DRF

DRF（Django Rest Framework）是可以快速基于Restful开发的Django应用的插件，功能非常多，被广泛应用。

安装

```
$ pip install djangorestframework
```

Django 需要 2.2+

注册

settings.py中增加

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

序列化器

采用前后端分离后

- 序列化：后端发给前端的数据就是Json，核心就是把数据结构序列化成Json发给浏览器端
  - 字典 => Json字符串
  - 更进一步，实例 => 字典，字典交给Response类序列化成Json
- 反序列化：前端发给后端的数据依然是request请求，但是提交的数据是Json，需要反序列化
  - Json字符串 => 字典

一般建议使用字典包封装数据。

序列化器类

参考 https://www.django-rest-framework.org/api-guide/serializers/

rest_framework.serializers.BaseSerializer是序列化器类的基类

```
class BaseSerializer(Field):
    def __init__(self, instance=None, data=empty, **kwargs):
        self.instance = instance # 如果送第一个参数表示实例，
        if data is not empty:
            self.initial_data = data #
        self.partial = kwargs.pop('partial', False)
        self._context = kwargs.pop('context', {})
        kwargs.pop('many', None)   # many表示数据是多个还是一个
        super().__init__(**kwargs)

    def is_valid(self, raise_exception=False):
        """反序列化时校验数据，也就是对浏览器端提交的数据进行验证"""

    @property
```

```
    def data(self):
        if hasattr(self, 'initial_data') and not hasattr(self,
'_validated_data'):
        # 如果有浏览器提交数据但没有校验过数据，先调用is_valid()校验
            msg = (
                'When a serializer is passed a `data` keyword argument you '
                'must call `.is_valid()` before attempting to access the '
                'serialized `.data` representation.\n'
                'You should either call `.is_valid()` first, '
                'or access `.initial_data` instead.'
            )
            raise AssertionError(msg)

        if not hasattr(self, '_data'): # 没有_data立即开始计算
            if self.instance is not None and not getattr(self, '_errors',
None):
            # 序列化。有实例，没有错误
                self._data = self.to_representation(self.instance)
            elif hasattr(self, '_validated_data') and not getattr(self,
'_errors', None):
            # 反序列化。is_valid()校验后的数据(提交的数据被验证过了)且无错误
                self._data = self.to_representation(self.validated_data)
            else: # 否则?
                self._data = self.get_initial()
        return self._data

    @property
    def validated_data(self):
        """校验后获取反序列化的数据"""

class Serializer(BaseSerializer, metaclass=SerializerMetaclass):
    @property
    def data(self):
        ret = super().data
        return ReturnDict(ret, serializer=self) # OrderedDict
```

序列化器原理

序列化器

在应用目录下构建serializers.py，根据Model类Employee编写EmpSerializer。

需要什么字段就在EmpSerializer中定义什么属性

```
from rest_framework import serializers
from .models import Employee

class EmpSerializer(serializers.Serializer):
    emp_no = serializers.IntegerField()
    birth_date = serializers.DateField()
    first_name = serializers.CharField(max_length=14) # max_length反序列化验证
用
    last_name = serializers.CharField(max_length=16)
    gender = serializers.ChoiceField(choices=Employee.Gender.choices)
    hire_date = serializers.DateField()
```

```
11
12 print('~' * 30)
13 print(EmpSerializer())
14 print('~' * 30)
```

序列化

```
1  import os
2  import django
3
4  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
5  django.setup(set_prefix=False)
6  # 所有测试代码，都要在上面4行之下
7
8  from employee.models import Employee
9  from employee.serializers import EmpSerializer
10
11 emgr = Employee.objects
12 # 单个对象
13 emp = emgr.get(pk=10010)
14 print(emp)
15 # 给实例准备序列化为Json字符串
16 serializer = EmpSerializer(instance=emp)
17 data = serializer.data # 序列化
18 print(type(data), data)
19
20 # 查询集，多个实例，使用many
21 emps = emgr.filter(pk__gt=10017)
22 print(*emps)
23 serializer = EmpSerializer(emps, many=True)
24 data = serializer.data
25 print(type(data), data) # 列表套着字典
```

反序列化

```
1  import os
2  import django
3
4  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
5  django.setup(set_prefix=False)
6  # 所有测试代码，都要在上面4行之下
7
8  from employee.models import Employee
9  from employee.serializers import EmpSerializer
10
11 emgr = Employee.objects
12 # POST方法提交上来的数据被request封装为字典
13 data = {
14     'emp_no': 10010, 'birth_date': '1963-06-01',
15     'first_name': 'Duangkaew', 'last_name': 'Piveteau',
16     'gender': '2', 'hire_date': '1989-08-24'
17 } # 注意，这是字典数据是data，不是Employee的实例
```

```python
# gender的2故意使用了字符串

serializer = EmpSerializer(data=data)
# print(serializer.data) # 没有调用.is_valid()直接报错，第一步需要验证
validated = serializer.is_valid(raise_exception=True)
# is_valid方法调用内部成功会把数据放在_validated_data属性
# raise_exception验证失败是否抛异常
print(validated)
print(serializer.data)
```

校验

对浏览器端提交的数据一定要校验，所以，**校验**是入库前必须要做的。

字段选项参数校验

- 字符串长度
  - 长度测试min_length和max_length

```
t1 = serializers.CharField(label="长度限制和必须", min_length=4, max_length=8)

测试 "t1": "ab"
错误 {'t1': [ErrorDetail(string='Ensure this field has at least 4 characters.', code='min_length')]}
注意，异常返回的[]列表，说明一个字段可以有多个校验器，这和前端开发校验一样
```

- 字段值是否必须
  - 字段默认require=True，也就是说必须提供

```
t1 = serializers.CharField(label="长度限制和必须", min_length=4, max_length=8)

测试 Post的数据不提供t1
错误 {'t1': [ErrorDetail(string='This field is required.', code='required')]}
```

- 只读
  - read_only不校验该字段，只会出现在序列化中，仅仅只能给人看
  - read_only和require不可以同时为True
  - read_only=True表示序列化时可以序列化t2这个属性的数据，反序列化不使用它，提供了值也不校验也不输出，serializer.data输出结果中没有t2

```
t2 = serializers.CharField(read_only=True)
```

- 只写
  - write_only=True，表示反序列化用，要校验。不会被序列化，不返回给浏览器

```
t3 = serializers.CharField(write_only=True)
```

对**序列化器**略作修改做测试，这个测试主要测试的是反序列化的校验

```
from rest_framework import serializers
from .models import Employee

class EmpSerializer(serializers.Serializer):
    emp_no = serializers.IntegerField()
    birth_date = serializers.DateField()
    first_name = serializers.CharField(max_length=14)
    last_name = serializers.CharField(max_length=16)
    gender = serializers.ChoiceField(choices=Employee.Gender.choices)
    hire_date = serializers.DateField()
    t1 = serializers.CharField(label="长度限制和必须", min_length=4,
max_length=8)
    t2 = serializers.CharField(read_only=True)
    t3 = serializers.CharField(write_only=True)
```

```
import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
django.setup(set_prefix=False)
# 所有测试代码，都要在上面4行之下

from employee.models import Employee
from employee.serializers import EmpSerializer

emgr = Employee.objects
# 单个对象
emp = emgr.get(pk=10010)
print(emp.__dict__)
# 为序列化凑几个字段
emp.t1 = 't1'
emp.t2 = 't2'
emp.t3 = 't3'

# 给实例准备序列化为Json字符串
serializer = EmpSerializer(instance=emp)
data = serializer.data # 序列化
print(data) # 字典
```

序列化结果如下：

- 序列化时，不校验t1
- t2是read_only，所以有它
- t3是write_only，不参与序列化，所以没有它

```
{'emp_no': 10010, 'birth_date': '1963-06-01', 'first_name': 'Duangkaew',
'last_name': 'Piveteau', 'gender': 2, 'hire_date': '1989-08-24', 't1': 't1',
't2': 't2'}
```

```
import os
import django
```

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
django.setup(set_prefix=False)
# 所有测试代码，都要在上面4行之下

from employee.models import Employee
from employee.serializers import EmpSerializer

emgr = Employee.objects

# POST方法提交上来的数据被request封装为字典
data = {
    'emp_no': 10010, 'birth_date': '1963-06-01',
    'first_name': 'Duangkaew', 'last_name': 'Piveteau',
    'gender': "2", 'hire_date': '1989-08-24',
    't1':'abcd',
    't2':'t2++', # t2给不给无所谓
    't3':'t3=='
} # 注意，这是字典数据是data，不是Employee的实例
serializer = EmpSerializer(data=data)
# print(serializer.data) # 没有调用.is_valid()直接报错，第一步需要验证
validated = serializer.is_valid(raise_exception=True)
# is_valid方法调用内部成功会把数据放在_validated_data属性
# raise_exception验证失败是否抛异常
print(validated)
print(serializer.data)
```

反序列化结果如下：

- 校验t1
- t2是read_only，所以没有它
- t3是write_only，反序列化必须有它，还要校验它

```
{'emp_no': 10010, 'birth_date': datetime.date(1963, 6, 1), 'first_name':
'Duangkaew', 'last_name': 'Piveteau', 'gender': 2, 'hire_date':
datetime.date(1989, 8, 24), 't1': 'abcd', 't3': 't3=='}
```

字段级校验器

参考 https://www.django-rest-framework.org/api-guide/serializers/#field-level-validation

这是针对某一个字段的校验，在序列化器类中增加validate_<字段名>方法，校验失败抛异常`serializers.ValidationError`，成功返回正确的值

```
from rest_framework import serializers
from .models import Employee

class EmpSerializer(serializers.Serializer):
    emp_no = serializers.IntegerField()
    birth_date = serializers.DateField()
    first_name = serializers.CharField() # 使用单独字段的校验器
    last_name = serializers.CharField(max_length=16)
    gender = serializers.ChoiceField(choices=Employee.Gender.choices)
    hire_date = serializers.DateField()

```

```
    def validate_first_name(self, value):
        print(value, '+++++')
        if 4 <= len(value) <= 14:
            return value
        raise serializers.ValidationError('The length must be between 4 and
14')
```

还有一种写法，使用字段选项参数`validators=[validator, ...]`

```
from rest_framework import serializers
from .models import Employee

def validate_fn(value):
    print(value, '+++++')
    if 4 <= len(value) <= 14:
        return value
    raise serializers.ValidationError('The length must be between 4 and 14')

class EmpSerializer(serializers.Serializer):
    emp_no = serializers.IntegerField()
    birth_date = serializers.DateField()
    first_name = serializers.CharField(validators=[validate_fn]) # 使用校验器
    last_name = serializers.CharField(max_length=16)
    gender = serializers.ChoiceField(choices=Employee.Gender.choices)
    hire_date = serializers.DateField()
```

除非有复用的必要，不要把校验器定义在外部。

```
import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
django.setup(set_prefix=False)
# 所有测试代码，都要在上面4行之下

from employee.models import Employee
from employee.serializers import EmpSerializer

emgr = Employee.objects

# POST方法提交上来的数据被request封装为字典
data = {
    'emp_no': 10010, 'birth_date': '1963-06-01',
    'first_name': 'Duangkaew', 'last_name': 'Piveteau',
    'gender': "2", 'hire_date': '1989-08-24',
} # 注意，这是字典数据是data，不是Employee的实例
serializer = EmpSerializer(data=data)
# print(serializer.data) # 没有调用.is_valid()直接报错，第一步需要验证
validated = serializer.is_valid(raise_exception=True)
# is_valid方法调用内部成功会把数据放在_validated_data属性
# raise_exception验证失败是否抛异常
print(validated)
print(serializer.data)
```

对象级校验器

参考 https://www.django-rest-framework.org/api-guide/serializers/#object-level-validation

对象级校验器就是对实例所有字段数据的校验

```
from rest_framework import serializers
from .models import Employee


class EmpSerializer(serializers.Serializer):
    emp_no = serializers.IntegerField()
    birth_date = serializers.DateField()
    first_name = serializers.CharField() # 使用校验器
    last_name = serializers.CharField(max_length=16)
    gender = serializers.ChoiceField(choices=Employee.Gender.choices)
    hire_date = serializers.DateField()

    def validate_first_name(self, value):
        if 4 <= len(value) <= 14:
            return value
        raise serializers.ValidationError('The length must be between 4 and 14')

    def validate(self, data):
        print(data, '====') # 字典
        last_name = data.get('last_name', '')
        if len(last_name) < 2:
            raise serializers.ValidationError('The length must be greater than 1')
        return data
```

入库

参考 https://www.django-rest-framework.org/api-guide/serializers/#saving-instances

校验后的数据是安全的，可以写入数据库。

原理

BaseSerializer中定义了save方法

- save()之前一定要.is_valid()
- BaseSerializer(instance=None, data=empty)，无实例就是新增，调用create；有实例就是更新，调用update。最终返回实例
- 在BaseSerializer中，create、update是未实现的抽象方法，Serializer也没有实现这2个方法。言下之意，就是需要用户自己实现。

```
from rest_framework import serializers
from .models import Employee


class EmpSerializer(serializers.Serializer):
    emp_no = serializers.IntegerField()
    birth_date = serializers.DateField()
    first_name = serializers.CharField(max_length=14)
```

```
    last_name = serializers.CharField(max_length=16)
    gender = serializers.ChoiceField(choices=Employee.Gender.choices)
    hire_date = serializers.DateField()

    # 基类只负责调用create或update，但未实现
    def create(self, validated_data):
        # validated_data校验过的数据
        # 管理器实现了create方法完成数据新增
        return Employee.objects.create(**validated_data)

    def update(self, instance:Employee, validated_data):
        instance.first_name = validated_data.get('first_name',
instance.first_name)
        instance.last_name = validated_data.get('last_name',
instance.last_name)
        instance.gender = validated_data.get('gender', instance.gender)
        instance.save()
        return instance
```

增

```
import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
django.setup(set_prefix=False)
# 所有测试代码，都要在上面4行之下

from employee.models import Employee
from employee.serializers import EmpSerializer

emgr = Employee.objects

# POST方法提交上来的数据被request封装为字典
data = {
    'emp_no': 10021, 'birth_date': '1963-06-01',
    'first_name': 'san', 'last_name': 'zhang',
    'gender': "1", 'hire_date': '1989-08-24',
} # 注意，这是字典数据是data，不是Employee的实例

# 只有data没有实例，是新增
serializer = EmpSerializer(data=data)
validated = serializer.is_valid(raise_exception=True)
#print(serializer.data) # save前不能调用data，只能使用validated_data查看
serializer.save()
print(serializer.data)
```

改

先查再改：先查返回结果填充实例，然后根据这个实例修改数据

```
import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'salary.settings')
```

```
django.setup(set_prefix=False)
# 所有测试代码，都要在上面4行之下

from employee.models import Employee
from employee.serializers import EmpSerializer

emgr = Employee.objects
emp = emgr.get(pk=10021)
print(emp)
# POST方法提交上来的数据被request封装为字典
data = {
    'emp_no': 10021, 'birth_date': '1963-06-01',
    'first_name': 'si', 'last_name': 'li',
    'gender': 1, 'hire_date': '1989-08-24',
} # 注意，这是字典数据是data，不是Employee的实例

# 有实例，有data，是更新
serializer = EmpSerializer(emp, data=data)
validated = serializer.is_valid(raise_exception=True)
#print(serializer.data) # save前不能调用data，只能使用validated_data查看
serializer.save()
print(serializer.data)
```

总结

- 序列化
  - 查库：查询数据库得到单个实例或多个实例
  - 构造序列化器实例：构造序列化器实例
  - 输出：使用data属性获取字典
- 反序列化
  - 反序列化：将Json数据反序列化为字典
  - 构造序列化器实例：使用字典构造序列化器实例
  - **校验**：调用is_valid方法校验各个字段的值
  - **入库**：调用save方法

从前面的知识点可以明白DRF的基本原理，也感觉到了Serializer类比较简陋，很多东西都需要自己写代码。

可以观察到，序列化器字段定义和Model类中的严重重复，能否利用Model类来简化?

所有的增、改都差不多，能否把create、update也实现了?