API 測試

重要：後面會用到

完成上述 ORM 設定後，Django 會自動產生可用於資料表 CRUD 的 API

語法	說明
資料表.objects.all()	取得所有資料
資料表.objects.order_by('欄位名稱')	- 取得所有資料並依指定欄位遞增排序
資料表.objects.order_by('-欄位名稱')	- 取得所有資料並依指定欄位遞減排序
資料表.objects.get(查詢條件)	取得"一筆"資料
資料表.objects.filter(查詢條件)	取得"所有"符合條件資料

語法	說明
資料表.objects.all()	取得所有資料
資料表.objects.order_by('欄位名稱')	- 取得所有資料並依指定欄位遞增排序
資料表.objects.order_by('-欄位名稱')	- 取得所有資料並依指定欄位遞減排序
資料表.objects.get(查詢條件)	取得"一筆"資料
資料表.objects.filter(查詢條件)	取得"所有"符合條件資料

==========

以互動模式測試 API

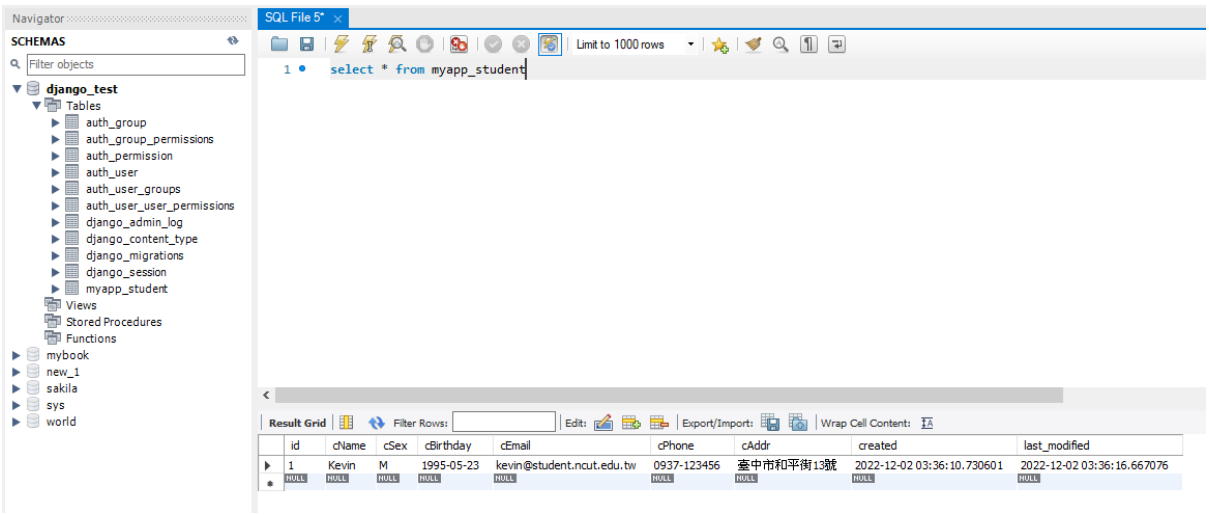
以下述指令進入 Python 的互動模式進行測試

(django) C:\Users\226083\django_test\myweb>py manage.py shell

進入後，可輸入以下>>>之後的指令，測試資料庫 API (測試都正常)

```
>>> from myapp.models import student
>>> from django.utils import timezone
>>> student.objects.all()
<QuerySet []>
>>> s1 = student(cName="Kevin", cSex="M", cBirthday='1995-05-23', cEmail="kevin
@student.ncut.edu.tw", cPhone="0937-123456", cAddr="臺中市和平街13號", last_modifi
ed=timezone.now(),created=timezone.now())
>>> s1.save()
>>> s1.id
1
>>> s1.cBirthday
'1995-05-23'
>>> s1.cName
'Kevin'
>>> s1.cName = "John"
>>> s1.cName
'John'
>>> student.objects.order_by('id')
<QuerySet [<student: Kevin>]>
>>> s1.save()
>>> student.objects.order_by('-id')
<QuerySet [<student: John>]>
>>>student.objects.get(cName="John")
<student: John>
>>> student.objects.filter(cName="John")
<QuerySet [<student: John>]>
>>> s1.delete()
(1, {'myapp.student': 1})
```

執行完”s1.save()”之後，先進資料庫看看是否有成功寫入(成功)



==========

補充：

把上面指令一條條條分清楚

```
# import會用到的套件
>>> from myapp.models import student
>>> from django.utils import timezone
```

重點：

```
>>> student.objects.all()
<QuerySet []> #回傳資料類型
```

object是Manager類型的對象

定義在from django.db import models中，是默認生成的，也就是objects = Modes.Manage() 。用途是數據庫和模型對象交互的接口(api)

因為這時是透過”Django”連結MYSQL資料庫，所以如果要對資料庫進行操作 查詢、新增、刪除等動作，都需要先經過”Django”的處理，也就是”object”

XXX.objects.all()：

all 會返回 “QuerySet” (<QuerySet []>)

程序並沒有真的在數據庫中執行SQL語句查詢數據

功能為取得所有資料(包含欄位)

“QuerySet”是什麼?

QuerySet在Django框架下，代表的是資料庫裡面的資料集合

比較實際的說法，QuerySet就是對應資料表(Table)上的所有資料紀錄

==========

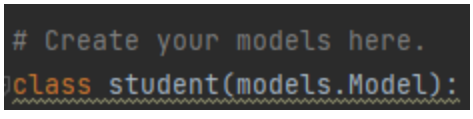
已經使用”object.all()”，透過”django”連結到MYSQL資料庫

接下來就可以開始對資料庫進行操作

```
s1 = student(cName="Kevin",
cSex="M",
cBirthday='1995-05-23',
cEmail="kevin@student.ncut.edu.tw",
cPhone="0937-123456",
cAddr="臺中市和平街13號",
last_modified=timezone.now(),
created=timezone.now())


>>> s1.save()
```

student是前面建立資料庫的class 名稱



後面(按照資料庫欄位順序)新增資料

注意：

前面資料欄位的設定，cBirthday = models.DateField('生日',null=False)

設定DateField屬性，填寫格式是西元-月份-日期

例如：'1995-05-23'

重要：

```
s1.save()
```

新增完資料之後，一定要儲存(.save())，不然不會寫入資料庫

==========

查詢：

```
# 查詢s1這筆資料裡面"id"的欄位值
>>> s1.id
1

# 查詢s1這筆資料裡面"cBirthday"的欄位值
>>> s1.cBirthday
'1995-05-23'

# 查詢s1這筆資料裡面"cName"的欄位值
>>> s1.cName
'Kevin'

# 指定值"John"，放進s1這筆資料裡面"cName"的欄位內
# 注意 這邊沒執行.save()動作，所以會改變的只有s1這筆資料，這個改動不會寫入資料庫
# 有執行save()，改動才會寫入資料庫
>>> s1.cName = "John"
>>> s1.cName
'John'
```

==========

進階查詢：

```
# 因為上面執行""s1.cName = "John"之後，沒有執行"save()"
# 所以資料庫內顯示的欄位值還是'Kevin'
>>> student.objects.order_by('id')
<QuerySet [<student: Kevin>]>


# 執行儲存資料，新資料會寫入資料庫
>>> s1.save()


# 執行"save()"後，上面做的變動就會被寫入資料庫
# 這時查詢就會顯示變動後的結果
>>> student.objects.order_by('-id')
<QuerySet [<student: John>]>

>>>student.objects.get(cName="John")
<student: John>

>>> student.objects.filter(cName="John")
<QuerySet [<student: John>]>

>>> s1.delete()
(1, {'myapp.student': 1})
```

排序：

objects.order_by()：

顯示某欄位的資料，並起按照順序排列

如果在欄位名稱前面加上"-"，則會反排序

例如：

student.objects.order_by('id')

student.objects.order_by('-id')

objects.get()、objects.filter()：**欄位查詢技巧**

get()返回的是Model對象，類型為列表

例如：<student: John>

filter()返回的是“QuerySet”

例如：<QuerySet [<student: John>]>

使用get、filter方法會直接執行sql語句獲取數據

all 則否(上面有說過)

以下範例說明：

```
Restaurant.objects.filter(name__contains='餐廳')
```

可以過濾出所有name欄位包含'餐廳'的所有資料

這個查詢語句會對應到SQL語言的

```
SELECT  * FROM ... WHERE name LIKE '%餐廳%';
```

常見的雙底線欄位查詢(field lookups)技巧:

field lookups	說明	範例
exact	與指定的值完全相等	`Restaurant.objects.filter(name__exact='goodgoodeat')`
iexact	同exact，但不分大小寫	`Restaurant.objects.filter(name__iexact='GoOdgoodEat')`
contains	包含指定的值	`Restaurant.objects.filter(name__contains='eat')`
icontains	同contains，但不分大小寫	`Restaurant.objects.filter(name__icontains='Eat')`
in	在指定的串列中	`Restaurant.objects.filter(name__in=['goodgoodeat','badbadeat'])`
gt	大於指定的值	`Food.objects.filter(price__gt=100)`
gte	大於等於指定的值	`Food.objects.filter(price__gte=100)`

lt	小於指定的值	`Food.objects.filter(price__lt=100)`
lte	小於等於指定的值	`Food.objects.filter(price__lte=100)`
startswith	以指定的字串開頭	`Food.objects.filter(name__startswith='宮保')`
endswith	以指定的字串結尾	`Food.objects.filter(price__endswith='雞丁')`

field lookups	說明	範例
exact	與指定的值完全相等	`Restaurant.objects.filter(name__exact='goodgoodeat')`
iexact	同exact，但不分大小寫	`Restaurant.objects.filter(name__iexact='GoOdgoodEat')`
contains	包含指定的值	`Restaurant.objects.filter(name__contains='eat')`
icontains	同contains，但不分大小寫	`Restaurant.objects.filter(name__icontains='Eat')`
in	在指定的串列中	`Restaurant.objects.filter(name__in=['goodgoodeat','badbadeat'])`
gt	大於指定的值	`Food.objects.filter(price__gt=100)`
gte	大於等於指定的值	`Food.objects.filter(price__gte=100)`
lt	小於指定的值	`Food.objects.filter(price__lt=100)`
lte	小於等於指定的值	`Food.objects.filter(price__lte=100)`
startswith	以指定的字串開頭	`Food.objects.filter(name__startswith='宮保')`
endswith	以指定的字串結尾	`Food.objects.filter(price__endswith='雞丁')`

==========

```
# 刪除
# 注意 執行刪除這個動作，會直接作用到資料庫上，不需要執行save()

>>> s1.delete()
(1, {'myapp.student': 1})
```

==========

樣板測試 API

新增 myweb\urls.py 以下第 26 與 27 行程式碼

(寫入網址路徑)

```
from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home),
    path('hi/<username>/', views.hiname),      # 傳遞字串參數 username
    path('age/<int:year>/', views.age),        # 傳遞數值參數 year
    path('hello/', views.hello_view),
    path('getName/<username>/', views.getOneByName), # 傳遞字串參數 username
    path('getAll/', views.getAll),
    # path(r'^admin/', admin.site.urls),
    # path(r'^$', sayhello),
]
```

新增 myapp\views.py 以下第 4-6 行程式碼

注意：

from myapp.models import student

後續會操作到資料庫中的”student”數據庫(student.objects.get())

所以必須加入這行，不然無法抓到該數據庫內的資料

```
from myapp.models import student
from django.http import Http404
from django.contrib import auth
```

還有以下第 31 行開始之程式碼

```
def getOneByName(request, username):
    title = "顯示一筆資料"
    # unit = get_object_or_404(student, cName=username)
    try:
        unit = student.objects.get(cName=username)
```

```
    except student.DoesNotExist:
        raise Http404("查無此學生")
    except:
        raise Http404("讀取錯誤")
    return render(request, 'listone.html', locals() )


def getAll(request):
    title = "顯示全部資料"
    # students = get_list_or_404(student)
    try:
        students = student.objects.all()
    except student.DoesNotExist:
        raise Http404("查無學生資料")
    except:
        raise Http404("讀取錯誤")
    return render(request, 'listall.html', locals() )
```

==========

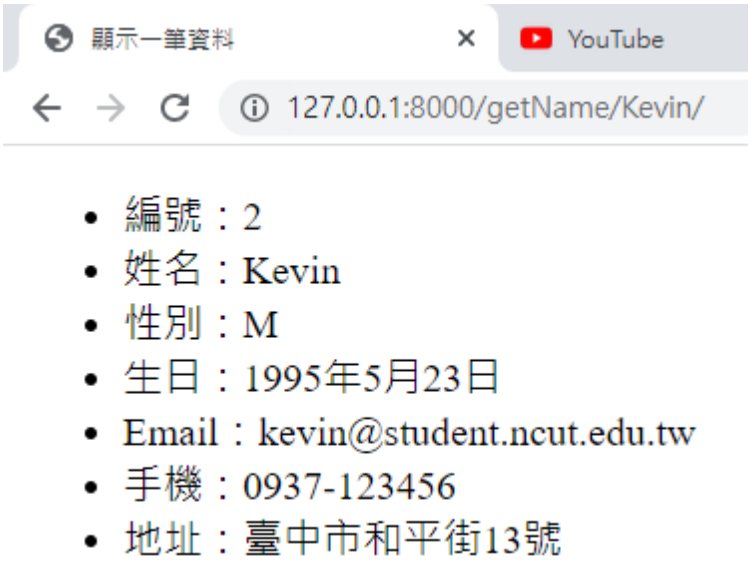
啟動伺服器測試 API

到PyCharm裡面執行比較順手，記得Terminal要選取”Command Prompt”

```
(django) C:\Users\226083\django_test\myweb>py manage.py runserver
```

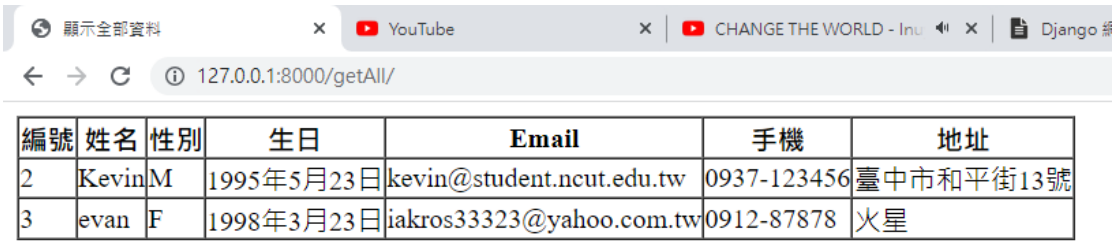
打開瀏覽器，輸入網址：http://127.0.0.1:8000/getName/Kevin/，出現畫面如下：

(”Kevin”這筆資料之前沒刪掉)



打開瀏覽器，輸入網址：http://127.0.0.1:8000/getAll/，出現畫面如下：

(另一筆資料是後來建立的，用來測試是否能查到全部資料)



編號	姓名	性別	生日	Email	手機	地址
2	Kevin	M	1995年5月23日	kevin@student.ncut.edu.tw	0937-123456	臺中市和平街13號
3	evan	F	1998年3月23日	iakros33323@yahoo.com.tw	0912-87878	火星