26-用户认证-注册

用户注册就是**创建用户对象**，将用户的个人信息保存到数据库里。回顾一下 Django 的 MVT 经典开发流程，对用户注册功能来说，首先创建用户模型（M），这一步我们已经完成了。编写注册视图函数（V），并将为视图函数绑定对应的 URL。编写注册模板（T），模板中提供一个注册表单给用户。Django 用户系统内置了登录、修改密码、找回密码等视图，但是唯独用户注册的视图函数没有提供，这一部分需要我们自己来写。

1、编写用户注册表单

Django 已经**内置了一个用户注册表单**：django.contrib.auth.forms.UserCreationForm，不过这个表单的一个小问题是它**关联的是 django 内置的 User 模型**，从它的源码中可以看出：

```
class UserCreationForm(forms.ModelForm):
    ...
    class Meta:
        model = User
        fields = ("username",)
        field_classes = {'username': UsernameField}
```

问题就出在内部类 `Meta` 的 `model` 属性，它的值对应的是 auth.User，因此无法用于我们自定义的 User 模型。好在表单实际上就是一个 Python 类，因此我们可以继承它，对它做一点小小的修改就可以了。

表单的代码通常写在 forms.py 文件里，因此在 users 应用下新建一个 forms.py 文件用于存放表单代码，然后写上如下代码：

【users/forms.py】

```
from django.contrib.auth.forms import UserCreationForm
# 第 1 种方法 from .models import User

# 第 2 种方法 from django.contrib.auth.models import User


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

UserCreationForm 的 Meta 内部类下的 model 属性对应的是 auth.User 模型。而 RegisterForm 通过覆写父类 model 属性的值，将其改为 users.User。

此外 fields 用于指定表单的字段，**这些指定的字段在模板中会被渲染成表单控件（即一些 <input> 等表单控件）。** UserCreationForm 中只指定了 fields = ("username",)，即用户名，此外还有两个字段密码和确认密码在 UserCreationForm 的属性中指定。所以默认的表单渲染后只有用户名（username）、密码、确认密码三个表单控件。我们还希望用户注册时提供邮箱地址，所以在 fields 中增加了 email 字段。

注意：**虽然 model 属性的值都被指定为 User，但一个是 auth.User，另一个是 users.User。**

2、编写用户注册视图函数

首先来分析一下注册函数的逻辑。用户在注册表单里填写注册信息，然后通过表单将这些信息提交给服务器。视图函数从用户提交的数据提取用户的注册信息，然后验证这些数据的合法性。如果数据合法，就新建一个用户对象，将用户的数据保存到数据库，否则就将错误信息返回给用户，提示用户对提交的信息进行修改。过程就是这么简单，下面是对应的代码（视图函数的代码通常写在 views.py 文件里）：

【users/views.py】

```
from django.shortcuts import render, redirect
from .forms import RegisterForm

def register(request):
    # 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）、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（所谓的注册，就是把用户信息放在数据库中）
            form.save()

            # 注册成功，跳转回首页
            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()
```

把用户填写的表单信息写进表单对象中

表单对象能直接save，是因为在Meta当中指定了model

```
    # 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
    return render(request, 'users/register.html', context={'form': form})
```

注意以上视图是**处理表单的经典流程**，即：

```
def form_process_view(request):
    if request.method == 'POST':
        # 请求为 POST，利用用户提交的数据构造一个绑定了数据的表单
        form = Form(request.POST)

        if form.is_valid():
            # 表单数据合法
            # 进行其它处理...
            # 跳转
            return redirect('/')
    else:
        # 请求不是 POST，构造一个空表单
        form = Form()

    # 渲染模板
    # 如果不是 POST 请求，则渲染的是一个空的表单
    # 如果用户通过表单提交数据，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
    return render(request, 'template.html', context={'form': form})
```

以上逻辑代码稍加修改就可以应用于各种表单处理。

3、设置 URL 模式

视图函数需要和对应的 URL 绑定，这样当用户访问某个 URL 时，Django 才知道调用哪个视图函数处理用户请求。首先在 users 应用下新建一个 urls.py 文件用于设置注册视图函数的 URL 模式。

【users/urls.py】

```
from django.conf.urls import url
from . import views

app_name = 'users'
urlpatterns = [
    url(r'^register/', views.register, name='register'),
]
```

`app_name = 'users'` 为这个 urls 模块设置命名空间。

接下来需要在工程的 urls.py 文件里包含 users 应用的 URL 模式。打开 blogproject/ 目录下的 urls.py 文件，将 users.urls.py 包含进来：

【blogproject/urls.py】

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

urlpatterns = [
    # ...
    url(r'^accounts/', include('users.urls')),
]
```

4、编写注册页面模板

我们在视图函数中渲染了 users/register.html，不过目前这个模板文件还不存在，我们这就来创建它。习惯将模板文件放在项目根目录（manage.py 所在目录）的 templates/ 目录下，然后在 templates/ 目录下再新建各个和应用同名的文件夹，用于存放该应用下的模板文件。当然模板放在哪里是无关紧要的，具体视项目而定，只要通过配置模板路径使 Django 能够找到模板文件即可。

接下来就是在 register.html 模板中渲染表单了，具体代码如下：

【templates/users/register.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>注册</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>注册</h3>
                <form class="form" action="{% url 'users:register' %}"
method="post">
```

```
                    {% csrf_token %}
                    {% 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">注册
</button>
                        <input type="hidden" name="next" value="{{ next }}"/>
                </form>
                <div class="flex-center top-gap text-small">
                    <a href="{% url 'login' %}">已有账号登录</a>
                </div>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

标准的带表单模板的写法

为了使注册页面更加美观，引入了 mobi.css 提供样式支持。其它的代码请忽略，重点只关注表单部分：

```
                <form class="form" action="{% url 'users:register' %}"
method="post">
                    {% csrf_token %}
                    {% 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">注册
</button>
                        <input type="hidden" name="next" value="{{ next }}"/>
                </form>
```

在 Django 中使用表单，必须注意以下几点：

- 设置表单的 action 属性。这个例子中，表单的数据将提交给 URL /users/register/，然后 Django 调用对应的视图函数 `register` 进行处理。这里我们使用了 {% url %} 模板标签，防止 URL 硬编码。关于 {% url %} 模板标签，可以看这篇文章中的介绍 博客文章详情页。
- 设置表单的 method 属性，通常提交 表单数据都是通过 post 方法提交。
- 在表单中加入 **{% csrf_token %} 模板标签。这个模板标签的用途就是用于防止跨站请求伪造攻击，提高网站的安全性。**至于什么是跨站请求伪造，感兴趣的可以搜索相关资料查阅。这里只需记住只要使用了表单，一定要在表单中加 {% csrf_token %} 模板标签，否则 Django 将不允许你提交表单数据。

接下来就是表单的控件部分。对表单 form（这是一个模板变量，是 `RegisterForm` 的一个实例，我们在 `register` 视图函数中将它传递给模板的。）进行循环就可以得到表单的各个控件：

- {{ field.label_tag }} 是相应控件的 label 标签
- {{ field }} 是相应的表单控件
- {{ field.errors }} 是表单的错误（如果有的话）
- {{ field.help_text|safe }} 是控件相关的帮助信息

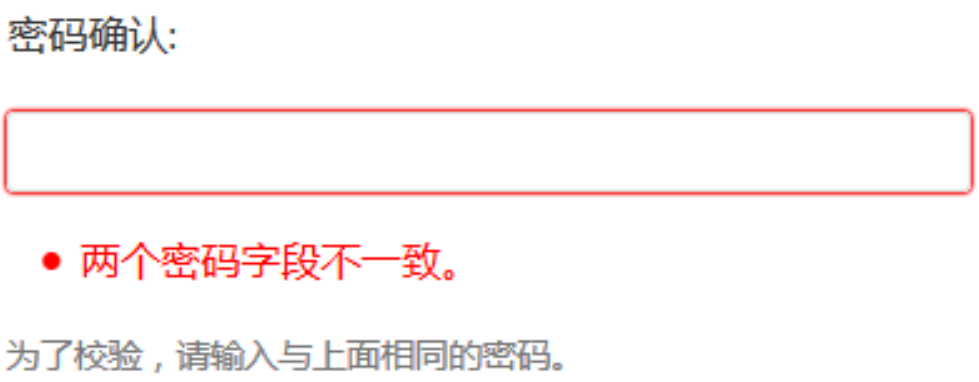
例如 `RegisterForm` 表单有用户名字段，渲染后的表单控件为：

```
<label for="id_username">用户名:</label><!-- 对应 {{ field.label_tag }} -->
<input type="text" name="username" id="id_username" autofocus required maxlength="150" /><!-- 对应 {{ field }} -->
<p class="help text-small text-muted">必填。150 个字符或者更少。包含字母，数字和仅有的 @/./+/-/_符号。</p><!-- 对应 {{ field.help_text }} -->
```

你可以按 F12 看看表单的源代码，对比一下表单控件是哪一部分渲染而成的。这种表单渲染方式是一种比较通用的做法，你可以把它当做一个模板，稍作修改就可以应用与其它需要渲染表单的模板中。

OK，运行开发服务器，访问 http://127.0.0.1:8000/users/register/ ，就可以看到渲染的用户注册表单了。

你可以尝试注册一个用户，或者尝试故意输错一些信息，看看表单渲染的错误信息是什么样的，比如我故意输入两次不同的密码，得到一个错误信息提示：



5、在 Admin 后台查看用户是否注册成功

如果表单数据没有错误，提交表单后就会跳转到首页，由于我们没有写任何处理首页的视图函数，所以得到一个 404 错误。不过没有关系，我么你现在只关心用户是否注册成功。那么怎么查看用户是否已经注册成功呢？可以去 Django Admin 后台看看是否有用户新注册的数据。为了在 Admin 后台查看用户数据，首先需要注册用户模型。打开 users/admin.py 文件，在里面注册 users.User 模型：

【users/admin.py】

```
from django.contrib import admin
from .models import User

admin.site.register(User)
```

为了进入后台，还要创建一个超级管理员用户，使用 `python manage.py createsuperuser` 创建一个管理员账户即可。

浏览器输入 http://127.0.0.1:8000/admin/ ，登录管理员账户，可以查看到注册的用户信息了，比如在我的后台可以看到三个用户：

其中有一个是使用 createsuperuser 命令创建的管理员账户，另外两个是注册的新用户。

至此，注册功能已经完成了。用户注册后就要登录，接下来就是如何提供用户登录功能了。