Django 前端模板
及 crispy form 美化等

by 捉不住的鼬鼠

2018-10-22

目　录

1 开始

这是紧接前面一篇 Django 用户登录注册等操作的文档，主要计划是学习 crispy form 等第三方库的使用，为前端进行美化。一开始接触的一些 CSS 和 bootstrap 用得不熟练，所以还是借助这些库和 django 模板来生成前端页面。像 django admin 和 xadmin 这样的后台管理，功能非常完善，页面要做很多改变所以还不如自己设计。django bootstrap 库感觉文档不多，crispy form 以前页有些接触，还能自己配置不同前端框架，所以这里应该会主要用 crispy form 结合模板对 django 前端进行设计美化。

不加修饰的前端页面太粗糙了，差不多就是这样的：

这里有一个网站，专门针对 django 工具包进行分析，如 Auth 模块及前端框架等：
https://djangopackages.org/grids/g/twitter-bootstrap/

比如 django 中 bootstrap 使用情况：

PACKAGE	DJANGO-CRISPY-FORMS	DJANGO-BOOTSTRAP3	DJANGO-BOOTSTRAP-TOOLKIT
Description	The best way to have DRY Django forms. The app provides a tag and filter that lets you quickly render ...	Bootstrap 3 integration with Django.	Bootstrap support for Django projects
Category	App	App	App
# Using This	54△	26△	5△
Python 3?	✔	✔	✖
Development Status	Production/Stable	Production/Stable	Production/Stable
Last updated	Oct. 4, 2018, 6:03 a.m.	Aug. 30, 2018, 4:15 a.m.	July 6, 2018, 3:24 a.m.

也可以看出 crispy forms 的使用很多，django-bootstrap 则紧跟其后。

2 第一个例子

2.1 安装

参考:
https://simpleisbetterthancomplex.com/tutorial/2018/08/13/how-to-use-bootstrap-4-forms-with-django.html
使用 pip 安装 crispy form:
pip install django-crispy-forms
然后 INSTALLED_APPS 中添加 crispy_forms，然后再添加一个单独的变量:
CRISPY_TEMPLATE_PACK = 'bootstrap4'

2.2 前端页面

2.2.1 base.html

这里采用 CDN 方式加载 bootstrap4，没添加 js 因为用不到。base.html 如下:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
    <title>Django People</title>
  </head>
  <body>
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-8">
          <h1 class="mt-2">Django People</h1>
          <hr class="mt-0 mb-4">
          {% block content %}
          {% endblock %}
        </div>
      </div>
```

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

initial-scale=1 指定初始比例显示，shrink-to-fit=no 阻止页面适配视口过程中缩放。

接着 justify-content 指定横向对齐方式，mt 和 mb 类以前也不知道，这是 bootstrap 中新添加的，就是 margin top 和 bottom，值为 0.5rem。

2.2.2 person_form.html

如下：

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

{% block content %}
  <form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit" class="btn btn-success">Save person</button>
  </form>
{% endblock %}
```

2.3 models.py

```
from django.db import models


class Person(models.Model):
    name = models.CharField(max_length=130)
    email = models.EmailField(blank=True)
    job_title = models.CharField(max_length=30, blank=True)
    bio = models.TextField(blank=True)
```

2.4 views.py

```
from django.views.generic import CreateView
from .models import Person


class PersonCreateView(CreateView):
    model = Person
    template_name = 'person_form.html'
    fields = ('name', 'email', 'job_title', 'bio')
```

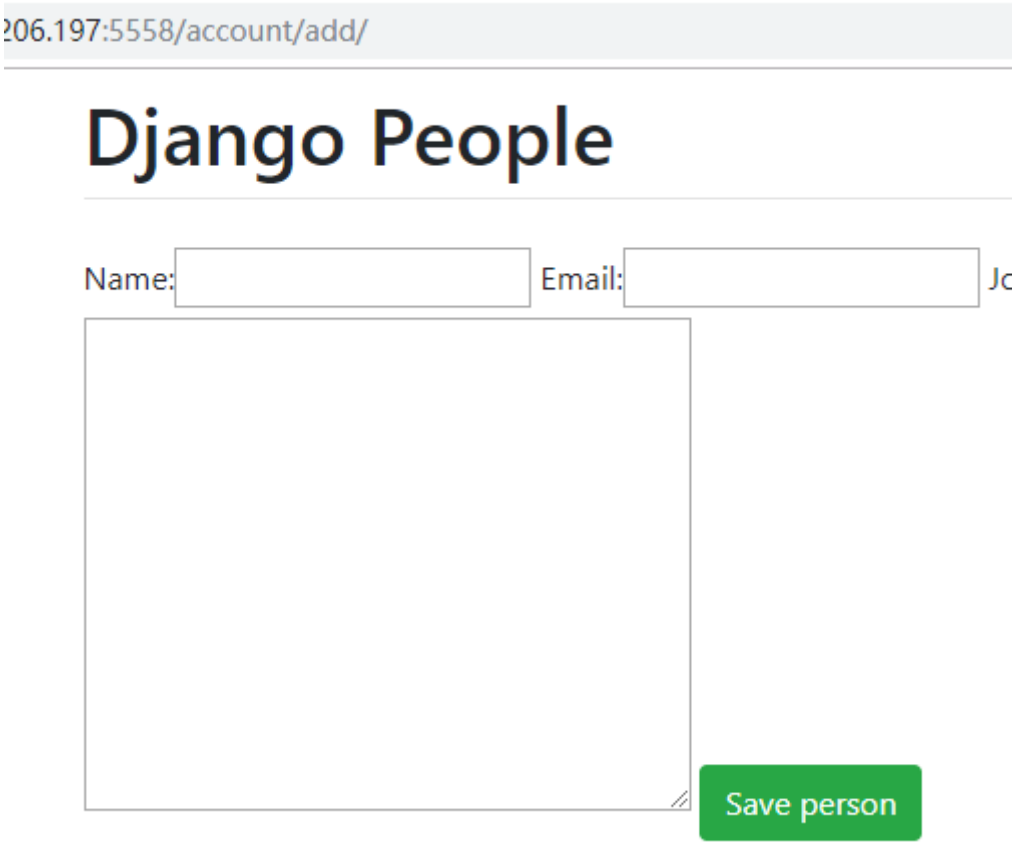
2.5 url.py

path('account/add/', views.PersonCreateView.as_view()),

2.6 效果

对比下这些代码，发现用了 CreateView 这个类是之前没见过的。相对于以前的做法，如 url 中配置，views 里面 render 返回，这里用了 as_view 方法。这样只需 views.py 中的简单几行就能对前端请求返回，省去了写 request 和 render 的部分。另外，该部分的 template_name 是我修改的，因为 django 会默认，而我的 html 路径和命名不是默认的。

这样我们就能通过下面的 url 访问该表格了：



add 是默认路由，添加一个记录的时候用。这里还没有 bootstrap 效果，但是也比自己写一个 html 什么都不加要好，比如那个绿色的按钮自带了基本样式。

2.7 添加 bootstrap 样式

将 person 页面修改如下：

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

{% load crispy_forms_tags %}
```

```
{% block content %}
  <form method="post" novalidate>
    {% csrf_token %}
    {{ form|crispy }}
    <button type="submit" class="btn btn-success">Save person</button>
  </form>
{% endblock %}
```

重新访问，效果如下：

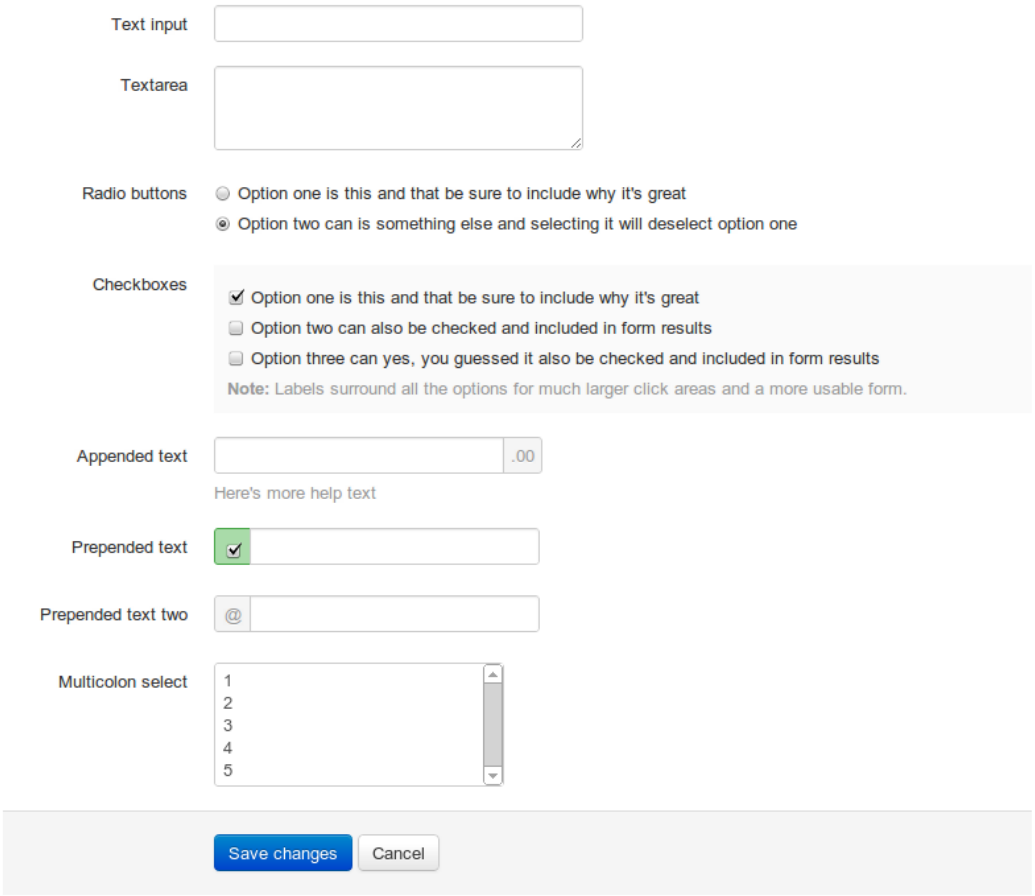
多了圆角效果，换行对齐都很好。

2.8 帮助类及更新

前面的 createview 是创建，同样可以做一些更新操作，然后添加 helper 功能，这部分暂时不看了。

3 第二个例子

crispy form 自己 GitHub 主页就有个例子：
https://github.com/django-crispy-forms/django-crispy-forms
不抄代码了，效果看下：



这里展示的几种控件基本够用了，需要的时候参考抄一下。

4 改造我的项目

上一篇的文档记录没加前端效果，因此很粗糙。这里我接上，做一个较好的用户系统。

4.1 settings.py

为了做得比较标准，设置：
LANGUAGE_CODE = 'zh-hans'

TIME_ZONE = 'Asia/Shanghai'

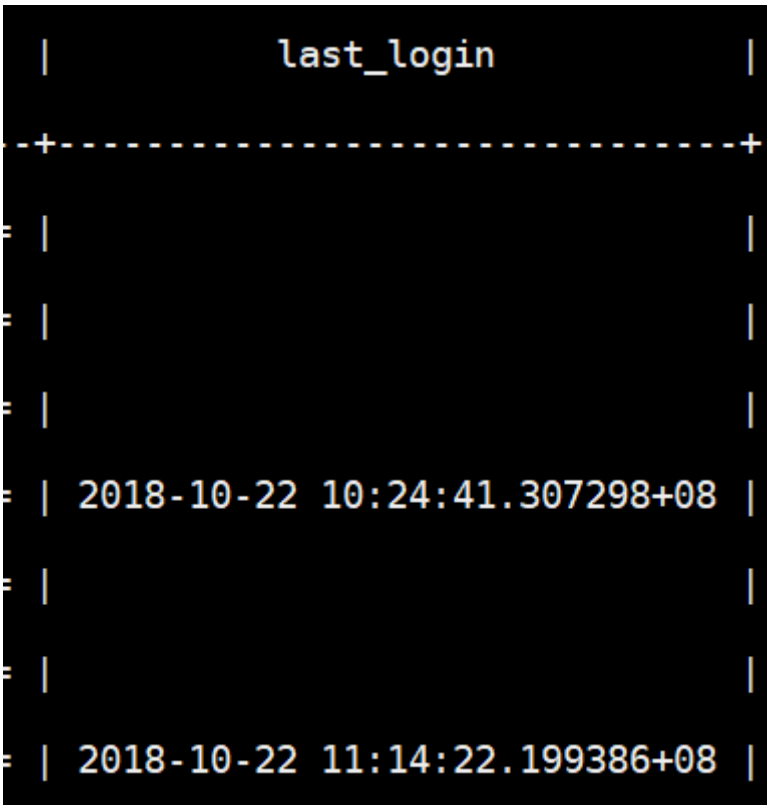
USE_I18N = True

USE_L10N = True

USE_TZ = True
这里，如果不用 zh-hans 而是 zh-cn，则会报错：

No translation files found

选择时区为上海，切设置使用时区为 True。根据先前的经验，这样设置，Django 自带的用户系统记录的是带时区的时间信息（使用 PostgreSQL），如：



这正是我需要的，接着：

```
ALLOWED_HOSTS = ['*']

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crispy_forms',
    'accounts',
]

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR + '/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
```

```
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'platform',
        'USER': 'dddddd',
        'PASSWORD': 'ssssss',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

结合之前学习的例子，这里采用继承 AbstractUser 扩展自己的用户模型的做法。添加：

```
AUTH_USER_MODEL = 'accounts.User'
```

accounts 是自己的 app 的名字，因为 views 文件中要设置 login_required 装饰器，需要 settings.py 中添加：

```
LOGIN_URL = '/accounts/login/'
```

这里是自己定义的跳转登录 url，我使用了绝对路径。

接着，因为要用 crispy form，设置：

```
CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

另外后面还有邮箱激活等，所以需要设置邮箱：

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_SSL = True
EMAIL_HOST = 'smtp.qq.com'
EMAIL_PORT = 465
EMAIL_HOST_USER = '1862@qq.com'
EMAIL_HOST_PASSWORD = 'itdiueeec'
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

另外，django 默认是用户名密码登录，为了用户名加邮箱登录，需要自己实现验证后端，并设置：

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend',
    'accounts.backends.EmailBackend',
)
```

4.2 models.py

目前还未确定用户有多少属性，暂且扩展一个昵称属性：

```
from django.db import models
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    nickname = models.CharField(max_length=50, blank=True)

    class Meta(AbstractUser.Meta):
        pass
```

4.3 base.html

先设计一个基础页面：

```
<!doctype html>
<html lang="zh-cmn-Hans">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <script src="/static/js/jquery-3.3.1.min.js"></script>
    <script src="/static/js/popper-1.14.4.min.js"></script>
    <script src="/static/front/bootstrap-4.1.3/js/bootstrap.min.js"></script>
    <link href="/static/front/bootstrap-4.1.3/css/bootstrap.min.css" rel="stylesheet" type="text/css">
    <link href="/static/css/footer.css" rel="stylesheet" type="text/css">
    <title>{{ page_title }}</title>
</head>

<body>
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
        <a class="navbar-brand" href="#">Huawei UCD</a>

        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" href="#">我的平台</a>
            </li>

            <li class="nav-item">
                <a class="nav-link" href="#">个人中心</a>
```

```
            </li>

            <li class="nav-item">
                <a class="nav-link" href="#">关于</a>
            </li>
        </ul>
    </nav>

    <br>

    <div class="container">
        <div class="row justify-content-center">
            <div class="col-8">
                {% block content %}
                {% endblock %}
            </div>
        </div>
    </div>

    <br>
    <br>
    <br>

    <footer class="section footer-classic context-dark bg-image" style="background: #2d3246;">
        <div class="container">
            <div class="row row-30">
                <div class="col-md-4 col-xl-5">
                    <div class="pr-xl-4">
                        <br>
                        <br>
                        <p>我们是富于创新性的团队,我们致力于设计软件应用于手机和 PC 等平台,推进音乐及视频等方面的应用.</p>
                        <!-- Rights-->
                        <p class="rights"><span>©  </span><span class="copyright-year">2018</span><span> </span><span>WWW</span><span>. </span><span>All Rights Reserved.</span></p>
                    </div>
                </div>

                <div class="col-md-4">
                    <br>
                    <h5>Contacts</h5>
```

```
                        <dl class="contact-list">
                            <dt>地址:</dt>
                            <dd>上海市浦东新区</dd>
                        </dl>

                        <dl class="contact-list">
                            <dt>email:</dt>
                            <dd><a href="mailto:#">ys@gmail.com</a></dd>
                        </dl>

                        <dl class="contact-list">
                            <dt>电话:</dt>
                            <dd><a  href="tel:#">+86  18739</a>  <span> 或 </span>  <a
href="tel:#">+89 12345678910</a>
                            </dd>
                        </dl>
                    </div>

                    <div class="col-md-4 col-xl-3">
                        <br>
                        <h5>Links</h5>

                        <ul class="nav-list">
                            <li><a href="#">关于</a></li>
                            <li><a href="#">项目</a></li>
                            <li><a href="#">博客</a></li>
                            <li><a href="#">联系我们</a></li>
                            <li><a href="#">报价</a></li>
                        </ul>
                    </div>
                </div>
            </div>
        </footer>
</body>

</html>
```

这里遇到的一个问题是 bootstrap4 用了 popper.js，我本来就没用过 bootstrap4，更没用过 popper, django 中我替换掉 cdn 方式改用本地文件配置 bootstrap 的时候出了点问题。bootstrap 的 css 和 jquery.js 放置好, 需要 popper.min.js。这个文件应该取自 dist/umd 目录，而不是 dist 目录下的，参考:

https://stackoverflow.com/questions/45694811/how-to-use-popper-js-with-bootstrap-4-beta

You want to use the dist target specified in the `package.json` file as `main` entry.

In this case, you are looking for the `umd` build (`dist/umd/popper.js`)

What's UMD?

> The UMD pattern typically attempts to offer compatibility with the most popular script loaders of the day (e.g RequireJS amongst others). In many cases it uses AMD as a base, with special-casing added to handle CommonJS compatibility.

This means that an UMD bundle can be loaded via `<script>` tag and get injected inside the global scope (`window`), but also work if required with a CommonJS loader such as RequireJS.

主要就是兼容性的问题吧，都是 js 的有关内容，不深究。

4.4 forms.py

这个之前的文档也有，这里放一个登录部分：

```
class LoginForm(forms.Form):
    username = forms.CharField(required=True,
                               label='用户名/邮箱',
                               error_messages={'required': '请输入用户名/邮箱'},
                               widget=forms.TextInput(attrs={'placeholder': '用户名/邮箱'}),)

    password = forms.CharField(required=True,
                               label='密码',
                               error_messages={'required': '请输入密码'},
                               widget=forms.PasswordInput(attrs={'placeholder': '密码'}))

    def clean(self):
        if not self.is_valid():
            raise forms.ValidationError('用户名和密码为必填项')

        else:
            cleaned_data = super(LoginForm, self).clean()

        return cleaned_data
```

这里和之前的一点不一样的是，username 被修改为了用户名和邮箱，因为一般的做法是可以用用户名也可以用邮箱填写，然后匹配密码登录。因此这里修改了，而且对应 views 中要实现邮箱登录验证功能，因为 django 默认的 authenticate 仅支持用户名和密码组合验证。

4.6 url.py

name 应该设置下，因为可以用 reverse 方法处理 name 得到 url，在后期有修改的时候非常方便。

先预设了几个:

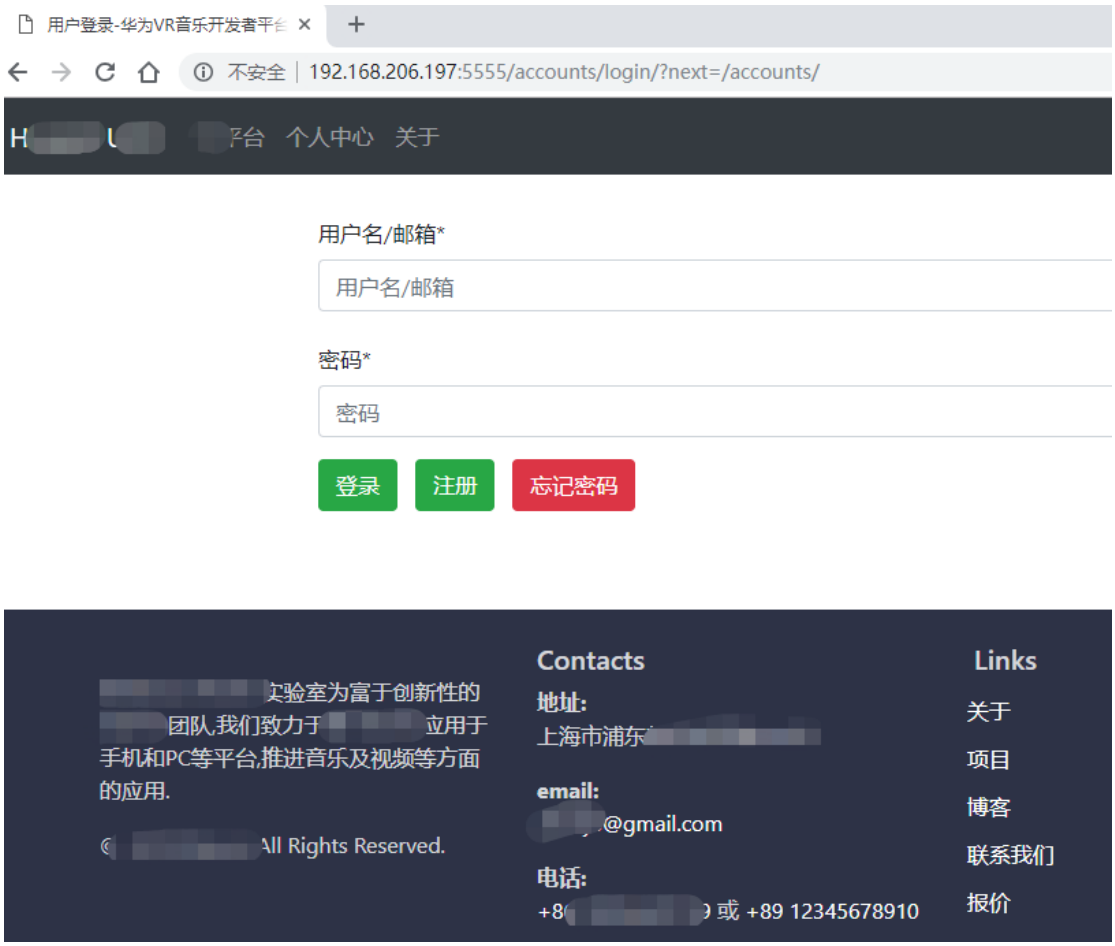
```
urlpatterns = [
    path('accounts/', views.index, name='index'),
    path('accounts/signup/', views.signup, name='signup'),
    re_path('accounts/activate/(?P<token>\w+.[-_\w]*\w+.[-_\w]*\w+)/$',
views.activate_user, name='active_user'),
    path('accounts/login/', views.login, name='login'),
    path('accounts/logout/', views.logout, name='logout'),
    path('accounts/change_pwd/', views.change_pwd, name='change_pwd'),
    path('accounts/reset_pwd/', views.reset_pwd, name='reset_pwd'),
]

urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

4.5 views.py

login 先不管 POST，直接返回:

```
def login(request):
    form = LoginForm()
    context = {'page_title': '用户登录-开发者平台', 'form': form}
    return render(request, 'login.html', context)
```

这里以后还是要修改的，到这里已经可以看看效果了，如下:



这里地址栏出了 next 因为我访问的 index:

```
@login_required
def index(request):
    username = request.session.get('username')

    return render(request, 'index.html', {'page_title': '用户中心-开发者平台', 'username': username})
```

被装饰器直接重定向到了登录界面，也就是上面的。

4.6 其他补充

我已经写过一篇文档，为了减少重复，为了代码不占用太多地方，其他功能和界面就不写了，只会记录一些遇到的问题，作为补充。

4.6.1 注册邮箱审查

以前都没研究过 django 的登录审查，比如:

django 自带的，也就是 contrib/auth 中已经实现了这样的验证功能，这些在 settings 中也有体现，只是以前都未注意：

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

这些检查包括了相似度、长度合法性和其他验证，非常全面。

在实际开发中，我需要用户邮箱，而 django 自带的并不含有邮箱审查，测试即可得知，一个用户名注册了一个邮箱，另外用一个用户名注册，同样使用该邮箱仍然可以成功注册。这是不合理的，所以我需要自己写一个邮箱审查。

参考 django/contrib/auth/forms.py 中的创建用户表单：

```
class UserCreationForm(forms.ModelForm):
    def _post_clean(self):
        super()._post_clean()
        # Validate the password after self.instance is updated with form data
        # by super().
        password = self.cleaned_data.get('password2')
        if password:
            try:
                password_validation.validate_password(password, self.instance)
            except forms.ValidationError as error:
                self.add_error('password2', error)
```

这里主要是重写_post_clean 方法，所以我自己写的如下：

```
from django.core.exceptions import ValidationError
```

```
    def email_validation(email):
        raise ValidationError(
            '测试',
            code='email_test',
        )
```

这里只是为了测试，就是对所有邮箱都抛错，对应的表单变为：

```
class SignUpForm(UserCreationForm):
    email = forms.EmailField(required=True)

    nickname = forms.CharField(required=False,
                               label='昵称',
                               widget=forms.TextInput(attrs={'placeholder': '昵称'}))

    def _post_clean(self):
        super()._post_clean()

        email = self.cleaned_data.get('email')

        if email:
            try:
                email_validation(email)
            except forms.ValidationError as error:
                self.add_error('email', error)

    def clean(self):
        if not self.is_valid():
            raise forms.ValidationError('请按要求填写注册信息')

        else:
            cleaned_data = super(SignUpForm, self).clean()

        return cleaned_data

    class Meta(UserCreationForm.Meta):
        model = User
        fields = ("username", "email")
```

所以提交时检测：

这里用自己写的 email 指定为必填项，同时加入 fields 中表明使用原生的进行验证，也就是格式是否有效等，这样可以添加自己的功能切不失去原有的验证功能。

这里路线已经清晰了，具体验证，如已经被占用等审查条件的代码就不贴了。

4.6.2 修改密码

根据之前的学习经验，注册、登录和注销等都没用问题。现在设计修改密码的部分耽误了一些时间，就是我尝试从 auth/forms 中直接用 AdminPasswordChangeForm 或者 SetPasswordForm，如：

form = SetPasswordForm(user, data=request.POST)

但是这样存在的问题是仅有密码和确认密码两项，无旧密码输入项。

接着用了自带的 auth/views 里面的 password_change，发现是有旧密码输入项的。比较得知使用错了 form，而且 django 建议使用 PasswordChangeView，所以我也就换成这个最新的了，如下：

from django.contrib.auth import views as auth_views

path('accounts/change_pwd/', auth_views.PasswordChangeView.as_view(template_name='change_pwd.html', success_url='/accounts/change_pwd_done/')),

4.6.3 session 有效期

默认的有效期是两周，我想要修改为 30 分钟。网上的教程一般是在 settings 中设置：

SESSION_COOKIE_AGE = 60 * 30 # 30 分钟

SESSION_SAVE_EVERY_REQUEST = True

SESSION_EXPIRE_AT_BROWSER_CLOSE = True # 关闭浏览器，则 COOKIE 失效

这里参考的是：

https://blog.csdn.net/daivon_up/article/details/77771502

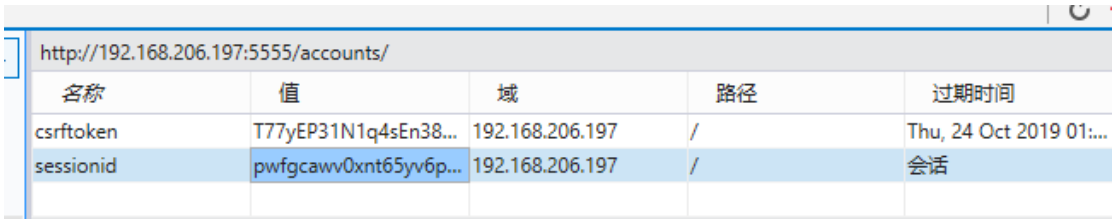
但是实际测试中发现有效期很奇怪：

192.168.206.197	/	1969-12-31T23:59:59.000Z

居然是 1969 年的有效期了，关键是系统状态确实是登录了，这点我有一点不理解。如果过期时间是 1969，那当前登录态肯定是要被踢下来的才对。当前用的是 Chrome，找了下发现有人说这个问题：

https://productforums.google.com/forum/#!topic/webmasters/CycnXQ-z_UM

也就是这个可能不作数，不影响什么。比如换个浏览器，如 Edge，显示正常：



所以以上设置是有效的。

另外，根据 django 文档说的，默认情况下，django 用了 django.contrib.sessions.models.Session 存储 session 信息到数据库，因此我默认的 settings 文件中是没用到 SESSION_ENGINE 变量的，这样同样工作正常。

经过实际测试，以上设置确实可以 30 分钟后自动退出登录状态。

4.6.4 自定义重置密码

自定义密码重置部分我想要用自带的，但是邮箱不存在的时候不提示。其实这样是合理的，减少了被攻击的可能。我还写了博客记录，因为以前都没意识到：

https://blog.csdn.net/u012911347/article/details/83378671

只不过这里根据设计需求，我们需要提醒用户，如果重置密码的时候邮箱不存在。因此我就要改写自带的模块，主要就是 forms 文件：

```
class PasswordResetFormModified(PasswordResetForm):
    def is_valid(self):
        valid = super(PasswordResetForm, self).is_valid()

        if valid:
            email_exist = False

            for _ in self.get_users(self.cleaned_data.get('email')):
                email_exist = True
                break

            if not email_exist:
                self.add_error('email', '该邮箱不存在')
                valid = False

        return valid
```

这里就是重写 is_valid 方法，判断该 email 是否存在，没有的话则用 add_error 添加错误信息。

url 中：

```
path('accounts/reset_pwd/',                    views.PasswordResetViewModified.as_view(),
```

name='reset_pwd'),
path('accounts/reset_pwd_done/', views.reset_pwd_done, name='reset_pwd_done'),
path('accounts/reset_pwd_confirm/<uidb64>/<token>/',

auth_views.PasswordResetConfirmView.as_view(template_name='accounts/password_reset_confirm.html',

success_url='/accounts/reset_pwd_complete/'),
name='reset_pwd_confirm'),
path('accounts/reset_pwd_complete/', views.password_reset_complete, name='password_reset_complete'),

这个涉及比较多，提交时 reset，发送了重置密码邮件时 done，用户在点击链接后到了输入新密码的地方是 confirm，重置成功了是 complete。

reset_pwd 对应我自己写的：

```
class PasswordResetViewModified(PasswordResetView):
    form_class = PasswordResetFormModified
    email_template_name = 'accounts/password_reset_email.html'
    subject_template_name = 'accounts/subject_template_name.txt'
    success_url = '/accounts/reset_pwd_done/'
    template_name = 'accounts/reset_pwd.html'
    title = '密码重置'
```

主要就是这样，其中 password_reset_email.html 如下：

```
您正在申请开发者平台重置密码,注册邮箱为 {{ email }}. 请点击以下链接进行密码重置:
{{ protocol}}://{{ domain }}{% url 'reset_pwd_confirm' uidb64=uid token=token %}
```

这里用的自带的 token 生成器。

5 总结

django 用户模块之前写了文档学习，这里对 crispy form 进行应用获得较好的前端效果。另外也对邮箱部分和重置密码等进行了学习，这都是从新手角度记录的，比较详细。

到这里也算对完成了一个 django 用户系统，而不是像以前那样就会用一个 django admin 或者 xadmin 那样了。这也不是仅仅会数据库增删改查或者写一些用户无关 api，算是真的发挥了 django 的用处，不然仅仅作为一个简单的 web 服务器就浪费了。

以后计划再写一篇 django 文档，目标是学习记录在线修改头像、其他信息修改和复杂页面展示等。