

Form表单

一、概要

通常情况下，我们需要自己手动在HTML页面中，编写form标签和其内的其它元素。但这费时费力，而且有可能写得不太恰当，数据验证也比较麻烦。有鉴于此，Django在内部集成了一个表单模块，专门帮助我们快速处理表单相关的内容。

Django的表单模块给我们提供了下面三个主要功能：

- 准备和重构数据用于页面渲染
- 为数据创建HTML表单元素
- 接收和处理用户从表单发送过来的数据

二、Form相关的对象包括

- Widget：用来渲染成HTML元素的部件，如：forms.Textarea对应HTML中的 `<textarea>` 标签
- Field：Form对象中的一个字段，如：EmailField表示email字段，如果这个字段不是有效的Email地址格式，就会产生错误。
- Form：一系列Field对象的集合，负责验证和显示HTML元素
- Form Media：用来渲染表单的CSS和JavaScript资源。

三、基本使用

1、说明

Form对象封装了一系列Field和验证规则，Form类都必须直接或间接继承自django.forms.Form，定义Form有两种方式：

- 直接继承Form

```
class XXXForm(forms.Form):  
    pass
```

- 结合Model，继承django.forms.ModelForm

```

class XXX(models.Model):
    字段 = models.CharField(max_length=30)
    字段 = models.CharField(max_length=20)

class XXXForm(ModelForm):
    class Meta:
        model = XXX
        field = ('字段', '字段') # 只显示model中指定的字段，显示所有是用'__all__'

```

3、示例代码

1. models

```

class Shop(models.Model):
    title = models.CharField('标题', max_length=30)
    content = models.CharField('内容', max_length=20)
    class Meta:
        db_table = 'T_SHOP'

```

2. ModelForm

```

class ShopForm(ModelForm):
    class Meta:
        model = models.Shop
        fields = ('title', 'content') # 只显示model中指定的字段

```

3. views

```

from django.shortcuts import render

# Create your views here.
from .forms import ShopForm

def add_shop(request):
    if request.method == "POST":
        form = ShopForm(request.POST)
        if form.is_valid(): # 所有验证都通过
            # 处理表单数据
            title = form.cleaned_data['title']
            print(title)

```

```
        # content = form.cleaned_data['content']
        # 保存数据
        form.save()
        return render(request, 'shop/add_shop.html',
{"shop_form": form})
    else:
        form = ShopForm()
        return render(request, 'shop/add_shop.html', {"shop_form":
form})
```

4. 模板中使用

```
<form action="{% url 'add' %}" method="post">
    {% csrf_token %}
    {{ shop_form }}
    <input type="submit" value="提交"/>
</form>
```

四、常用的field类


- 核心字段参数

参数名	说明
required	给字段添加必填属性，不能空着，若要表示一个字段不是必需的，设置required=False
label	label参数用来给字段添加‘人类友好’的提示信息。如果没有设置这个参数，那么就用字段的首字母大写名字
label_suffix	Django默认为上面的label参数后面加个冒号后缀，如果想自定义，可以使用 <code>label_suffix</code> 参数
initial	为HTML页面中表单元定义初始值。也就是input元素的value参数的值
widget	指定渲染Widget时使用的widget类，也就是这个form字段在HTML页面中是显示为文本输入框？密码输入框？单选按钮？多选框？还是别的
help_text	该参数用于设置字段的辅助描述文本
error_messages	该参数允许你覆盖字段引发异常时的默认信息。传递的是一个字典，其值为你想覆盖的错误信息
validators	指定一个列表，其中包含了为字段进行验证的函数
localize	localize参数帮助实现表单数据输入的本地化。
disabled	设置有该属性的字段在前端页面中将显示为不可编辑状态。

- 核心字段

字段名	参数
BooleanField	默认的Widget: CheckboxInput 空值: False 规范: Python 的True 或 False。 错误信息的键: required
IntegerField	默认的Widget: 当Field.localize是False时为NumberInput，否则为TextInput。 空值: None 规范化为: Python 整数或长整数。 验证给定值是一个整数。允许前导和尾随空格，类似Python的int()

	<p>函数。</p> <p>错误信息的键: max_value, invalid, required, min_value</p>
CharField	<p>默认的Widget: TextInput</p> <p>空值: "" (一个空字符串)</p> <p>规范化为: 一个Unicode 对象。如果提供, 验证max_length 或 min_length。否则, 所有的输入都是合法的。</p> <p>错误信息的键: required, max_length, min_length</p>
ChoiceField	<p>默认的Widget: Select</p> <p>空值: '' (一个空字符串)</p> <p>规范化为: 一个Unicode 对象。验证给定的值在选项列表中存在。</p> <p>错误信息的键: required, invalid_choice</p> <p>invalid_choice: 错误消息可能包含%(value)s, 它将被选择的选项替换掉。接收一个额外的必选参数: choices用来作为该字段选项的一个二元组组成的可迭代对象 (例如, 列表或元组) 或者一个可调用对</p>
DateField	<p>默认的Widget: DateInput</p> <p>空值: None</p> <p>规范化为: 一个Python datetime.date 对象。</p> <p>错误信息的键: required, invalid</p> <p>input_formats: 一个格式的列表, 验证给出的值是一个 datetime.date、datetime.datetime 或指定日期格式的字符串。如果不提供, 默认日期格式: ['%Y-%m-%d', '%m/%d/%Y', '%m/%d/%y']</p>
DateTimeField	<p>默认的Widget: DateInput</p> <p>空值: None</p> <p>规范化为: 一个Python datetime.datetime对象。</p> <p>错误信息的键: required, invalid</p> <p>input_formats: 默认的格式['%Y-%m-%d %H:%M:%S', '%Y-%m-%d %H:%M', '%Y-%m-%d', '%m/%d/%Y %H:%M:%S', '%m/%d/%Y %H:%M', '%m/%d/%Y', '%m/%d/%y %H:%M:%S', '%m/%d/%y %H:%M', '%m/%d/%y']</p>
DecimalField	<p>默认的Widget: 当Field.localize是False时为NumberInput, 否则为TextInput。</p> <p>空值: None</p> <p>规范化为: Python decimal对象。</p> <p>错误信息的键: max_whole_digits, max_digits, max_decimal_places, max_value, invalid, required, min_value</p> <p>可选参数: max_value,min_value:允许的值范围, 需要赋值 decimal.Decimal对象, 不能直接给个整数类型。</p> <p>max_digits: 值允许的最大位数 (小数点之前和之后的数字总共的</p>

	<p>位数，前导的零将被删除）。</p> <p>decimal_places: 允许的最大小数位。</p>
FloatField	<p>默认的Widget: 当Field.localize是False时为NumberInput, 否则为TextInput。</p> <p>空值: None</p> <p>规范化为: Float 对象。</p> <p>验证给定的值是一个浮点数。</p> <p>错误信息的键: max_value, invalid, required, min_value</p> <p>可选的参数:max_value和min_value</p>
EmailField	<p>默认的Widget: EmailInput</p> <p>空值: " (一个空字符串)</p> <p>规范化为: Unicode 对象。使用正则表达式验证给出的值是一个合法的邮件地址。</p> <p>错误信息的键: required, invalid</p> <p>两个可选的参数用于验证, max_length 和min_length。</p>
ImageField	<p>默认小部件: ClearableFileInput</p> <p>空值: None</p> <p>规范化为: UploadedFile 将文件内容和文件名称封装到单个对象中的对象。</p> <p>验证文件数据是否已绑定到表单, 并且该文件是Pillow可以理解的图像格式。</p> <p>错误信息键: required, invalid, missing, empty, invalid_image</p>
FileField	<p>默认小部件: ClearableFileInput</p> <p>空值: None</p> <p>规范化为: UploadedFile 将文件内容和文件名称封装到单个对象中的对象。</p> <p>可以验证非空文件数据已被绑定到表单。</p> <p>错误信息键: required, invalid, missing, empty, max_length</p>
ModelChoiceField	<p>默认的Widget: Select</p> <p>空值: None</p> <p>规范化为: 一个模型实例。</p> <p>验证给定的id存在于查询集中。</p> <p>错误信息的键: required, invalid_choice 可以选择一个单独的模型对像, 适用于表示一个外键字段。ModelChoiceField默认widet不适用选择数量很大的情况, 在大于100项时应该避免使用它。</p> <p>可选参数:</p> <p>empty_label:默认情况下, ModelChoiceField使用的 </p>

--	--

五、Form常用属性和方法

名称	说明	示例
cleaned_data (字典)	表单中验证通过的干净数据	form.cleaned_data form.cleaned_data.get('username')
changed_data	有变化的字段的列表	form.changed_data
fields (字典)	表单中的字段属性	form.fiedls['username']
is_bound	表单是否绑定数据	form.is_bound
errors(字典)	错误信息	form.errors
is_valid()	表单中数据是否验证通过，通过返回True，否则返回False	form.is_valid()
has_changed()	检查表单数据是否已从初始数据更改	form.has_changed()
errors.as_json(escape_html=False)	返回JSON序列化后的错误信息字典	form.errors.as_json()

六、表单渲染的选项

1、概要

对于 `<label>/<input>` 对，还有几个输出选项：

- `{{ form.as_table }}` 以表格的形式将它们渲染在 `<tr>` 标签中
- `{{ form.as_p }}` 将它们渲染在 `<p>` 标签中
- `{{ form.as_ul }}` 将它们渲染在 `` 标签中

注意，你必须自己提供 `<table>` 或 `` 元素。

- 常用渲染项：

有用的属性包括： `{{ field }}`

`{{ field.label }}`

该领域的标签，例如。Email address

```
{{ field.label_tag }}
```

字段的标签包含在适当的HTML `<label>`标记中。这包括表格`label_suffix`。例如，默认`label_suffix`值为冒号：

```
<label for="id_email">Email address:</label>
```

```
{{ field.id_for_label }}
```

将用于此字段的ID（`id_email`在上面的示例中）。如果您手动构建标签，则可能需要使用此代替`label_tag`。例如，如果你有一些内联JavaScript并且想要避免硬编码字段的ID，它也很有用。

```
{{ field.value }}
```

该字段的值。例如`someone@example.com`。

```
{{ field.html_name }}
```

将在输入元素的名称字段中使用的字段的名称。这会将表单前缀考虑在内，如果已设置的话。

```
{{ field.help_text }}
```

与该字段关联的任何帮助文本。

```
{{ field.errors }}
```

输出包含与此字段对应的任何验证错误的`a`。您可以使用循环自定义错误的表示。在这种情况下，循环中的每个对象都是包含错误消息的简单字符串。`<ul`

```
class="errorlist">{% for error in field.errors %}
```

```
{{ field.is_hidden }}
```

True如果表单字段是隐藏字段， False则此属性。它作为模板变量并不是特别有用，但在条件测试中可能很有用，例如：

```
{% if field.is_hidden %}
```

```
    {# Do something special #}
```

```
{% endif %}
```

```
{{ field.field }}
```

Field来自此BoundField包装的表单类的实例。您可以使用它来访问 Field属性，例如 。`{{ char_field.field.max_length }}`

2、渲染

1. form.as_ul

```
<form action="/add/" method="post">
    {% csrf_token %}
    <ul>
        {{ form.as_ul }}
    </ul>
    <input type="submit" value="注册一个学生">
</form>
```


2. 手动渲染

```
<form action="/add/" method="post">
  {% csrf_token %}
  <div>
    <label for="{{ form.name.id_for_label }}">姓名:</label>
    {{ form.name }}
    {{ form.name.errors }}
  </div>
  <div>
    <label for="{{ form.sex.id_for_label }}">性别:</label>
    {{ form.sex }}
    {{ form.sex.errors }}
  </div>
  <div>
    <label for="{{ form.age.id_for_label }}">年龄:</label>
    {{ form.age }}
    {{ form.age.errors }}
  </div>
  <input type="submit">
</form>
```

3. 循环渲染

```
{% for field in form %}
  <div class="fieldWrapper">
    {{ field.errors }}
    {{ field.label_tag }} {{ field }}
    {% if field.help_text %}
    <p class="help">{{ field.help_text|safe }}</p>
    {% endif %}
  </div>
{% endfor %}
```

4. 循环隐藏和可见字段

```

{# Include the hidden fields #}
{% for hidden in form.hidden_fields %}
{{ hidden }}
{% endfor %}
{# Include the visible fields #}
{% for field in form.visible_fields %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
    </div>
{% endfor %}

```

5. 可重用的表单模板

```

# In your form template:
{% include "form_snippet.html" %}

# In form_snippet.html:
{% for field in form %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
    </div>
{% endfor %}

```

七、重写验证

1. 说明
2. 举个栗子
3. 注意事项
 - 函数名就必须为clean_字段名
 - 必须有返回值
 - 只能拿自己当前字段值
 - raise ValidationError('xxx')

```

class UserForm(forms.Form):
    # 自定义方法（局部钩子），密码必须包含字母和数字

```

```

def clean_password(self):
    if self.cleaned_data.get('password').isdigit() or
self.cleaned_data.get('password').isalpha():
        raise ValidationError('密码必须包含数字和字母')
    else:
        return self.cleaned_data['password']

def clean_valid_code(self): # 检验验证码正确；之前生成的验证码
保存在了session中
    if self.cleaned_data.get('valid_code').upper() ==
self.request.session.get('valid_code'):
        return self.cleaned_data['valid_code']
    else:
        raise ValidationError('验证码不正确')

# 自定义方法（全局钩子，检验两个字段），检验两次密码一致；
def clean(self):
    if self.cleaned_data.get('password') !=
self.cleaned_data.get('password2'):
        raise ValidationError('密码不一致')
    else:
        return self.cleaned_data

# 注意，上面的字典取值用get，因为假如在clean_password中判断失败，
那么没有返回值，最下面的clean方法直接取值就会失败

```

完整代码

1. 示例代码

```

class UserForm(forms.Form):
    name = forms.BooleanField(label='用户名', required=True,
error_messages={'required': u'必选'})
    password = forms.CharField(label='密码',
widget=forms.PasswordInput(attrs={'placeholder': '请输入密码'}))
    confirm_password = forms.CharField(label='密码',
widget=forms.PasswordInput(attrs={'placeholder': '请再次输入密
码'}))
    # 下拉框
    city = forms.ChoiceField(choices=[(1, '上海'), (2, '北京'),
(3, '广州')])
    create_date = forms.DateField(label='选择时间', input_formats=
['%Y-%m-%d'])

```

```

        price = forms.DecimalField(label='价格', max_digits=10,
decimal_places=2)
        head = forms.FileField(allow_empty_file=True)
        img = forms.ImageField(allow_empty_file=True)
        email = forms.EmailField(required=False,
                                error_messages={'required': u'邮箱不
能为空',
                                                'invalid': u'邮箱格式
错误'},
                                widget=forms.TextInput(
                                    attrs={'class': "form-control",
                                           'placeholder': u'邮箱'})
                                )

        address =
forms.ModelChoiceField(queryset=models.Address.objects.filter(uid
=2)
                                , empty_label=None
                                , to_field_name=None)

    def clean_password(self):
        if self.cleaned_data.get('password').isdigit() or
self.cleaned_data.get('password').isalpha():
            raise ValidationError('密码必须包含数字和字母')
        else:
            return self.cleaned_data['password']

    def clean_valid_code(self):
        if self.cleaned_data.get('valid_code').upper() ==
self.request.session.get('valid_code'):
            return self.cleaned_data['valid_code']
        else:
            raise ValidationError('验证码不正确')

    def clean(self):
        if self.cleaned_data.get('password') !=
self.cleaned_data.get('confirm_password'):
            raise ValidationError('密码不一致')
        else:
            return self.cleaned_data

```

八、综合

```
#-----models.py
from django.db import models
class Info(models.Model):
    name = models.CharField(max_length=64)
    sex = models.CharField(max_length=64)
    birthday = models.CharField(max_length=64)
    age=models.CharField(max_length=64)
    qualification=models.CharField(max_length=64)
    job=models.CharField(max_length=64)
    email=models.CharField(max_length=64,default='')
class Hobby(models.Model):
    item=models.CharField(max_length=64)

#-----form.py

from django import forms
from app01 import models
from django.core.exceptions import ValidationError

class InfoForm(forms.Form):
    def validate_name(value):
        try:
            models.Info.objects.get(name=value)
            raise ValidationError('%s 的信息已经存在!'%value)
        except models.Info.DoesNotExist:
            pass
    sex_choice=((0,'男'),
               (1,'女'))#select的数据可以像这样写,也可以在另外一张表中动态去拿
    name = forms.CharField(validators=[validate_name],label='姓名',error_messages={'required':'必填'})

    age = forms.CharField(label='年龄',error_messages={'required':'必填'})

    # sex = forms.CharField(label='性别',error_messages=
    {'required':'必填'},,)
```

```

sex=forms.IntegerField(widget=forms.widgets.Select(choices=sex_choice
,
    attrs={'class':'setform2'} ))
    birthday = forms.CharField(label='生日',error_messages=
{'required':'必填'})

    qualification = forms.CharField(label='学历',error_messages=
{'required':'必填'},widget=forms.TextInput(attrs=
{'class':'formset','placeholder':'本科' }))
    email=forms.EmailField(max_length=100,min_length=10)
    job = forms.CharField(label='工作',error_messages={'required':'必
填'})

    def __init__(self,*args,**kwargs):
        super(Info_form,self).__init__(*args,**kwargs)
self.fields['hobby']=forms.CharField(widget=forms.widgets.Select(choi
ces=models.Hobby.objects.values_list('id','item')))

#-----views.py
from django.shortcuts import render,HttpResponse

def add_info(req):
    if req.method=='POST':
        Info_form_obj=Info_form(req.POST)
        if Info_form_obj.is_valid():

Info.objects.create(name=Info_form_obj.cleaned_data['name'],

age=Info_form_obj.cleaned_data['age'],

sex=Info_form_obj.cleaned_data['sex'],

birthday=Info_form_obj.cleaned_data['birthday'],

qualification=Info_form_obj.cleaned_data['qualification'],
                    job=Info_form_obj.cleaned_data['job']
                    )
        return HttpResponse('添加成功!')
    else:
        error_obj=Info_form_obj.errors
        print('*****')

```

```

        print(type(error_obj))#<class
'django.forms.utils.ErrorDict'>
        print(error_obj['name'][0])#必填
        print(error_obj.get('age'))#<ul class="errorlist"><li>必
填</li></ul>
        return render(req, 'add_info.html',
{'form_obj':Info_form_obj, 'error_obj':error_obj})
        Info_form_obj=Info_form()
        return render(req, 'add_info.html', {'form_obj':Info_form_obj})

#-----add_info.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>添加个人信息</title>
    <style>
        .formset{
            color: rebeccapurple;
            border: dashed cadetblue;
        }
    </style>
</head>
<body>
    <form action="{% url 'add_info' %}" method="post">
        <p>姓名{{ form_obj.name }}{{ error_obj.name.0 }}</p>
        <p>年龄{{ form_obj.age }}{{ error_obj.age.0 }}</p>
        <p>生日{{ form_obj.birthday }}{{ error_obj.birthday.0 }}</p>
        <p>工作{{ form_obj.job }}<span>{{ error_obj.job }}</span>
    </p>
        <p>学历{{ form_obj.qualification }}<span>{{
error_obj.qualification }}</span></p>
        <p>性别{{ form_obj.sex }}<span>{{ error_obj.sex }}</span>
    </p>
        <p>邮箱{{ form_obj.email }}<span>{{ error_obj.email }}
    </span></p>
        <p>爱好{{ form_obj.hobby }}<span>{{ error_obj.hobby }}
    </span></p>
        {{ form_obj.as_p }}
        <input type="submit" value="提交"><br>
        {% csrf_token %}
    </form>

```

```
</body>
```

```
</html>
```