Django ModelForm用法详解

在上一节《Django Form基于Model定义表单》我们使用 ModelForm 定义了一个基于模型 UserInfo 的一张表单，整个实现的过程也非常的类似于 Form 对象的实现过程，在本节我们将对 ModelForm 中涉及的一些细节做一下介绍，那么我们先从 Meta 元数据项开始。

1. 常用的Meta选项

在上节一中，我们 class Meta 中使用了一些元数据项，比如说 exclude、labels 以及 fields，当然还有些其他的选项，在 Django 官方网站 ModelForm 的定义如下所示

```
1. modelform_factory (model, form = ModelForm, fields = None, exclude = None, formfield_callback = None, widgets = None, localized_fields = None, labels = None, help_texts = None, error_messages = None, field_classes
```

在这里给大家依次进行介绍这些 Meta 选项。

1) fields

其为列表或元组类型，与 exclude 相反，它指定当前的表单应该包含哪些字段，如果要所有的 Model 字段都包含在表单中，可以设定 fields='__all__'。ModelForm 的定义中必须要包含 fields 或 exclude 选项，否则将会抛出异常，同时给出错误提示：

Creating a ModelForm without either the'fields'attribute or the'exclude'attribute is prohibited。

2) labels

其为字典类型，用于定义表单字段的名称（输入框左边显示的名称）。表单字段的名称首先会使用 Model 字段定义设置的 verbose_name，如果没有设置，则直接使用字段名。因此当没有定义 verbo se_name 时，就可以使用 labels 选项来指定字段名。例如：

```
1. labels={
2. 'title':'标题', 、
3. 'price':'价格'
4. }
```

3) help_texts

其为字典类型，用于给表单字段添加帮助信息。目前页面中表单字段的帮助信息（输入框下方显示的内容）来自 Model字段的 help_texts 定义，如果没有定义则什么都不显示。help_texts 的定义方式与 labels 选项类似，例如：

```
1. help_texts={
2. "title":"书籍的名称"
3. "price':"书籍价格"
4. }
```

4) widgets

其为字典类型，用于定义表单字段选用的控件。默认情况下，ModelForm 会根据Model字段的类型映射表单 Field 类，因此会应用 Field 类中默认定义的 widgets。这个选项用于自定义控件类型，例如：

```
1. class Meta:
2.     model=UserInfo
3.     fields="__all__"
4.     widgets={"password':widgets.PasswordInput()}
```

5) field_classes

字典类型，用于指定表单字段使用的 Field 类型。默认情况下，对于 title 字段，ModelForm 会将它映射为 fields.CharField 类型。可以根据需要改变这种默认行为，例如，将 title 设置为如下类型：

```
1. field_calss={"title":forms.URLField}
```

6) error_messages

字典类型，用来指定表单字段校验规则，即验证失败时的报错信息。

上面的字段只是常用的 Mata 选项，若大家感兴趣，同样也可以参阅官方文档《Model Form Functions》对此处的知识点进行学习。

2. ModelForm的save()方法

在上一节《Django Form基于Model定义表单》的 user_add_form 视图函数中，它将所有的 Model 字段都定义在 ModelForm 中，此时，字段值通过校验 is_valid 之后，我们使用了 ModelForm 提供的 save 方法实现了 Model 对象的保存。ModelForm 的 save 方法定义于它的基类 BaseModelForm中，其源代码在 Django 中实现如下：

```
1. def save(self, commit=True):
2.     if self.errors:
3.         raise ValueError(
4.             "The %s could not be %s because the data didn't validate." % (
5.                 self.instance._meta.object_name,
6.                 'created' if self.instance._state.adding else 'changed',
7.             )
8.         )
9.     if commit:
10.         # 除了保存当前Model实例, 还会保存多对多关系数据
11.         self.instance.save()
12.         self._save_m2m()
13.     else:
14.         # 将保存多对多数据方法赋值给save_m2m, save返回后可以手动调用
15.         # saving of m2m data.
16.         self.save_m2m = self._save_m2m
17.     return self.instance
18. save.alters_data = True
```

通过源码不难理解它的实现逻辑，下面主要讲解一下 save 方法的基本使用，主要有三个场景。如下所示：

1) 通过页面 Post 提交过来的数据，通过 form 接收 ，然后直接保存到数据库，同时能够产生对应的 models 的一个新对象，如下所示：

```
1. f = BookForm(request.POST)
2. new_book = f.save()
```

2) 从数据库中取出 models 的对象，然后通过 form 参数 instance 方法能够实例化该 form，这个主要用来查看具体的信息，如下所示：

```
1. a = Book.objects.get(id=1)
2. f = BookForm(instance=a)
3. f.save()
```

3) 如果既有 Post 又有 instance，则以 Post 提交数据为主，这个主要用来修改具体的信息。如下所示：

```
1. a = Book.objects.get(id=1)
2. f = BookForm(request.POST, instance=a)
3. f.save()
```

save 方法接受一个 commit 参数，默认为 True，可以实现 Model 实例的保存以及多对多关系数据的保存。如果在使用 save 方法时设置了 commit 为 False，则不会执行保存动作。此时，可以对返回的实例对象做一些操作后，再执行 save() 方法。

```
1. user=UserModelForm(request.Post)
2. if user.is_valid():
3.     user=user.save(commit=False)
4.     user.username=request.username
5.     user.save()
6.     return render(request,'index/user_add.html',locals())
```

本节内容也至此完毕，在下一节《Django表单系统工作原理详述》，我们将分析一下 Django 表单系统的工作原理，由浅入深，各位小伙伴要做好心里准备哦！