

# 編寫你的第一個 Django 應用，第 7 部分

這篇教學承接 [教學第 6 部分](#) 結束的地方。我們繼續修改在線投票應用，這次我們專注於自定義我們在 [教學第 2 部分](#) 初見過的 Django 自動產生管理的過程。



## 從哪裡取得協助：

如果你在閱讀本教學的過程中有任何疑問，可以前往FAQ的[doc:Getting Help](#)的版塊。

## 自定義管理表單

透過 `admin.site.register(Question)` 註冊 `Question` 模型，Django 能夠構建一個預設的表單用於展示。通常來說，你期望能自定義表單的外觀和工作方式。你可以在註冊模型時將這些設置告訴 Django。

讓我們透過重排欄表單上的欄位來看看它是怎麼工作的。用以下內容替換 `admin.site.register(Question)`：

```
polls/admin.py

from django.contrib import admin

from .models import Question

class QuestionAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question_text']

admin.site.register(Question, QuestionAdmin)
```



你需要遵循以下流程——建立一個模型管理類，接著將其作為第二個參數傳給 `admin.site.register()` ——在你需要修改模型的管理管理選項時這麼做。

以上修改使得 "Publication date" 欄位顯示在 "Question" 欄位之前：

Home > Polls > Questions > What's up?

### Change question

Date published:      Date: 2015-09-06      Today | 📅  
Time: 21:16:20      Now | ⌚

Question text:      What's up?

這在只有兩個欄位時並不特別引人注意，但對於擁有數十個欄位的表單來說，為表單選擇一個直觀的排序方法就是個很有用的細節。

說到擁有數十個欄位的表單，你可能更期望將表單分為幾個欄位集：

```
polls/admin.py

from django.contrib import admin

from .models import Question
```



```
class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date']}),
    ]

admin.site.register(Question, QuestionAdmin)
```

**fieldsets** 元組中的第一個元素是欄位集的標題。以下是我們的表單現在的樣子：

Home › Polls › Questions › What's up?

Change question


Question text:

What's up?


Date information

Date published:

Date: 2015-09-06

Today | 

Time: 21:16:20

Now | 

## 增加關聯的對象

好了，現在我們有了投票的管理頁。不過，一個 **Question** 有多個 **Choice**，但管理頁卻沒有顯示多個選項。

好了。

有兩個方法可以解決這個問題。第一個就是仿照我們向管理注冊 **Question** 一樣注冊 **Choice**：

```
polls/admin.py

from django.contrib import admin

from .models import Choice, Question
# ...
admin.site.register(Choice)
```



現在 "Choices" 在 Django 管理頁中是一個可用的選項了。“增加選項”的表單看起來像這樣：

Home › Polls › Choices › Add choice

Add choice

Question:

-----

Choice text:

Votes:

0

在這個表單中，"Question" 欄位是一個套件含資料庫中所有投票的選擇框。Django 知道要將 ForeignKey 在管理中以選擇框 `<select>` 的形式展示。此時，我們只有一個投票。

同時也注意下 "Question" 旁邊的“增加”按鈕。每個使用 **ForeignKey** 關聯到另一個對象的對象會自動取得這個功能。當你點擊“增加”按鈕時，你會見到一個套件含“增加投票”的表單。如果你在這個對話框中增加了一個投票，並點擊了“儲存”，Django 會將其儲存至資料庫，並動態地在你正在查看的“增加選項”表單中選中它。

不過，這是一種很低效地增加“選項”的方法。更好的辦法是在你建立“投票”對象時直接增加好幾個選項。讓我們實現它。

移除調用 **register()** 注冊 **Choice** 模型的程式。隨後，像這樣修改 **Question** 的注冊程式：

```
polls/admin.py

from django.contrib import admin

from .models import Choice, Question


class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)
```

這會告訴 Django：“**Choice** 對象要在 **Question** 管理頁面編輯。預設提供 3 個足夠的選項欄位。”

載入“增加投票”頁面來看看它看起來的樣子：

## Add question

Question text:

### Date information (Hide)

Date published:

Date:

Today



Time:

Now



### CHOICES

Choice: #1

Choice text:

Votes:

0

Choice: #2

Choice text:

Votes:

0

Choice: #3

Choice text:

Votes:

0

[+ Add another Choice](#)

Save and add another

Save and continue editing

SAVE

它看起來像這樣：有三個關聯的選項位置——由 **extra** 定義，且每次你返回任意已建立的對象的“修改”頁面時，你會見到三個新的位置。

在三個位置的末端，你會看到一個“增加新選項”的按鈕。如果你單擊它，一個新的位置會被增加。如果你想移除已有的位置，可以點擊位置右上角的X。注意，你不能移除原始的3個位置。以下圖片展示了一個已增加的位置：

CHOICES	
Choice: #1	
Choice text:	<input type="text"/>
Votes:	<input type="text" value="0"/>
Choice: #2	
Choice text:	<input type="text"/>
Votes:	<input type="text" value="0"/>
Choice: #3	
Choice text:	<input type="text"/>
Votes:	<input type="text" value="0"/>
Choice: #4 <span>✕</span>	
Choice text:	<input type="text"/>
Votes:	<input type="text" value="0"/>
<a href="#">+ Add another Choice</a>	

不過，仍然有點小問題。它佔據了大量的螢幕區域來顯示所有關聯的 **Choice** 對象的欄位。對於這個問題，Django 提供了一種表格式的單行顯示關聯對象的方法。要使用它，只需按如下形式修改 **ChoiceInline** 申明：

polls/admin.py

```
class ChoiceInline(admin.TabularInline):  
    #...
```

透過 **TabularInline**（取代 **StackedInline**），關聯對象以一種表格式的方式展示，顯得更加緊湊：

CHOICES		
CHOICE TEXT	VOTES	DELETE?
<input type="text"/>	<input type="text" value="0"/>	
<input type="text"/>	<input type="text" value="0"/>	
<input type="text"/>	<input type="text" value="0"/>	
<a href="#">+ Add another Choice</a>		
<div><button>Save and add another</button><button>Save and continue editing</button><button>SAVE</button></div>		

注意這裡有一個額外的“刪除？”欄，這允許移除透過“增加新選項”按鈕增加的，或是已被儲存的行。

## 自定義管理更改欄表

現在投票的管理頁看起來很不錯，讓我們對“更改欄表”頁面進行一些調整——改成一個能展示系統中所有投票的頁面。

以下是它此時的外觀：

Home > Polls > Questions

Select question to change

ADD QUESTION +

Action:  Go 0 of 1 selected

☐ QUESTION

☐ What's up?

1 question

預設情況下，Django 顯示每個對象的 `str()` 返回的值。但有時如果我們能夠顯示單個欄位，它會更有協助。為此，使用 `list_display` 管理選項，它是一個套件含要顯示的欄位名的元組，在更改欄表頁中以欄的形式展示這個對象：

```
polls/admin.py

class QuestionAdmin(admin.ModelAdmin):
    # ...
    list_display = ('question_text', 'pub_date')
```

另外，讓我們把 [教學第 2 部分](#) 中的 `was_published_recently()` 方法也加上：

```
polls/admin.py

class QuestionAdmin(admin.ModelAdmin):
    # ...
    list_display = ('question_text', 'pub_date', 'was_published_recently')
```

現在修改投票的欄表頁看起來像這樣：

Home > Polls > Questions

Select question to change

Action:  Go 0 of 1 selected

<input type="checkbox"/> QUESTION TEXT	DATE PUBLISHED	WAS PUBLISHED RECENTLY
<input type="checkbox"/> What's up?	Sept. 3, 2015, 9:16 p.m.	False

1 question

你可以點擊欄標題來對這些行進行排序——除了 `was_published_recently` 這個欄，因為沒有實現排序方法。順便看下這個欄的標題 `was_published_recently`，預設就是方法名（用空格替換下劃線），該欄的每行都以字符串形式展示出處。

你可以透過給這個方法（在 `polls/models.py` 中）一些屬性來達到優化的目的，像這樣：

```
polls/models.py

class Question(models.Model):
    # ...
    def was_published_recently(self):
        now = timezone.now()
```

```
        return now - datetime.timedelta(days=1) <= self.pub_date <= now
    was_published_recently.admin_order_field = 'pub_date'
    was_published_recently.boolean = True
    was_published_recently.short_description = 'Published recently?'
```

更多關於這些方法屬性的資訊，參見 [list\\_display](#)。

再次編輯文件 `polls/admin.py`，優化 `Question` 變更頁：過濾器，使用 [list\\_filter](#)。將以下程式增加至 `QuestionAdmin`：

```
list_filter = ['pub_date']
```

這樣做增加了一個“過濾器”側邊欄，允許人們以 `pub_date` 欄位來過濾欄表：

Home > Polls > Questions

Select question to change

ADD QUESTION +

Action:  Go 0 of 1 selected

<input type="checkbox"/>	QUESTION TEXT	DATE PUBLISHED	PUBLISHED RECENTLY?
<input type="checkbox"/>	What's up?	Sept. 3, 2015, 9:16 p.m.	

1 question

FILTER

By date published

Any date

Today

Past 7 days

This month

This year

展示的過濾器類型取決於你要過濾的欄位的類型。因為 `pub_date` 是類 `DateTimeField`，Django 知道要提供哪個過濾器：“任意時間”，“今天”，“過去7天”，“這個月”和“今年”。

這已經弄的很好了。讓我們再擴充些功能：

```
search_fields = ['question_text']
```

在欄表的頂部增加一個搜索框。當輸入待搜項時，Django 將搜索 `question_text` 欄位。你可以使用任意多的欄位——由於管理使用 **LIKE** 來查詢數據，將待搜索的欄位數限制為一個不會出問題大小，會便於資料庫進行查詢操作。

現在是給你的修改欄表頁增加分頁功能的好時機。預設每頁顯示 100 項。[變更頁分頁](#)，[搜索框](#)，[過濾器](#)，[日期層次結構](#)，和 [欄標題排序](#) 均以你期望的方式合作執行。

## 自定義管理界面和風格

在每個管理頁頂部顯示“Django 管理員”顯得很滑稽。這只是一串佔位文本。

不過，你可以透過 Django 的模板系統來修改。Django 的管理由自己驅動，且它的交互接口採用 Django 自己的模板系統。

## 自定義你的 工程的模板

在你的工程目錄（指套件含 `manage.py` 的那個文件夾）內建立一個名為 `templates` 的目錄。模板可放在你系統中任何 Django 能找到的位置。（誰啟動了 Django，Django 就以他的用戶身份執行。）不過，把你的模板放在工程內會帶來很大便利，推薦你這樣做。

打開你的設置文件（`mysite/settings.py`，牢記），在 `TEMPLATES` 設置中增加 `DIRS` 選項：

```
mysite/settings.py
```



```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

```

**DIRS** 是一個套件含多個系統目錄的文件欄表，用於在載入 Django 模板時使用，是一個待搜索路徑。



### 組織模板

就像靜態文件一樣，我們 可以把所有的模板文件放在一個大模板目錄內，這樣它也能工作的很好。但是，屬於特定應用的模板文件最好放在應用所屬的模板目錄（例如 **polls/templates**），而不是工程的模板目錄（**templates**）。我們會在 [建立可復用的應用教學](#) 中討論 為什麼我們要這樣做。

現在，在 **templates** 目錄內建立名為 **admin** 的目錄，隨後，將存放 Django 預設模板的目錄（**django/contrib/admin/templates**）內的模板文件 **admin/base\_site.html** 復制到這個目錄內。



### Django 的源文件在哪裡？

如果你不知道 Django 源碼在你系統的哪個位置，執行以下命令：

```
$ python -c "import django; print(django.__path__)"
```



接著，用你網頁網站的名字編輯替換文件內的 `{% site_header|default:_('Django administration') %}`（套件含大括號）。完成後，你應該看到如下程式：

```

{% block branding %}
<h1 id="site-name"><a href="{% url 'admin:index' %}">Polls Administration</a></h1>
{% endblock %}

```

我們會用這個方法來教你復寫模板。在一個實際工程中，你可能更期望使用 [django.contrib.admin.AdminSite.site\\_header](#) 來進行簡單的定制。

這個模板文件套件含很多類似 `{% block branding %}` 和 `{{ title }}` 的文本。`{%}` 和 `{{` 標籤是 Django 模板語言的一部分。當 Django 渲染 **admin/base\_site.html** 時，這個模板語言會被求值，產生最終的網頁，就像我們在 [教學第 3 部分](#) 所學的一樣。

注意，所有的 Django 預設管理模板均可被復寫。若要復寫模板，像你修改 **base\_site.html** 一樣修改其它文件——先將其從預設目錄中拷貝到你的自定義目錄，再做修改。

## 自定義你 應用的模板



機智的同學可能會問：[DIRS](#) 預設是空的，Django 是怎麼找到預設的管理模板的？因為 [APP\\_DIRS](#) 被置為 **True**，Django 會自動在每個應用套件內遞歸查找 **templates/** 子目錄（不要忘了 **django.contrib.admin** 也是一個應用）。

我們的投票應用不是非常複雜，所以無需自定義管理模板。不過，如果它變的更加複雜，需要修改 Django 的標準管理模板功能時，修改 *應用* 的模板會比 *工程* 的更加明智。這樣，在其它工程套件含這個投票應用時，可以確保它總是能找到需要的自定義模板文件。

更多關於 Django 如何查找模板的文件，參見 [載入模板文件](#)。

---

## 自定義管理主頁

在類似的說明中，你可能想要自定義 Django 管理索引頁的外觀。

預設情況下，它展示了所有設定在 [INSTALLED\\_APPS](#) 中，已透過管理應用注冊，按拼音排序的應用。你可能想對這個頁面的布局做重大的修改。畢竟，索引頁是管理的重要頁面，它應該便於使用。

需要自定義的模板是 **admin/index.html**。（像上一節修改 **admin/base\_site.html** 那樣修改此文件——從預設目錄中拷貝此文件至自定義模板目錄）。打開此文件，你將看到它使用了一個叫做 **app\_list** 的模板變數。這個變數套件含了每個安裝的 Django 應用。你可以用任何你期望的硬編碼連結（連結至特定對象的管理頁）取代使用這個變數。

---

## 接下來要做什麼？

初學者教學到這就結束了。隨後，你可能想閱讀 [下一步看什麼](#)，看看下一步能做什麼。

如果你很熟悉 Python 打套件，且對學習如何把投票應用改成“可復用應用”感興趣，查看 [進階教學：如何建立可復用應用](#)。

---

[◀ 編寫你的第一個 Django 應用，第 6 部分](#)

[進階指南：如何編寫可重用程式](#) ▶

---