# **EP 02 - JSON 직렬화**

모든 프로그래밍 언어의 통신에서 데이터는 필히 문자열로 표현되어야만 합니다.

* 송신자 : 객체를 문자열로 변환하여, 데이터 전송 => 이를 직렬화 (Serialization) 라고 합니다.
* 수신자 : 수신한 문자열을 다시 객체로 변환하여, 활용 => 이를 비직렬화 혹은 역직렬화 (Deserialization) 라고 합니다.

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

## **JSON**

보통의 웹애플리케이션에서는 일반적으로 웹브라우저가 주 클라이언트 프로그램이기 때문에, 주로 **HTML**포맷으로 통신을 합니다. 그런데, 최근의 API 서버에는 대개 **JSON 포맷**으로 통신을 수행합니다. 그렇기에 항상 장고 API 뷰 함수에서는 최종적으로 JSON포맷의 문자열 응답을 해야 합니다.

* JSON 포맷 : 다른 언어/플랫폼과 통신할 때 주로 사용합니다. 표준 라이브러리 **json**이 제공됩니다.
  + pickle에 비해 직렬화를 지원하는 데이터타입의 수가 적습니다. 공통 데이터타입에 한해서만 직렬화를 지원합니다.
* PICKLE 포맷 : 파이썬 전용 포맷으로서 파이썬 시스템끼리 통신할 때 사용합니다만, 최근 파이썬끼리 통신에 json포맷도 많이 사용합니다. 표준 라이브러리 **pickle**이 제공됩니다.
  + JSON에서 지원하지않는 파이썬 데이터타입을 지원합니다.
  + 파이썬 버전 특성을 탑니다.

다음과 같은 파이썬 객체가 있습니다.

In [1]:

post\_list = [  
 {'message': 'hello askdjango'},  
]

다음과 같이 JSON 포맷의 문자열로 직렬화를 할 수 있습니다.

In [2]:

**import** **json**  
  
json\_string = json.dumps(post\_list)  
json\_string

Out[2]:

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

거꾸로 비직렬화도 가능합니다.

In [3]:

json.loads(json\_string)

Out[3]:

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

JSON 직렬화와 유사한 방식으로, PICKLE 포맷의 문자열로 직렬화를 할 수 있습니다.

In [4]:

**import** **pickle**  
  
pickle\_data = pickle.dumps(post\_list)  
pickle\_data

Out[4]:

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

이 역시 비직렬화가 가능합니다

In [5]:

pickle.loads(pickle\_data)

Out[5]:

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

json/pickle 라이브러리는 파이썬 표준 라이브러리로서 파이썬 표준 데이터타입에 대한 직렬화/비직렬화를 수행해줍니다. 이는 파이썬 표준 데이터타입에 대해서는 각각의 타입에 대해서 직렬화/비직렬화 룰을 파이썬이 지원해주고 있기 때문입니다. 하지만 장고 Model/QuerySet과 같은 **파이썬 언어 외부타입**에 대해서는 **파이썬의 json 모듈**은 직렬화/비직렬화 Rule을 모르기에 직렬화가 불가합니다.

이미 만들어둔 장고 프로젝트가 있으시다면, 장고 쉘을 통해 확인해보실 수 있습니다. User 모델 인스턴스에 대해 JSON직렬화를 수행했는데, **TypeError**가 발생했으며 **"Objects of type 'User' is not JSON serializable."** 메세지가 나옵니다.

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

이제 장고의 데이터타입에 대해 JSON 직렬화를 수행하는 방법에 대해서 살펴보겠습니다.

## **Django 프로젝트 기본 셋업**

본 에피소드를 시작하기에 앞서, Jupyter Notebook을 통해 직렬화 연습을 해보기 위해, [Jupyter Notebook에서 Django 프로젝트 세팅해서 모델 돌려보기](https://nomade.kr/doc/django/jupyter-notebook%EC%97%90%EC%84%9C-django-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EC%84%B8%ED%8C%85%ED%95%B4%EC%84%9C-%EB%AA%A8%EB%8D%B8-%EB%8F%8C%EB%A0%A4%EB%B3%B4%EA%B8%B0/) 내역을 먼저 수행해주세요. 해당 내역을 잘 수행하셨다면, 다음 코드처럼 Post모델을 통해 DB 쿼리하실 수 있어요.

설명은 해당 포스팅에 잘 나와있구요. 코드만 모아서 한 번에 실행해보겠습니다. :)

In [1]:

*# 최소한의 settings 설정*  
**import** **django**  
**import** **os**  
  
SECRET\_KEY = 'askdjango' *# 임의 문자열*  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.sqlite3',  
 'NAME': ':memory:',  
 }  
}  
ROOT\_URLCONF = '\_\_main\_\_'  
  
urls = []  
  
os.environ['DJANGO\_SETTINGS\_MODULE'] = '\_\_main\_\_'  
  
django.setup()  
  
  
*# 모델 정의*  
**from** **django.db** **import** models  
  
**class** **Post**(models.Model):  
 title = models.CharField(max\_length=100)  
 content = models.TextField()  
 created\_at = models.DateTimeField(auto\_now\_add=**True**)  
 updated\_at = models.DateTimeField(auto\_now=**True**)  
   
 **class** **Meta**:  
 app\_label = 'api' *# 앱이 따로 없으므로, app label을 필히 지정해줘야합니다.*  
   
 **def** \_\_str\_\_(self):  
 **return** self.title  
  
  
*# DB TABLE 생성*  
**from** **django.db** **import** connection  
  
table\_name = Post.\_meta.db\_table  
  
**with** connection.cursor() **as** cursor:  
 cursor.execute('''  
CREATE TABLE "**{}**"  
 ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
 "title" varchar(100) NOT NULL,  
 "content" text NOT NULL,  
 "created\_at" datetime NOT NULL,  
 "updated\_at" datetime NOT NULL);  
'''.format(table\_name))  
  
  
*# 데이터 추가*  
Post.objects.create(  
 title='횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭',  
 content='교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.')  
  
Post.objects.create(  
 title="'디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?",  
 content='옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.')  
  
Post.objects.create(  
 title='저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는',  
 content='늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?')

Out[1]:

<Post: 저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는>

자. 이제 DB에서 쿼리도 잘 됩니다. :D

In [2]:

Post.objects.all()

Out[2]:

<QuerySet [<Post: 횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭>, <Post: '디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?>, <Post: 저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는>]>

In [5]:

**for** post **in** Post.objects.filter():  
 print(post.id, post.title, ':', len(post.content), '글자')

1 횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭 : 90 글자  
2 '디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는? : 42 글자  
3 저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는 : 121 글자

## **장고의 JSON 직렬화**

장고에서는 파이썬 표준 라이브러리 json 모듈을 그대로 쓰지 않고, django/core/serizliers/json.py의 DjangoJSONEncoder 클래스를 통한 직렬화를 수행합니다.

DjangoJSONEncoder는 json.JSONEncoder를 상속받았으며, 다음 타입에 대한 직렬화를 추가로 지원합니다.

* datetime.datetime
* datetime.date
* datetime.time
* datetime.timedelta
* decimal.Decimal, uuid.UUID

그런데, 이는 파이썬 기본 데이터 타입에 대한 직렬화가 추가되었을 뿐, 장고 데이터타입인 **QuerySet과 Model 인스턴스에 대한 직렬화는 지원하지 않습니다**. 장고는 웹 애플리케이션을 만들기 위한 웹프레임워크이고 웹 애플리케이션 개발에서는 JSON직렬화할 일이 적긴 합니다. 그렇지만 기본에서 제공해주면 좋았을 텐데요 ... 아쉽습니다. 이 가려운 부분을 djangorestframework가 긁어줍니다. :D 이는 뒤에서 살펴보구요.

우선 장고 기본에서 제공해주는 DjangoJSONEncoder를 실행해봅시다.

In [9]:

**import** **json**  
**from** **django.core.serializers.json** **import** DjangoJSONEncoder

In [6]:

data = Post.objects.all()  
data

Out[6]:

<QuerySet [<Post: 횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭>, <Post: '디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?>, <Post: 저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는>]>

이렇게 직렬화할 데이터를 QuerySet으로 준비합니다. 그리고 직렬화를 수행해봅니다.

TypeError: Object of type 'QuerySet' is not JSON serializable 예외가 발생할 거예요. :(

In [11]:

json.dumps(data, cls=DjangoJSONEncoder)

---------------------------------------------------------------------------  
TypeError Traceback (most recent call last)  
<ipython-input-11-1e0c35bdbf0d> in <module>()  
----> 1 json.dumps(data, cls=DjangoJSONEncoder)  
  
/Users/allieus/anaconda/lib/python3.6/json/\_\_init\_\_.py in dumps(obj, skipkeys, ensure\_ascii, check\_circular, allow\_nan, cls, indent, separators, default, sort\_keys, \*\*kw)  
 236 check\_circular=check\_circular, allow\_nan=allow\_nan, indent=indent,  
 237 separators=separators, default=default, sort\_keys=sort\_keys,  
--> 238 \*\*kw).encode(obj)  
 239   
 240   
  
/Users/allieus/anaconda/lib/python3.6/json/encoder.py in encode(self, o)  
 197 # exceptions aren't as detailed. The list call should be roughly  
 198 # equivalent to the PySequence\_Fast that ''.join() would do.  
--> 199 chunks = self.iterencode(o, \_one\_shot=True)  
 200 if not isinstance(chunks, (list, tuple)):  
 201 chunks = list(chunks)  
  
/Users/allieus/anaconda/lib/python3.6/json/encoder.py in iterencode(self, o, \_one\_shot)  
 255 self.key\_separator, self.item\_separator, self.sort\_keys,  
 256 self.skipkeys, \_one\_shot)  
--> 257 return \_iterencode(o, 0)  
 258   
 259 def \_make\_iterencode(markers, \_default, \_encoder, \_indent, \_floatstr,  
  
/Users/allieus/anaconda/lib/python3.6/site-packages/django/core/serializers/json.py in default(self, o)  
 122 return bool(o)  
 123 else:  
--> 124 return super(DjangoJSONEncoder, self).default(o)  
  
/Users/allieus/anaconda/lib/python3.6/json/encoder.py in default(self, o)  
 178 """  
 179 raise TypeError("Object of type '%s' is not JSON serializable" %  
--> 180 o.\_\_class\_\_.\_\_name\_\_)  
 181   
 182 def encode(self, o):  
  
TypeError: Object of type 'QuerySet' is not JSON serializable

왜죠? 왜일까요? **DjangoJSONEncoder**는 **QuerySet**의 직렬화/비직렬화방법을 모르고 있기 때문에, **not JSON serializable** 오류가 발생한 겁니다. 그렇다면, 어떻게 해야할까요?

QuerySet을 파이썬 표준 데이터타입의 값으로 한땀 한땀 직접 변환을 할 수 있겠습니다. 이는 json모듈이 하던 일을 직접 하는 것이죠.

In [14]:

data = [  
 {'id': post.id, 'title': post.title, 'content': post.content}  
 **for** post **in** Post.objects.all()]  
  
json.dumps(data, cls=DjangoJSONEncoder, ensure\_ascii=**False**)

Out[14]:

'[{"id": 1, "title": "횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭", "content": "교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다."}, {"id": 2, "title": "\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?", "content": "옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다."}, {"id": 3, "title": "저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는", "content": "늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?"}]'

In [6]:

**import** **json**  
  
mydata = ['안녕', '파이썬']  
json.dumps(mydata)

Out[6]:

'["\\uc548\\ub155", "\\ud30c\\uc774\\uc36c"]'

In [7]:

json.dumps(mydata, ensure\_ascii=**False**)

Out[7]:

'["안녕", "파이썬"]'

"소곤소곤. json에게 직렬화 방법을 알려줄 수도 있어요. 어떻게 하느냐??? DjangoJSONEncoder가 직렬화 방법을 알고 있기에, 이를 확장하면 됩니다. 다음 2가지 타입을 지원할 수 있도록 해보겠습니다.

* QuerySet 타입 : tuple 타입으로 변환
* Post 타입 : dict 타입으로 변환

In [15]:

**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}  
 **return** super().default(obj)  
  
data = Post.objects.all()  
  
*# 직렬화할 때, 직렬화를 수행해줄 JSON Encoder를 지정해줍니다.*  
json.dumps(data, cls=MyJSONEncoder, ensure\_ascii=**False**)

Out[15]:

'[{"id": 1, "title": "횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭", "content": "교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다."}, {"id": 2, "title": "\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?", "content": "옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다."}, {"id": 3, "title": "저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는", "content": "늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?"}]'

django-rest-framework에서도 커스텀 JSON Encoder를 만드는 방식으로, JSON 인코딩을 처리하고 있습니다.

## **rest\_framework.renderer.JSONRender 의 직렬화 방식**

rest\_framework/utils/encoders.py 의 JSONEncoder 클래스를 통한 직렬화를 수행합니다.

JSONEncoder는 장고의 DjangoJSONEncoder를 상속받지는 않고, json.JSONEncoder를 직접 상속받아 다음 타입에 대한 직렬화를 추가로 지원합니다.

* 파이썬 표준 데이터 타입
  + datetime.datetime 타입
  + datetime.date 타입
  + datetime.time 타입
  + datetime.timedelta 타입
  + decimal.Decimal 타입
  + uuid.UUID 타입
  + six.binary\_type 타입
  + \_\_getitem\_\_ 함수를 지원할 경우, dict(obj)의 리턴값을 취함
  + \_\_iter\_\_ 함수를 지원할 경우, tuple(item for item in obj)의 리턴값을 취함
* 장고 데이터 타입
  + QuerySet 타입일 경우, tuple(obj)의 리턴값을 취함.
  + .tolist 함수를 가질 경우, obj.tolist()의 리턴값을 취함.

QuerySet에 대한 직렬화를 지원해줍니다만, Model 타입에 대한 직렬화는 없습니다. 이는 ModelSerializer의 도움을 받습니다.

rest\_framework/renderer.py내 JSONRenderer는 json.dumps 함수에 대한 래핑 클래스입니다. 보다 편리한 JSON 직렬화를 도와줍니다. 다음 코드로 직렬화를 수행하실 수 있어요. utf8 인코딩도 추가로 수행해줍니다.

In [21]:

**from** **rest\_framework.renderers** **import** JSONRenderer  
  
data = {'이름': 'AskDjango'}  
json\_utf8\_string = JSONRenderer().render(data)  
  
json\_utf8\_string.decode('utf8') *# 출력포맷 조정을 위한 목적일 뿐, 실제 서비스에서는 decode하지 않습니다.*

Out[21]:

'{"이름":"AskDjango"}'

위에서 살펴보셨다시피, JSONRenderer은 rest\_framework.utils.encoders.JSONEncoder를 사용합니다. JSONEncoder는 Model 타입에 대한 직렬화를 지원하지 않기에 직렬화에 실패합니다.

In [18]:

**from** **rest\_framework.renderers** **import** JSONRenderer  
  
data = Post.objects.all()

In [19]:

JSONRenderer().render(data)

---------------------------------------------------------------------------  
TypeError Traceback (most recent call last)  
<ipython-input-19-b1c25b0e5300> in <module>()  
----> 1 JSONRenderer().render(data)  
  
/Users/allieus/anaconda/lib/python3.6/site-packages/rest\_framework/renderers.py in render(self, data, accepted\_media\_type, renderer\_context)  
 103 data, cls=self.encoder\_class,  
 104 indent=indent, ensure\_ascii=self.ensure\_ascii,  
--> 105 separators=separators  
 106 )  
 107   
  
/Users/allieus/anaconda/lib/python3.6/json/\_\_init\_\_.py in dumps(obj, skipkeys, ensure\_ascii, check\_circular, allow\_nan, cls, indent, separators, default, sort\_keys, \*\*kw)  
 236 check\_circular=check\_circular, allow\_nan=allow\_nan, indent=indent,  
 237 separators=separators, default=default, sort\_keys=sort\_keys,  
--> 238 \*\*kw).encode(obj)  
 239   
 240   
  
/Users/allieus/anaconda/lib/python3.6/json/encoder.py in encode(self, o)  
 197 # exceptions aren't as detailed. The list call should be roughly  
 198 # equivalent to the PySequence\_Fast that ''.join() would do.  
--> 199 chunks = self.iterencode(o, \_one\_shot=True)  
 200 if not isinstance(chunks, (list, tuple)):  
 201 chunks = list(chunks)  
  
/Users/allieus/anaconda/lib/python3.6/json/encoder.py in iterencode(self, o, \_one\_shot)  
 255 self.key\_separator, self.item\_separator, self.sort\_keys,  
 256 self.skipkeys, \_one\_shot)  
--> 257 return \_iterencode(o, 0)  
 258   
 259 def \_make\_iterencode(markers, \_default, \_encoder, \_indent, \_floatstr,  
  
/Users/allieus/anaconda/lib/python3.6/site-packages/rest\_framework/utils/encoders.py in default(self, obj)  
 68 elif hasattr(obj, '\_\_iter\_\_'):  
 69 return tuple(item for item in obj)  
---> 70 return super(JSONEncoder, self).default(obj)  
  
/Users/allieus/anaconda/lib/python3.6/json/encoder.py in default(self, o)  
 178 """  
 179 raise TypeError("Object of type '%s' is not JSON serializable" %  
--> 180 o.\_\_class\_\_.\_\_name\_\_)  
 181   
 182 def encode(self, o):  
  
TypeError: Object of type 'Post' is not JSON serializable

이 역시, 직접 직렬화를 한땀한땀할 수도 있겠지만 ...

In [20]:

data = [  
 {'id': post.id, 'title': post.title, 'content': post.content}  
 **for** post **in** Post.objects.all()]  
  
json\_utf8\_string = JSONRenderer().render(data)  
  
json\_utf8\_string.decode('utf8') *# 출력포맷 조정을 위한 목적일 뿐, 실제 서비스에서는 decode하지 않습니다.*

Out[20]:

'[{"id":1,"title":"횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭","content":"교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다."},{"id":2,"title":"\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?","content":"옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다."},{"id":3,"title":"저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는","content":"늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?"}]'

이 역시, django-rest-framework에서 사용하는 JSONEncoder를 확장해 볼수도 있겠지만 ...

In [22]:

**from** **rest\_framework.renderers** **import** JSONRenderer  
**from** **rest\_framework.utils.encoders** **import** JSONEncoder  
  
**class** **MyJSONEncoder**(JSONEncoder):  
 **def** default(self, obj):  
 **if** isinstance(obj, Post):  
 **return** {'id': obj.id, 'title': obj.title, 'content': obj.content}  
 **return** super().default(obj)  
  
data = Post.objects.all()  
  
renderer = JSONRenderer()  
renderer.encoder\_class = MyJSONEncoder  
json\_utf8\_string = renderer.render(data)  
  
json\_utf8\_string.decode('utf8')

Out[22]:

'[{"id":1,"title":"횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭","content":"교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다."},{"id":2,"title":"\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?","content":"옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다."},{"id":3,"title":"저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는","content":"늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?"}]'

## **ModelSerializer를 통한 JSON 직렬화**

django-rest-framework에서는 일반적으로 ModelSerializer를 통해 JSONRenderer에서 변환가능한 형태로 먼저 데이터를 변환합니다.

Serializer는 장고의 Form과 유사하며, ModelSerializer는 장고의 ModelForm과 유사합니다. 역할 면에서 Serializer는 POST요청만 처리하는 Form이라 할 수 있습니다.

|  |  |
| --- | --- |
| **Django Form/ModelForm** | **Django Serializer/ModelSerializer** |
| 폼 필드 지정 혹은 모델로부터 읽어오기 | 좌동 |
| Form HTML을 생성 | JSON 문자열을 생성 |
| 입력된 데이터에 대한 유효성 검사 및 획득 | 좌동 |

Tip: Form/ModelForm이 가물가물하신 분은 [장고 기본편 VOD](https://nomade.kr/vod/django/)의 해당 에피소드를 다시 복습해보세요. :)

다음과 같이 ModelSerializer를 정의합니다. ModelForm과 거의 판박이입니다. :D

In [24]:

**from** **rest\_framework.serializers** **import** ModelSerializer  
  
*# Post모델에 대한 ModelSerializer 정의*  
**class** **PostModelSerializer**(ModelSerializer):  
 **class** **Meta**:  
 model = Post  
 fields = '\_\_all\_\_'

다음과 같이 Post 모델 인스턴스에 대해서도 dict타입으로 변환을 지원합니다. PostModelSerializer에 Post객체를 넘겨보세요.

In [31]:

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

Out[31]:

<Post: 횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭>

In [32]:

serializer = PostModelSerializer(post)  
serializer.data

Out[32]:

ReturnDict([('id', 1),  
 ('title', '횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭'),  
 ('content',  
 '교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.'),  
 ('created\_at', '2017-10-16T14:15:12.102064'),  
 ('updated\_at', '2017-10-16T14:15:12.102093')])

Tip: 위 serializer.data는 ReturnDict타입입니다. OrderedDict을 상속받았으며, 생성자를 통해 serializer필드를 추가로 받습니다.

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

### **QuerySet 변환 지원**

ModelSerializer는 QuerySet에 대해서도 변환을 지원해줍니다. ModelSerializer의 many 인자는 디폴트 False입니다. many=True 인자를 지정해줘야만 QuerySet을 처리합니다.

In [33]:

serializer = PostModelSerializer(Post.objects.all(), many=**True**) *# QuerySet을 지정할 경우, 필히 many=True 지정*  
  
*# 지정된 Model Instance 필드를 통해 list/OrderedDict 획득*  
serializer.data

Out[33]:

[OrderedDict([('id', 1), ('title', '횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭'), ('content', '교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.'), ('created\_at', '2017-10-16T14:15:12.102064'), ('updated\_at', '2017-10-16T14:15:12.102093')]), OrderedDict([('id', 2), ('title', "'디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?"), ('content', '옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.'), ('created\_at', '2017-10-16T14:15:12.102431'), ('updated\_at', '2017-10-16T14:15:12.102446')]), OrderedDict([('id', 3), ('title', '저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는'), ('content', '늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?'), ('created\_at', '2017-10-16T14:15:12.102714'), ('updated\_at', '2017-10-16T14:15:12.102729')])]

In [35]:

**import** **json**  
  
json.dumps(serializer.data, ensure\_ascii=**False**)

Out[35]:

'[{"id": 1, "title": "횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭", "content": "교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.", "created\_at": "2017-10-16T14:15:12.102064", "updated\_at": "2017-10-16T14:15:12.102093"}, {"id": 2, "title": "\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?", "content": "옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.", "created\_at": "2017-10-16T14:15:12.102431", "updated\_at": "2017-10-16T14:15:12.102446"}, {"id": 3, "title": "저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는", "content": "늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?", "created\_at": "2017-10-16T14:15:12.102714", "updated\_at": "2017-10-16T14:15:12.102729"}]'

In [36]:

**from** **rest\_framework.renderers** **import** JSONRenderer  
  
json\_utf8\_string = JSONRenderer().render(serializer.data)  
  
json\_utf8\_string.decode('utf8')

Out[36]:

'[{"id":1,"title":"횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭","content":"교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.","created\_at":"2017-10-16T14:15:12.102064","updated\_at":"2017-10-16T14:15:12.102093"},{"id":2,"title":"\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?","content":"옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.","created\_at":"2017-10-16T14:15:12.102431","updated\_at":"2017-10-16T14:15:12.102446"},{"id":3,"title":"저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는","content":"늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?","created\_at":"2017-10-16T14:15:12.102714","updated\_at":"2017-10-16T14:15:12.102729"}]'

# **뷰에서의 Json 응답**

## **장고 스타일**

JSON 포맷으로 직렬화된 문자열은 장고 뷰를 통해서 응답이 이뤄져야 합니다. 다음 2가지가 가능합니다.

1. json.dumps를 통해 직렬화된 문자열을 HttpResponse를 통해 응답
2. json.dumps기능을 제공하는 JsonResponse를 즉시 사용

이 중에 2번째 방법을 사용해보겠습니다. 이때 JsonResponse는 장고의 DjangoJSONEncoder를 사용하고 있으니, QuerySet에 대해서는 직렬화가 불가능합니다. 그래서 위에서 정의한 MyJSONEncoder를 활용해보겠습니다.

In [38]:

*# 직렬화할 QuerySet 준비*  
data = Post.objects.all()  
data

Out[38]:

<QuerySet [<Post: 횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭>, <Post: '디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?>, <Post: 저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는>]>

JsonResponse에 넘겨줄 인자를 준비합니다.

* encoder (디폴트: DjangoJSONEncoder) : JSON 인코딩을 수행할 클래스
* safe (디폴트: True) : 변환할 데이터의 dict타입 체킹을 목적으로 합니다. 데이터가 dict타입이 아닐 경우에는 필히 False를 지정해주세요. 미지정 시에 TypeError 예외를 발생시킵니다.
* json\_dumps\_params (디폴트: None) : json.dumps에 넘겨질 인자
* kwargs (디폴트: {}) : 부모 클래스인 HttpResponse에 넘겨질 인자

In [40]:

encoder = MyJSONEncoder  
safe = **False** *# True: data가 dict일 경우, False: dict이 아닐 경우*  
json\_dumps\_params = {'ensure\_ascii': **False**}  
kwargs = {} *# HttpResponse에 전해지는 Keyword 인자*

다음과 같이 Http 응답을 생성하고, 그 응답바디를 출력해봅시다.

In [42]:

**from** **django.http** **import** JsonResponse  
  
response = JsonResponse(data, encoder, safe, json\_dumps\_params, \*\*kwargs)  
  
print(response)  
response.content.decode('utf8')

<JsonResponse status\_code=200, "application/json">

Out[42]:

'[{"id": 1, "title": "횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭", "content": "교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다."}, {"id": 2, "title": "\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?", "content": "옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다."}, {"id": 3, "title": "저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는", "content": "늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?"}]'

## **django-rest-framework 스타일 (맛보기)**

django-rest-framework에서의 JSON 직렬화 코드를 러프하게 살펴보겠습니다. 이어지는 코드가 많아보이지만, 실제로 사용될 때에는 코드가 아주 심플합니다. 디폴트 세팅된 항목들이 사용되기 때문입니다.

In [59]:

*# 변환할 데이터로서 QuerySet을 준비*  
queryset = Post.objects.all()

In [60]:

*# queryset을 통해 ModelSerializer 준비*  
serializer = PostModelSerializer(queryset, many=**True**)  
serializer

Out[60]:

PostModelSerializer(<QuerySet [<Post: 횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭>, <Post: '디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?>, <Post: 저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는>]>, many=True):  
 id = IntegerField(label='ID', read\_only=True)  
 title = CharField(max\_length=100)  
 content = CharField(style={'base\_template': 'textarea.html'})  
 created\_at = DateTimeField(read\_only=True)  
 updated\_at = DateTimeField(read\_only=True)

In [61]:

*# 짠~ 이렇게, 직렬화할 데이터가 뽑아졌습니다.*  
serializer.data

Out[61]:

[OrderedDict([('id', 1), ('title', '횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭'), ('content', '교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.'), ('created\_at', '2017-10-16T14:15:12.102064'), ('updated\_at', '2017-10-16T14:15:12.102093')]), OrderedDict([('id', 2), ('title', "'디지털세대, 아날로그에 빠지다'…아날로그 인기 이유는?"), ('content', '옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.'), ('created\_at', '2017-10-16T14:15:12.102431'), ('updated\_at', '2017-10-16T14:15:12.102446')]), OrderedDict([('id', 3), ('title', '저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는'), ('content', '늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?'), ('created\_at', '2017-10-16T14:15:12.102714'), ('updated\_at', '2017-10-16T14:15:12.102729')])]

뷰에서는 Response를 통해 응답을 생성합니다. 이는 HttpResponse를 상속받은 클래스입니다. Response는 단순히 JSON 직렬화뿐만 아니라, HTTP요청에 따라 다양한 포맷으로 변환(Render)하여 응답을 생성할 수 있습니다.

In [63]:

**from** **rest\_framework.response** **import** Response  
  
response = Response(serializer.data)  
response

Out[63]:

<Response status\_code=200, "text/html; charset=utf-8">

Response 객체에 변환에 필요한 속성을 지정해줘야합니다. 실제 요청을 처리하는 코드에서는 APIView 클래스에 의해서 디폴트 지정이 되므로, 대개 수동으로 지정할 일은 없습니다.

In [64]:

**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**}

In [67]:

response

Out[67]:

<Response status\_code=200, "application/json">

response 객체는 아직 변환할 준비만 하고 있을 뿐, 아직 JSON 직렬화 변환은 수행하지 않았습니다. .rendered\_content 속성에 접근할 때, 변환이 이뤄집니다.

In [68]:

response.rendered\_content.decode('utf8')

Out[68]:

'[{"id":1,"title":"횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭","content":"교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.","created\_at":"2017-10-16T14:15:12.102064","updated\_at":"2017-10-16T14:15:12.102093"},{"id":2,"title":"\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?","content":"옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.","created\_at":"2017-10-16T14:15:12.102431","updated\_at":"2017-10-16T14:15:12.102446"},{"id":3,"title":"저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는","content":"늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?","created\_at":"2017-10-16T14:15:12.102714","updated\_at":"2017-10-16T14:15:12.102729"}]'

## **실전에서의 Response 활용**

위에서는 디테일하게 django-rest-framework 뷰에서의 JSON 직렬화 순서에 대해서 살펴봤는데요. 실제로는 다음과 같이 간결하게 사용합니다.

In [84]:

**from** **rest\_framework** **import** generics  
  
**class** **PostListAPIView**(generics.ListAPIView):  
 queryset = Post.objects.all()  
 serializer\_class = PostModelSerializer

이렇게 간결하게 정의한 뷰 만으로 다음과 같이 JSON 응답을 만들어낼 수 있습니다.

In [86]:

**from** **django.http** **import** HttpRequest  
  
**class** **DummyUser**:  
 **pass**  
  
request = HttpRequest()  
request.user = DummyUser()  
request.method = 'GET'  
  
response = PostListAPIView.as\_view()(request)  
response.rendered\_content.decode('utf8')

Out[86]:

'[{"id":1,"title":"횡단보도 보행자 없으면 우회전 가능?…혼란 빚은 까닭","content":"교차로에서 우회전할 때 횡단 보도를 건너는 사람이 없다면 보행자 신호가 녹색이더라도 진입할 수 있을까요? 이 문제를 놓고 대법원과 경찰의 판단이 다른 상황입니다.","created\_at":"2017-10-16T14:15:12.102064","updated\_at":"2017-10-16T14:15:12.102093"},{"id":2,"title":"\'디지털세대, 아날로그에 빠지다\'…아날로그 인기 이유는?","content":"옛 방식을 고집하는 아날로그 공간들이 젊은 층을 중심으로 주목받고 있습니다.","created\_at":"2017-10-16T14:15:12.102431","updated\_at":"2017-10-16T14:15:12.102446"},{"id":3,"title":"저녁 줄였는데 누구는 살 빠지고, 난 안 빠지고…이유는","content":"늦은 시간에 야식 먹으면 다 살로 간다고 하죠? 그래서 야식 증후군이란 말까지 생겼습니다. 또 아침은 많이 먹고 저녁은 되도록 적게 먹는 것이 다이어트의 지름길이라고 생각하기도 합니다. 이게 다 얼마나 맞는 말일까요?","created\_at":"2017-10-16T14:15:12.102714","updated\_at":"2017-10-16T14:15:12.102729"}]'