

一、forms 介绍

在 Django 中，forms 是用于处理用户输入的表单。forms 允许你定义表单字段、验证规则和渲染表单。forms 还提供了一些方便的方法来处理表单数据，例如清洁和验证用户输入。

二、创建 forms

要创建一个 forms，你可以使用  forms.Form  类或继承自  forms.ModelForm  类。 forms.Form  类用于创建自定义表单，而  forms.ModelForm  类用于创建与模型相关联的表单。

三、案例：创建一个用户注册 form

假设你有一个用户模型，如下所示：

from django.db import models

class User(models.Model):

    name = models.CharField(max\_length=100)

    email = models.EmailField(unique=True)

    password = models.CharField(max\_length=255)

你可以创建一个与 User 模型相关联的 form，如下所示：

from django import forms

from.models import User

class UserForm(forms.ModelForm):

    class Meta:

        model = User

        fields = ['name', 'email', 'password']

在这个 form 中，我们使用  forms.ModelForm  类继承，并指定了模型和要显示的字段。 Meta  类用于配置 form 的元数据，例如模型和字段。

四、渲染 forms

要渲染一个 forms，你可以使用 Django 的模板系统。在模板中，你可以使用  {{ form.as\_p }}  来渲染 form。

以下是一个示例模板：

<!DOCTYPE html>

<html>

<head>

    <title>用户注册</title>

</head>

<body>

    <h1>用户注册</h1>

    <form action="/register/" method="post">

        {% csrf\_token %}

        {{ form.as\_p }}

        <button type="submit">注册</button>

    </form>

</body>

</html>

在这个模板中，我们使用  {{ form.as\_p }}  来渲染 form。 as\_p  方法会将 form 渲染为 HTML 表格。

五、验证 forms

在 Django 中，forms 提供了一些验证规则，你可以在 form 中定义这些规则。例如，你可以要求某个字段必填，或者要求某个字段符合特定的格式。

以下是一个示例：

from django import forms

from.models import User

class UserForm(forms.ModelForm):

    class Meta:

        model = User

        fields = ['name', 'email', 'password']

    def clean\_password(self):

        password = self.cleaned\_data['password']

        if len(password) < 8:

            raise forms.ValidationError("密码必须至少包含 8 个字符。")

        return password

在这个 form 中，我们定义了一个  clean\_password  方法，用于验证密码字段。如果密码长度小于 8 个字符，验证会失败，并显示错误消息。

六、处理 forms 数据

当用户提交表单时，你需要处理表单数据。在 Django 中，你可以在视图中处理表单数据。

以下是一个示例视图：

from django.shortcuts import render, redirect

from.forms import UserForm

from.models import User

def register\_view(request):

    if request.method == 'POST':

        form = UserForm(request.POST)

        if form.is\_valid():

            form.save()

            return redirect('/login/')

    else:

        form = UserForm()

    context = {

        'form': form,

    }

    return render(request, 'register.html', context)

在这个视图中，我们首先检查请求方法是否为 POST。如果是 POST 请求，我们使用  request.POST  获取表单数据，并使用  form.is\_valid()  验证表单数据。如果验证通过，我们使用  form.save()  保存表单数据。如果验证失败，我们渲染表单模板，并显示错误消息。

七、总结

在这份文档中，我们学习了如何使用 Django 的 forms 来处理用户输入。我们创建了一个 form，并使用 Django 的模板系统渲染 form。我们还学习了如何验证 forms，并处理表单数据。

好的，我可以为你提供更多关于 Django forms 的讲解。

一、forms 的作用

forms 是 Django 提供的一个用于处理用户输入的模块。它提供了一个简单而强大的机制，用于创建、渲染和验证表单。

forms 的主要作用是将用户输入的数据转换为 Python 对象，并对这些数据进行验证。验证可以确保用户输入的数据符合预期的格式和规则。

forms 还可以将验证后的 Python 对象转换为 HTML 表单，以便在浏览器中显示给用户。

二、forms 的基本使用

在 Django 中，使用 forms 非常简单。你可以按照以下步骤来创建和使用 forms：

1. 创建一个 forms 类：使用 Python 类来定义表单字段和验证规则。

2. 渲染表单：使用 Django 的模板系统来渲染表单。

3. 处理表单数据：在视图中处理表单数据。

三、forms 的字段

forms 类中的字段表示表单中的输入字段。Django 提供了多种内置的字段类型，例如：

1. CharField：用于输入文本。

2. IntegerField：用于输入整数。

3. DateField：用于输入日期。

4. EmailField：用于输入电子邮件地址。

5. FileField：用于上传文件。

你可以根据需要选择合适的字段类型，并在 forms 类中定义它们。

四、forms 的验证

forms 类中的验证规则用于确保用户输入的数据符合预期的格式和规则。Django 提供了多种内置的验证器，例如：

1. Required：确保字段不为空。

2. MaxLength：限制字段的最大长度。

3. MinLength：限制字段的最小长度。

4. URLValidator：验证字段是否为有效的 URL。

5. EmailValidator：验证字段是否为有效的电子邮件地址。

你可以根据需要选择合适的验证器，并在 forms 类中定义它们。

五、forms 的渲染

在 Django 中，你可以使用模板系统来渲染 forms。在模板中，你可以使用  {{ form.as\_p }}  来渲染 forms，这将显示一个 HTML 表单。

你还可以使用  {{ form.as\_ul }}  来渲染 forms，这将显示一个列表形式的表单。

六、forms 的处理

在 Django 中，你可以在视图中处理 forms 数据。当用户提交表单时，视图将收到一个  HttpRequest  对象，其中包含表单数据。

你可以使用  form = UserForm(request.POST)  来获取表单数据，并使用  form.is\_valid()  来验证表单数据。

如果表单数据有效，你可以使用  form.cleaned\_data  来获取验证后的数据。

七、forms 的高级特性

除了上述基本特性之外，forms 还提供了一些高级特性，例如：

1. 字段标签：你可以自定义表单字段的标签。

2. 字段帮助文本：你可以添加字段的帮助文本。

3. 字段前缀和后缀：你可以添加字段的前缀和后缀。

4. 表单组：你可以将多个表单字段分组。

5. 表单修饰符：你可以在表单上添加修饰符，例如  readonly 、 disabled  等。

八、forms 的示例

下面是一个完整的 forms 示例，演示了如何创建、渲染和处理 forms：

from django import forms

from.models import User

class UserForm(forms.ModelForm):

    class Meta:

        model = Users

        fields = ['name', 'email', 'password']

def register\_view(request):

    if request.method == 'POST':

        form = UserForm(request.POST)

        if form.is\_valid():

            form.save()

            return redirect('/login/')

    else:

        form = UserForm()

    context = {

        'form': form,

    }

    return render(request, 'register.html', context)

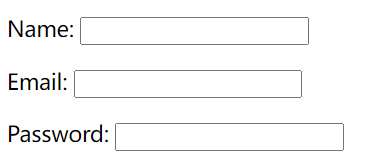
在上述示例中，我们创建了一个  UserForm  类，继承自  forms.ModelForm 。我们在  Meta  类中指定了表单的模型和字段。

然后，我们创建了一个  register\_view  视图，用于处理用户注册。在视图中，我们首先检查请求方法是否为 POST。如果是 POST 请求，我们使用  request.POST  获取表单数据，并使用  form.is\_valid()  验证表单数据。如果表单数据有效，我们使用  form.save()  保存表单数据。

如果请求方法不是 POST 请求，我们创建一个新的  UserForm  对象，并将其添加到上下文字典中。

最后，我们使用  render  函数渲染  register.html  模板，并将上下文字典传递给模板。

在  register.html  模板中，我们使用  {{ form.as\_p }}  来渲染表单。



九、总结

forms 是 Django 提供的一个用于处理用户输入的模块。它提供了一个简单而强大的机制，用于创建、渲染和验证表单。

forms 的主要作用是将用户输入的数据转换为 Python 对象，并对这些数据进行验证。验证可以确保用户输入的数据符合预期的格式和规则。

forms 还可以将验证后的 Python 对象转换为 HTML 表单，以便在浏览器中显示给用户。

希望这个文档对你有所帮助！如果你还有其他问题，请随时提问。

**自定义表单输入的widget**

Django forms的每个字段你都可以选择你喜欢的输入widget，比如多选，复选框。你还可以定义每个widget的css属性。如果你不指定，Django会使用默认的widget，有时比较丑。

比如下面这段代码定义了表单username字段的输入控件为Textarea，还指定了其样式css。

class RegistrationForm(forms.Form):

    username = forms.CharField(label='Username', max\_length=255,

        widget=forms.Textarea(

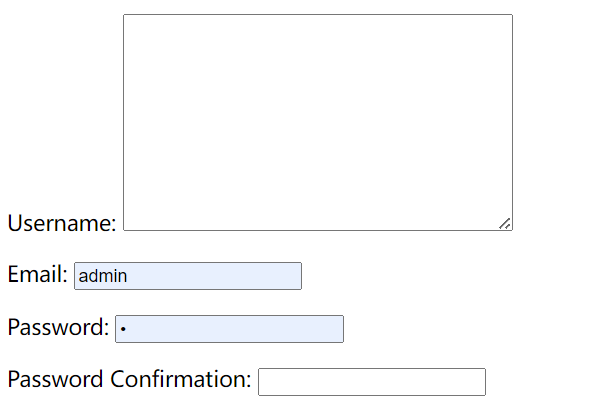
            attrs={'class': 'custom'},

        ),)

    email = forms.EmailField(label='Email',)

    password1 = forms.CharField(label='Password', widget=forms.PasswordInput)

    password2 = forms.CharField(label='Password Confirmation', widget=forms.PasswordInput)



设置widget可以是你的表单大大美化，方便用户选择输入。比如下面案例里对年份使用了SelectDateWidget，颜色则使用了复选框CheckboxSelectMultiple。单选可以用RadioSelect和Select。常见文本输入可以用TextInput和TextArea。

from django import forms

BIRTH\_YEAR\_CHOICES = ('1980', '1981', '1982')

COLORS\_CHOICES = (

('blue', 'Blue'),

('green', 'Green'),

('black', 'Black'),

)

class SimpleForm(forms.Form):

birth\_year = forms.DateField(widget=forms.SelectDateWidget(years=BIRTH\_YEAR\_CHOICES))

favorite\_colors = forms.MultipleChoiceField(

required=False,

widget=forms.CheckboxSelectMultiple,

choices=COLORS\_CHOICES,

)

以下是 Django 表单集 Formset 的教学文档，包含一个案例。

一、表单集 Formset 简介

在 Django 中，表单集 Formset 是一种用于处理多个相似表单的工具。它允许你在同一个页面上显示和处理多个表单，并且提供了一些方便的功能，例如自动生成表单的数量、管理表单的状态等。

二、使用表单集 Formset 的步骤

1. 导入 Formset 类：

from django.forms import formset

2. 创建一个表单集 Formset 实例，并指定表单的类和数量：

UserFormSet = formset(Form, extra=2)

在上述代码中， Form 是你要使用的表单类， extra=2 表示额外生成 2 个空表单。

3. 在模板中显示表单集：

<form method="post">

    {{ formset.management\_form }}

    {% for form in formset %}

        {{ form.as\_p }}

    {% endfor %}

    <input type="submit" value="提交">

</form>

在上述代码中， formset.management\_form 用于显示表单集的管理表单， formset 是表单集的实例， form.as\_p 用于显示单个表单。

4. 在视图中处理表单集：

def my\_view(request):

    if request.method == 'POST':

        formset = UserFormSet(request.POST)

        if formset.is\_valid():

            # 处理有效的表单数据

            for form in formset.forms:

                # 处理每个表单的数据

                pass

            return redirect('success\_url')

    else:

        formset = UserFormSet()

    context = {

        'formset': formset,

    }

    return render(request, 'my\_template.html', context)

在上述代码中，首先检查请求方法是否为 POST。如果是 POST 请求，调用 formset.is\_valid() 方法来验证表单集的数据。如果数据有效，通过循环处理每个表单的数据。如果请求方法不是 POST 请求，创建一个新的表单集实例。

三、案例

假设我们有一个用户管理页面，用户可以在该页面上添加或修改多个用户的信息。我们将使用 Django 的表单集 Formset 来实现这个功能。

首先，我们需要创建一个用户表单 UserForm ：

from django import forms

class UserForm(forms.Form):

    name = forms.CharField(max\_length=100)

    email = forms.EmailField(max\_length=200)

    password = forms.CharField(max\_length=200, widget=forms.PasswordInput)

然后，我们在视图中处理表单集：

from django.shortcuts import render, redirect

from django.forms import formset

def my\_view(request):

    if request.method == 'POST':

        formset = UserFormSet(request.POST)

        if formset.is\_valid():

            # 处理有效的表单数据

            for form in formset.forms:

                # 处理每个表单的数据

                pass

            return redirect('success\_url')

    else:

        formset = UserFormSet()

    context = {

        'formset': formset,

    }

    return render(request, 'my\_template.html', context)

在上述代码中，我们首先检查请求方法是否为 POST。如果是 POST 请求，调用 formset.is\_valid() 方法来验证表单集的数据。如果数据有效，通过循环处理每个表单的数据。如果请求方法不是 POST 请求，创建一个新的表单集实例。

最后，在模板中显示表单集：

<!DOCTYPE html>

<html>

<head>

    <title>用户管理页面</title>

</head>

<body>

    <h1>用户管理页面</h1>

    <form method="post">

        {{ formset.management\_form }}

        {% for form in formset %}

            {{ form.as\_p }}

        {% endfor %}

        <input type="submit" value="提交">

    </form>

</body>

</html>

在上述模板中，我们使用 {{ formset.management\_form }} 来显示表单集的管理表单，使用 {% for form in formset %} 来循环显示每个表单。

通过以上步骤，我们就可以使用 Django 的表单集 Formset 来处理多个相似表单的情况。