requirements : Python 3.7, Django 2.1.1 and Django Rest Framework 3.9.4.

前置作業 ： 建立 Django 專案，建立 Django app，建立 serializers.py 檔案

前置作業可以參考 djangorestframework 官方網站中的 quickstart :

https://www.django-rest-framework.org/tutorial/quickstart/

這個測試主要為測試不同的 serializer 寫法的效能

主要使用 Django 內建的 User 資料來進行各種 serializer 的效能測試

為了單獨測試 serializer 的效能，所以只會單獨使用一個 instance，避免 database 的外在因素影響，也不會完整的使用到 API 的功能

第一個測試，不使用 rest_framework 提供的 serializer，單純將 User instance 從 django.contrib.auth.models.User 轉換成 dictionary 的資料類型後 return

```python
from typing import Dict, Any

from django.contrib.auth.models import User


def serialize_user(user: User) -> Dict[str, Any]:
    return {
        'id': user.id,
        'last_login': user.last_login.isoformat() if user.last_login is not None else None,
        'is_superuser': user.is_superuser,
        'username': user.username,
        'first_name': user.first_name,
        'last_name': user.last_name,
        'email': user.email,
        'is_staff': user.is_staff,
        'is_active': user.is_active,
        'date_joined': user.date_joined.isoformat(),
    }
```

將 serializer 的程式碼建立在 app 中建立的 serializers.py 檔案中，然後使用 python manage.py shell 進行效能測試 ：

>> from django.contrib.auth.models import User

>> u = User.objects.create_user(username="EdgarChang123" , first_name="Chang" , last_name="Edgar" , email="Edgar123@gmail.com")

>> import cProfile

>> from main.serializers import serialize_user

>> cProfile.runctx('for i in range(5000) : serialize_user(u)',globals(),locals(),sort='tottime')

```
>>> u = User.objects.create_user(username = "kevin Chang" , first_name = "Chang" , last_name = "kevin" , email = "kenvin123456@gmail.com")
>>> import cProfile
>>> from main.serializers import serialize_user
>>> cProfile.runctx('for i in range(5000): serialize_user(u)' , globals() , locals() , sort='tottime')
         15003 function calls in 0.020 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     5000    0.013    0.000    0.013    0.000 {method 'isoformat' of 'datetime.datetime' objects}
     5000    0.005    0.000    0.018    0.000 serializers.py:5(serialize_user)
        1    0.002    0.002    0.020    0.020 <string>:1(<module>)
     5000    0.001    0.000    0.001    0.000 __init__.py:223(utcoffset)
        1    0.000    0.000    0.020    0.020 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

總共花費 0.02 秒完成 5000 次的 serialize

主要的測試工具是 cProfile

https://docs.python.org/3/library/profile.html

改用成使用 rest_framework 提供的 ModelSerializer

```python
class UserModelSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = [
            'id',
            'last_login',
            'is_superuser',
            'username',
            'first_name',
            'last_name',
            'email',
            'is_staff',
            'is_active',
            'date_joined',
        ]
```

```
>>> cProfile.runctx('for i in range(5000): UserModelSerializer(u).data',globals(),locals(),sort='tottime')
         19140291 function calls (18820229 primitive calls) in 19.389 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    85000    2.594    0.000    6.967    0.000 functional.py:82(__prepare_class__)
  7955003    2.255    0.000    2.255    0.000 {built-in method builtins.hasattr}
  1080000    1.744    0.000    1.744    0.000 functional.py:102(__promise__)
    50000    0.885    0.000    7.361    0.000 field_mapping.py:66(get_field_kwargs)
    55000    0.791    0.000    1.023    0.000 fields.py:319(__init__)
1135004/1135003    0.679    0.000    0.706    0.000 {built-in method builtins.getattr}
     5000    0.560    0.000   17.746    0.004 serializers.py:992(get_fields)
  1240002    0.541    0.000    0.541    0.000 {built-in method builtins.setattr}
    20000    0.501    0.000    0.778    0.000 {built-in method builtins.__build_class__}
   210000    0.484    0.000    1.416    0.000 trans_real.py:275(gettext)
420000/210000    0.433    0.000    0.521    0.000 gettext.py:451(gettext)
    20000    0.409    0.000    7.336    0.000 fields.py:762(__init__)
    75000    0.408    0.000    3.295    0.000 functional.py:191(wrapper)
    50000    0.302    0.000    4.146    0.000 field_mapping.py:46(needs_label)
  1200012    0.297    0.000    0.354    0.000 {built-in method builtins.isinstance}
    50000    0.285    0.000    7.940    0.000 serializers.py:1197(build_standard_field)
    20000    0.277    0.000    0.277    0.000 functional.py:57(__proxy__)
    55000    0.258    0.000    1.296    0.000 text.py:14(capfirst)
    85000    0.195    0.000    7.161    0.000 functional.py:66(__init__)
   210000    0.180    0.000    1.596    0.000 __init__.py:74(gettext)
   115000    0.174    0.000    1.273    0.000 functional.py:105(__wrapper__)
   495000    0.160    0.000    0.160    0.000 {method 'replace' of 'str' objects}
   310000    0.157    0.000    0.157    0.000 {method 'update' of 'dict' objects}
   750001    0.153    0.000    0.153    0.000 {method 'get' of 'dict' objects}
    50000    0.148    0.000    0.202    0.000 field_mapping.py:29(__getitem__)
    55000    0.147    0.000    0.173    0.000 fields.py:616(__new__)
65000/45000    0.133    0.000    0.150    0.000 deconstruct.py:14(__new__)
     5000    0.130    0.000   19.077    0.004 serializers.py:508(to_representation)
    50000    0.127    0.000    0.170    0.000 fields.py:365(bind)
    20000    0.125    0.000    1.495    0.000 functional.py:49(lazy)
140000/50000    0.125    0.000    1.363    0.000 functional.py:112(__text_cast)
    85000    0.119    0.000    7.280    0.000 functional.py:159(__wrapper__)
    50000    0.118    0.000    8.058    0.000 serializers.py:1174(build_field)
    20000    0.109    0.000    0.206    0.000 functools.py:37(update_wrapper)
    20000    0.108    0.000    0.358    0.000 functools.py:186(total_ordering)
     5000    0.108    0.000   18.113    0.004 serializers.py:353(fields)
```

總共花費 19.389 秒，和前面的測試相比，在 functional.py 中花費了大量時間

來看看如何處理這效能問題

在 python manage.py shell 直接建立 UserModelSerializer 的物件 serializer，然後 print(repr(serializer))，觀察一下

UserModelSerializer():

id = IntegerField(label='ID', read_only=True)

last_login = DateTimeField(allow_null=True, required=False)

is_superuser = BooleanField(help_text='Designates that this user has all permissions without explicitly assigning them.', label='Superuser status', required=False)

username = CharField(help_text='Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.', max_length=150, validators=[<django.contrib.auth.validators.UnicodeUsernameValidator object>, <UniqueValidator(queryset=User.objects.all())>])

first_name = CharField(allow_blank=True, max_length=30, required=False)

last_name = CharField(allow_blank=True, max_length=150, required=False)

email = EmailField(allow_blank=True, label='Email address', max_length=254, required=False)

is_staff = BooleanField(help_text='Designates whether the user can log into this admin site.', label='Staff status', required=False)

is_active = BooleanField(help_text='Designates whether this user should be treated as active. Unselect this instead of deleting accounts.', label='Active', required=False)

date_joined = DateTimeField(required=False)

最主要的參數是 label、read_only、allow_null、required、help_text、allow_blank，以及在 username 的 validators

validators 會對效能造成較大的影響，UniqueValidator 驗證器使用了 https://github.com/django/django/blob/master/django/utils/functional.py 中的 lazy function 進行驗證

在 lazy function 中，class __proxy__(Promise)使用了@total_ordering 這個裝飾器，在官方文件中寫到這個裝飾器會造成效能降低

@total_ordering 官方文件：

https://docs.python.org/3.8/library/functools.html

對於這些 Core arguments 可以參考官方文件 ：

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

開源碼:

https://github.com/encode/django-rest-framework/blob/82f256989559b95a0cd5ba318a95f7c66945436a/rest_framework/fields.py

因為 ModelSerializer 只會對 writeable field 添加 Field validations，所以接下來使用 ModelSerializer，但是改成 readonly

```
class UserReadOnlyModelSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = [
            'id',
            'last_login',
            'is_superuser',
            'username',
            'first_name',
            'last_name',
            'email',
            'is_staff',
            'is_active',
            'date_joined',
        ]
        read_only_fields = fields
```

```
>>> cProfile.runctx('for i in range(5000): UserReadOnlyModelSerializer(u).data',globals(),locals(),sort='tottime')
         14835003 function calls (14535003 primitive calls) in 11.968 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  6090000    1.218    0.000    1.218    0.000 {built-in method builtins.hasattr}
    65000    1.180    0.000    2.375    0.000 functional.py:82(__prepare_class__)
    50000    0.816    0.000    6.822    0.000 field_mapping.py:66(get_field_kwargs)
    55000    0.637    0.000    0.840    0.000 fields.py:319(__init__)
   835000    0.536    0.000    0.559    0.000 {built-in method builtins.getattr}
     5000    0.476    0.000   10.468    0.002 serializers.py:992(get_fields)
   210000    0.432    0.000    1.287    0.000 trans_real.py:275(gettext)
420000/210000    0.408    0.000    0.489    0.000 gettext.py:451(gettext)
    75000    0.340    0.000    3.155    0.000 functional.py:191(wrapper)
  1205000    0.282    0.000    0.334    0.000 {built-in method builtins.isinstance}
    50000    0.257    0.000    7.328    0.000 serializers.py:1197(build_standard_field)
    55000    0.228    0.000    1.225    0.000 text.py:14(capfirst)
    50000    0.221    0.000    0.390    0.000 serializers.py:1310(include_extra_kwargs)
    50000    0.195    0.000    3.921    0.000 field_mapping.py:46(needs_label)
    20000    0.170    0.000    0.621    0.000 fields.py:762(__init__)
   210000    0.166    0.000    1.453    0.000 __init__.py:74(gettext)
   745000    0.160    0.000    0.160    0.000 {method 'pop' of 'dict' objects}
   800000    0.159    0.000    0.159    0.000 {method 'get' of 'dict' objects}
   495000    0.159    0.000    0.159    0.000 {method 'replace' of 'str' objects}
   115000    0.153    0.000    1.171    0.000 functional.py:105(__wrapper__)
    55000    0.146    0.000    0.172    0.000 fields.py:616(__new__)
    65000    0.133    0.000    2.508    0.000 functional.py:66(__init__)
   290000    0.132    0.000    0.132    0.000 {method 'update' of 'dict' objects}
    50000    0.125    0.000    0.168    0.000 field_mapping.py:29(__getitem__)
   150000    0.123    0.000    0.141    0.000 functional.py:193(<genexpr>)
    50000    0.123    0.000    0.160    0.000 fields.py:365(bind)
140000/50000    0.117    0.000    1.289    0.000 functional.py:112(__text_cast)
     5000    0.115    0.000   11.666    0.002 serializers.py:508(to_representation)
    50000    0.114    0.000    7.442    0.000 serializers.py:1174(build_field)
     5000    0.100    0.000   10.803    0.002 serializers.py:353(fields)
     5000    0.097    0.000    0.122    0.000 serializers.py:1439(_get_model_fields)
    65000    0.080    0.000    2.587    0.000 functional.py:159(__wrapper__)
   135000    0.076    0.000    0.156    0.000 {built-in method builtins.next}
    50000    0.075    0.000    0.153    0.000 fields.py:55(is_simple_callable)
     5000    0.070    0.000    0.096    0.000 model_meta.py:72(_get_forward_relationships)
     5000    0.069    0.000    0.198    0.000 serializers.py:1365(get_uniqueness_extra_kwargs)
    45000    0.068    0.000    1.398    0.000 functional.py:134(__eq__)
```

Serializer 的 .data() : Returns the outgoing primitive representation.

總共花費 11.968 秒，主要少了 functional.py 中的__promise__

再來，field_mapping.py 和 fields.py 也花費了一定的時間，試試使用一般的

serializer

這次不要使用 ModelSerializer

```python
class UserSerializer(serializers.Serializer):
    id = serializers.IntegerField()
    last_login = serializers.DateTimeField()
    is_superuser = serializers.BooleanField()
    username = serializers.CharField()
    first_name = serializers.CharField()
    last_name = serializers.CharField()
    email = serializers.EmailField()
    is_staff = serializers.BooleanField()
    is_active = serializers.BooleanField()
    date_joined = serializers.DateTimeField()
```

```
>>> cProfile.runctx('for i in range(5000): UserSerializer(u).data',globals(),locals(),sort='tottime')
         3110003 function calls (3010003 primitive calls) in 2.952 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    55000    0.460    0.000    0.608    0.000 fields.py:319(__init__)
105000/5000    0.258    0.000    1.740    0.000 copy.py:132(deepcopy)
    50000    0.183    0.000    1.212    0.000 fields.py:626(__deepcopy__)
    20000    0.130    0.000    0.470    0.000 fields.py:762(__init__)
   310000    0.127    0.000    0.127    0.000 {built-in method builtins.getattr}
    50000    0.120    0.000    0.170    0.000 fields.py:365(bind)
     5000    0.102    0.000    2.713    0.001 serializers.py:508(to_representation)
   235000    0.080    0.000    0.080    0.000 {method 'update' of 'dict' objects}
    25000    0.073    0.000    0.080    0.000 deconstruct.py:14(__new__)
     5000    0.071    0.000    1.675    0.000 copy.py:268(_reconstruct)
    55000    0.071    0.000    0.091    0.000 fields.py:616(__new__)
   260000    0.066    0.000    0.103    0.000 {built-in method builtins.isinstance}
    50000    0.063    0.000    0.128    0.000 fields.py:55(is_simple_callable)
    55000    0.060    0.000    0.081    0.000 copy.py:252(_keep_alive)
    50000    0.054    0.000    0.054    0.000 fields.py:635(<listcomp>)
     5000    0.054    0.000    2.038    0.000 serializers.py:353(fields)
    50000    0.054    0.000    0.265    0.000 fields.py:89(get_attribute)
    50000    0.053    0.000    0.223    0.000 serializer_helpers.py:145(__setitem__)
   215000    0.041    0.000    0.041    0.000 {method 'get' of 'dict' objects}
     5000    0.033    0.000    0.034    0.000 {method 'isoformat' of 'datetime.datetime' objects}
     5000    0.031    0.000    0.135    0.000 fields.py:1223(to_representation)
     5000    0.030    0.000    0.032    0.000 serializer_helpers.py:18(__init__)
    50000    0.030    0.000    0.295    0.000 fields.py:447(get_attribute)
     5000    0.029    0.000    0.218    0.000 fields.py:813(__init__)
    55000    0.028    0.000    0.045    0.000 _collections_abc.py:760(__iter__)
    25000    0.027    0.000    0.049    0.000 fields.py:401(validators)
        1    0.027    0.027    2.952    2.952 <string>:1(<module>)
    80000    0.027    0.000    0.027    0.000 {built-in method __new__ of type object at 0xaab820}
     5000    0.024    0.000    0.069    0.000 serializers.py:376(<listcomp>)
   170000    0.024    0.000    0.024    0.000 {built-in method builtins.id}
     5000    0.023    0.000    2.743    0.001 serializers.py:248(data)
    50000    0.022    0.000    0.033    0.000 inspect.py:158(isfunction)
   115000    0.021    0.000    0.021    0.000 {method 'pop' of 'dict' objects}
    50000    0.021    0.000    0.021    0.000 {built-in method _abc._abc_instancecheck}
     5000    0.021    0.000    2.796    0.001 serializers.py:561(data)
```

總共花費 2.952 秒，很明顯，多花點時間寫 Field 是非常值得的

試試看改成 Readonly，但是在 ModelSerializer 改成 readonly 可以提升效能的原因是因為 Field validation，但是現在手刻 Serializer 本來就沒有 Field validation，所以預計並不會提升效能

```python
class UserReadOnlySerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    last_login = serializers.DateTimeField(read_only=True)
    is_superuser = serializers.BooleanField(read_only=True)
    username = serializers.CharField(read_only=True)
    first_name = serializers.CharField(read_only=True)
    last_name = serializers.CharField(read_only=True)
    email = serializers.EmailField(read_only=True)
    is_staff = serializers.BooleanField(read_only=True)
    is_active = serializers.BooleanField(read_only=True)
    date_joined = serializers.DateTimeField(read_only=True)
```

```
>>> cProfile.runctx('for i in range(5000): UserReadOnlySerializer(u).data',globals(),locals(),sort='tottime')
         3360003 function calls (3210003 primitive calls) in 3.174 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    55000    0.446    0.000    0.598    0.000 fields.py:319(__init__)
155000/5000    0.323    0.000    1.941    0.000 copy.py:132(deepcopy)
    50000    0.218    0.000    1.393    0.000 fields.py:626(__deepcopy__)
   310000    0.132    0.000    0.132    0.000 {built-in method builtins.getattr}
    20000    0.128    0.000    0.462    0.000 fields.py:762(__init__)
    50000    0.117    0.000    0.166    0.000 fields.py:365(bind)
     5000    0.107    0.000    2.932    0.001 serializers.py:508(to_representation)
    55000    0.087    0.000    0.107    0.000 copy.py:252(_keep_alive)
    25000    0.084    0.000    0.091    0.000 deconstruct.py:14(__new__)
   235000    0.078    0.000    0.078    0.000 {method 'update' of 'dict' objects}
    55000    0.073    0.000    0.089    0.000 fields.py:616(__new__)
     5000    0.072    0.000    1.876    0.000 copy.py:268(_reconstruct)
   260000    0.067    0.000    0.104    0.000 {built-in method builtins.isinstance}
    50000    0.063    0.000    0.129    0.000 fields.py:55(is_simple_callable)
   315000    0.058    0.000    0.058    0.000 {method 'get' of 'dict' objects}
    50000    0.057    0.000    0.161    0.000 fields.py:638(<dictcomp>)
    50000    0.056    0.000    0.223    0.000 serializer_helpers.py:145(__setitem__)
     5000    0.055    0.000    2.239    0.000 serializers.py:353(fields)
    50000    0.055    0.000    0.267    0.000 fields.py:89(get_attribute)
    50000    0.051    0.000    0.051    0.000 fields.py:635(<listcomp>)
     5000    0.036    0.000    0.142    0.000 fields.py:1223(to_representation)
     5000    0.034    0.000    0.035    0.000 {method 'isoformat' of 'datetime.datetime' objects}
   220000    0.034    0.000    0.034    0.000 {built-in method builtins.id}
     5000    0.030    0.000    0.032    0.000 serializer_helpers.py:18(__init__)
    50000    0.030    0.000    0.297    0.000 fields.py:447(get_attribute)
    55000    0.029    0.000    0.046    0.000 _collections_abc.py:760(__iter__)
    25000    0.029    0.000    0.053    0.000 fields.py:401(validators)
     5000    0.025    0.000    0.071    0.000 serializers.py:376(<listcomp>)
     5000    0.024    0.000    0.216    0.000 fields.py:813(__init__)
        1    0.024    0.024    3.174    3.174 <string>:1(<module>)
    80000    0.023    0.000    0.023    0.000 {built-in method __new__ of type object at 0xaab820}
   115000    0.023    0.000    0.023    0.000 {method 'pop' of 'dict' objects}
    50000    0.022    0.000    0.033    0.000 inspect.py:158(isfunction)
     5000    0.021    0.000    0.035    0.000 serializers.py:120(__new__)
    10000    0.021    0.000    0.114    0.000 fields.py:1163(__init__)
     5000    0.020    0.000    3.009    0.001 serializers.py:561(data)
```

總共花費 3.174 秒

測試一下十分方便的 HyperlinkedModelSerializer 使用 ReadOnly

```python
class UserReadOnlyHyperlinkedModelSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = [
            'id',
            'last_login',
            'is_superuser',
            'username',
            'first_name',
            'last_name',
            'email',
            'is_staff',
            'is_active',
            'date_joined',
        ]
        read_only_fields = fields
```

```
>>> cProfile.runctx('for i in range(5000): UserReadOnlyHyperlinkedModelSerializer(u).data',globals(),locals(),sort='tottime')
         14845287 function calls (14545226 primitive calls) in 14.610 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  6090003    1.498    0.000    1.498    0.000 {built-in method builtins.hasattr}
    65000    1.365    0.000    2.843    0.000 functional.py:82(__prepare_class__)
    50000    0.972    0.000    8.222    0.000 field_mapping.py:66(get_field_kwargs)
    55000    0.834    0.000    1.089    0.000 fields.py:319(__init__)
840004/840003    0.765    0.000    0.793    0.000 {built-in method builtins.getattr}
     5000    0.616    0.000   12.718    0.003 serializers.py:992(get_fields)
   210000    0.537    0.000    1.556    0.000 trans_real.py:275(gettext)
420000/210000    0.458    0.000    0.552    0.000 gettext.py:451(gettext)
    75000    0.413    0.000    3.732    0.000 functional.py:191(wrapper)
  1205012    0.353    0.000    0.412    0.000 {built-in method builtins.isinstance}
    50000    0.324    0.000    8.853    0.000 serializers.py:1197(build_standard_field)
    55000    0.277    0.000    1.437    0.000 text.py:14(capfirst)
    50000    0.255    0.000    0.464    0.000 serializers.py:1310(include_extra_kwargs)
    50000    0.240    0.000    4.670    0.000 field_mapping.py:46(needs_label)
   210000    0.205    0.000    1.761    0.000 __init__.py:74(gettext)
   495000    0.195    0.000    0.195    0.000 {method 'replace' of 'str' objects}
    20000    0.194    0.000    0.764    0.000 fields.py:762(__init__)
   800001    0.192    0.000    0.192    0.000 {method 'get' of 'dict' objects}
   745000    0.190    0.000    0.190    0.000 {method 'pop' of 'dict' objects}
   115000    0.189    0.000    1.519    0.000 functional.py:105(__wrapper__)
    55000    0.184    0.000    0.213    0.000 fields.py:616(__new__)
    50000    0.156    0.000    0.212    0.000 field_mapping.py:29(__getitem__)
   295000    0.153    0.000    0.153    0.000 {method 'update' of 'dict' objects}
    65000    0.149    0.000    2.992    0.000 functional.py:66(__init__)
     5000    0.143    0.000   14.222    0.003 serializers.py:508(to_representation)
    50000    0.141    0.000    0.187    0.000 fields.py:365(bind)
     5000    0.139    0.000   13.132    0.003 serializers.py:353(fields)
    50000    0.129    0.000    8.982    0.000 serializers.py:1174(build_field)
140000/50000    0.126    0.000    1.497    0.000 functional.py:112(__text_cast)
   150000    0.104    0.000    0.127    0.000 functional.py:193(<genexpr>)
   135000    0.093    0.000    0.194    0.000 {built-in method builtins.next}
    50000    0.093    0.000    0.186    0.000 fields.py:55(is_simple_callable)
    65000    0.090    0.000    3.082    0.000 functional.py:159(__wrapper__)
     5000    0.090    0.000    0.120    0.000 serializers.py:1439(_get_model_fields)
     5000    0.088    0.000    0.218    0.000 serializers.py:1365(get_uniqueness_extra_kwargs)
    45000    0.084    0.000    1.624    0.000 functional.py:134(__eq__)
```

花費總共 14.618 秒

但是出現像 username 這種需要是 unique 的 field，還是會需要驗證

```python
from rest_framework.validators import UniqueValidator

class UserSerializer(serializers.Serializer):
    id = serializers.IntegerField()
    last_login = serializers.DateTimeField()
    is_superuser = serializers.BooleanField()
    username = serializers.CharField(validators=[UniqueValidator(queryset=User.objects.all())])
    first_name = serializers.CharField()
    last_name = serializers.CharField()
    email = serializers.EmailField()
    is_staff = serializers.BooleanField()
    is_active = serializers.BooleanField()
    date_joined = serializers.DateTimeField()
```

這次改成只使用 Serializer，然後自己加上 validators，再來測試一次效能

```
>>> cProfile.runctx('for i in range(5000) : UserSerializer(u).data',globals(),locals(),sort='tottime')
         3110059 function calls (3010031 primitive calls) in 2.864 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    55000    0.421    0.000    0.572    0.000 fields.py:319(__init__)
105000/5000    0.241    0.000    1.660    0.000 copy.py:132(deepcopy)
    50000    0.208    0.000    1.161    0.000 fields.py:626(__deepcopy__)
310003/310002    0.131    0.000    0.131    0.000 {built-in method builtins.getattr}
    50000    0.118    0.000    0.171    0.000 fields.py:365(bind)
    20000    0.113    0.000    0.423    0.000 fields.py:762(__init__)
     5000    0.094    0.000    2.628    0.001 serializers.py:508(to_representation)
   235000    0.084    0.000    0.084    0.000 {method 'update' of 'dict' objects}
    25000    0.080    0.000    0.086    0.000 deconstruct.py:14(__new__)
    55000    0.072    0.000    0.087    0.000 fields.py:616(__new__)
     5000    0.069    0.000    1.602    0.000 copy.py:268(_reconstruct)
   260000    0.063    0.000    0.099    0.000 {built-in method builtins.isinstance}
    50000    0.062    0.000    0.267    0.000 fields.py:89(get_attribute)
    50000    0.061    0.000    0.126    0.000 fields.py:55(is_simple_callable)
    50000    0.056    0.000    0.227    0.000 serializer_helpers.py:145(__setitem__)
     5000    0.056    0.000    1.962    0.000 serializers.py:353(fields)
    55000    0.051    0.000    0.072    0.000 copy.py:252(_keep_alive)
    50000    0.049    0.000    0.049    0.000 fields.py:635(<listcomp>)
   215000    0.041    0.000    0.041    0.000 {method 'get' of 'dict' objects}
    50000    0.034    0.000    0.301    0.000 fields.py:447(get_attribute)
     5000    0.034    0.000    0.036    0.000 serializer_helpers.py:18(__init__)
     5000    0.032    0.000    0.033    0.000 {method 'isoformat' of 'datetime.datetime' objects}
    55000    0.029    0.000    0.046    0.000 _collections_abc.py:760(__iter__)
     5000    0.025    0.000    0.128    0.000 fields.py:1223(to_representation)
     5000    0.025    0.000    0.071    0.000 serializers.py:376(<listcomp>)
   170000    0.024    0.000    0.024    0.000 {built-in method builtins.id}
    50000    0.023    0.000    0.034    0.000 inspect.py:158(isfunction)
    25000    0.023    0.000    0.041    0.000 fields.py:401(validators)
     5000    0.022    0.000    0.194    0.000 fields.py:813(__init__)
        1    0.022    0.022    2.864    2.864 <string>:1(<module>)
   115000    0.021    0.000    0.021    0.000 {method 'pop' of 'dict' objects}
    80000    0.021    0.000    0.021    0.000 {built-in method __new__ of type object at 0xaab820}
     5000    0.020    0.000    2.708    0.001 serializers.py:561(data)
    50000    0.020    0.000    0.020    0.000 {built-in method _abc._abc_instancecheck}
    50000    0.020    0.000    0.020    0.000 {method 'replace' of 'str' objects}
     5000    0.018    0.000    0.107    0.000 serializers.py:111(__init__)
    50000    0.018    0.000    0.018    0.000 {method 'split' of 'str' objects}
```

只花費了 2.864 秒，看來就算加上 validators，只要節省 field_mapping.py 和 fields.py 的時間，依然可以提升很多的效能

本文章參考自：https://hakibenita.com/django-rest-framework-slow