Django Form表单实现自定义字段

在《Django Form表单内置字段详解》一节中，我们介绍了 Django Form 表单的内置字段，但是在某些业务场景下，内置字段无法满足我们的业务需求，此时就需要我们使用到 Django Form 表单提供的自定义字段的功能。不仅如此，Django Form 表单还给开发者提供了自定义验证规则的功能，可以让开发者对内置字段或者自定义字段添加额外的校验逻辑，从而满足实际的开发任务需要。在本节我们先对如何实现自定义 Form 表单字段做讲解。

1. 实现自定义Field字段

1) Field基类clean方法源码解读

在介绍 Form 表单的内置字段时，我们提及过每一个 Field 字段都一个 clean 方法。Field 字段通过 clean() 校验并获取字段值，所以自定义 Field 字段，简而言之就是自己去实现 clean 方法。自定义字段也要继承 Field 类，那么我看一看，Field 类中的 clean 方法是怎么实现数据的“清洗”作用的，clean 方法的源码如下所示：

```
def clean(self, value):
    """
    验证给定的值并将其“清洗”后的值作为合适的Python对象返回。
    对任何错误抛出ValidationError。
    """
    value = self.to_python(value)
    self.validate(value)
    self.run_validators(value)
    return value
```

从 源码分析来看，clean 方法接受一个 value 参数，分别经过三个方法的处理，返回已“清洗”的 value 或者是抛出异常 ValidationError。这个三个方法的源码如下所示：

```
def to_python(self, value):
    return value

def validate(self, value):
    if value in self.empty_values and self.required:
        raise ValidationError(self.error_messages['required'], code='required')

def run_validators(self, value):
    if value in self.empty_values:
        return
    errors = []
```

```
    for v in self.validators:
        try:
            v(value)
        except ValidationError as e:
            if hasattr(e, 'code') and e.code in self.error_messages:
                e.message = self.error_messages[e.code]
            errors.extend(e.error_list)
    if errors:
        raise ValidationError(errors)
```

我们分析一下这三个方法的作用。分别如下所示：

- to_python：实现数据的转换，将传递进来的 value 转换成需要的 Pyhton 对象，例如，对于 TimeField，它的 to_python 方法会将 value 转换成 Python 的 datetime.time 对象。
- validate：验证经过转换的 value 是否合法，如果不合法，需要抛出 ValidationError 异常。Field 中实现的 validate 方法只是简单地对 required 属性限制的条件进行验证，如果 required 为 True，且 value 为空值的时候，则会抛出异常。
- run_validators：这个方法会执行当前实例中包含的验证器，如果出现错误，则会抛出 ValidationError 异常。

所以从上面 clean 方法的源码分析来看，只要去实现 clean 中调用的方法，就能够实现自定义数据转换成 Python 对象和数据的校验。

2) 实现自定义BookField字段

那么接下来我们就亲自去实现一个自定字段 BookField 字段，该字段的功能是输入书籍的 id 后，可以获取 Book 的实例对象，它的代码如下所示：

```
class BookField(forms.Field):
    default_error_message={
        'invalid':'Enter a whole number',
        'not_exist':'Book Not Exist',
    }
    def to_python(self,num):
        try:
            num=int(str(num).strip())
            return Book.objects.get(id=num)
        except (ValueError,TypeError):
            raise ValidationError(self.error_messages['invalid'],code='invalid')
        except Exception:
            raise  ValidationError(self.error_messages['not_exist'],code='not_exist')
```

在编写代码的过程中，我们一定要善于参考 Django Field 类的源码，首先理解定义内置字段的过程，然后照葫芦画瓢，实现自定义 Field 字段。BookField 字段继承自 Field 基类，它重写了 to_python 方法，把 num 当做主键去查询 Book 实例，然后将其作实例对象返回。如下所示：

```
In [1]: from index.forms import BookField
In [2]: x=BookField()
In [3]: x.clean(1)
Out[3]: <Book: title: Python Django pub:PubName object (8) price:59.00>
```

3) 继承自forms.CharField实现自定义字段

继承基类 Field 去自定义表单字段可能考虑比较多的问题，所以通常自定义 Field 都会继承自 CharField 或者 IntergerField 等内置字段，即 Field 的子类。举一个简单的例子，给字符串添加固定的前缀词语，如下所示：

```
class AddstrField(forms.CharField):
    def clean(self,value):
        return 'C语言中文网 %s'% super().clean(value)
```

AddstrField 该字段在 clean 方法中使用 super 方法调用了父类的 clean()，也就是使用了 CharField 的数据校验方法，这样就很大程度上简化了该功能实现过程，实例演示如下所示：

```
In [1]: from index.forms import AddstrField
In [2]: x=AddstrField()
In [3]: x.clean('hello')
Out[3]: 'C语言中文网 hello'
```

4) 使用ValidationError验证器校验数据

验证器（validators）是一个可以调用的对象，接受一个参数，并验证参数是否符合预期，如果不符合预期就会抛出 ValidationError 异常。现在我们就使用验证器来校验数据的合法性，首先编写一个验证器，如下所示：

```
#自定义一个验证偶数的验证器，否则抛出异常
def even_validator(value):
    if value % 2  !=0:
        raise ValidationError('%d is not a even number'% value)
#编写 EvenField字段，只可以接受偶数，否则抛出异常ValidationError
class EvenField(forms.IntegerField):
    #使用构造函数__init__ 对其进行初始化，并添加验证器规则
    def __init__(self,**kwargs):
        super().__init__(validators=[even_validator],**kwargs)
```

实例演示如下所示：

```
In [1]: from index.forms import EvenField
In [2]: x=EvenField()
In [3]: x.clean("1")
....
ValidationError: ['1 is not a even number']
In [5]: x.clean(2)
Out[5]: 2
```

本节我们讲解 Django Form 表单的自定义字段的实现方法。通过本节的学习，你可能会领略到读源码的重要性，善于学习的人，总会找到合适的方法去学习自己想要掌握的知识，在下一节中，我们将讲解 Django Form 表单如何实现自定义验证规则。