一、验证器Validators

在Django REST framework 序列化器中进行验证和 在Django的 `ModelForm` 类上进行验证有一点点区别.

对于 `ModelForm` ，一部分验证在表单上执行，一部分在模型实例上执行。而使用REST框架时，验证完全在序列化类上执行。 它有下面的优点：

- 它引入了适当的关注点分离，使代码行为更加明显。
- 在使用快捷方式ModelSerializer类和使用显式的序列化类之间切换很容易。用于ModelSerializer的任何验证行为都很容易复制。
- 打印序列化程序实例可以显示它应用的验证规则。没有对模型实例调用额外的隐藏的验证行为。

当你使用（继承） `ModelSerializer` 类的时候，所有的验证动作都是自动的。而如果你使用（继承）更底层的 `Serializer` 类的时候，你需要显式地定义验证器地规则。

我们以下面的模型为例子，来演示如何在DRF中进行验证工作，这个模型有一个字段是unique的：

```
class CustomerReportRecord(models.Model):
    time_raised = models.DateTimeField(default=timezone.now, editable=False)
    reference = models.CharField(unique=True, max_length=20) # 注意这个字段
    description = models.TextField()
```

下面则是一个序列化类，继承了ModelSerializer:

```
class CustomerReportSerializer(serializers.ModelSerializer):
    class Meta:
        model = CustomerReportRecord
```

使用 `python manage.py shell` 进入shell环境，尝试下面的操作：

```
>>> from project.example.serializers import CustomerReportSerializer
>>> serializer = CustomerReportSerializer()
>>> print(repr(serializer))
CustomerReportSerializer():
    id = IntegerField(label='ID', read_only=True)  # 会自动生成这个字段，并且只读
    time_raised = DateTimeField(read_only=True)
    reference = CharField(max_length=20, validators=[<UniqueValidator(queryset=CustomerReportRecord.objects.all())>])
    description = CharField(style={'type': 'textarea'})
```

注意其中的 `reference` 字段，它被显式地指定了一个unique验证。

REST framework包含了一系列验证器类，这些类不属于Django核心原生的，它们都位于DRF的validators模块里。

UniqueValidator

用于对 `unique=True` 约束进行验证的验证器。 主要参数如下：

- `queryset` *必填参数* - 需要满足unique约束的查询集。
- `message` - 可选参数。当验证失败时自定义的错误信息。
- `lookup` - 查询的匹配方法或者说匹配模式，默认是 `'exact'` 。

看下面的例子：

```
from rest_framework.validators import UniqueValidator

slug = SlugField(
    max_length=100,
    validators=[UniqueValidator(queryset=BlogPost.objects.all())]
)
```

UniqueTogetherValidator

用于 `unique_together` 联合唯一的验证。参数如下：

- `queryset` *必填参数* - 验证器应用的查询集
- `fields` *必填参数* - 一个字段名的列表或元组，它们需要成为联合唯一不重复的集合，必须是序列化类中的字段。

- `message` - 可选参数。当验证失败时自定义的错误信息。

在序列化类中，可以参考下面的使用方式

```
from rest_framework.validators import UniqueTogetherValidator

class ExampleSerializer(serializers.Serializer):
    # ...
    class Meta:
        validators = [
            UniqueTogetherValidator(
                queryset=ToDoItem.objects.all(),
                fields=('list', 'position')
            )
        ]
```

UniqueForDateValidator、UniqueForMonthValidator、UniqueForYearValidator

以上三个验证器用于验证 `unique_for_date`，`unique_for_month` 和 `unique_for_year` 约束。参数如下：

- `queryset` 同上，必填参数。
- `field` 必填参数，实施验证的字段，字段的值必须在要求的时间范围内。
- `date_field` 必填参数-用于决定时间范围的字段。
- `message` - 错误信息，可选。

参考范例：

```
from rest_framework.validators import UniqueForYearValidator

class ExampleSerializer(serializers.Serializer):
    # ...
    class Meta:
        # Blog posts should have a slug that is unique for the current year.
        validators = [
            UniqueForYearValidator(
                queryset=BlogPostItem.objects.all(),
                field='slug',
                date_field='published'
            )
        ]
```

二、高级的字段默认值

在序列化时，跨多个字段应用的验证程序有时可能需要API客户端不应提供的字段输入，但 *可作为验证程序的输入* 使用。

两种可能的应用场景：

- 使用 `HiddenField` 字段。这种字段会出现在 `validated_data` 中，但是不会用在序列化的输出表示中。
- 使用带有 `read_only=True` 属性的标准字段，但是它同时还带有 `default=…` 参数。此时，该字段将用于序列化的输出表示，但无法被用户直接设置。

REST框架提供了一些在此情景中可能有用的默认值。

- **CurrentUserDefault**

用于表示当前用户的默认类。为了使用它，在实例化序列化类时，“request”必须作为上下文字典的一部分提供。

```
owner = serializers.HiddenField(
    default=serializers.CurrentUserDefault()
)
```

- **CreateOnlyDefault**

可用于设置的默认类，只能在创建对象的期间设置默认参数。在更新期间，该字段将被省略。

它只接收一个参数，也就是创建对象时使用的默认值或可调用对象。

```
created_at = serializers.DateTimeField(
    default=serializers.CreateOnlyDefault(timezone.now)
)
```

三、移除默认的验证器

在某些不明确的情况下，可能需要显式处理验证过程，而不是依赖“modelserializer”类生成的默认序列化类。在这些情况下，可以通过将序列化类的'meta.validators'属性指定为空列表来禁用自动生成的验证器。

例如：

```
class BillingRecordSerializer(serializers.ModelSerializer):
    def validate(self, data):
        #在这里自定义验证过程，或者在视图中。

    class Meta:
        fields = ('client', 'date', 'amount')
        extra_kwargs = {'client': {'required': False}}
        validators = []  # 移除默认的验证器
```

四、自定义验证器

你可以使用任何Django原生的验证器或者自定义一个验证器。

- 基于函数的验证器

看下面的例子，当验证失败的时候，需要弹出 `serializers.ValidationError` 异常：

```
def even_number(value):
    if value % 2 != 0:
        raise serializers.ValidationError('This field must be an even number.')
```

也可以通过为 `Serializer` 的子类添加 `.validate_<field_name>` 方法，自定义字段级别的验证器。

- 基于类的验证器

使用 `__call__` 方法编写基于类的验证器。

```
class MultipleOf(object):
    def __init__(self, base):
        self.base = base

    def __call__(self, value):
        if value % self.base != 0:
            message = 'This field must be a multiple of %d.' % self.base
            raise serializers.ValidationError(message)
```

在编写验证器的过程中，有时候需要使用一些上下文环境，可以在基于类的验证器中声明一个 `set_context` 方法来实现这一目的。

```
def set_context(self, serializer_field):
    # Determine if this is an update or a create operation.
    # In `__call__` we can then use that information to modify the validation behavior.
    self.is_update = serializer_field.parent.instance is not None
```