序列化器的字段

DRF在Django字段类型的基础上，派生出了自己的一系列字段类型以及字段参数。

序列化器的字段类型用于处理原始值和内部数据类型之间的转换。它们还用于验证输入值，以及从父对象检索和设置值。

注意： 虽然序列化器的字段类型是在 `fields.py` 模块中声明的，但按照惯例，我们一般导入 `from rest_framework import serializers` ，然后使用 `serializers.<FieldName>` 的方式引用字段类型。

一、字段的核心参数

下面是字段类型的通用参数，适合所有字段类型：

`read_only`

该参数默认为False，设置为True则将字段变为只读。

被设置成只读的字段可以包含在API输出中，但在创建或更新操作期间不应包含在输入中。即使你在序列化的时候为一个 `read_only` 字段提供值，也会被忽略，也就是说这个时候的 `validated_data` 中不会包括你发送过来的被设置为只读字段的数据。

`write_only`

默认为 `False` ，将其设置为 `True` 则表示只写不读。

`required`

默认为 `True` ，表示该字段必填，不可空缺，如果在反序列化期间未提供字段的值，则会引发错误。

设置为 `False` 表示该字段的内容可以空缺。

`default`

为字段设置默认值。请注意，在字段中同时设置 `default` 和 `required` 关键字参数是无效的，并且会引发错误。

allow_null

该参数的默认值为 `False`，表示该字段不允许为空。如果设置为True，表示字段的值可以是空的，或者接受一个None。类似于Django原生字段参数中的 `blank` 和 `null`。

source

该参数是一种指定字段的值如何填充的方法。指定了这个参数，那么这个字段可能就不需要从请求中获取，也可以在序列化的时候自动获取对应的值。单词source应该理解为数据的来源，值从哪里获得的意思，通常是对应model模型中的某个字段。

这个参数的值可能是一个只接受 `self` 参数的方法，例如 `URLField(source='get_absolute_url')`，或者是使用点分表示来遍历属性，例如 `EmailField(source='user.email')`。使用点分表示法序列化字段时，如果对象不存在或在属性遍历的结果为空，则可能需要提供 `default` 参数值。

设置 `source='*'` 具有特殊含义，用于指示整个对象应该传递到该字段。这对于创建嵌套表示或者需要访问整个对象以确定输出表示的字段非常有用。

默认为字段的名称。

```
1  user_role = serializers.CharField(source='user_type')  # 获取model中user_type
   字段的内容
2  url = serializers.URLField(source='get_absolute_url')  # 获取完整url地址
3  group_id = serializers.CharField(source='group.id')  # 获取外键关联的组的id
4  students = serializers.CharField(source='student.all') # 获取多对多关联的所有学
   生，该做法有缺陷
5  serializers.CharField(source='user_type')
```

validators

该参数用于为字段指定专门的验证器，可以是多个验证器的列表。它会引发验证错误或只是简单地返回。

error_messages

错误消息的错误代码字典。也就是说，你可以自定义错误信息。

label

一个简短的文本字符串，可用作HTML表单或其他描述性元素中生成字段的标签。和Django原生的一个意思。

help_text

一个文本字符串，可用作HTML表单或其他描述性元素中对字段的描述文字。和Django原生的一个意思。

initial

该参数用于预先填充HTML表单字段的值，也就是给字段一个初始化值。也可以将一个callable可调用方法传递给它：

```
import datetime
from rest_framework import serializers
class ExampleSerializer(serializers.Serializer):
    day = serializers.DateField(initial=datetime.date.today)
```

style

该参数的值必须为键值对的字典，可用于控制渲染器应如何渲染字段。也就是为字段添加额外的CSS或HTML控制。

这里的两个例子是 `'input_type'` 和 `'base_template'` ：

```
# Use <input type="password"> for the input.
password = serializers.CharField(
    style={'input_type': 'password'}
)

# Use a radio input instead of a select input.
color_channel = serializers.ChoiceField(
    choices=['red', 'green', 'blue'],
    style={'base_template': 'radio.html'}
)
```

下面开始介绍DRF中各种类型的字段。

二、布尔类型字段

BooleanField

普通的布尔字段。也就是值只能为True或者False。

对应于 `django.db.models.fields.BooleanField` 。

签名： `BooleanField()`

NullBooleanField

接受 `None` 作为有效值的布尔字段，也就是说这个字段也可以为空。

对应于 `django.db.models.fields.NullBooleanField` 。

签名： `NullBooleanField()`

三、字符串类型字段

CharField

最基本的文本类型。常用的参数是验证文本是否短于 `max_length` 和长于 `min_length` 。注意，不同于Django原生，这两个参数非必须。

对应于 `django.db.models.fields.CharField` 或 `django.db.models.fields.TextField` 。

签名： `CharField(max_length=None, min_length=None, allow_blank=False, trim_whitespace=True)`

- `max_length` - 字段最大长度
- `min_length` - 字段最小长度
- `allow_blank` - 字段是否可以为空，默认为 `False` 。
- `trim_whitespace` - 如果设置为 `True` 则会自动修剪字符串的前导和尾随的空格，相当于自动应用了一个字符串的strip方法。默认为 `True` 。

虽然 `allow_null` 参数也可用于字符串字段，但不鼓励同时设置 `allow_blank=True` 和 `allow_null=True`，这可能会导致数据不一致和微妙的应用程序错误。

EmailField

文本格式，验证为有效的电子邮件地址。

对应于 `django.db.models.fields.EmailField`

签名： `EmailField(max_length=None, min_length=None, allow_blank=False)`

RegexField

文本格式，字段的值必须与regex参数指定的正则表达式相匹配。

对应于 `django.forms.fields.RegexField`。

签名： `RegexField(regex, max_length=None, min_length=None, allow_blank=False)`

必填的第一位置参数 `regex` 可以是字符串，也可以是编译过的python正则表达式对象。

此字段使用Django的 `django.core.validators.RegexValidator` 进行验证。

SlugField

一个 `RegexField` 字段，根据 `[a-zA-Z0-9_-]+` 正则模式验证输入。也就是定死了正则表达式。

对应于 `django.db.models.fields.SlugField`。

签名： `SlugField(max_length=50, min_length=None, allow_blank=False)`

URLField

一个 `RegexField` 类型字段，规定字段必须是个合法的URL类型的字符串，根据URL匹配模式验证输入。类似 `http://<host>/<path>` 的形式。

对应于 `django.db.models.fields.URLField` 。使用Django的 `django.core.validators.URLValidator` 进行验证。

签名： `URLField(max_length=200, min_length=None, allow_blank=False)`

UUIDField

确保输入的字段是有效的UUID字符串。 `to_internal_value` 方法将返回一个 `uuid.UUID` 实例。在输出时，该字段将以规范的以短横线连接的格式返回一个字符串，例如：

```
"de305d54-75b4-431b-adb2-eb6b9e546013"
```

签名： `UUIDField(format='hex_verbose')`

- format：指定uuid值的表示格式
  - `'hex_verbose'` - 十六进制表示，包括连字符： `"5ce0e9a5-5ffa-654b-cee0-1238041fb31a"`
  - `'hex'` - UUID的紧凑十六进制表示形式，不包括连字符： `"5ce0e9a55ffa654bcee01238041fb31a"`
  - `'int'` - UUID的128位整数表示： `"123456789012312313134124512351145145114"`
  - `'urn'` - UUID的RFC 4122 URN表示： `"urn:uuid:5ce0e9a5-5ffa-654b-cee0-1238041fb31a"`

FilePathField

文件路径类型字段，其选择仅限于文件系统上某个目录中的文件名

对应于 `django.forms.fields.FilePathField` 。

签名： `FilePathField(path, match=None, recursive=False, allow_files=True, allow_folders=False, required=None, **kwargs)`

- `path` - 此FilePathField应从中选择的目录的绝对文件系统路径。
- `match` - 作为字符串的正则表达式，FilePathField用它过滤文件名。
- `recursive` - 指定是否应递归搜索路径的所有子目录。默认是 `False` 。
- `allow_files` - 指定是否应包括指定位置的文件。默认是 `True` 。
- `allow_folders` - 指定是否应包括指定位置的文件夹。默认是 `False` 。它和 `allow_files` 必须有一个为 `True` 。

IPAddressField

确保输入是有效的IPv4或IPv6的字符串。

对应于 `django.forms.fields.IPAddressField` 和 `django.forms.fields.GenericIPAddressField` 。

签名： `IPAddressField(protocol='both', unpack_ipv4=False, **options)`

- `protocol` ：接受协议的类型。可接受的值是“both”（默认）、“IPv4”或“IPv6”。匹配不区分大小写。
- `unpack_ipv4` ：解包IPv4映射地址，如:: ffff：192.0.2.1。如果启用此选项，则该地址将解压缩到192.0.2.1。默认为禁用。只能在协议设置为“both”时使用。

四、数字类型字段

IntegerField

整数。

对应于 `django.db.models.fields.IntegerField` ，`django.db.models.fields.SmallIntegerField` ，`django.db.models.fields.PositiveIntegerField` 和 `django.db.models.fields.PositiveSmallIntegerField` 。

签名： `IntegerField(max_value=None, min_value=None)`

- `max_value` 允许的最大值。
- `min_value` 允许的最小值。

FloatField

浮点数。

对应于 `django.db.models.fields.FloatField` 。

签名： `FloatField(max_value=None, min_value=None)`

- `max_value` 验证提供的数字是否不大于此值。

- `min_value` 验证提供的数字是否不低于此值。

DecimalField

科学十进制表示，Python的 `Decimal` 实例。

对应于 `django.db.models.fields.DecimalField` 。

签名： `DecimalField(max_digits, decimal_places, coerce_to_string=None, max_value=None, min_value=None)`

- `max_digits` 数字中允许的最大位数。它必须是 `None` 或大于或等于 `decimal_places` 的整数。
- `decimal_places` 与数字一起存储的小数位数。
- `max_value` ： 验证提供的数字是否不大于此值。
- `min_value` ： 验证提供的数字是否不低于此值。
- `localize` ： 默认为 `False` 。设置为 `True` ，则根据当前区域设置启用输入和输出的本地化。这也将迫使 `coerce_to_string` 设置为 `True` 。请注意，如果已在设置文件中进行了 `USE_L10N=True` 设置，则会启用数据格式设置。
- `rounding` ： 设置量化到配置精度时使用的舍入模式。默认为 `None` 。

五、日期和时间类型字段

DateTimeField

日期和时间表示。

对应于 `django.db.models.fields.DateTimeField` 。

签名： `DateTimeField(format=api_settings.DATETIME_FORMAT, input_formats=None, default_timezone=None)`

- `format` - 时间字符串的格式。如果未指定，则默认为settings中的 `DATETIME_FORMAT` 设置。
- `input_formats` - 用于解析日期的输入格式的字符串列表。如果未指定， `DATETIME_INPUT_FORMATS` 将使用默认设置 `['iso-8601']` 。
- `default_timezone` - 默认时区

`auto_now` 和 `auto_now_add` 模型字段

使用 `ModelSerializer` 或 `HyperlinkedModelSerializer` 序列化器时，请注意，带有 `auto_now=True` 或 `auto_now_add=True` 的模型字段将自动设置 `read_only=True` 参数。如果要覆盖此行为，则需要在序列化类中显式声明一个 `DateTimeField` 字段。例如：

```
class CommentSerializer(serializers.ModelSerializer):
    created = serializers.DateTimeField()

    class Meta:
        model = Comment
```

DateField

日期类型字段。

对应于 `django.db.models.fields.DateField`

签名： `DateField(format=api_settings.DATE_FORMAT, input_formats=None)`

TimeField

时间类型字段。

对应于 `django.db.models.fields.TimeField`

签名： `TimeField(format=api_settings.TIME_FORMAT, input_formats=None)`

DurationField

持续时间长度类型的字段。

对应于 `django.db.models.fields.DurationField`

如果你定义了这种类型的字段，那么在 `validated_data` 变量中将包含一个 `datetime.timedelta` 实例，以 `'[DD] [HH:[MM:]]ss[.uuuuuu]'` 格式的字符串形式表示。

签名： `DurationField(max_value=None, min_value=None)`

- `max_value` 验证提供的持续时间不大于此值。
- `min_value` 验证提供的持续时间不小于此值。

六、选择类型字段

ChoiceField

接受有限选择集中的值的字段。

如果相应的模型字段包含 `choices=…` 参数，则 `ModelSerializer` 会自动生成一个对应的字段。

签名： `ChoiceField(choices)`

- `choices` - 有效值列表或 `(key, display_name)` 元组列表。
- `allow_blank` - 如果设置为 `True` 则应将空字符串视为有效值。如果设置为 `False` 则空字符串被视为无效并将引发验证错误。默认为 `False` 。
- `html_cutoff` - 如果设置，这将是HTML下拉列表标签将显示的最大选择数。
- `html_cutoff_text` - 如果设置，如果在HTML选择下拉列表中截断了最大项目数，则将显示文本指示符。默认为 `"More than {count} items…"`

`allow_blank` 与 `allow_null` 两个参数都可以用于 `ChoiceField` 字段，但建议只使用一个，而不是同时用两个。 `allow_blank` 应是文本选择的首选， `allow_null` 对应数字或其他非文本选择。

MultipleChoiceField

可多选的类型字段。

可以接受零个、一个或多个值的字段。 `to_internal_value` 方法可以返回包含所选值的集合。

签名： `MultipleChoiceField(choices)`

- `choices` - 有效值列表或 `(key, display_name)` 元组列表，必填参数。
- `allow_blank` - 如果设置为 `True` 则应将空字符串视为有效值。如果设置为 `False` 则空字符串被视为无效并将引发验证错误。默认为 `False` 。
- `html_cutoff` - 同上

- `html_cutoff_text` - 同上

正如 `ChoiceField`， `allow_blank` 和 `allow_null` 只能二选一。

七、文件和图片字段

`FileField` 和 `ImageField` 字段仅适用于使用 `MultiPartParser` 或 `FileUploadParser` 解析器的情况。大多数解析器（例如JSON）不支持文件上载。Django原生的FILE_UPLOAD_HANDLERS用于上传文件的处理。

FileField

文件字段。执行Django标准的FileField验证。

对应于 `django.forms.fields.FileField` 。

签名： `FileField(max_length=None, allow_empty_file=False, use_url=UPLOADED_FILES_USE_URL)`

- `max_length` - 指定文件名的最大长度。
- `allow_empty_file` - 指定是否允许空文件。
- `use_url` - 如果设置为 `True` ，则URL字符串将用于输出表示。如果设置为 `False` ，则文件名字符串值将用于输出表示。默认为settings中 `UPLOADED_FILES_USE_URL` 设置的值，除非另有设置。

ImageField

图像字段。将上载的文件内容验证为与已知图像格式匹配。

对应于 `django.forms.fields.ImageField` 。

签名： `ImageField(max_length=None, allow_empty_file=False, use_url=UPLOADED_FILES_USE_URL)`

- `max_length` - 指定文件名的最大长度。
- `allow_empty_file` - 指定是否允许空文件。
- `use_url` - 同上

使用此字段需要提前安装 `Pillow` 包或 `PIL` 包。建议使用 `Pillow` ， 因为 `PIL` 不再维护。

八、复合类型字段

ListField

对象列表的字段类型。

签名：`ListField(child=<A_FIELD_INSTANCE>, min_length=None, max_length=None)`

- `child` - 用于验证列表中对象的字段实例。如果未提供此参数，则不会进行验证。
- `min_length` - 列表包含的元素数量不少于该值。
- `max_length` - 列表包含的元素个数不超过该值。

例如，要验证整数列表，可以使用以下内容：

```
scores = serializers.ListField(
   child=serializers.IntegerField(min_value=0, max_value=100)
)
```

`ListField` 还支持编写可重用的列表字段类。

```
class StringListField(serializers.ListField):
    child = serializers.CharField()
```

现在可以在整个应用程序中重用我们自定义的 `StringListField` 类，而无需为其提供 `child` 参数。

DictField

对象字典的字段类。 `DictField` 中的键始终假定为字符串值。

签名：`DictField(child=<A_FIELD_INSTANCE>)`

- `child` - 同上，但对元素个数没有要求

例如，要创建一个验证字符串到字符串的映射的字段，可以编写如下内容：

```
document = DictField(child=CharField())
```

也可以自定义字段类。例如：

```
class DocumentField(DictField):
    child = CharField()
```

HStoreField

预配置的 `DictField` 。与Django的postgres `HStoreField` 兼容。

签名： `HStoreField(child=<A_FIELD_INSTANCE>)`

- `child` - 用于验证字典中的值的字段实例。默认子字段接受空字符串和空值。

请注意，子字段**必须**是 `CharField` 实例。

JSONField

json字段类，用于验证传入数据是否有效的JSON结构。

签名： `JSONField(binary)`

- `binary` - 如果设置为 `True` 则该字段将输出并验证JSON编码的字符串，而不是原始数据结构。默认为 `False` 。

九、其他类型字段

ReadOnlyField

自读类型，它只返回字段的值而不进行修改。

默认情况下，此字段用于 `ModelSerializer` 包含与属性相关的字段名称而不是模型字段。

签名： `ReadOnlyField()`

例如，如果 `has_expired` 是 `Account` 模型上的属性，则以下序列化程序会自动将其生成为 `ReadOnlyField` ：

```
class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        fields = ('id', 'account_name', 'has_expired')
```

HiddenField

隐藏的字段，它不根据用户输入获取值，而是从默认值或可调用的值中获取值。

签名：`HiddenField()`

例如，要包含始终将当前时间作为序列化程序验证数据的一部分提供的字段：

```
modified = serializers.HiddenField(default=timezone.now)
```

ModelField

可以绑定到任意模型字段的通用字段。

此字段用于`ModelSerializer`对应于自定义模型字段类。一般我们不使用

签名：`ModelField(model_field=<Django ModelField instance>)`

SerializerMethodField

一个只读字段。它通过调用附加到的序列化类上的方法来获取值。可用于将任何类型的数据添加到序列化对象中。

这相当于自定义字段数据。

签名：`SerializerMethodField(method_name=None)`

- `method_name` - 要调用的序列化程序上方法的名称。如果不包含，则默认为`get_<field_name>`。

```python
class MySerializer(serializers.Serializer):

    extra_info = serializers.SerializerMethodField()

    def get_extra_info(self, obj): # 注意这个方法的名字,obj参数表示每一条数据库记录
        return {
            'email': 'xxxx',
            'xxxx': 'xxxx',
            'status':obj.related_filed.some_filed,
        }
```