_(self):

return str(self.user)

### 会员等级

可以根据捐衣次数或者兑换次数，或者消费积分的累积，享受不同的折扣兑换，增加用户对网站的粘性

### 我的捐赠

可以展示我捐赠衣服的次数，时间，上传的照片等其它信息

### 我的订单

显示我使用积分商城的兑换情况

将用户的订单信息传递给template

class ProfileDetailView(FormMixin, DetailView):

model = UserModel

template\_name = "personalcenter/profile\_detail.html"

form\_class = MyUserForm

def get\_context\_data(self, \*args, \*\*kwargs):

context = super(ProfileDetailView, self).get\_context\_data(\*args, \*\*kwargs)

context["form"] = self.form\_class(instance = self.get\_object())

upload\_file\_form = UploadFileForm()

context["upload\_form"] = upload\_file\_form

+ usermodel = UserModel.objects.get(id=self.kwargs.get("id"))

+ from orders.models import UserCheckout

+ usercheckout = UserCheckout.objects.get(user = usermodel)

+ context["object\_list"] = Order.objects.filter(user = usercheckout)

return context

在profile\_detail.html里面加上order信息的显示，具体的积分兑换情况等后面实现了再加

<h3>Your Orders</h3>

<table class='table'>

<tr>

<th>Order Number</th>

<th>Item Count</th>

<th>Exchange</th>

</tr>

{% for object in object\_list %}

<tr>

<td>

<a href='{{ object.get\_absolute\_url }}'>#{% if object.order\_id %}{{ object.order\_id }}{% else %}{{ object.id }}{% endif %}</a>

</td>

<td>

{{ object.cart.items.count }}

</td>

<td>0</td>

</tr>

{% endfor %}

</table>



## 设置

### 个人资料修改

修改个人信息资料，包含会员性别，生日，昵称等



### 密码修改

修改登录密码



### 意见反馈

给网站提出意见



### 注销登录

注销当前用户登录信息



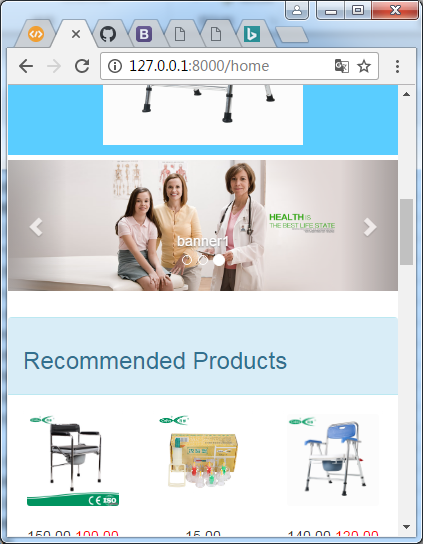
## 活动消息

### 动态消息展示

可以显示后台发送的动态消息等 scroll spy

### 轮播图

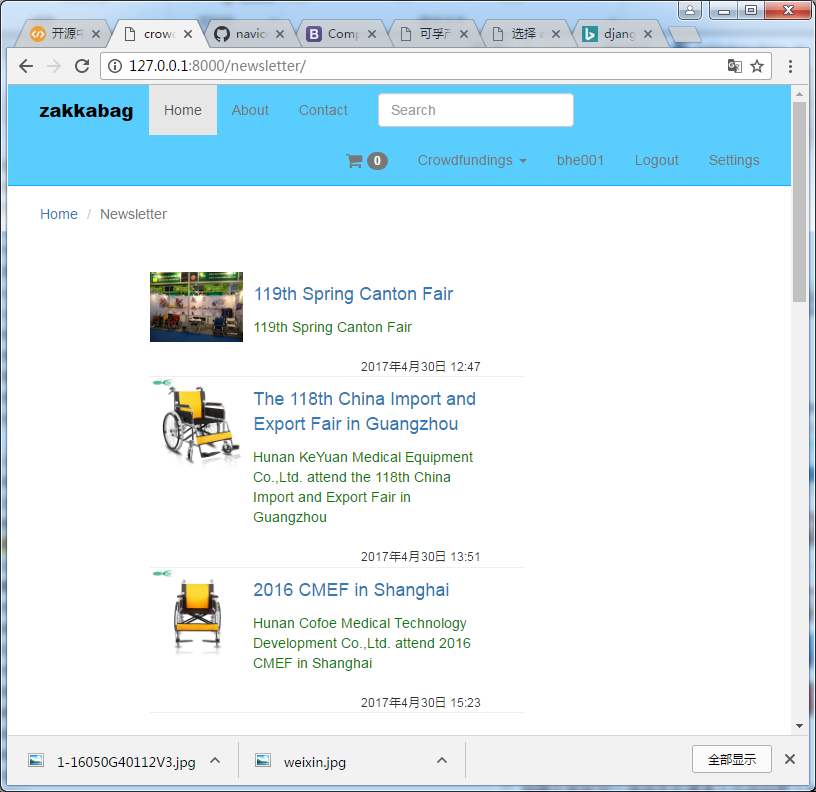
轮播切换广告图片，可以放自己活动也可以作为广告位营销



全屏显示

### 活动专区

展示活动内容



## 捐衣模块

### 拍照上传

捐赠衣服前可以调用手机摄像头功能拍照上传

### 捐衣信息填报

填写捐赠件数等其它信息，还可以选择具体哪个捐赠衣服箱，方便工作人员审核和统计

目录

upload >> user

products >> category >>

>> featured\_product

banner >> filename\*

crowdfunding >> username

profile >> username

footer >> filename\*

## 众筹模块 (产品模块代替)

### 众筹列表

可以列表展示众筹信息

<http://127.0.0.1:8000/crowdfundings/>

### 众筹分类

可以根据大病救助，助学，创业分类展示众筹信息

### 众筹详情

可以展示众筹内容的展示，包含众筹的标题，内容以及图片和详情等具体信息

### 筹款动态

可以显示筹款的动态信息

### 众筹评论

对本条众筹可以添加评论信息

### 众筹收藏

可以收藏众筹信息，方便之后可以快速找到

### 众筹分享

可以把众筹信息，分享到主流的第三方平台上比如微信，朋友圈，微博，qq等

### 在线支付

可以通过在线支付支持爱心众筹，微网站支持微信一种支付方式

### 举报

如果发现不是真实的众筹信息，可以通过举报按钮提交举报，工作人员可以后台审核举报信息

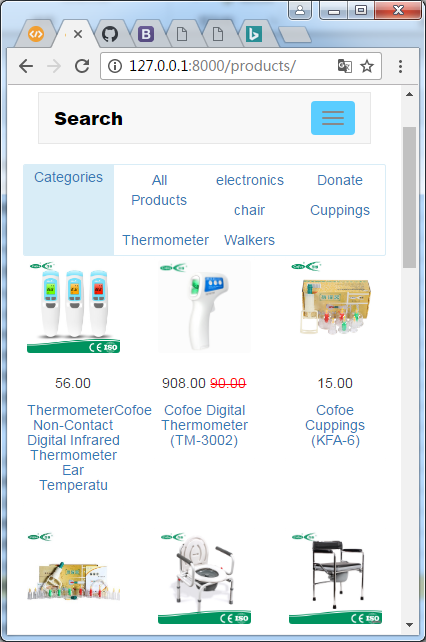
### 发布众筹

可以通过平台发起大病救助，扶贫助学，支持创业等众筹项目，包含金额，用途，截止日期，标题，详情，照片，身份证信息等资料

## 积分商城

### 商品列表

展示积分商城的商品，包含图片，标题，所需兑换积分等信息



### 商品分类

可以根据分类展示不同的兑换商品



### 商品详情

展示单个积分商城商品的详细信息



### 积分兑换

通过积分兑换模块兑换所选择的商品，自动扣取积分



### 积分兑换规则

可以通过后台配置积分规则，前台此模块显示兑换的规则

# 公益平台管理系统（PC Web）

## 会员管理

### 会员认证管理

会员认证管理是对于会员的认证信息进行管理，可调整会员的认证方式包括手机认证、邮箱认证等其他第三方认证，关联微信openid,同时对会员的认证相关信息进行维护。

### 会员资料管理

会员资料管理是对会员的个人资料进行管理，可查看会员相关资料信息包括姓名、性别、手机、注册时间、邮箱、生日、剩余积分、等信息，可通过姓名、性别、手机号、等信息进行查询。提供新增会员，并根据需要进行查看详情、修改、删除功能。

### 会员积分管理

会员积分管理是对会员积分，兑换相关业务进行管理，可记录回溯兑换业务信息，包括手机号码、会员名称、兑换时间、兑换金额、本次积分、兑换商品、办理渠道、结算方式等，可通过电话以及选择相应开始时间到结束时间进行查询；可支撑会员的退积分业务处理。如果客户有ERP系统，各笔消费相关业务操作记录券可通过接口同步至ERP系统。

### 会员捐衣管理

对于会员捐衣纪录进行管理

### 会员等级管理

会员等级是可以新增会员等级，设置会员等级名称，消费金额，演出折扣，配置会员权益说明的功能。

### 会员协议管理

对线上平台出现的会员协议内容进行管理，可编辑修改会员协议内容。

## 微网站内容管理

### 微网站轮播图管理

对微网站轮播图进行新增，修改，删除，更换位置等管理

### 微网站活动管理

对微网站活动列表以及内容进行新增，修改，删除等管理

## app内容管理

### app轮播图管理

对app轮播图进行新增，修改，删除，更换位置等管理

### app活动管理

对app活动列表以及内容进行新增，修改，删除等管理

## 消息推送

### 全局消息推送

可以通过后台给全部会员发布消息

### 分组会员消息推送

可以根据对组员分类推送不同消息，比如对参加同个众筹的会员推送众筹的进度情况

## 众筹管理

### 官方发起众筹

官方发起的众筹的可以通过管理后台发布

### 众筹管理

对于平台上用户发起的众筹进度进行监控管理

### 审核管理

对用户提交的的众筹信息进行审核和对外发布，可以进行删除和管理

### 评论管理

管理对众筹发布的评论信息

### 举报管理

对于用户举报的众筹信息进行评估监控可以对其核实，删除等管理

### 支付管理

管理用户对众筹投入的资金，管控到期众筹的资金发放

## 商城管理

### 商品推荐管理

对积分商城推荐的商品进行推荐管理，可以更换推荐的位置，删除推荐等

### 商品上下架管理

对积分商城的商品进行上架和下架的维护和库存管理

### 商品详情管理

对金粉商城的商品进行商品内容，图片，详细信息的内容管理

### 订单管理

管理积分兑换的订单进行管理

## 图片上传 (后台)

需求：添加新产品，其中包括上传产品图片。需要动态的更新上传进度。

<https://docs.djangoproject.com/en/1.8/topics/http/file-uploads/#handling-uploaded-files-with-a-model>

**Handling uploaded files with a model**[**¶**](https://docs.djangoproject.com/en/1.8/topics/http/file-uploads/#handling-uploaded-files-with-a-model)

If you’re saving a file on a [**Model**](https://docs.djangoproject.com/en/1.8/ref/models/instances/#django.db.models.Model) with a [**FileField**](https://docs.djangoproject.com/en/1.8/ref/models/fields/#django.db.models.FileField), using a [**ModelForm**](https://docs.djangoproject.com/en/1.8/topics/forms/modelforms/#django.forms.ModelForm) makes this process much easier. The file object will be saved to the location specified by the [**upload\_to**](https://docs.djangoproject.com/en/1.8/ref/models/fields/#django.db.models.FileField.upload_to) argument of the corresponding [**FileField**](https://docs.djangoproject.com/en/1.8/ref/models/fields/#django.db.models.FileField) when calling **form.save()**:

**from** **django.http** **import** HttpResponseRedirect

**from** **django.shortcuts** **import** render

**from** **.forms** **import** ModelFormWithFileField

**def** upload\_file(request):

**if** request.method == 'POST':

form = ModelFormWithFileField(request.POST, request.FILES)

**if** form.is\_valid():

*# file is saved*

form.save()

**return** HttpResponseRedirect('/success/url/')

**else**:

form = ModelFormWithFileField()

**return** render(request, 'upload.html', {'form': form})

1. 添加表单Form

注意: 在表单中加入[enctype](http://www.w3school.com.cn/tags/att_form_enctype.asp)="multipart/form-data；如果不加enctype="multipart/form-data，得不到图片信息，只有路径

FILES是个字典，它包含每个FileField的键 （或者ImageField，FileField的子类）。这样的话就可以用request.FILES['File']来存放表单中的这些数据了。

注意request.FILES只有在请求方法为POST，并且发送请求的<form>拥有enctype="multipart/form-data属性时，才会包含数据。否则request.FILES为空

创建ProductImageForm用于图片的上传，里面唯一的field是image

products/forms.py

class ProductImageForm(forms.ModelForm):

class Meta:

model = ProductImage

fields = [

'image',

]

在模板里面添加两个form，ProductCreateForm和ProductImageForm，分别用于正常的产品信息的创建和图片的上传。

products/templates/products/product\_create.html

-<form method='POST' action=''>{% csrf\_token %}

+<form method='POST' enctype="multipart/form-data" action=''>{% csrf\_token %}

{{ form|crispy }}

+{{image\_form.as\_p}}

<input class = 'btn btn-primary' type='submit' value='Submit' />

</form>

products/views.py

在get\_context\_data里面把ProductImageForm传给server

class ProductCreateView(CreateView):

template\_name = 'products/product\_create.html'

form\_class = ProductCreateForm

success\_url = "/products"

def get\_context\_data(self, \*args, \*\*kwargs):

context = super(ProductCreateView, self).get\_context\_data(\*args, \*\*kwargs)

context["image\_form"] = ProductImageForm()#➊➋➌➍➎➏➐➑➒➓

return context

def post(self, request, \*args, \*\*kwargs):

postresult = super(ProductCreateView, self).post(request, \*args, \*\*kwargs)

if 0: #方法1 – 手动处理文件保存

filename=request.FILES['image'] #➊➋➌➍➎➏➐➑➒➓

from PIL import Image #➊➋➌➍➎➏➐➑➒➓

if filename:

img=Image.open(filename)

title = self.object.title

slug = slugify(title)

basename, file\_extension = filename.name.split(".")

new\_filename = "%s-%s.%s" %(slug, self.object.id, file\_extension)

from django.conf import settings

import os

photoname = os.path.join("products", slug, new\_filename)

photopath = os.path.join(settings.MEDIA\_ROOT, "products", slug)

if not os.path.exists(photopath):

os.makedirs(photopath) #➊➋➌➍➎➏➐➑➒➓

img.save(os.path.join(settings.MEDIA\_ROOT, photoname)) #➊➋➌➍➎➏➐➑➒➓

ProductImage.objects.create(product = self.object,

image = photoname) #➊➋➌➍➎➏➐➑➒➓

# BELOW ALSO WORKS 方法2 – 利用form处理文件保存

else:

imageForm = ProductImageForm(request.POST, request.FILES) #➊➋➌➍➎➏➐➑➒➓

if imageForm.is\_valid():

productImage = imageForm.save(commit=False) #➊➋➌➍➎➏➐➑➒➓

productImage**.**product **=** self**.**object

productImage**.**save**()**

**return** postresult

**return** postresult

<https://docs.djangoproject.com/en/1.2/topics/http/file-uploads/>

[django上传图片](http://www.cnblogs.com/RoundGirl/p/5395658.html)

[python image模块](http://www.cnblogs.com/DjangoBlog/p/3557744.html)

[Python创建目录文件夹](http://www.qttc.net/201209207.html)

[[豆瓣]浅论Django文件上传](https://www.douban.com/group/topic/21131470/)

[Django Ajax 文件上传及进度显示](http://onlypython.group.iteye.com/group/wiki/1517-django-ajax-file-uploads-and-progress-display)

[Ajax 和 jQuery 实现进度条＋上传文件到Django](http://blog.csdn.net/dimply/article/details/42425373)

[ajax异步上传图片 ajax 异步上传带进度条视频并提取缩略图](http://www.alonely.com.cn/Ajax/20161025/58632.html)

[Django and AJAX image uploads](https://blog.mozilla.org/webdev/2010/09/17/django-and-ajax-image-uploads/) | [Source Code](https://github.com/pcraciunoiu/kitsune/)

[Django image upload](http://cloudinary.com/documentation/django_image_upload)

[jQuery File Upload Demo](https://blueimp.github.io/jQuery-File-Upload/) | [wiki](https://github.com/blueimp/jQuery-File-Upload/wiki) | [Source Code](https://github.com/blueimp/jQuery-File-Upload)

[Django jQuery File Upload](https://github.com/sigurdga/django-jquery-file-upload) django基于jQuery File Upload的实现

[jQuery File Upload](http://www.cnblogs.com/cart55free99/p/3823357.html)

[Node.js&NPM的安装与配置](http://www.infoq.com/cn/articles/nodejs-npm-install-config)

[Django “./manage.py bower install” tells me bower isn't installed when it is](https://stackoverflow.com/questions/38808548/django-manage-py-bower-install-tells-me-bower-isnt-installed-when-it-is)

<https://fossies.org/windows/www/node-v8.0.0-win-x64.zip/>

[Windows环境下的NodeJS+NPM+Bower安装配置](http://jingyan.baidu.com/article/2d5afd69e243cc85a2e28efa.html)

## 用户扩展

用户扩展包括以下几种方法

* (1) 直接修改django 源码
* (2) 把django 的user以及认证部分的源代码拷贝到自己的app下面
* (3) OneToOneField扩展
* (4) Profile Model (1.5之后不再支持)
* (5) AbstractBaseUser, PermissionsMixin开始派生出一个自定用户Model，并且实现自定义的BaseUserManager (1.5之后)
* (6) ProfileBase(type)
* (7) 继承User

(1)(2) 每次升级都要修改，不合适

(7)

**class** **MyUser(**AbstractBaseUser**,** PermissionsMixin**):**

**class** **MyUser(**AbstractUser**):**

**class** **MyUserManager(**BaseUserManager**):**

## 扩展用户授权登录

### 指定authentication backend

Django维护了一个“authentication backends”列表，通过它在授权。Django会遍历所有的backends直到授权接收。

authentication backends在settings里的[AUTHENTICATION\_BACKENDS](https://docs.djangoproject.com/en/dev/ref/settings/#std:setting-AUTHENTICATION_BACKENDS)指定，可以是任意路径，默认的backends是

['django.contrib.auth.backends.ModelBackend']

[AUTHENTICATION\_BACKENDS](https://docs.djangoproject.com/en/dev/ref/settings/#std:setting-AUTHENTICATION_BACKENDS)的顺序是有影响的，如果多个backends的用户名和密码都能验证通过，django在第一次匹配后就会停止。

如果backend抛出[PermissionDenied](https://docs.djangoproject.com/en/dev/ref/exceptions/#django.core.exceptions.PermissionDenied)异常，授权检查会立即停止，不会再检查后面的backend。

注意：如果授权成功，django会把这个授权方式保存到session里，session周期里的下一次接入还是用这种方式。如果要强迫用不同方法授权，一个简单的方法是调用Session.objects.all().delete().

*Settings.py*

AUTHENTICATION\_BACKENDS **=** **(**

'personalcenter.backends.auth.WechatBackend'**,**

'django.contrib.auth.backends.ModelBackend'**,**

**)**

AUTH\_USER\_MODEL **=** 'personalcenter.MyUser'

### 实现authentication backend

必须实现的方法

get\_user(user\_id)

authenticate(request, \*\*credentials)

不知道为什么？username登录时的request.user是有值的，但是wechat登录永远是anonymous，这个从一开始的render就开始了

\_cached\_user为AnonymousUser: AnonymousUser

>>问题查清楚了，下面这个wechat auth backend函数没写好，之前返回None，这个函数在eclipse上打断点也进不去不知道为什么

>>好像也不是这个问题，突然就好了

personalcenter.auth.py

class WechatBackend(object):

def get\_user(self, user\_id):

try:

return UserModel.objects.get(pk=user\_id)

except UserModel.DoesNotExist:

return None

**def** authenticate**(**self**,** request**,** user**):**

obj **=** **None**

profile **=** **None**

**try:**

profile **=** WechatUserProfile**.**objects**.**get**(**openid **=** user**[**'openid'**])**

**if** profile**:**

obj **=** profile**.**user

**except** WechatUserProfile**.**DoesNotExist**:**

profile **=** WechatUserProfile**.**objects**.**create**(**

user **=** **None,** openid **=** user**[**'openid'**])**

profile**.**unionid **=** user**[**'unionid'**]**

profile**.**city **=** user**[**'city'**]**

profile**.**country **=** user**[**'country'**]**

profile**.**language **=** user**[**'language'**]**

**if** 1 **==** user**[**'sex'**]:**

profile**.**sex **=** 'male'

**else:**

profile**.**sex **=** 'female'

profile**.**nickname **=** user**[**'nickname'**]**

profile**.**headimgurl **=** user**[**'headimgurl'**]**

profile**.**save**()**

request**.**session**[**'wechat\_id'**]** **=** profile**.**id

request**.**user **=** obj

**return** obj

### wechat\_id的保存方法

Workaround方法 – 可能已解决user为空的情况

class openidmiddleware():

def process\_request(self, request):

wechat\_id = request.session.get("wechat\_id")

if wechat\_id:

UserModel = get\_user\_model()

try:

request.user = UserModel.objects.get(pk=wechat\_id)

except UserModel.DoesNotExist:

pass

class WechatBackend(object):

def authenticate(self, request, code, app\_id=None, app\_secret=None, redirect\_uri=None):

request.session['wechat\_id'] = obj.id

return None

### 绑定WechatUser和User

personalcenter.urls

urlpatterns **=** **[**

url**(**r'^linktowechat$'**,** accountlinktowechat**,** name**=**'link\_to\_wechat'**),**

**def** accountlinktowechat**(**request**):**

user **=** auth**.**get\_user**(**request**)**

wechat\_id **=** request**.**session**.**get**(**"wechat\_id"**)**

wechat **=** **None**

**if** wechat\_id**:**

**try:**

wechat **=** WechatUserProfile**.**objects**.**get**(**pk**=**wechat\_id**)**

**wechat.user = user** #这儿将WechatUser和User绑定起来

wechat**.**save**()**

**return** redirect**(**reverse**(**"home"**,** kwargs**={}))**

**except:**

**pass**

**return** redirect**(**reverse**(**"home"**,** kwargs**={}))**

#### 激活成功后绑定当前微信用户

templates.registration.activation\_complete.html

{% block content %}

<p class = "lead">

{% trans "Your account is now activated." %}

{% if not user.is\_authenticated %}

{% trans "You can log in." %}

{% else %}

<a href='{% url "link\_to\_wechat" %}'>Link to wechat</a>

{% endif %}

</p>

{% endblock %}

#### 微信登录后绑定当前登录用户

request middleware里面把wechat信息添加进去，这样在template里面能判断微信用户和普通用户是否已绑定

这儿额外在request里面加了一个变量wechat，否则就要给registration添加额外的context (第三方库)

**from** django**.**contrib**.**auth **import** get\_user\_model

**from** personalcenter**.**models **import** WechatUserProfile

UserModel **=** get\_user\_model**()**

**class** **openidmiddleware():**

**def** process\_request**(**self**,** request**):**

**if** request**.**user**.**is\_anonymous**:**

wechat\_id **=** request**.**session**.**get**(**"wechat\_id"**)**

**if** wechat\_id**:**

**try:**

request**.**wechat **=** WechatUserProfile**.**objects**.**get**(**pk**=**wechat\_id**)**

**except:**

**pass**

如果当前WechatUser指定并没有绑定User，则提示绑定

templates.registration.login.html

{% if request.user.is\_anonymous and request.wechat %}

You need to login to continue

{% endif %}

<form method=**"post"** action=**""**>

**{% csrf\_token %}**

**{{ form|crispy }}**

<input class = **"btn btn-block btn-primary"** type=**"submit"** value=**"{% trans 'Log in' %}"** />

**{% if request.user.is\_anonymous and request.wechat %}**

<input type=**"hidden"** name=**"next"** value=**"{% url 'link\_to\_wechat' %}"** /> <!-- workaround here, maybe we need to change it in login function later-->

**{% else %}**

<input type=**"hidden"** name=**"next"** value=**"{{ next }}"** />

**{% endif %}**

</form>

### 解绑WechatUser和User

personalcenter.urls

urlpatterns **=** **[**

url**(**r'^unlinkfromwechat$'**,** account\_unlink\_from\_wechat**,** name**=**'unlink\_from\_wechat'**),**

**def** account\_unlink\_from\_wechat**(**request**):**

wechat\_id **=** request**.**session**.**get**(**"wechat\_id"**)**

wechat **=** **None**

**if** wechat\_id**:**

**try:**

wechat **=** WechatUserProfile**.**objects**.**get**(**pk**=**wechat\_id**)**

wechat**.**user **=** **None**

wechat**.**save**()**

**except:**

**pass**

**return** redirect**(**reverse**(**"home"**,** kwargs**={}))** # to change to ajax

### django实现

#### authenticate

下面分析django是如何实现authentication的

authenticate会直接调用这个函数

django.contrib.auth.\_\_init\_\_

**def** authenticate**(\*\***credentials**):**

"""

If the given credentials are valid, return a User object.

"""

**for** backend**,** backend\_path **in** \_get\_backends**(**return\_tuples**=True):**

**try:**

inspect**.**getcallargs**(**backend**.**authenticate**,** **\*\***credentials**)** # 遍历backend

**except** TypeError**:**

# This backend doesn't accept these credentials as arguments. Try the next one.

**continue**

**try:**

user **=** backend**.**authenticate**(\*\***credentials**)** # 调用backend的authenticate函数

**except** PermissionDenied**:**

# This backend says to stop in our tracks - this user should not be allowed in at all.

**return** **None**

**if** user **is** **None:**

**continue**

# Annotate the user object with the path of the backend.

user**.**backend **=** backend\_path

**return** user

# The credentials supplied are invalid to all backends, fire signal

user\_login\_failed**.**send**(**sender**=**\_\_name\_\_**,**

credentials**=**\_clean\_credentials**(**credentials**))**

demo代码里backend的authenticate的参数包含request，调用authenticate时，必须request=request, 否则django.contrib.auth.\_\_init\_\_的authenticate**(\*\***credentials**)**函数在getcallargs这一步会出错，因为request这个参数不能传进**\*\***credentials

<https://docs.djangoproject.com/en/dev/topics/auth/customizing/#writing-an-authentication-backend>

* [Keyword Arguments](https://docs.python.org/2/tutorial/controlflow.html#keyword-arguments)
* [python 中参数\*args, \*\*kwargs](http://blog.csdn.net/anhuidelinger/article/details/10011013)

网上例子的练习，实际并没有用

*Auth.py (personalcenter\backends)*

class SettingsBackend(object):

**def** authenticate**(**self**,** request**,** username**=None,** password**=None):**

login\_valid **=** **(**settings**.**ADMIN\_LOGIN **==** username**)**

pwd\_valid **=** check\_password**(**password**,** settings**.**ADMIN\_PASSWORD**)**

**if** login\_valid **and** pwd\_valid**:**

**try:**

user **=** User**.**objects**.**get**(**username**=**username**)**

**except** User**.**DoesNotExist**:**

# Create a new user. There's no need to set a password

# because only the password from settings.py is checked.

user **=** User**(**username**=**username**)**

user**.**is\_staff **=** **True**

user**.**is\_superuser **=** **True**

user**.**save**()**

**return** user

**return** **None**

**def** get\_user**(**self**,** user\_id**):**

**try:**

**return** User**.**objects**.**get**(**pk**=**user\_id**)**

**except** User**.**DoesNotExist**:**

**return** **None**

<https://docs.djangoproject.com/en/dev/topics/auth/default/#how-to-log-a-user-in>

**from** django**.**contrib**.**auth **import** authenticate**,** login

**def** my\_view**(**request**):**

username **=** request**.**POST**[**'username'**]**

password **=** request**.**POST**[**'password'**]**

user **=** authenticate**(**request= request**,** username**=**username**,** password**=**password**)**

**if** user **is** **not** **None:**

login**(**request**,** user**)**

# Redirect to a success page.

**...**

**else:**

# Return an 'invalid login' error message.

**...**

*django.contrib.auth.backends.py*

class ModelBackend(object):

"""

Authenticates against settings.AUTH\_USER\_MODEL.

"""

def authenticate(self, username=None, password=None, \*\*kwargs):

UserModel = get\_user\_model()

if username is None:

username = kwargs.get(UserModel.USERNAME\_FIELD)

try:

user = UserModel.\_default\_manager.get\_by\_natural\_key(username)

if user.check\_password(password):

return user

except UserModel.DoesNotExist:

# Run the default password hasher once to reduce the timing

# difference between an existing and a non-existing user (#20760).

UserModel().set\_password(password)

#### login

*\_\_init\_\_.py (site-packages\django\contrib\auth)*

SESSION\_KEY **=** '\_auth\_user\_id'

BACKEND\_SESSION\_KEY **=** '\_auth\_user\_backend'

HASH\_SESSION\_KEY **=** '\_auth\_user\_hash'

REDIRECT\_FIELD\_NAME **=** 'next'

**def** login**(**request**,** user**):**

"""

Persist a user id and a backend in the request. This way a user doesn't

have to reauthenticate on every request. Note that data set during

the anonymous session is retained when the user logs in.

"""

session\_auth\_hash **=** ''

**if** user **is** **None:**

user **=** request**.**user

**if** hasattr**(**user**,** 'get\_session\_auth\_hash'**):**

session\_auth\_hash **=** user**.**get\_session\_auth\_hash**()**

**if** SESSION\_KEY **in** request**.**session**:**

**if** \_get\_user\_session\_key**(**request**)** **!=** user**.**pk **or** **(**

session\_auth\_hash **and**

request**.**session**.**get**(**HASH\_SESSION\_KEY**)** **!=** session\_auth\_hash**):**

# To avoid reusing another user's session, create a new, empty

# session if the existing session corresponds to a different

# authenticated user.

request**.**session**.**flush**()**

**else:**

request**.**session**.**cycle\_key**()**

# 正常用户登录时，会保存这个用户session，这样在response的消息里的request里面会带出去user信息

request**.**session**[**SESSION\_KEY**]** **=** user**.**\_meta**.**pk**.**value\_to\_string**(**user**)**

request**.**session**[**BACKEND\_SESSION\_KEY**]** **=** user**.**backend

request**.**session**[**HASH\_SESSION\_KEY**]** **=** session\_auth\_hash

**if** hasattr**(**request**,** 'user'**):**

request**.**user **=** user

rotate\_token**(**request**)**

user\_logged\_in**.**send**(**sender**=**user**.**\_\_class\_\_**,** request**=**request**,** user**=**user**)**

{{user}}

{{request.user}}

这两个user分别是由下面的middleware传进来的

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

#### get\_user

*django/contrib/auth/\_\_init\_\_*

**def** get\_user**(**request**):**

"""

Returns the user model instance associated with the given request session.

If no user is retrieved an instance of `AnonymousUser` is returned.

"""

**from** **.**models **import** AnonymousUser

user **=** **None**

**try:**

user\_id **=** \_get\_user\_session\_key**(**request**)**

backend\_path **=** request**.**session**[**BACKEND\_SESSION\_KEY**]**

**except** KeyError**:**

**pass**

**else:**

**if** backend\_path **in** settings**.**AUTHENTICATION\_BACKENDS**:**

backend **=** load\_backend**(**backend\_path**)**

user **=** backend**.**get\_user**(**user\_id**)**

# Verify the session

**if** **(**'django.contrib.auth.middleware.SessionAuthenticationMiddleware'

**in** settings**.**MIDDLEWARE\_CLASSES **and** hasattr**(**user**,** 'get\_session\_auth\_hash'**)):**

session\_hash **=** request**.**session**.**get**(**HASH\_SESSION\_KEY**)**

session\_hash\_verified **=** session\_hash **and** constant\_time\_compare**(**

session\_hash**,**

user**.**get\_session\_auth\_hash**()**

**)**

**if** **not** session\_hash\_verified**:**

request**.**session**.**flush**()**

user **=** **None**

**return** user **or** AnonymousUser**()**

*django/contrib/auth/backends.py*

class ModelBackend(object):

def get\_user(self, user\_id):

UserModel = get\_user\_model()

try:

return UserModel.\_default\_manager.get(pk=user\_id)

except UserModel.DoesNotExist:

return None

*django/contrib/auth/forms.py*

class AuthenticationForm(forms.Form):

def get\_user(self):

return self.user\_cache

*django/contrib/auth/middleware.py*

**def** get\_user**(**request**):**

**if** **not** hasattr**(**request**,** '\_cached\_user'**):**

request**.**\_cached\_user **=** auth**.**get\_user**(**request**)**

**return** request**.**\_cached\_user

\_cached\_user when will it be used?

#### auth

*django.contrib.auth.context\_processors.auth*

**def** auth**(**request**):**

"""

Returns context variables required by apps that use Django's authentication

system.

If there is no 'user' attribute in the request, uses AnonymousUser (from

django.contrib.auth).

"""

**if** hasattr**(**request**,** 'user'**):**

user **=** request**.**user

**else:**

**from** django**.**contrib**.**auth**.**models **import** AnonymousUser

user **=** AnonymousUser**()**

**return** **{**

'user'**:** user**,**

'perms'**:** PermWrapper**(**user**),**

**}**

#### middleware

*Middleware.py (site-packages\django\contrib\auth*

**def** get\_user**(**request**):**

**if** **not** hasattr**(**request**,** '\_cached\_user'**):**

request**.**\_cached\_user **=** auth**.**get\_user**(**request**)**

**return** request**.**\_cached\_user

**class** **AuthenticationMiddleware(**object**):**

**def** process\_request**(**self**,** request**):**

**assert** hasattr**(**request**,** 'session'**),** **(**

"The Django authentication middleware requires session middleware "

"to be installed. Edit your MIDDLEWARE\_CLASSES setting to insert "

"'django.contrib.sessions.middleware.SessionMiddleware' before "

"'django.contrib.auth.middleware.AuthenticationMiddleware'."

**)**

request**.**user **=** SimpleLazyObject**(lambda:** get\_user**(**request**))**

class ThreadingMixIn:

"""Mix-in class to handle each request in a new thread."""

# Decides how threads will act upon termination of the

# main process

daemon\_threads = False

def process\_request\_thread(self, request, client\_address):

"""Same as in BaseServer but as a thread.

In addition, exception handling is done here.

"""

try:

self.finish\_request(request, client\_address)

self.shutdown\_request(request)

except:

self.handle\_error(request, client\_address)

self.shutdown\_request(request)

def process\_request(self, request, client\_address):

"""Start a new thread to process the request."""

t = threading.Thread(target = self.process\_request\_thread,

args = (request, client\_address))

t.daemon = self.daemon\_threads

t.start()

class BaseHandler(object):

**def** get\_response**(**self**,** request**):**

"Returns an HttpResponse object for the given HttpRequest"

# Setup default url resolver for this thread, this code is outside

# the try/except so we don't get a spurious "unbound local

# variable" exception in the event an exception is raised before

# resolver is set

urlconf **=** settings**.**ROOT\_URLCONF

urlresolvers**.**set\_urlconf**(**urlconf**)**

resolver **=** urlresolvers**.**RegexURLResolver**(**r'^/'**,** urlconf**)**

**try:**

response **=** **None**

# Apply request middleware

**for** middleware\_method **in** self**.**\_request\_middleware**:**

response **=** middleware\_method**(**request**)**

**if** response**:**

**break**

**if** response **is** **None:**

**if** hasattr**(**request**,** 'urlconf'**):**

# Reset url resolver with a custom urlconf.

urlconf **=** request**.**urlconf

urlresolvers**.**set\_urlconf**(**urlconf**)**

resolver **=** urlresolvers**.**RegexURLResolver**(**r'^/'**,** urlconf**)**

resolver\_match **=** resolver**.**resolve**(**request**.**path\_info**)**

callback**,** callback\_args**,** callback\_kwargs **=** resolver\_match

request**.**resolver\_match **=** resolver\_match

# Apply view middleware

**for** middleware\_method **in** self**.**\_view\_middleware**:**

response **=** middleware\_method**(**request**,** callback**,** callback\_args**,** callback\_kwargs**)**

**if** response**:**

**break**

**if** response **is** **None:**

wrapped\_callback **=** self**.**make\_view\_atomic**(**callback**)**

**try:**

response = wrapped\_callback(request, \*callback\_args, \*\*callback\_kwargs) # url对应链接

**except** Exception **as** e**:**

# If the view raised an exception, run it through exception

# middleware, and if the exception middleware returns a

# response, use that. Otherwise, reraise the exception.

**for** middleware\_method **in** self**.**\_exception\_middleware**:**

response **=** middleware\_method**(**request**,** e**)**

**if** response**:**

**break**

**if** response **is** **None:**

**raise**

# Complain if the view returned None (a common error).

**if** response **is** **None:**

**if** isinstance**(**callback**,** types**.**FunctionType**):** # FBV

view\_name **=** callback**.**\_\_name\_\_

**else:** # CBV

view\_name **=** callback**.**\_\_class\_\_**.**\_\_name\_\_ **+** '.\_\_call\_\_'

**raise** ValueError**(**"The view %s.%s didn't return an HttpResponse object. It returned None instead."

**%** **(**callback**.**\_\_module\_\_**,** view\_name**))**

# If the response supports deferred rendering, apply template

# response middleware and then render the response

**if** hasattr**(**response**,** 'render'**)** **and** callable**(**response**.**render**):**

**for** middleware\_method **in** self**.**\_template\_response\_middleware**:**

response **=** middleware\_method**(**request**,** response**)**

# Complain if the template response middleware returned None (a common error).

**if** response **is** **None:**

**raise** ValueError**(**

"%s.process\_template\_response didn't return an "

"HttpResponse object. It returned None instead."

**%** **(**middleware\_method**.**\_\_self\_\_**.**\_\_class\_\_**.**\_\_name\_\_**))**

response **=** response**.**render**()**

**except** http**.**Http404 **as** e**:**

logger**.**warning**(**'Not Found: %s'**,** request**.**path**,**

extra**={**

'status\_code'**:** 404**,**

'request'**:** request

**})**

**if** settings**.**DEBUG**:**

response **=** debug**.**technical\_404\_response**(**request**,** e**)**

**else:**

response **=** self**.**get\_exception\_response**(**request**,** resolver**,** 404**)**

**except** PermissionDenied**:**

logger**.**warning**(**

'Forbidden (Permission denied): %s'**,** request**.**path**,**

extra**={**

'status\_code'**:** 403**,**

'request'**:** request

**})**

response **=** self**.**get\_exception\_response**(**request**,** resolver**,** 403**)**

**except** MultiPartParserError**:**

logger**.**warning**(**

'Bad request (Unable to parse request body): %s'**,** request**.**path**,**

extra**={**

'status\_code'**:** 400**,**

'request'**:** request

**})**

response **=** self**.**get\_exception\_response**(**request**,** resolver**,** 400**)**

**except** SuspiciousOperation **as** e**:**

# The request logger receives events for any problematic request

# The security logger receives events for all SuspiciousOperations

security\_logger **=** logging**.**getLogger**(**'django.security.%s' **%**

e**.**\_\_class\_\_**.**\_\_name\_\_**)**

security\_logger**.**error**(**

force\_text**(**e**),**

extra**={**

'status\_code'**:** 400**,**

'request'**:** request

**})**

**if** settings**.**DEBUG**:**

**return** debug**.**technical\_500\_response**(**request**,** **\***sys**.**exc\_info**(),** status\_code**=**400**)**

response **=** self**.**get\_exception\_response**(**request**,** resolver**,** 400**)**

**except** SystemExit**:**

# Allow sys.exit() to actually exit. See tickets #1023 and #4701

**raise**

**except:** # Handle everything else.

# Get the exception info now, in case another exception is thrown later.

signals**.**got\_request\_exception**.**send**(**sender**=**self**.**\_\_class\_\_**,** request**=**request**)**

response **=** self**.**handle\_uncaught\_exception**(**request**,** resolver**,** sys**.**exc\_info**())**

**try:**

# Apply response middleware, regardless of the response

**for** middleware\_method **in** self**.**\_response\_middleware**:**

response **=** middleware\_method**(**request**,** response**)**

# Complain if the response middleware returned None (a common error).

**if** response **is** **None:**

**raise** ValueError**(**

"%s.process\_response didn't return an "

"HttpResponse object. It returned None instead."

**%** **(**middleware\_method**.**\_\_self\_\_**.**\_\_class\_\_**.**\_\_name\_\_**))**

response **=** self**.**apply\_response\_fixes**(**request**,** response**)**

**except:** # Any exception should be gathered and handled

signals**.**got\_request\_exception**.**send**(**sender**=**self**.**\_\_class\_\_**,** request**=**request**)**

response **=** self**.**handle\_uncaught\_exception**(**request**,** resolver**,** sys**.**exc\_info**())**

response**.**\_closable\_objects**.**append**(**request**)**

**return** response

#### render

render(request, *"home.html"*, context)

*Shortcuts.py (site-packages\django)*

render -> Engine.render\_to\_string -> Template.render -> Context. bind\_template

### 超级用户

修改command，我只是调整了create\_user的参数

### 参考文档

<https://docs.djangoproject.com/en/1.5/ref/settings/#auth-profile-module>

<https://docs.djangoproject.com/en/dev/topics/auth/customizing/#substituting-a-custom-user-model>

[Django中扩展User模型](http://blog.csdn.net/watsy/article/details/15506351)

[非profile方式扩展Django User Model](http://onlypython.group.iteye.com/group/wiki/1519-expansion-django-user-model-by-non-profile-way)

[django user 权限](http://www.cnblogs.com/smallcoderhujin/p/3193103.html)

[django 注册、登录及第三方接口程序(2)：扩展User表](http://www.tuicool.com/articles/aMBJNjj)

[扩展django的User的部分方法](http://www.cnblogs.com/captain-cp/p/3590213.html)

[django admin框架使用系列之三:扩展user model](http://www.yihaomen.com/article/python/334.htm)

[Django扩展user表并使用email登陆](http://www.python88.com/topic/395/)

[Django 扩展 User model 的字段有什么好的方法？](https://segmentfault.com/q/1010000000610904)

[django使用email进行身份验证](http://guangboo.org/2013/03/27/authentication-using-email-in-django)

[自定义的用户认证如何登录使用Django自带的后台管理?](https://www.oschina.net/question/2341837_235437)

[How to log a user in](https://docs.djangoproject.com/en/dev/topics/auth/default/#how-to-log-a-user-in)

<https://docs.djangoproject.com/en/dev/topics/db/models/#proxy-models>

## 评论

django.contrib.comments在django1.8已删除

[ImportError: No module named comments django](http://stackoverflow.com/questions/32986302/importerror-no-module-named-comments-django)

Django comments are deprecated in latest version. You can either install older django version (such as 1.5). Or the better solution is to install comments from external repository like so:

pip install django-contrib-comments

and then change all imports like this:

**from** django**.**contrib**.**comments**.**models **import** Comment # old

**from** django\_comments**.**models **import** Comment # new

Django1.9版本使用django\_comments而非“django.contrib.comments”，直接pip install django-contrib-comments即可。

可选的库包括

* mptt
* threadedcomments

最后选用mptt

生成的数据库效果图





**object\_pk**

Mptt第一级的object\_pk是crowdfunding，但是子评论的object\_pk是父评论

注: 9是手动改的level变成1，变成8的子，用于测试目的。所以object\_id=2，因为创建是level为0

评论显示的树形结构，每一个comments都是一个Node

|  |  |  |  |
| --- | --- | --- | --- |
| 8 |  |  |  |
|  | 9 |  |  |
|  |  | 12 |  |
|  |  | 13 |  |
|  |  |  | 20 |
|  | 11 |  |  |
|  | 14 |  |  |
| 10 |  |  |  |
|  | 15 |  |  |
|  | 16 |  |  |
|  | 17 |  |  |
|  | 19 |  |  |
| 18 |  |  |  |

html结构如下



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | pypi | github | Document |  |
| django-mptt |  | [django-mptt](https://github.com/django-mptt/django-mptt/) | [readthedocs](http://django-mptt.readthedocs.io/en/latest/tutorial.html#getting-started) | [Django mptt介绍以及使用](http://qinxuye.me/article/introduction-and-usage-of-django-mptt/)  mfitzp.io   * [Django AJAX threaded-comments using only jQuery](http://mfitzp.io/article/django-ajax-threaded-comments-using-only-jquery/) * [Django Threaded Comments with mptt](http://mfitzp.io/article/django-threaded-comments/" \o "Django Threaded Comments) * [Syntax highlighting with Django and Markdown](http://mfitzp.io/article/syntax-highlighting-with-django-and-markdown/) |
| django-threadedcomments | [django-threadedcomments](https://pypi.python.org/pypi/django-threadedcomments) | [django-threadedcomments](https://github.com/HonzaKral/django-threadedcomments) |  | [Eric Florenzano’s Blog - Category: django-threadedcomments](http://eflorenzano.com/blog/categories/django-threadedcomments/) |
| django-comments-threaded |  | [django-comments-threaded](https://github.com/marazmiki/django-comments-threaded) |  |  |
| django-contrib-comments / django\_comments |  | [readthedocs](http://django-comments-xtd.readthedocs.io/en/latest/) |  | [Django 自带comment评论库使用](http://www.cnblogs.com/weiming-cheng/p/5344309.html)  [Django个人博客开发（五）](https://www.dannysite.com/blog/16/) |
| djangocms-comments | [djangocms-comments](https://pypi.python.org/pypi/djangocms-comments/0.2.2) |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Command** | **Example** | **Comments** |
| Fetching comment counts: | get\_comment\_count | {% get\_comment\_count for [object] as [varname] %}  {% get\_comment\_count for [object] as [varname] root\_only %}  {% get\_comment\_count for [app].[model] [id] as [varname] %}  {% get\_comment\_count for [app].[model] [id] as [varname] root\_only %} |  |
| Fetching the comments list: | get\_comment\_list | {% get\_comment\_list for [object] as [varname] %}  {% get\_comment\_list for [object] as [varname] flat %}  {% get\_comment\_list for [object] as [varname] root\_only %} | Gets the list of comments for the given params and populates the template  context with a variable containing that value, whose name is defined by the  'as' clause. |
| Rendering the comments list: | render\_comment\_list | {% render\_comment\_list for [object] %}  {% render\_comment\_list for [object] root\_only %}  {% render\_comment\_list for [app].[model] [id] %}  {% render\_comment\_list for [app].[model] [id] root\_only %} | Render the comment list (as returned by ``{% get\_comment\_list %}``)  through the ``comments/list.html`` template |
| Fetching the comment form: | get\_comment\_form | {% get\_comment\_form for [object] as [varname] %}  {% get\_comment\_form for [object] as [varname] with [parent\_id] %}  {% get\_comment\_form for [app].[model] [id] as [varname] %}  {% get\_comment\_form for [app].[model] [id] as [varname] with [parent\_id] %} | Get a (new) form object to post a new comment. |
| Rendering the comment form: | render\_comment\_form | {% render\_comment\_form for [object] %}  {% render\_comment\_form for [object] with [parent\_id] %}  {% render\_comment\_form for [app].[model] [id] %}  {% render\_comment\_form for [app].[model] [id] with [parent\_id] %} | Render the comment form (as returned by ``{% render\_comment\_form %}``) through  the ``**comments/form.html**`` template. |
| Rendering the whole tree: |  | {% for comment in comment\_list|fill\_tree|annotate\_tree %}  {% ifchanged comment.parent\_id %}{% else %}</li>{% endifchanged %}  {% if not comment.open and not comment.close %}</li>{% endif %}  {% if comment.open %}<ul>{% endif %}  <li id="c{{ comment.id }}">  ...  {% for close in comment.close %}</li></ul>{% endfor %}  {% endfor %} | The fill\_tree filter is required for pagination, it ensures that the parents of the first comment are included as well.  The annotate\_tree filter adds the open and close properties to the comment. |
|  |  |  |  |

### Requirement

django-contrib-comments / django\_comments

django-mptt 支持threaded

在setting里添加APP，comments是自定义的app

INSTALLED\_APPS = (

'django\_comments',

# 'django.contrib.comments',

'mptt',

'comments',

)

同时，在settings.py添加

COMMENTS\_APP **=** 'comments'

创建MPTTComments，同时从MPTTModel和Comment继承，并添加新的域parent

**from** django.contrib.comments.models **import** Comment

**from** mptt.models **import** MPTTModel, TreeForeignKey

**class** **MPTTComment**(MPTTModel, Comment):

""" Threaded comments - Add support for the parent comment store and MPTT traversal"""

*# a link to comment that is being replied, if one exists*

parent **=** TreeForeignKey('self', null**=**True, blank**=**True, related\_name**=**'children')

**class** **MPTTMeta**:

*# comments on one level will be ordered by date of creation*

order\_insertion\_by**=**['submit\_date']

**class** **Meta**:

ordering**=**['tree\_id','lft']

同样的，我们也要继承CommentForm，添加新的域parent

**from** django **import** forms

**from** django.contrib.admin **import** widgets

**from** django.contrib.comments.forms **import** CommentForm

**from** models **import** MPTTComment

**class** **MPTTCommentForm**(CommentForm):

parent **=** forms**.**ModelChoiceField(queryset**=**MPTTComment**.**objects**.**all(), required**=**False, widget**=**forms**.**HiddenInput)

**def** **get\_comment\_model**(self):

*# Use our custom comment model instead of the built-in one.*

**return** MPTTComment

**def** **get\_comment\_create\_data**(self):

*# Use the data of the superclass, and add in the parent field field*

data **=** super(MPTTCommentForm, self)**.**get\_comment\_create\_data()

data['parent'] **=** self**.**cleaned\_data['parent']

**return** data

Django comments extensions work on a simple principle: you tell the comments app about what you’ve changed through the use of some simple hooks. Django then uses these hooks to lookup your altered models and forms and overrides the default set. So now that we’ve created our modified forms and models, we need to point Django at them.

在\_\_init\_\_.py文件中添加钩子功能，指定model和form

**from** models **import** MPTTComment

**from** forms **import** MPTTCommentForm

**def** **get\_model**():

**return** MPTTComment

**def** **get\_form**():

**return** MPTTCommentForm

接下来创建一个处理评论的模板，这个模板最好是一个独立的文件comments\_tree.html，可以在其他文件中进行调用

{% with <yourmodel> as object %}{% include "comments\_tree.html" %}{% endwith %}

这样模板里面可以固定要object作为对象名

***{%*** **load** comments ***%}***

***{%*** **load** mptt\_tags ***%}***

***{%*** **get\_comment\_list** for object **as** comments ***%}***

***{%*** **if** comments ***%}***

***{%*** **recursetree** comments ***%}***

<a name="c***{{*** node.id ***}}***"></a>

***{{*** node.comment ***}}***

***{{*** node.user ***}}***

***{{*** node.submit\_date**|timesince** ***}}*** ago

<a href="***{{*** object.get\_absolute\_url ***}}***#c***{{*** node.id ***}}***">#</a>

***{%*** **render\_comment\_form** for object ***%}***

*{# recursion! children of a given comment #}*

***{%*** **if** **not** node.is\_leaf\_node ***%}***

***<ul class="children">***

***{{*** children ***}}***

</ul>

***{%*** **endif** ***%}***

***{%*** **endrecursetree** ***%}***

***{%*** **endif** ***%}***

***{%*** **render\_comment\_form** for object ***%}***

***<ul class="children">***可以呈现缩进效果



或者可以使用另外一个命令full\_tree\_for\_model comments.MPTTComment，这个可以获取所有的comments

原来我用上面的方法没搞定，所以用了下面的workaroud，这儿只是介绍这种方法

node.object\_pk == object.id|int2str 可以过滤出根节点，所有的根节点的object都是被依附的对象本身，子评论的objet\_PK变成父评论

找到根节点之后，叶节点只要满足node.level>0就可以了。其他object的node的level随便也大于0，但是它们的父节点永远都不满足条件，所以它们本身永远不会有机会出现

***{%*** **load** comments ***%}***

***{%*** **load** mptt\_tags ***%}***

{% full\_tree\_for\_model comments.MPTTComment as comments %}

***{%*** **if** comments ***%}***

***{%*** **recursetree** comments ***%}***

{% if node.level > 0 or node.object\_pk == object.id|int2str %}

*Comments…*

***{%*** **endif** ***%}***

***{%*** **endrecursetree** ***%}***

***{%*** **endif** ***%}***

***{%*** **render\_comment\_form** for object ***%}***

最后，当回复comments时，需要再comments form里添加parent\_id，添加comments/form.html

***{%*** **load** comments i18n ***%}***

<form action="***{%*** **comment\_form\_target** ***%}***" method="post">***{%*** **csrf\_token** ***%}***

***{{*** form.object\_pk ***}}***

***{{*** form.content\_type ***}}***

***{{*** form.timestamp ***}}***

***{{*** form.security\_hash ***}}***

***{%*** **if** node.id ***%}***

<input type="hidden" name="parent" id="parent\_id" value="***{{*** node.id ***}}***" />

***{%*** **endif** ***%}***

<input type='hidden' name='next' value='{% url "Crowdfunding\_comments" pk=object.id %}'/>

***{{*** form.comment ***}}***

<input type="submit">

</form>

关键点是添加parent\_id. 这个文件可以从django contrib库里拷贝过来再编辑。

Crowdfunding\_comments会返回html，内容在ajax处理, 尝试过用default模板的{{next}}，不工作

**form的render可以参考这个原文件**

### 添加Ajax提交功能

功能都是由模板或者templatetag完成，不需要改后台

首先给form添加id，便于Jquery操作，form和它的wrapper都添加了id

form.html

<div class**=**"comment-form" id**=**"comment-form-{{ node.id }}" {% if node.id %}style**=**"display:none;"{% endif %}>

<form action**=**"{% comment\_form\_target %}" method**=**"post" id**=**"comment-form-{{ node.id }}f">

接下来修改评论输出文件，在**{% recursetree comments %}**下面添加

<div {% if request.REQUEST.c|add:"0" == node.id %}id**=**"newly\_posted\_comment"{% endif %}>

...comment content...

</div>

request.REQUEST.c会寻找query字符串里的c值, add:"0"会把它转为整形值、

这个效果是给post的评论添加id，便于找到ajax返回的html

添加**\_ajax\_comments.html**用于处理ajax请求

{% load comments %}

<script type="text/javascript" charset="utf-8">

(function( $ ){

$.fn.bindPostCommentHandler = function() {

// We get passed a list of forms; iterate and get a unique id for each

// attach a submit trigger to handle saving and returning

this.each(function() {

$(this).submit(function() {

commentform = this;

commentwrap = $(this).parent();

$.ajax({

type: "POST",

data: $(commentform).serialize(),

url: "{% comment\_form\_target %}",

cache: false,

dataType: "html",

success: function(html, textStatus) {

// Extract the form from the returned html

postedcomment = $(html).find('#newly\_posted\_comment');

$(commentform).replaceWith(postedcomment.html());

$(commentwrap).hide();

$(commentwrap).slideDown(600);

$(commentwrap).find('.comment-form form').bindPostCommentHandler();

},

error: function (XMLHttpRequest, textStatus, errorThrown) {

$(commentform).replaceWith('Your comment was unable to be posted at this time. We apologise for the inconvenience.');

}

});

return false;

});

}); //each

};

})( jQuery );

$(function() {

$('.comment-form form').bindPostCommentHandler();

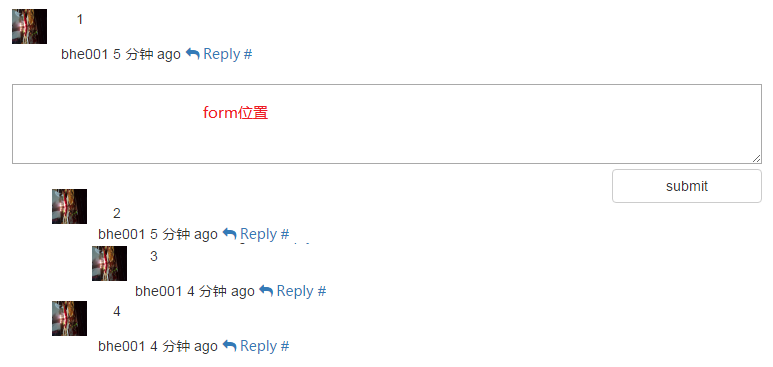
});

</script>

Success的处理中，根据#newly\_posted\_comment从返回的html中找到当前的评论，并将它replace到form中去。

这种方法效率比较高，但是问题是座位嵌套显示时会有问题，因为form的位置并不是新的评论要插入的位置

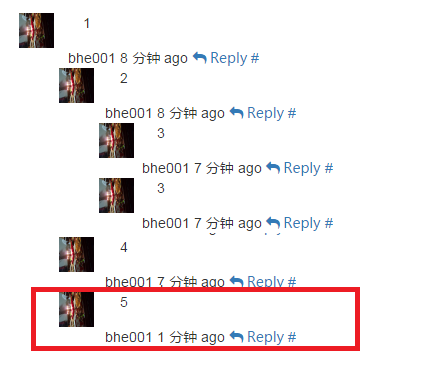
Form位置



实际插入位置



期望插入位置



之前我也实现过一个全部代替的方式，如下

#cmt\_show是整个评论的wrapper，在comments\_tree.html里

postedcomment = $(html).find('#cmt\_show');

posted\_comment\_html=postedcomment.html();

console.log(posted\_comment\_html);

posted\_comment\_wrap\_html="<div id='cmt\_show'>"+posted\_comment\_html+"</div>";

$('#cmt\_show').replaceWith(posted\_comment\_wrap\_html);

$('.comment-form form').bindPostCommentHandler();

### 添加level0 form

自定义 comments/comments\_form.html,也可以直接用django\_comments/templates/comments/form.html，这也是**render\_comment\_form**的默认的template

{% load comments i18n %}

<form action="{% comment\_form\_target %}" method="post">{% csrf\_token %}

{% if next %}

<div><input type="hidden" name="next" value="{{ next }}"/></div>{% endif %}

{% for field in form %}

{% if field.is\_hidden %}

<div>{{ field }}</div>

{% else %}

{% if field.errors %}{{ field.errors }}{% endif %}

<p

{% if field.errors %} class="error"{% endif %}

{% ifequal field.name "honeypot" %} style="display:none;"{% endifequal %}>

{{ field.label\_tag }} {{ field }}

</p>

{% endif %}

{% endfor %}

<p class="submit">

<input type="submit" name="post" class="submit-post" value="{% trans "Post" %}"/>

<input type="submit" name="preview" class="submit-preview" value="{% trans "Preview" %}"/>

</p>

</form>

### 遗留问题

* Post之后刷新页面会多次自动提交
* Post comments只有Chrome能工作，其他的都不行
* 嵌套的顺序是反的，应该是父在最外面
* 只有登录情况下才能post comments

[django中的多级评论](http://www.cnblogs.com/ccorz/p/5985534.html)

[Django中使用jQuery做评论的无刷新提交](http://www.cnblogs.com/restran/archive/2012/04/26/2469839.html)

[Django使用Ajax实现页面无刷新评论回复功能](http://www.cnblogs.com/mfc-itblog/p/5188900.html)

[分层评论的实现思路,thread comments 实现思路,数据库存树状结构](http://www.yihaomen.com/article/python/255.htm)

社会化评论

[django-blog-zinnia集成多说评论框](http://bookshadow.com/weblog/2014/06/04/django-blog-zinnia-add-duoshuo/)

[Disqus\_百度百科](https://www.baidu.com/link?url=U33F7R5Dlgc7qtGN0njPDD9_gO-Dzn3xf1zgeVeIR4S4wLpr_pt4N03camgNEW4l50SFSUEaHZSTIfxcr6VxD_&wd=&eqid=d9d7b7a80003b0790000000458440bd3)

[国内社会化评论(类DISQUS)插件哪个好?](https://www.baidu.com/link?url=VPTczqkq7L-HDLvGHU7UfJ4PTkb8CNzKMlJitI7kyskHi9TXfI9npoopHK06W4Cy&wd=&eqid=e510e6f10003aa960000000458440c69)

畅言 <http://changyan.kuaizhan.com/>

### sblog

[django 简易博客开发 4 comments库使用及ajax支持](http://www.cnblogs.com/cacique/archive/2012/10/03/2710803.html)

[sblog](https://github.com/gusibi/sblog)怎么做comments的ajax更新的

*templates/simpleblog/blog\_add\_comments.html*

<div id="comment\_form">

<form class="form-horizontal" action="{% comment\_form\_target %}" method="post">

<fieldset>

<div class="form-actions">

<input class="btn btn-info" type="submit" name="submit" value="Post">

{# <input class="btn btn-info" type="submit" name="preview" value="Preview"> #}

<input type='hidden' name='next' value='/simpleblog/blog/{{ blog.id }}/commentshow/'/>

</div>

</fieldset>

</form>

</div>

*simpleblog/urls.py*

url**(**r'^blog/(?P<id>\d+)/commentshow/$'**,** 'blog\_show\_comment'**,** name**=**'showcomment'**),**

*simpleblog/views.py*

**def** blog\_show\_comment**(**request**,** id**=**''**):**

blog **=** Blog**.**objects**.**get**(**id**=**id**)**

**return** render\_to\_response**(**'blog\_comments\_show.html'**,** **{**"blog"**:** blog**})**

*templates/simpleblog/blog\_comments\_show.html*

{% load comments %}

{% load gravatar %}

{% load markup %}

<article id="cmt">

{% get\_comment\_count for blog as comment\_count %}

<h4 class="muted">{{ comment\_count }} Comments</h4>

<hr class="soften">

{% get\_comment\_list for blog as blog\_com %}

{% for comment in blog\_com %}

<div class="container-fluid none-padding">

<div class="row-fluid">

<div class="span1">

<img class="gravatar" src="{% gravatar\_url comment.user\_email %}">

</div>

<div class="span10">

<p class="muted"><small>{{ comment.user }}</small><small>{{ comment.submit\_date|date:"F,j,Y" }}</small></p>

{{ comment.comment|markdown:'safe,codehilite' }}

</div>

</div>

</div>

<hr class="soften">

{% endfor %}

</article>

*templates/simpleblog/blog\_show.html*

<script type="text/javascript" charset="utf-8">

function bindPostCommentHandler() {

$('#comment\_form form input.submit-preview').remove();

$('#comment\_form form').submit(function() {

$.ajax({

type: "POST",

data: $('#comment\_form form').serialize(),

url: "{% comment\_form\_target %}",

cache: false,

dataType: "html",

success: function(html, textStatus) {

$('#cmt').replaceWith(html);

$('#comment\_form form')[0].reset();

},

error: function (XMLHttpRequest, textStatus, errorThrown) {

$('#comment\_form form').replaceWith('Your comment was unable to be posted at this time. We apologise for the inconvenience.');

}

});

return false;

});

}

$(document).ready(function() {

bindPostCommentHandler();

});

</script>

**def** post\_comment**(**request**,** next**=None,** using**=None):**

我的方法

1. Form初始化为hide，点击回复显示
2. 整个comments替换
3. 替换后 整体ajax要重新加载

网上方法

1. Form动态添加，怎么解决node问题？
2. 只替换部分comment，form改成comment，form的位置可能要改
3. 只需要加载新增加部分的ajax

### Issue

Reverse for 'comments-post-comment' with arguments '()' and keyword arguments '{}' not found. 1 pattern(s) tried: ['crowdfundings/comments$post/$']

url**(**r'^comments$'**,** include**(**'django\_comments.urls'**)),**

url**(**r'^comments/'**,** include**(**'django\_comments.urls'**)),**

### CSS

[HTML 5 <article> 标签](http://www.w3school.com.cn/html5/html5_article.asp)

[Bootstrap Well](http://www.runoob.com/bootstrap/bootstrap-wells.html)

### Div

div并列及全屏

[CSS技巧！像table一样布局div](http://www.cnblogs.com/hnyei/archive/2011/09/19/2181442.html)

### JQuery

[jquery after append appendTo三个函数的区别](http://blog.csdn.net/ljyabc1028/article/details/7341665)

[jQuery选择器总结](http://www.cnblogs.com/onlys/articles/jQuery.html)

[jQuery 参考手册 - 遍历](http://www.w3school.com.cn/jquery/jquery_ref_traversing.asp)

[jquery选择器 之 获取父级元素、同级元素、子元素](http://blog.163.com/zhaoyanping_1125/blog/static/2013291532011112961637321/)

### templatetag

[django中自定义标签和过滤器](http://www.cnblogs.com/MnCu8261/p/5934203.html)

## JQuery

033 Shopping cart icon

088 Cart Count in Navbar

## Social Share

### 微信分享

[微信WeixinJSBridge API](http://www.cnblogs.com/txw1958/p/WeixinJSBridge-api.html)-方倍工作室

[微信公众平台开发(90) 分享到朋友圈和发送给好友](http://www.cnblogs.com/txw1958/p/MicroMessenger-browser.html)

[微信公众平台开发(107) 分享到朋友圈和发送给好友](http://www.cnblogs.com/txw1958/p/weixin-js-sharetimeline.html)

[Python微信-- 分享接口（分享到朋友圈、朋友、空间）](http://www.cnblogs.com/wasayezi/p/5275691.html)

[微信JS-SDK 分享到朋友圈 分享给朋友 分享到QQ 拍照或从手机相册中选图 识别音频并返回识别结果 使用微信内置地图查看位置](http://www.aichengxu.com/view/5250400)

[微信朋友圈分享页面（JS-SDK 1.0）](http://www.aichengxu.com/view/4578913)

[微信js sdk invalid signature签名错误 问题解决](https://my.oschina.net/u/2308739/blog/371414)

<http://jsbin.com/kalenahana/edit?html,js,output>

[微信JS SDK Demo](http://www.cnblogs.com/txw1958/p/weixin-js-sdk-demo.html)

[微信WeixinJSBridge API（屏蔽右上角按钮等）](http://www.2cto.com/weixin/201511/451592.html)

<http://mmlike.sinaapp.com/>

[微信公众号与HTML 5混合模式揭秘4——jssdk调用微信扫一扫](https://segmentfault.com/a/1190000004038089)

[微信公众平台开发(104) 自定义菜单扫一扫、发图片、发地理位置](http://www.cnblogs.com/txw1958/p/weixin-menu-new-type.html)

微信WeixinJSBridge API。在 微信分享页面时，用到的调用 Native 功能的 API。

微信6.0.2取消了分享接口WeixinJSBridge

<http://demo.open.weixin.qq.com/jssdk>

#➊➋➌➍➎➏➐➑➒➓

## 验证码 Captcha

## 工具

<https://www.zhihu.com/question/20009756/answer/14705540>

切图部分用fireworks

如果写单写js，用webstorm比较好一点。

vim + firebug + firebug autocomplete + fiddler + 虚拟机（或者IEtester）vim: 编辑器，快速、轻量级、无穷的定制性，有了vim，你会很喜欢coding的感觉，知道什么是“十指如飞”；firebug+firebug autocomplete：这是firefox上调试网页的必备工具，当然在不同浏览器上有不同的相似工具，例如safari和chrome下的开发者工具，opera下的dragonfly；另外，如果熟悉vim的话，推荐在firefox下实用 pentadactyl 插件；fiddler：用于页面调试，可以用本地的js文件替换某个服务器上的js文件。虚拟机+IETester：用于调试ie6~~

html+css 还是用DW , notepad++

JS用eclipse

调试 Chrome 开发人员工具

## 短信

[短信验证码平台哪家速度快又稳定，推荐下？](https://www.zhihu.com/question/25009882)

django-sendsms==0.2.3

<http://www.yunpian.com/>

## 兼容py3

from \_\_future\_\_ import unicode\_literals

## 临时

class User(AbstractUser):

class Meta(AbstractUser.Meta):

swappable = 'AUTH\_USER\_MODEL'

**swappable** is an "intentionally undocumented" feature which is currently under development / in-test. It's used to handle "I have a base abstract model which has some foreign-key relationships." Slightly more detail is available from [Django's ticketing system](https://code.djangoproject.com/ticket/19103) and [github](https://github.com/wq/django-swappable-models). Because it's a "stealth alpha" feature, it's not guaranteed to work (for anything other than User), and understanding the detailed operation will likely require diving into source code. It works with **AUTH\_USER\_MODEL** because the User model and **swappable** flag were developed together, specifically for each other.

user = UserModel.\_default\_manager.get\_by\_natural\_key(username)

def get\_by\_natural\_key(self, username):

return self.get(\*\*{self.model.USERNAME\_FIELD: username})

\*\*解决了get后面名字不确定的问题，也是之前困扰voith的一个问题

UserModel.\_default\_manager

MyUserManager: personalcenter.MyUser.objects

UserModel.\_base\_manager

Manager: personalcenter.MyUser.\_base\_manager

执行get时会做validation，即使值一致但是validation不过也会返回失败

比如电话号码+86135000000000，之前validation时没有+86，添加的值也不会带+86.

Validation规则变化之后，即使搜索135000000000（值在数据库存在）也会返回