DRF: Сериализаторы и валидаторы для связанных моделей | Я.Шпора

Для примеров будут использованы связанные модели `Cat` и `Owner`:

```
# cats/models.py
from django.db import models
from django.contrib.auth.models import User


# Расширенная модель пользователя:
class Owner(User):
    first_name = models.CharField(max_length=128)
    last_name = models.CharField(max_length=128)

    def __str__(self):
        return f'{self.first_name} {self.last_name}'


class Cat(models.Model):
    name = models.CharField(max_length=16)
    color = models.CharField(max_length=16)
    birth_year = models.IntegerField()
    owner = models.ForeignKey(
        Owner,
                related_name='cats',
                on_delete=models.CASCADE
        )

    def __str__(self):
        return self.name
```

Если в полях типа `ForeighnKey` указан параметр `related_name`, то это имя можно указывать в полях сериализатора (поле `cats` в сериализаторе

`OwnerSerializer`).

```
class OwnerSerializer(serializers.ModelSerializer):

    class Meta:
        model = Owner
        fields = ('first_name', 'last_name', 'cats')
```

Для связанных полей модели сериализатор по умолчанию применяет тип `PrimaryKeyRelatedField`; этот тип поля в сериализаторе оперирует первичными ключами (id) связанного объекта.

Тип поля StringRelatedField

В модели `Cat` строковое представление объекта задано магическим методом `__str__` модели. В сериализаторе можно вместо первичного ключа получить строковое представление связанной модели.

Это реализуется через поле типа `StringRelatedField`.

```
class Cat(models.Model):
    ...
    owner = models.ForeignKey(
        Owner,
                related_name='cats',
                on_delete=models.CASCADE
        )

    def __str__(self):
        return self.name


class OwnerSerializer(serializers.ModelSerializer):
    cats = serializers.StringRelatedField(many=True, read_only=Tru

    class Meta:
        model = Owner
        fields = ('first_name', 'last_name', 'cats')
```

При описании типа поля переданы аргументы `many=True` и `read_only=True`:

- Для поля `cats` в модели **Owner** установлена связь «один-ко-многим», следовательно, полю `cats` в сериализаторе надо разрешить обработку списка объектов — аргумент `many=True`.
- Поле типа `StringRelatedField` не поддерживает операции записи, поэтому для него указан параметр `read_only=True`.

Вложенный сериализатор

В работе со связанными моделями `Cat` и `Owner` сериализатор `CatSerializer` может получить лишь **ссылку** на связанный объект, но не сам объект. Чтобы получить доступ к объекту и ко всем его полям, можно использовать вложенные сериализаторы.

Продемонстрируем через связанные модели `Achievement` и `Cat` связь «многие-ко-многим» реализована через вспомогательную модель `AchievementCat`:

```
class Achievement(models.Model):
    name = models.CharField(max_length=64)

    def __str__(self):
        return self.name


class Cat(models.Model):
    ...
    # Связь реализована через вспомогательную модель AchievementCa
    achievements = models.ManyToManyField(Achievement, through='Ac

    def __str__(self):
        return self.name


# В этой модели будут связаны id котика и id его достижения
class AchievementCat(models.Model):
    achievement = models.ForeignKey(Achievement, on_delete=models
    cat = models.ForeignKey(Cat, on_delete=models.CASCADE)
```

```
    def __str__(self):
        return f'{self.achievement} {self.cat}'
```

Каждое достижение может принадлежать любому количеству котиков, и каждый котик может обладать любым количеством достижений; это связь **«многие-ко-многим»**.

Сериализатор для модели `Achievement` будет выглядеть так:

```
class AchievementSerializer(serializers.ModelSerializer):

    class Meta:
        model = Achievement
        fields = ('id', 'name')
```

Чтобы в сериализаторе `CatSerializer` получить не **ссылки** на объекты `Achievement`, а объекты целиком — поле `achievements` надо объявить через вложенный сериализатор.

```
...
class CatSerializer(serializers.ModelSerializer):
    owner = serializers.StringRelatedField(read_only=True)
    # Переопределяем поле achievements
    achievements = AchievementSerializer(read_only=True, many=True

    class Meta:
        model = Cat
        fields = ('id', 'name', 'color', 'birth_year', 'owner', 'a
```

Операции записи с вложенными сериализаторами

Вложенные сериализаторы по умолчанию доступны только для чтения. Чтобы настроить сохранение данных, нужно переопределить метод `create()` в сериализаторе.

```
from .models import Achievement, AchievementCat, Cat, Owner

...
```

```
class CatSerializer(serializers.ModelSerializer):
    achievements = AchievementSerializer(many=True)  # Убрали reac
    ...

    class Meta:
        model = Cat
        fields = ('id', 'name', 'color', 'birth_year', 'owner', 'a

    ...

    def create(self, validated_data):
        # Извлечём список 'achievements' из словаря validated_data
        achievements = validated_data.pop('achievements')

        # Создадим нового котика без достижений
        cat = Cat.objects.create(**validated_data)

        # Для каждого достижения из списка достижений...
        for achievement in achievements:
            # ...создадим новую запись или получим существующий эк
            current_achievement, status = Achievement.objects.get_
                **achievement)
            # Помещаем ссылку на каждое достижение во вспомогатель
            # указываем, к какому котику оно относится:
            AchievementCat.objects.create(
                achievement=current_achievement, cat=cat)
        return cat
```

Исходные данные из запроса в сериализаторе

Получить доступ к данным, которые были переданы в сериализатор, можно через словарь `serializer.initial_data`.

Необязательные поля в сериализаторе

Поле сериализатора можно объявить необязательным; для этого устанавливается параметр `required` в значение `False`.

```
class CatSerializer(serializers.ModelSerializer):
    achievements = AchievementSerializer(many=True, required=False
    ...
```

Если есть необязательное поле — код метода `create()` может быть таким:

```
...

    def create(self, validated_data):
        # Если в исходном запросе не передано значение поля achiev
        if 'achievements' not in self.initial_data:
            # ...создаём запись о котике без его достижений:
            cat = Cat.objects.create(**validated_data)
            return cat
        ...
```

Поле SerializerMethodField: обработка данных в сериализаторе

SerializerMethodField — поле для чтения, связанное с определённым методом, в котором будет вычислено значение этого поля. Метод, который будет вызван для поля `<имя_поля>`, по умолчанию должен называться `get_<имя_поля>`.

```
...
import datetime as dt
...

class CatSerializer(serializers.ModelSerializer):
    achievements = AchievementSerializer(many=True)
    age = serializers.SerializerMethodField()

    class Meta:
        model = Cat
        fields = ('id', 'name', 'color', 'birth_year', 'owner', 'a
                  'age')
```

```
    def get_age(self, obj):
        return dt.datetime.now().year - obj.birth_year
```

Переименование полей: параметр source

Сериализатор, унаследованный от `ModelSerializer`, по умолчанию использует те же названия полей, что и в модели, с которой он работает. Эти же имена служат ключами в ответе API.

При необходимости, имя поля можно изменить с помощью параметра **source**:

```
new_field_name = serializers.CharField(source='old_field_name')
```

Валидаторы

Валидация на уровне поля выполняется методами сериализатора; названия методов должны соответствовать шаблону `validate_<имя поля>`. Такие методы автоматически вызываются при получении данных.

```
class CatSerializer(serializers.ModelSerializer):
    achievements = AchievementSerializer(many=True, required=False
    color = serializers.ChoiceField(choices=CHOICES)
    age = serializers.SerializerMethodField()
    owner = serializers.PrimaryKeyRelatedField(read_only=True)

    class Meta:
        model = Cat
        fields = ('id', 'name', 'color', 'birth_year',
                  'achievements', 'owner', 'age')

    def validate_birth_year(self, value):
        year = dt.date.today().year
        if not (year - 40 < value <= year):
            raise serializers.ValidationError('Проверьте год рожде
        return value
```

Валидация на уровне объекта осуществляется с помощью определения метода `validate()` в сериализаторе.

```
class CatSerializer(serializers.ModelSerializer):
    achievements = AchievementSerializer(many=True, required=False
    color = serializers.ChoiceField(choices=CHOICES)
    age = serializers.SerializerMethodField()
    owner = serializers.PrimaryKeyRelatedField(read_only=True)

    class Meta:
        model = Cat
        fields = ('id', 'name', 'color', 'birth_year',
            'achievements', 'owner', 'age')

    ...

    def validate(self, data):
        if data['color'] == data['name']:
            raise serializers.ValidationError(
                'Имя не может совпадать с цветом!')
        return data
```

Встроенные валидаторы для сериализаторов

Для сериализаторов в DRF есть несколько встроенных классов-валидаторов, среди них есть `UniqueValidator` и `UniqueTogetherValidator`.

- **UniqueValidator** обеспечивает проверку уникальности значения поля.

  В моделях такое ограничение описывается параметром `unique=True` для поля.

- **UniqueTogetherValidator** обеспечивает проверку уникальности комбинации. Например, если проверка уникальности идёт по полям `name` и `color` — в базе может быть только одна запись с `name='Мурзик'` и `color='Black'`, пусть даже остальные поля будут у них различаться.

Валидаторы указываются в классе `Meta` сериализатора, в опциональном поле `validators`; значением этого поля должен быть список валидаторов.

```
class CatSerializer(serializers.ModelSerializer):
    achievements = AchievementSerializer(many=True, required=False
    color = serializers.ChoiceField(choices=CHOICES)
```

```
    age = serializers.SerializerMethodField()
    owner = serializers.PrimaryKeyRelatedField(read_only=True)

    class Meta:
        model = Cat
        fields = ('id', 'name', 'color', 'birth_year', 'achievemer
                  'age')
        # У одного владельца не может быть двух котиков с одинаков
        validators = [
            UniqueTogetherValidator(
                queryset=Cat.objects.all(),
                fields=('name', 'owner')
            )
        ]

    ...
```

Скрытые поля и значения по умолчанию

Даже если поле необязательное для заполнения, оно будет использоваться для валидации. Если поле не представлено, валидация будет провалена.

Для такого поля можно указать значение по умолчанию.

Для поля сериализатора значение по умолчанию можно указать двумя способами. На примере поля `owner` сериализатора `CatSerializer`:

1. Переопределить поле `owner`, указав для него тип `HiddenField` (скрытое поле), и передать в него значение по умолчанию `CurrentUserDefault`. `CurrentUserDefault` возвращает объект пользователя, который сделал запрос.

   Поле типа `HiddenField` не принимает данные из запроса, а получает значение по умолчанию и передаёт его в словарь `validated_data` сериализатора.

   ```
   class CatSerializer(serializers.ModelSerializer):
       ...
       owner = serializers.HiddenField(default=serializers.Curren
       ...
   ```

2. Можно переопределить поле `owner`, указав ему любой тип из библиотеки DRF, но обязательно передать параметры `read_only=True` и `default=<значение_по_умолчанию>` (для поля `owner` это будет `default=serializers.CurrentUserDefault()`).

```
class CatSerializer(serializers.ModelSerializer):
    ...
    owner = serializers.PrimaryKeyRelatedField(
        read_only=True, default=serializers.CurrentUserDefault
    ...
```

Любой из этих подходов решит проблему; записи будут добавляться в БД, валидация `UniqueTogetherValidator` отработает корректно. Второй вариант предпочтительнее: код получается более понятным.

Практикум