Django Form表单API详解

Django 表单系统功能非常强大，它可以实现对字段的验证，而且还可以根据字段定义生成 HTML，在本节我们通过一些 Form 表单的 API，对上述功能进行详细的讲解，通过本节知识的学习，你会对 Django 表单系统有更加深入的认识，在本节中我们会穿插介绍一些小的应用实例以便于读者更好的理解这些 API 。

1. Form表单检查数据绑定

在《Django表单系统初体验》中我们使用类的方式创建了一个登陆表单，并在视图函数中，通过是实例化类对象，成功的创建了一张用户登录表单。

```
1. from django import forms
2. class LoginForm(forms.Form):
3.     username=forms.CharField(label="用户名",min_length=6,max_length=12)
4.     password=forms.CharField(label="用户密码",min_length=8)
```

在实际的开发工作中使用类的方式是非常方便，可以帮助开发者减少编写重复的代码的工作，而且代码也显得更为整洁，所以这种方式也是我们推荐使用的。

1) Form.is_bound

Django Form 表单有两种表现形式：一种是该表单绑定了数据，那么它就可以实现表单数据的验证并生成数据渲染的 HTML 表单；而另外一种是未绑定数据的，由于不存在数据所以直接生成空白的 HTML 表单。如果需要绑定数据，需要使用字典的形式传递待绑定的数据。实例如下所示：

```
1. In [1]: from django import forms
2.     ...: class LoginForm(forms.Form):
3.     ...:     title=forms.CharField(label="书名")
4. In [2]: f= LoginForm()#未绑定数据
5. In [3]: f.is_bound #检查是否绑定数据
6. Out[3]: False
7. In [4]: f= LoginForm({"title":"jango"})#字典传递数据
8. In [5]: f.is_bound
9. Out[15]: True #已经绑定数据
```

在此字典中，键是字段名称即 title，该字段名称与 LoginForm 类中的属性相对应。这些值是您要验证的数据。这些数据通常是字符串，但是有时也要看具体的 Field 字段类型，因为此时数据类型会有所不同。在上述代码中，我们使用 is_bound() 来区分表单是否绑定了数据。若返回 True 则说明绑定了数据，反之则未绑定。

如果你有一个绑定的 Form 实例，但是你想更改数据或者你想给一个未绑定的 Form 表单绑定一些数据，此时你需要创建一个新 Form 实例。因为 Form 实例一旦创建，它的数据将不可变。
那么各位小伙伴可以试一试，如果传递的是个空字典又会怎么样呢？猜一猜它的返回值是 True 还是 False 呢？

提示：传递空字典会创建具有空数据的绑定形式。

2. Form表单验证数据方法

1) Form.clean()

前面介绍过对于单个 Field 字段有 clean() 方法，它起到得到“清洁的数据”的作用，而这里的 clean() 是对于整个 HTML 表单来讲的，这一点需要大家注意。通常该方法用于自定义验证功能。如果想实现自定义验证机制，那么你需要重写这个 clean 方法。实例如下所示：

```
1. from django import forms
2. class ContactForm(forms.Form):
3.     def clean(self):
4.         cleaned_data = super().clean() #继承clean()方法
5.         username= cleaned_data.get("username") #获取值
```

```
        password = cleaned_data.get("password")
        if username and password: #两者必须同时满足才可以
            if "Mrcao" not in username: #验证用户名字
                raise forms.ValidationError(
                       "USE_is  error or password
                   is error"
                )
```

2) Form.is_valid()

Form 对象的主要任务是验证数据。对于绑定的 Form 实例调用 is_valid() 方法来验证指定的数据是否有效并符合规则，若有效则返回一个布尔值 True。实例如下所示：

```
In [1]: from django import forms
   ...: class LoginForm(forms.Form):
   ...:     title=forms.CharField(label="书名")
   ...:
In [2]: f=LoginForm({'title':'Django'})
In [3]: f.is_valid()
Out[3]: True
```

若将 title 的传递值设置为""的时候，则返回值是 False，因为 CharField 的 required 参数规定了此字段是必填项。所以这里的验证输入值是无效值。示例如下所示：

```
In [1]: from django import forms
   ...: class LoginForm(forms.Form):
   ...:     title=forms.CharField(label="书名")
In [2]: f=LoginForm({'title':'Django'})
In [3]: f=LoginForm({'title':''})
In [4]: f.is_valid()
Out[4]: False
```

3. Form表单处理字段错误信息方法

1) Form.add_error(field, error)

它代表向表单特定字段添加错误信息。field 参数为字段的名称。如果值为None，error 将作为 Form.non_field_errors() 的一个非字段错误即没有该字段相关联的错误类型。

2) Form.has_error(field, code=None)

判断某个字段是否具有指定 code 的错误。当 code 为 None 时，如果字段有任何错误它都将返回 True。这里的 code 代表字段的参数，比如 required、invalid 等，我们可以通过 Form.errors.as_json(escape_html=False) 来看一下（该方法用于返回 JSON序列化后的错误信息字典），实例如下：

```
In [10]: f.errors.as_json()
Out[10]: '{"title": [{"message": "¥¥u8fd9¥¥u4e2a¥¥u5b57¥¥u6bb5¥¥u662f¥¥u5fc5¥¥u586b¥¥u9879¥¥u3002", "code": "required"}]}'
In [11]: f.errors.as_data()#返回一个字典，它将字段映射到原始的ValidationError实例
Out[11]: {'title': [ValidationError(['这个字段是必填项。'])]}
In [12]: f.has_error(field="title",code="required")
Out[12]: True
```

4. Form表单检查数据是否被修改

1) Form.has_changed()

当你需要检查表单的数据是否从初始数据发生改变时，可以使用has_changed()方法。

```
In [19]: f=LoginForm({"title":"Python"},initial={'title':'Django'})
In [20]: f.has_changed()
Out[20]: True
```

2) Form.changed_data

它返回的是有变化的字段的列表。实例如下所示：

```
1. class LoginForm(forms.Form):
2.     title=forms.CharField(label="书名")
3.     user=forms.CharField(label="用户名")
4. f=LoginForm({"title":"c++","user":"yanchangsheng"},initial={"title":"Django","user":"caoxuesong"})
5. if f.has_changed():
6.     print("-".join(f.changed_data))
7. 输出结果是：title-user
```

5. 访问Form表单中的字段方法

1) Form.fields.values()

通过 fileds 属性访问表单的字段,它的返回值是字段的对象。实例如下所示：

```
1. f.fields.values()
2. odict_values([<django.forms.fields.CharField object at 0x087404F0>, <django.forms.fields.CharField object at 0x08740C30>])
3. f.fields['title']
4. <django.forms.fields.CharField at 0x87404f0>
```

6. cleaned_data实现数据访问

当一个 Form 实例创建成功后，我们就可以使用 cleaned_data 属性访问干净的数据，为什么要说干净的数据呢，因为这个数据将数据转换成 Python 字典的格式，更易于我们阅读，示例如下所示：

```
1. In [1]: f.cleaned_data
2. Out[1]: {'title': 'c++', 'user': 'yanchangsheng'}
```

如上所示通过 cleaned_data 属性实现字段值数据的访问，不过在正常的逻辑中，我们先会使用 is_valid() 方法对输入数据的合法性进行验证，然后再使用该属性得到干净的数据，若验证时存在不合法的数据，cleaned_data 方法将会自动清洗掉不它们，只显示合法的数据。

7. Form表单输出为HTML

1) print()方法输出表单

将 Form 渲染成 HTML 代码方法非的简单只需要 print() 即可，默认情况下，根据 form 类中字段的编写顺序，在 HTML 中以同样的顺序罗列。如下所以：

```
1. In [2]: f=LoginForm()
2. In [3]: print(f)
3. <tr><th><label for="id_title">书名:</label></th><td><input type="text" name="title" required id="id_title"></td></tr>
4. <tr><th><label for="id_user">用户名:</label></th><td><input type="text" name="user" required id="id_user"></td></tr>
```

不过在输出 HTML 时有以下注意事项：

- 为了灵活性，输出不包含<table>和</table>、<form>和</form>以及<input type="submit">标签。 需要我们自己手动添加。
- 每个字段类型都由一个默认的 HTML 标签展示。注意，这些只是默认的，可以使用 Widget 特别指定。
- 每个 HTML标签的 name 属性名直接从 LoginForm 类中获取。
- form 使用 HTML5 语法，顶部需添加 <!DOCTYPE html> 说明。
- <table>输出是 print 表单时的默认调用 as_table() 输出表格，但其他输出样式也可用。它们都可以作为表单对象上的方法使用。

2) 使用as_p()与as_ul()渲染HTML

上面我们介绍过，当 print() 输出的时候默认使用 as_table() 方法，这是 Form 表单对象的默认的输出格式。当然我们也可以使用其他方式进行输出，比如 as_p() 或者as_ul() 方法。实例如下所示：

```
In [1]: f.as_p()
Out[1]: '<p><label for="id_title">书名:</label> <input type="text" name="title" required id="id_title"></p>¥n<p><label for="id_user">用户名:</label> <input type="text" name="user" required id="id_user"></p>'
In [2]: f.as_ul()
Out[2]: '<li><label for="id_title">书名:</label> <input type="text" name="title" required id="id_title"></li>¥n<li><label for="id_user">用户名:</label> <input type="text" name="user" required id="id_user"></li>'
```

8. 实现错误信息添加CSS样式

如果为表单中的错误信息添加颜色或者加粗效果，那么会让其提示更加醒目，这时候就需要为其添加 CSS 样式。Django Form 表单提供了 Form.error_css_class 和 Form.required_css_class 这两个属性，使用它们就可以实现上述功能。实例如下所示：

```
In [6]: class LoginForm(forms.Form):
   ...:     error_css_class="error"
   ...:     required_css_class="required"
   ...:     title=forms.CharField(label="书名")
   ...:     user=forms.CharField(label="用户名")
   ...: f=LoginForm({"title":"c++","user":"yanchangsheng"},initial={"title":"Django","user":"caoxuesong"})
In [5]: print(f)
#按as_table()输出表格添加 required属性
<tr class="required"><th><label class="required" for="id_title">书名:</label></th><td><input type="text" name="title" value="c++" required id="id_title"></td></tr>
<tr class="required"><th><label class="required" for="id_user">用户名:</label></th><td><input type="text" name="user" value="yanchangsheng" required id="id_user"></td></tr>
#通过label_tag(attrs={})设置属性名
In [8]: f['title'].label_tag(attrs={'class': 'bar'})
Out[8]: '<label class="bar required" for="id_title">书名:</label>'
```

本节我们对 Django Form 表单中的常用 API 做了讲解，内容也许有点对，不过结合应用实例理解起来应该还算简单，如果你意犹未尽想继续深入学习 Form 表单的 API ，那么我建议你参考官方文档《Form API》。在下一节《Django Form表单完整使用流程》我们将使用这几节中学到的知识，完成一个完整的 Form 表单使用流程。