27-用户认证-登录

用户已经能够在我们的网站注册了，注册就是为了登录，接下来我们为用户提供登录功能。和注册不同的是，**Django 已经为我们写好了登录功能的全部代码**，我们不必像之前处理注册流程那样费劲了。只需几分钟的简单配置，就可为用户提供登录功能。接下来就来看看如何使用内置的登录功能。

1、引入内置的 URL 模型

Django 内置的登录、修改密码、找回密码等视图函数对应的 URL 模式位于 django.contrib.auth.urls.py 中，首先在工程的 urls.py 文件里包含这些 URL 模式。打开 blogproject/ 目录下的 urls.py 文件，将 django.contrib.auth.urls.py 包含进来：

【blogproject/urls.py】

```
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^accounts/', include('users.urls')),
    # 将 auth 应用中的 urls 模块包含进来
    url(r'^accounts/', include('django.contrib.auth.urls')),
]
```

这将包含以下的 URL 模式：

- 登录：^accounts/login/$ [name='login']
- 登出：^accounts/logout/$ [name='logout']
- 修改密码：^accounts/password_change/$ [name='password_change']
- 修改密码成功：^accounts/password_change/done/$ [name='password_change_done']
- 重设密码：^accounts/password_reset/$ [name='password_reset']
- 重设密码成功：^accounts/password_reset/done/$ [name='password_reset_done']
- 重设账户：^accounts/reset/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$ [name='password_reset_confirm']
- 重设账号成功^accounts/reset/done/$ [name='password_reset_complete']

最好将 url 地址设为 accounts/，因为默认的登录地址就是 accounts/login。

2、设置模板路径

默认的登录视图函数渲染的是 registration/login.html 模板（注意：这是默认的文件目录地址！），因此需要在 templates/ 目录下新建一个 registration 文件夹，再在 registration/ 目录下新建 login.html 模板文件。此时目录结构为：

```
blogproject/
  manage.py
  blogproject/
    __init__.py
    settings.py
    urls.py
    wsgi.py
  templates/
    users/
      register.html
    registration/
      login.html
```

3、编写登录模板

登录模板的代码和注册模板的代码十分类似：

【templates/registration/login.html】

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
    <title>登录</title>
    <link rel="stylesheet" href="https://unpkg.com/mobi.css/dist/mobi.min.css">
    <style>
        .errorlist {
            color: red;
        }
    </style>
</head>
<body>
<div class="flex-center">
    <div class="container">
        <div class="flex-center">
            <div class="unit-1-2 unit-1-on-mobile">
                <h1><a href="{% url 'blog:index' %}">Django Auth Example</a></h1>
                <h3>登录</h3>
                <form class="form" action="{% url 'login' %}" method="post">
                    {% csrf_token %}
                    {{ form.non_field_errors }}
```

登录的视图函数是已有的，不需要自己写

```
                    {% for field in form %}
                        {{ field.label_tag }}
                        {{ field }}
                        {{ field.errors }}
                        {% if field.help_text %}
                            <p class="help text-small text-
muted">{{ field.help_text|safe }}</p>
                        {% endif %}
                    {% endfor %}
                    <button type="submit" class="btn btn-primary btn-block">登录
</button>
                    <input type="hidden" name="next" value="{{ next }}"/>
                </form>
                <div class="flex-left top-gap text-small">
                    <div class="unit-2-3"><span>没有账号？<a href="{% url
'users:register' %}">立即注册</a></span></div>
                    <div class="unit-1-3 flex-right"><span><a href="{% url
'password_reset' %}">忘记密码？</a></span></div>
                </div>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

为了登录页面的美观引入 mobi.css 提供样式支持，其它代码请忽略，我们只关注表单部分的代码。

```
                <form class="form" action="{% url 'login' %}" method="post">
                    {% csrf_token %}
                    {{ form.non_field_errors }}
                    {% for field in form %}
                        {{ field.label_tag }}
                        {{ field }}
                        {{ field.errors }}
                        {% if field.help_text %}
                            <p class="help text-small text-
muted">{{ field.help_text|safe }}</p>
                        {% endif %}
                    {% endfor %}
                    <button type="submit" class="btn btn-primary btn-block">登录
</button>
                    <input type="hidden" name="next" value="{{ next }}"/>
                </form>
```

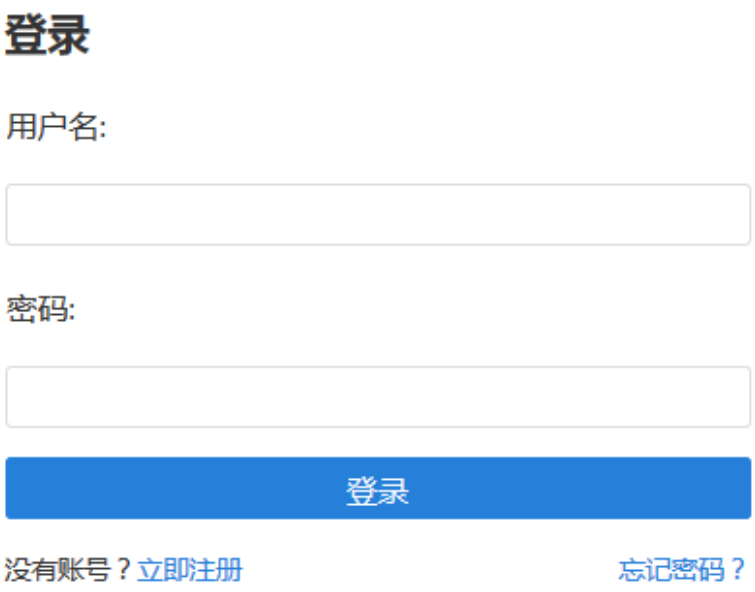
循环表单字段、渲染控件、渲染帮助信息等在注册表单部分已经讲过，登录表单中只引入了一个新的东西：**{{ form.non_field_errors }}，这显示的同样是表单错误，但是显示的表单错误是和具体的某个表单字段无关的。相对 {{ field.errors }}，这个则显示的是具体某个字段的错误。**比如对于字段 username，如果用户输入的 username 不符合要求，比如太长了或者太短了，表单会在 username 下方渲染这个错误。但有些表单错误不和任何具体的字段相关，比如用户输入的用户名和密码无法通过验证，这可能是用户输入的用户名不存在，也可能是用户输入的密码错误，因此这个错误信息将通过 {{ form.non_field_errors }} 渲染。

注意：**你可能觉得用户名不存在错误和 username 字段有关，密码错误和 password 字段有关。但是在现代的用户认证系统中，我们不为用户提供这么详细的信息，只是笼统地告知用户名不存在或者密码错误。这能提高一些用户账户的安全性。**

此外登录表单的 action 属性的值是 **{% url 'login' %}，即 auth 应用下的 `login` 视图函数（在 `django.contrib.auth.urls` 里）对应的 URL**，用户提交的表单数据将提交给这个 URL，Django 调用 `login` 视图函数来处理。

不要忘了加 {% csrf_token %} 模板标签。

现在打开开发服务器，在浏览器输入 http://127.0.0.1:8000/users/login/ ，你将看到一个用户登陆表单。



故意使用一个不存在的账户登录，或者故意输错密码，你将看到表单渲染的非字段相关的错误。

如果用户登录成功，你会发现默认被跳转到了 http://127.0.0.1:8000/accounts/profile/ 页面。由于我们没有写任何视图函数处理这个 URL，所以看到一个 404 错误。

要想把用户跳转回首页，可以在 settings 中做如下设置：

```
LOGOUT_REDIRECT_URL = '/'
LOGIN_REDIRECT_URL = '/'
```

4、如何在模板中判断用户是否已经登录

在模板中判断用户是否已经登录非常简单，使用 {% **if user.is_authenticated** %} 条件判断即可。

【templates/base.html】

```
                            <li class="cl-effect-11"><a
href="contact.html" data-hover="联系">联系</a></li>

                            {% if user.is_authenticated %}

                            <li class="cl-effect-11"><a href="{% url
'logout' %}">Logout</a></li>
```

```
                            {% else %}
                            <li class="cl-effect-11"><a href="{% url 'login' %}">Login</a></li>
                            {% endif %}
```

user.is_authenticated 当用户已经登录时返回 True，否则返回 False。所以已登录的用户将看到登出按钮，否则将看到登录按钮。

你也许奇怪我们在 index 视图中并没有传递 user 模板变量给 index.html，为什么可以在模板中引用 user 呢？这是因为 Django 的 auth 应用为我们设置了模板常量，所以**在任何模板中都可以引用 {{ user }}**。此外，我们之前提过的 **django.contrib.auth.middleware.AuthenticationMiddleware 为所有的请求 request 绑定了一个 user 属性。所以在模板中引用 {{ user }} 和 {{ request.user }} 是等价。**

OK 了！不过目前为止，如果你已经登录过了，想要看看未登录的效果会变得比较困难，因为我们还无法注销登录。下面就来给网站添加注销登录的功能。

5、注销登录

当用户想切换登录账号，或者想退出登录状态时，这时候就需要注销已登录的账号。如果你已经登陆，就会看到一个注销登录的按钮，点击该按钮就会跳转到首页，你将看到登录按钮，说明你已经成功注销登录状态了。

6、访问限制

要限制只有登录用户才能访问，需要使用 login_required 装饰器：

【blog/urls.py】

```
from django.contrib.auth.decorators import login_required

...
```

```
app_name = 'blog'

urlpatterns=[

    ...

    url(r'^post/(?P<pk>[0-
9]+)/$',login_required(views.PostDetailView.as_view()),name='detail'
),

]
```

7、页面跳转

对于一个网站来说，**比较好的用户体验是登录、注册和注销后跳转回用户之前访问的页面。**否则用户在你的网站东跳转西跳转好不容易找到了想看的内容，结果他已登录给他跳转回了首页，这会使用户非常愤怒（我在有些网站就遇到过）。接下来我们看看如何让登录、注册和注销后跳转回用户之前访问的页面。

（1）登录和注销后返回当前页面

在登录和注销的视图函数中，Django 已经为我们处理了跳转回用户之前访问页面的流程。其实现的原理是，在登录和注销的流程中，始终传递一个 next 参数记录用户之前访问页面的 URL。

为了在整个登录流程中记录 next 的值，还需要在**登录表单（注意是在表单中添加！！！）**中增加一个表单控件，用于传递 next 值。

【registration/login.html】

```
<form class="form" action="{% url 'login' %}" method="post">
 ...
 <button type="submit" class="btn btn-primary btn-block">登录</button>
 <input type="hidden" name="next" value="{{ next }}" />
</form>
```

即在表单中增加了一个隐藏的 input 控件，其值为 {{ next }}。这个 next 可以直接使用，即之前通过 URL 参数传递给登录视图函数的，然后登录视图函数又将该值传递给了 login.html 模板。这样在整个登录流程中，始终有一个记录着用户在登录前页面 URL 的变量 next 在视图和模板间来回传递，知道用户登录成功后再跳转回 next 记录的页面 URL。

现在你可以点击登录和注销的按钮来走一遍登录和注销流程，发现页面跳转已经符合我们的需求了。这样，整个登录和注销流程就形成了一个闭环。如果用户通过点击登录或者注销按钮登录和注销的话，在登录或者注销成功后就会被带回登录或者注销前的页面，否则将他带回网站首页。

（2）注册后返回当前页面

类似的，我们也希望用户注册后返回注册前页面。不过由于注册视图函数是我们自己写的，之前的处理方式是用户注册成功后将其带回网站首页，因此需要修改一下注册视图函数：

【users/views.py】

```
def register(request):
    # 从 get 或者 post 请求中获取 next 参数值
    # get 请求中，next 通过 url 传递，即 /?next=value
    # post 请求中，next 通过表单传递，即 <input type="hidden" name="next" value="{{ next }}"/>
    redirect_to = request.POST.get('next', request.GET.get('next', ''))

    # 只有当请求为 POST 时，才表示用户提交了注册信息
    if request.method == 'POST':
        # request.POST 是一个类字典数据结构，记录了用户提交的注册信息
        # 这里提交的就是用户名（username）、密码（password）、确认密码、邮箱（email）
        # 用这些数据实例化一个用户注册表单
        form = RegisterForm(request.POST)

        # 验证数据的合法性
        if form.is_valid():
            # 如果提交数据合法，调用表单的 save 方法将用户数据保存到数据库
            form.save()

            if redirect_to:
                return redirect(redirect_to)
            else:
                return redirect('/')
    else:
        # 请求不是 POST，表明用户正在访问注册页面，展示一个空的注册表单给用户
        form = RegisterForm()

    # 渲染模板
    # 如果用户正在访问注册页面，则渲染的是一个空的注册表单
    # 如果用户通过表单提交注册信息，但是数据验证不合法，则渲染的是一个带有错误信息的表单
    # 将记录用户注册前页面的 redirect_to 传给模板，以维持 next 参数在整个注册流程中的传递
    return render(request, 'users/register.html', context={'form': form, 'next': redirect_to})
```

逻辑非常简答，就是首先尝试从用户的 GET 或者 POST 请求中获取 next 参数值，即在注册成功后需要跳转的 URL，如果有值，注册成功后跳转到该 URL，否则跳转回首页。不要忘记将该值传给模板，以维持 next 参数在整个注册流程中的传递。

接下来修改注册表单，和登录表单的设置是一样的：

【templates/users/register.html】

```
<form class="form" action="{% url 'users:register' %}" method="post">
  ...
  <button type="submit" class="btn btn-primary btn-block">注册</button>
  <input type="hidden" name="next" value="{{ next }}" />
</form>
```

注意：在注册视图函数中，对 next 的任意值我们都进行了跳转，这可能导致一些安全问题。正确的做法应该是在跳转前，对需要跳转的 URL 做安全性检查。不过这里只作为一个示例，在实际项目中请仔细考虑可能的安全后果，以及添加必要的安全性检查代码。

OK，如此修改以后，用户的登录、注册和注销流程的用户体验可以形成一个比较良好闭环了。接下来就来实现修改密码的功能。