Ask Company

리액트와 함께 장고 시작하기 / 장고 DRF

JSON 직렬화

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

직렬화 (Serialization)

모든 프로그래밍 언어의 통신에서 데이터는 필히 문자열로 표현되어야만 합니다.

송신자 : 객체를 문자열로 변환하여, 데이터 전송 → 직렬화

수신자 : 수신한 문자열을 다시 객체로 변환하여, 활용 → 비직렬화

각 언어에서 모두 지원하는 직렬화 포맷 (JSON, XML 등) 도 있고, 특정 언어에서만 지원하는 직렬화 포맷 (파이썬은 Pickle) 이 있습니다.

JSON 변환

데이터를 같아도, 응답형식이 다를 수 있습니다.

보통의 웹 세상에서는 …

- GET 요청에 대해 HTML 포맷으로 응답
- POST 요청을 application/x-www-form-urlencoded 인코딩 혹은 multipart/form-data 인코딩으로 요청하고, HTML 포맷으로 응답

요즘의 API 서버에서는 대개 JSON 인코딩된 요청/응답

JSON 포맷과 PICKLE 포맷 (1/2)

JSON 포맷

- 다른 언어/플랫폼과 통신할 때 주로 사용.
- 표준 라이브러리 json 제공
- pickle에 비해 직렬화를 지원하는 데이터타입의 수가 적지만, 커스텀 Rule 지정도 가능

PICKLE 포맷

- 파이썬 전용 포맷으로서 파이썬 시스템끼리만 통신할 때 사용 가능
- 표준 라이브러리 pickle 제공 → json 라이브러리가 유사한 사용 방법
- 주의) 파이썬 버전 특성을 타는 경우가 있습니다.

JSON 포맷과 PICKLE 포맷 (2/2)

```python
import json

# 데이터 준비
post_list = [
    {'message': 'hello askdjango'},
]

# 직렬화
json_string = json.dumps(post_list)
print(json_string)
```

```
'[{"message": "hello askdjango"}]'
```

```python
# 비직렬화 → 다시 객체
print(json.loads(json_string))
```

```python
import pickle

# 데이터 준비
post_list = [
    {'message': 'hello askdjango'},
]

# 직렬화
pickle_bytes = pickle.dumps(post_list)
print(pickle_bytes)
```

```
b'\x80\x03]q\x00}q\x01X\x07\x00\x00\x00messageq\x02X\x0f\x00\x00\x00hello askdjangoq\x03sa.'
```

```python
# 비직렬화 → 다시 객체
print(pickle.loads(pickle_bytes))
```

TypeError: Object of type "???" is not JSON serializable

```
쉘> python3 manage.py shell

>>> import json
>>> from django.contrib.auth import get_user_model
>>> User = get_user_model()
>>> json.dumps(User.objects.first())

TypeError: Object of type 'User' is not JSON serializable
```

json/pickle 모두 파이썬 기본 라이브러리
→ 장고 타입 (Model/QuerySet 등) 에 대해서는 직렬화 Rule이 없습니다.

DjangoJSONEncoder를 통해 추가로 부여된 Rule

다음 타입에 대한 Rule을 추가로 구현

- datetime.datetime, datetime.date, datetime.time, datetime.timedelta
- decimal.Decimal, uuid.UUID, Promise

```
import json
from django.core.serializers.json import DjangoJSONEncoder
from django.contrib.auth import get_user_model

qs = get_user_model().objects.all()
json_string = json.dumps(qs, cls=DjangoJSONEncoder)
print(json_string)
```

역시 동일한 오류

https://github.com/django/django/blob/2.2.3/django/core/serializers/json.py#L76

어떻게 해결할 수 있을까? → 직접 변환

```
data = [
    {'id': post.id, 'title': post.title, 'content': post.content}
    for post in Post.objects.all()]

json.dumps(data, cls=DjangoJSONEncoder, ensure_ascii=False)
```

어떻게 해결할 수 있을까? → 직접 변환 Rule 지정하기

```python
from django.core.serializers.json import DjangoJSONEncoder
from django.db.models.query import QuerySet

# 커스텀 JSON Encoder를 정의
class MyJSONEncoder(DjangoJSONEncoder):
    def default(self, obj):
        if isinstance(obj, QuerySet):
            return tuple(obj)
        elif isinstance(obj, Post):
            return {'id': obj.id, 'title': obj.title, 'content': obj.content}
        elif hasattr(obj, 'as_dict'):
            return obj.as_dict()
        return super().default(obj)

data = Post.objects.all()

# 직렬화할 때, 직렬화를 수행해줄 JSON Encoder를 지정해줍니다.
json.dumps(data, cls=MyJSONEncoder, ensure_ascii=False)
```

DRF에서도 커스텀 JSON Encoder를 사용합니다.

rest_framework.renderer.JSONRender

rest_framework/utils/encoders.py의 JSONEncoder를 통한 직렬화

- 장고의 DjangoJSONEncoder를 상속받지 않고, json.JSONEncoder 상속을 통해 구현
- datetime.datetime/date/time/timedelta, decimal.Decimal, uuid.UUID, six.binary_type
- __getitem__ 속성을 지원할 경우 dict(obj) 변환
- __iter__ 속성을 지원할 경우, tuple 변환
- QuerySet 타입일 경우, tuple 변환
- .tolist 속성을 지원할 경우. obj.tolist() 반환

Model 타입은 미지원 → ModelSerializer를 통한 변환

https://github.com/encode/django-rest-framework/blob/3.11.0/rest_framework/utils/encoders.py

rest_framework.renderer.JSONRenderer

json.dumps에 대한 래핑 클래스 → 보다 편리한 직렬화 지원

UTF8 인코딩도 추가로 수행

```
from rest_framework.renderers import JSONRenderer

data = Post.objects.all()
JSONRenderer().render(data)
```

하지만 Model에 대한 변환 Rule은 아직 없어요.

ModelSerializer를 통한 JSON 직렬화

Serializer/ModelSerializer은 Form/ModelForm와 유사

→ 역할 면에서 Serializer는 POST요청만 처리하는 Form

장고 Form/ModelForm	DRF Serializer/ModelSerializer
폼 필드 지정 혹은 모델로부터 읽어오기	
Form 태그가 포함된 HTML을 생성	Form 데이터가 포함된 JSON 문자열을 생성
입력된 데이터에 대한 유효성 검사 및 획득	

ModelForm 유사한 ModelSerializer

```
from rest_framework.serializers import ModelSerializer

# Post모델에 대한 ModelSerializer 정의
class PostModelSerializer(ModelSerializer):
    class Meta:
        model = Post
        fields = '__all__'

post = Post.objects.first()  # Post 타입

serializer = PostModelSerializer(post)
serializer.data              # ➔ ReturnDict 타입
```

ReturnDict 타입

serializer.data의 데이터 타입

OrderDict을 상속받았으며, serializer를 키워드 인자로 받습니다.

```
class ReturnDict(OrderedDict):
    def __init__(self, *args, **kwargs):
        self.serializer = kwargs.pop('serializer')
        super().__init__(*args, **kwargs)
        # 생략
```

ModelSerializer의 QuerySet 변환 지원

```
qs = Post.objects.all()
serializer = PostModelSerializer(qs, many=True)

serializer.data  # list와 OrderedDict의 조합
```

Model 객체에 대해서는 필히 many=False 지정 (디폴트)
QuerySet 객체에 대해서는 필히 many=True 지정

→ 지정이 맞지 않으면 변환 에러 발생 (TypeError, AttributeError)

```
# 파이썬 기본 JSON 변환 사용 활용
import json
json_str_string = json.dumps(serializer.data, ensure_ascii=False)


# DRF에서 지원하는 JSON 변환 활용 → 변환 Rule이 추가된 Encoder
from rest_framework.renderers import JSONRenderer
json_utf8_string = JSONRenderer().render(serializer.data)
```

Serializer를 통한 Model Instance → OrderedDict

```python
class Serializer(BaseSerializer, metaclass=SerializerMetaclass):
    # ...
    def to_representation(self, instance):
        """
        Object instance -> Dict of primitive datatypes.
        """
        ret = OrderedDict()
        fields = self._readable_fields

        for field in fields:
            try:
                attribute = field.get_attribute(instance)
            except SkipField:
                continue

            # We skip `to_representation` for `None` values so that fields do
            # not have to explicitly deal with that case.
            #
            # For related fields with `use_pk_only_optimization` we need to
            # resolve the pk value.
            check_for_none = attribute.pk if isinstance(attribute, PKOnlyObject) else attribute
            if check_for_none is None:
                ret[field.field_name] = None
            else:
                ret[field.field_name] = field.to_representation(attribute)

        return ret
```

https://github.com/encode/django-rest-framework/blob/3.11.0/rest_framework/serializers.py#L507

View에서의 JSON 응답

장고 기본 View에서의 HttpResponse JSON 응답

모든 View는 HttpResponse 타입의 응답을 해야만 합니다.

일반적으로 다음 2가지 방법

1. 직접 json.dumps를 통해 직렬화된 문자열을 획득하여 HttpResponse를 통해 응답
2. 1번을 정리하여 JsonResponse 지원 → 내부적으로 json.dumps를 사용하며 DjangoJSONEncoder가 디폴트 지정

JsonResponse에서 QuerySet을 JSON 직렬화

이전에 정의한 MyJSONEncoder를 활용해봅시다.

```
qs = Post.objects.all()

# JsonResponse 생성자의 각종 인자 나열
encoder = MyJSONEncoder
safe = False                                    # True: data가 dict일 경우, False: dict이 아닐 경우
json_dumps_params = {'ensure_ascii': False}
kwargs = {}                                     # HttpResponse에 전해지는 Keyword 인자


from Django.http import JsonResponse

response = JsonResponse(qs, encoder, safe, json_dumps_params, **kwargs)
```

DRF를 통한 JSON 응답

DRF를 통한 HttpResponse JSON 응답

DRF Response 활용

```
qs = Post.objects.all()

serializer = PostModelSerializer(qs, many=True)

from rest_framework.response import Response
response = Response(serializer.data)   # Content-Type: text/html 디폴트 지정
```

Response에서는 "JSON 직렬화"가 Lazy하게 동작합니다.

실제 응답 생성 시에 .rendered_content 속성에 접근하며, 이때 변환이 이뤄집니다.

Response와 APIView

DRF의 모든 뷰는 APIView를 상속받습니다.

APIView를 통해 Response에 다양한 속성이 지정됩니다.

```
from rest_framework.views import APIView

renderer_cls = APIView.renderer_classes[0]
renderer_obj = renderer_cls()
response.accepted_renderer = renderer_obj                # JSON 변환을 위한 JSONRenderer 인스턴스

response.accepted_media_type = renderer_obj.media_type   # 'application/json'
response.renderer_context = {'view': None, 'args': (), 'kwargs': {}, 'request': None}

response  # <Response status_code=200, "application/json">
```

실제 DRF Serializer 활용

다음과 같이 간결하게 사용

```
from rest_framework import generics

class PostListAPIView(generics.ListAPIView):
    queryset = Post.objects.all()
    serializer_class = PostModelSerializer


post_list = PostListAPIView.as_view()
```

포스팅 조회 응답에 username 응답을 할려면? (1/2)

author = FK(User) 필드가 있을 때, Serializer에서는 FK 키값으로 응답

serializer.ReadOnlyField를 통해 FK의 필드값을 읽어올 수 있습니다.

```
from rest_framework import serializers
from .models import Post

class PostSerializer(serializers.ModelSerializer):
    username = serializers.ReadOnlyField(source='author.username')   # 추가

    class Meta:
        model = Post
        fields = ['pk', 'username', 'title', 'cotnent']              # 'username' 추가
```

포스팅 조회 응답에 username 응답을 할려면? (2/2)

StringRelatedField, SlugRelatedField 등 뿐만 아니라, 중첩된 Serializer를 통해서도 구현 가능

```
from django.contrib.auth import get_user_model
from rest_framework import serializers
from .models import Post

class AuthorSerializer(serializers.ModelSerializer):
    class Meta:
        model = get_user_model()
        fields = ['username']

class PostSerializer(serializers.ModelSerializer):
    author = AuthorSerializer()

    class Meta:
        model = Post
        fields = '__all__'
```

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

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

-

Ask Company