



django girls

學習指南

目錄

Django Girls 學習指南	0
Django 介紹	1
安裝 Django	2
Project and Apps	3
Views and URLconfs	4
Templates	5
Models	6
Admin	7
Django ORM	8
Template Tags	9
Dynamic URL	10
Deploy	11
What's Next?	12

Django Girls 學習指南

這份學習指南適合所有 Django 初學者，為了更好的學習效果，我們希望你能具備：

- Web 的初步認識
- 了解如何使用 Command Line
- 略懂 Python 基礎語法
- 看得懂簡單的 HTML / CSS

學習前準備

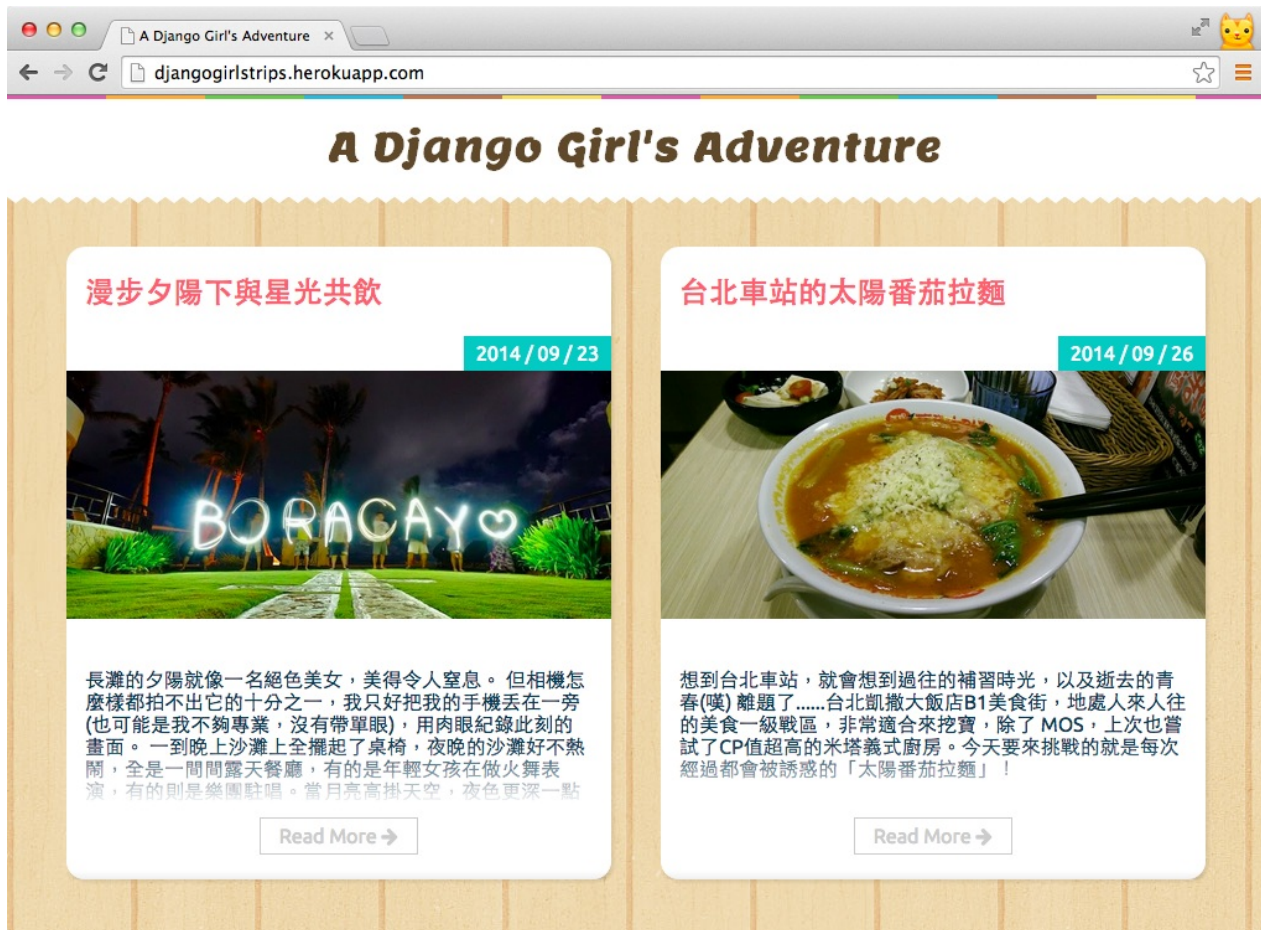
在使用這份指南前，請先準備好：

1. [安裝 Python 3.4](#)
2. [安裝 Git](#)
3. [Heroku 註冊與安裝設定](#)

學習範例

透過這份學習指南，你會學習到 Django 的程式架構，從創建一個專案，到最後將網站發佈到網路上，建立一個屬於自己的旅遊日記。

旅遊日記首頁



旅遊日記 - 單篇日記頁面

A Django Girl's Adventure

台北車站的太陽番茄拉麵

2014 / 09 / 26

台北車站



想到台北車站，就會想到過往的補習時光，以及逝去的青春(嘆) 離題了.....台北凱撒大飯店B1美食街，地處人來人往的美食一級戰區，非常適合來挖寶，除了 MOS，上次也嘗試了CP值超高的米塔義式廚房。今天要來挑戰的就是每次經過都會被誘惑的「太陽番茄拉麵」！



Django 介紹

[Django](#) (/ˈdʒæŋɡoʊ/ jang-goh) 可以說是 Python 最著名的 Web Framework，一些知名的網站如 [Pinterest](#), [Instagram](#), [Disqus](#) 等等都使用過它來開發。

它有以下的特色：

- 免費開放原始碼
- 著重快速開發、高效能
- 遵從 [DRY \(Don't Repeat Yourself \)](#) 守則，致力於淺顯易懂和優雅的程式碼
- 使用類似 Model–view–controller (MVC) pattern 的架構

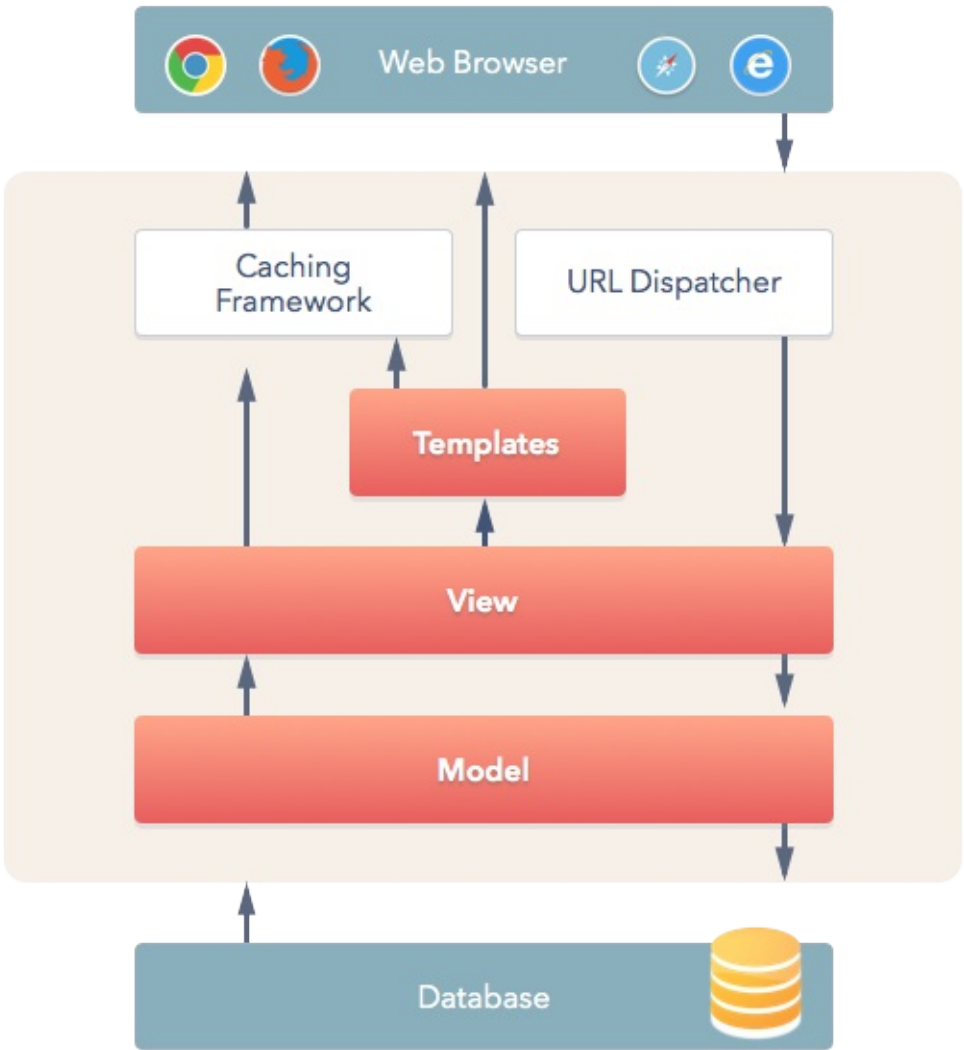
Web Framework

Web framework，簡單來說就是當你開發 Web 應用程式時所用的框架。它通常會提供：

1. 一個既定的程式骨架 -- 你必須按照它的規範寫程式，例如把資料庫相關的程式與跟畫面溝通的程式分開，而不是全部寫在同一個檔案。這對於程式的開發速度、再利用性、和程式可讀性等等都有相當大的好處。
2. 強大且豐富的函式庫 (**Libraries**) -- 通常會提供一些開發網站所需要且常用的功能，例如使用者認證、安全機制、URL mapping、資料庫連接等等。讓你在開發網站時可以直接使用函式庫，然後專注在客製化自己的功能。

Django 的 架構

如同一些比較著名的 Web framework，Django 同樣也使用了類似 MVC 的架構，只是在定義和解釋上略為不同，稱為 MTV (Model–Template–View)，我們可以透過下面這張圖來了解其運作方式：



安裝 Django

在這一章，我們會開始建立第一個 Django 專案，並瞭解如何使用虛擬環境。

首先，請開啟終端機，確定現在的位置是在家目錄底下：

我們先建立一個名為 `djangogirls` 的資料夾

```
mkdir djangogirls
```

並切換至剛剛建立的目錄

```
cd djangogirls
```

虛擬環境（virtualenv）

在安裝 Django 之前，我們要先建立一個虛擬環境（virtual environment）。

我們可以直接開始安裝 Django，但實務上，大多數人都會搭配使用虛擬環境。使用虛擬環境有許多優點：

- 你的專案會擁有一個專屬的獨立 Python 環境。
- 不需要 root 權限，就可以安裝新套件。
- 方便控管不同版本的套件，不用擔心升級套件會影響到其他專案。
- 如果需要多人協作或在不同機器上跑同一個專案時，使用虛擬環境也可以確保環境一致性。

創建虛擬環境

在較舊的 Python 版本中，建立處擬環境需要另外安裝。但 Python 3.3 已經加入 `venv` 模組，可以直接使用。

那我們立刻開始，首先要創建一個虛擬環境資料夾 `djangogirls_venv`。

Windows

如果有按照安裝教學，使用 **Django Environment** 開啟終端機後，輸入以下指令：

```
C:\Users\YOUR_NAME\djangogirls> python -m venv djangogirls_venv
```

Linux / OS X

Linux 或 OS X 需要使用 `python3` 來建立虛擬環境，指令如下：

```
~/djangogirls$ python3 -m venv djangogirls_venv
```

切換虛擬環境

虛擬環境建立完成後，我們可以透過 `activate` 這個 script 來啟動它。

記得未來在安裝新套件，或是要執行 Django 相關指令時，都要先啟動該專案的虛擬環境。

Windows

```
C:\Users\YOUR_NAME\djangogirls> djangogirls_venv\Scripts\activate
```



Linux / OS X

```
~/djangogirls$ source djangogirls_venv/bin/activate
```

如果無法使用 `source` 的話，可以用下列指令替代：

```
~/djangogirls$ . djangogirls_venv/bin/activate
```

目前的虛擬環境

如果看到前面多了 (虛擬資料夾名稱) , 則表示已經成功切換至該虛擬環境。

Windows

```
(djangogirls_venv) C:\Users\YOUR_NAME\djangogirls>
```

Linux / OS X

```
(djangogirls_venv) ~/djangogirls$
```

安裝 Django 1.8 最新版本

開始安裝

Python 3.4 預先安裝了 `pip` 這個強大的套件管理工具, 我們將使用它來安裝 Django :

```
(djangogirls_venv) ~/djangogirls$ pip install "django<1.9"
```

這裡需要特別注意, 我們使用的指令是 `"django <1.9 "`。這樣一來才可以確保我們安裝的是 **Django 1.8** 的最新版本

輸入了應該會看到如下的訊息, 表示安裝成功

```
Installing collected packages: django
Successfully installed django-1.8.6
```

註: 如果你看到以 *Fatal error in launcher* 開頭的輸出, 而不是上面的安裝成功訊息, 請改用 `python -m pip install "django<1.9"` 試試看。之後如果在使用 `pip` 時遇到類似問題, 也可以試著在前面加上 `python -m`。

確認安裝成功

最後, 讓我們最後來測試一下。

請在虛擬環境下指令輸入 `python`，進入互動式命令列環境

```
(djangogirls_venv) ~/djangogirls$ python
```

輸入以下的指令取得 Django 版本資訊：

```
>>> import django
>>> django.VERSION
(1, 8, 6, 'final', 0')
```

如果看見類似上面的訊息，就代表安裝成功囉！

Project and apps

每一個 Django project 裡面可以有多個 Django apps，可以想成是類似模組的概念。在實務上，通常會依功能分成不同 **app**，方便未來的維護和重複使用。

例如，我們要做一個類似 Facebook 這種網站時，依功能可能會有以下 apps：

- 使用者管理 -- accounts
- 好友管理 -- friends
- 塗鴉牆管理 -- timeline
- 動態消息管理 -- news

若未來我們需要寫個購物網站，而需要會員功能時，`accounts` app（使用者管理）就可以被重複使用。

這一章，你會學到如何使用 Django 命令列工具建立 Django project 和一個 Django app。

建立 Django project

建立專案資料夾 -- `startproject`

首先，使用 `django-admin.py` 來建立第一個 Django project `mysite`：

```
(djangogirls_venv) ~/djangogirls$ django-admin.py startproject mysite
```

此時會多了一個 **mysite** 資料夾。我們切換進去：

```
(djangogirls_venv) ~/djangogirls$ cd mysite
```

`startproject` 這個 Django 指令除了建立專案資料夾，也預設會建立一些常用檔案，你可以使用 `ls` 或 `dir /w` (Windows) 檢視檔案結構。

目前 project 的檔案結構如下：

```
mysite/
├─ manage.py
└─ mysite
    ├─ __init__.py
    ├─ settings.py
    ├─ urls.py
    └─ wsgi.py
```

瞭解 Django 的 Management commands

`manage.py` 是 Django 提供的命令列工具，我們可以利用它執行很多工作，例如同步資料庫、建立 app 等等，指令的使用方式如下：

```
python manage.py <command> [options]
```

如果你想要了解有什麼指令可以使用，輸入 `help` 或 `-h` 指令會列出所有指令列表：

```
python manage.py -h
```

而如果想了解其中一個指令，可以在指令名字後輸入 `-h`，你會看到簡單的指令介紹以及用法說明。以 `runserver` 為例：

```
(djangogirls_venv) ~/djangogirls/mysite$ python manage.py runserver
usage: manage.py runserver [-h] [--version] [-v {0,1,2,3}]
                           [--settings SETTINGS] [--pythonpath PYTHONPATH]
                           [--traceback] [--no-color] [--ipv6] [--noreload]
                           [--nostatic] [--insecure] [addrport]
```

Starts a lightweight Web server for development and also serves static files.

positional arguments:

addrport Optional port number, or ipaddr:port

optional arguments:

-h, --help show this help message and exit

--version show program's version number and exit

-v {0,1,2,3}, --verbosity {0,1,2,3} Verbosity level; 0=minimal output, 1=normal output, 2=verbose output, 3=very verbose output

--settings SETTINGS The Python path to a settings module, e.g. "myproject.settings.main". If this isn't provided, the DJANGO_SETTINGS_MODULE environment variable will be used.

--pythonpath PYTHONPATH A directory to add to the Python path, e.g. "/home/djangoprojects/myproject".

--traceback Raise on CommandError exceptions

--no-color Don't colorize the command output.

--ipv6, -6 Tells Django to use an IPv6 address.

--nothreading Tells Django to NOT use threading.

--noreload Tells Django to NOT use the auto-reloader.

--nostatic Tells Django to NOT automatically serve static files at STATIC_URL.

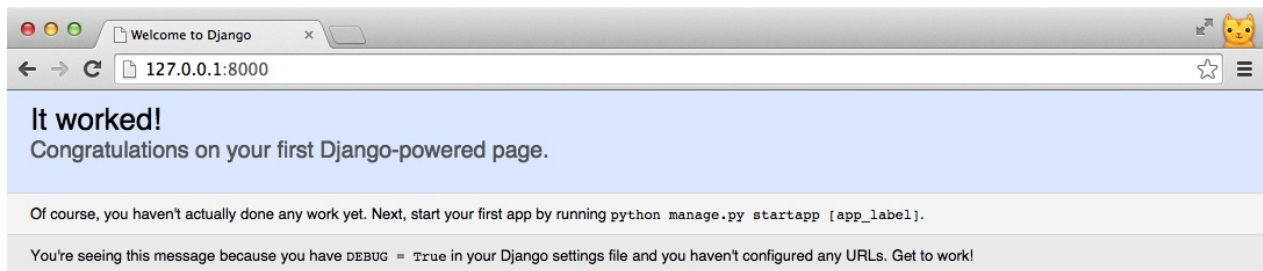
--insecure Allows serving static files even if DEBUG is False.

啟動開發伺服器 -- **runserver**

從說明中可以知道，`runserver` 會啟動一個簡單的 web server，方便於在開發階段使用：

```
(django girls_venv) ~/django girls/mysite$ python manage.py runserver
...
Django version 1.8.5, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

現在打開瀏覽器輸入 <http://127.0.0.1:8000/> 或是 <http://localhost:8000/>，會看到你的 django 專案已成功在 web server 上執行



最後我們可以在終端機按下 `CTRL+C`，關閉 web server 回到命令列。

如果無法看到成功畫面，瀏覽器上顯示錯誤訊息 - *A server error occurred. Please contact the administrator.*，請輸入：

```
(django girls_venv) ~/django girls/mysite$ python manage.py migrate
```

然後再次 `runserver` 啟動你的 web server，我們會在 **Django Models** 解釋 `migrate` 的作用。

建立 Django application (app)

讓我們利用 `startapp` 建立第一個 Django app -- **trips**:

```
(django girls_venv) ~/django girls/mysite$ python manage.py startapp
```


`startapp` 會按照你的命名建立一個同名資料夾和 app 預設的檔案結構如下：

```
trips
├── __init__.py
├── admin.py
├── migrations
├── models.py
├── tests.py
└── views.py
```

將新增的 **Django app** 加入設定檔

在前一個指令，我們透過 Django 命令列工具建立了 **trips** 這個 app。但若要讓 Django 知道要管理哪些 apps，還需再調整設定檔。

新增 app

打開 `mysite/settings.py`，找到 `INSTALLED_APPS`，調整如下：

```
# mysite/settings.py

...

# Application definition

INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'trips',
)
```

請注意 app 之間有時候需要特定先後順序。在此，我們將自訂的 `trips` 加在最後面。

預設安裝的 Django app

Django 已將常用的 app 設定為 `INSTALLED_APPS`。例如，`auth`（使用者認證）、`admin`（管理後台）... 等等，我們可依需求自行增減。

小結

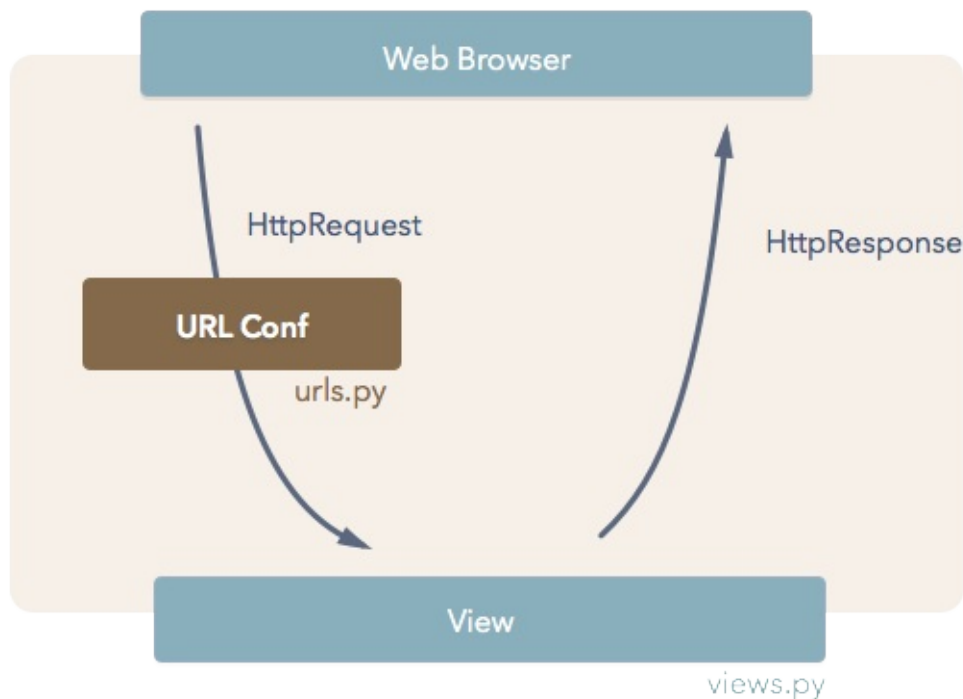
目前為止，我們使用 `startproject` 建立了一個名為 **mysite** 的 Django 專案，和一個名為 **trips** 的 Django app。

```
mysite
├── manage.py
├── mysite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── trips
    ├── __init__.py
    ├── admin.py
    ├── migrations
    ├── models.py
    ├── tests.py
    └── views.py
```

最後，我們回顧一下本章學到的指令

指令	說明
<code>django-admin.py startproject <project_name></code>	建立 Django 專案
<code>python manage.py -h <command_name></code>	查看 Django commands 的使用方法
<code>python manage.py runserver</code>	啟動開發伺服器
<code>python manage.py startapp <app_name></code>	新增 Django app

Views and URLconfs



在前面的介紹，我們有提到 Django 的 MTV 架構。其處理 request 的流程如下：

1. 瀏覽器送出 **HTTP request**
2. Django 依據 **URL configuration** 分配至對應的 View
3. View 進行資料庫的操作或其他運算，並回傳 `HttpResponse` 物件
4. 瀏覽器依據 **HTTP response** 顯示網頁畫面

這一章，我們將透過 **Hello World** 範例，瞭解 Django 如何處理一個 request 的流程。

Django Views

Django view 其實是一個 function，處理 `HttpRequest` 物件，並回傳 `HttpResponse` 物件，大致說明如下：

- 會收到 `HttpRequest` 參數：Django 從網頁接收到 request 後，會將 request 中的資訊封裝產生一個 `HttpRequest` 物件，並當成第一個參數，傳入對應的 view function。
- 需要回傳 `HttpResponse` 物件：`HttpResponse` 物件裡面包含：
 - `HttpResponse.content`
 - `HttpResponse.status_code` ...等

建立第一個 View

首先建立一個名為 `hello_world` 的 view。

在 `trips/views.py` 輸入下列程式碼：

```
# trips/views.py

from django.http import HttpResponse

def hello_world(request):
    return HttpResponse("Hello World!")
```

以上程式在做的事就是：

1. 從 `django.http` 模組中引用 `HttpResponse` 類別
2. 宣告 `hello_world` 這個 view
3. 當 `hello_world` 被呼叫時，回傳包含字串 **Hello World!** 的 `HttpResponse` 物件。

Django URL 設定

最後，Django 需要知道 **URL** 與 **view** 的對應關係。

例如：

有人瀏覽 <http://127.0.0.1:8000/hello/> 時，`hello_world()` 這個 view function 需要被執行。

而這個對應關係就是 **URL conf** (URL configuration)。

URL Conf

- 通常定義在 `urls.py`
 - 是一連串的規則 (URL patterns)
 - Django 收到 request 時，會一一比對 URL conf 中的規則，決定要執行哪個 view function
-

現在我們來設定 Hello World 範例的 URL conf。

首先打開 `mysite/urls.py`，先 import 剛剛寫的 view function，然後在 `urlpatterns` 中加入下面這行：

```
url(r'^hello/$', hello_world),
```

現在 `mysite/urls.py` 的內容應該會像下面這樣：

```
# mysite/urls.py

from django.conf.urls import include, url
from django.contrib import admin
# Import view functions from trips app.
from trips.views import hello_world

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^hello/$', hello_world),
]
```

以上程式透過 `url()` function 傳入兩個參數 `regex`，`view`：

```
url(regex, view)
```

- **regex** -- 定義的 URL 規則
 - 規則以 regular expression（正規表示式）來表達

- `r'^hello/$'` 代表的是 `hello/` 這種 URL
- **view** -- 對應的 view function
 - 指的是 `hello_world` 這個 view

測試 Hello World

現在啟動你的 web server。(如果剛剛沒關閉的話，通常 Django 會在你修改程式碼後，自動重新啟動 web server)

```
(djangogirls_venv) ~/djangogirls/mysite$ python manage.py runserver
```

在瀏覽器輸入 <http://127.0.0.1:8000/hello/>，你會看到網頁顯示我們在 `HttpResponse` 傳入的文字 `Hello World!`。



Templates

加上 HTML / CSS & 動態內容

上一章的例子，只是很簡單的顯示一行字串。讓我們加上一些 HTML/CSS 美化網頁，並動態顯示每次進來這個頁面的時間：

```
# trips/views.py

from datetime import datetime
from django.http import HttpResponse

def hello_world(request):
    output = """
        <!DOCTYPE html>
        <html>
            <head>
            </head>
            <body>
                Hello World! <em style="color:LightSeaGreen;">{current_time}</em>
            </body>
        </html>
    """.format(current_time=datetime.now())

    return HttpResponse(output)
```

1. 多行字串：

`"""..."""` 或是 `'''...'''` (三個雙引號或三個單引號) 是字串的多行寫法，這裡我們使用它表達 HTML，並維持原有的縮排。

2. 顯示目前時間：

為了顯示動態內容，我們 import `datetime` 時間模組，並用 `datetime.now()` 取得現在的時間。

3. 字串格式化：

使用 `format()` 格式化字串，將 `datetime.now()` 產生的值，代入 `{current_time}` 在字串中的位置。

現在啟動 web server，連至 <http://127.0.0.1:8000/hello/> 後，會發現網頁不再是純文字。除了加上了一些樣式外，也會顯示當下的時間。

你可以重新整理網頁，試試看時間有沒有改變



第一個 Template

在前一個例子，我們把 HTML/CSS 放在 View function 裡。但在實務上，我們會將前端的程式碼獨立出來，放在 `templates` 資料夾裡。不僅增加可讀性，也方便與設計師或前端工程師分工。

Template 資料夾

首先建立 Template 資料夾。開啟終端機 (如果不想關閉 web server，可以再開一個新的終端機視窗)，並確認目前所在位置為 `djangogirls/mysite/`。

新增一個名為 `templates` 的資料夾：

```
(djangogirls_venv) ~/djangogirls/mysite$ mkdir templates
```

設定 **Templates** 資料夾的位置

建立好資料夾以後，我們需要修改 `mysite/settings.py` 中的 `TEMPLATES` 設定：

```
# mysite/settings.py

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
        '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'
            ],
        },
    },
]
```

我們將 `'DIRS'` 原本的 `[]` 修改成：

```
[os.path.join(BASE_DIR, 'templates').replace('\\', '/')]
```

好讓 Django 找得到剛剛建立的 `templates` 資料夾。

建立第一個 **Template**

新增檔案 `templates/hello_world.html`，並將之前寫在 view function 中的 HTML 複製到 `hello_world.html`：

```
mysite
├─ mysite
├─ templates
│   └─ hello_world.html
├─ trips
└─ manage.py
```

為了區別，我們做了一些樣式上的調整：

```
<!-- hello_world.html -->

<!DOCTYPE html>
<html>
  <head>
    <title>I come from template!!</title>
    <style>
      body {
        background-color: lightyellow;
      }
      em {
        color: LightSeaGreen;
      }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
    <em>{{ current_time }}</em>
  </body>
</html>
```

在 **Template** 中顯示變數

以上 template 中，有個地方要特別注意：

```
<em>{{ current_time }}</em>
```

仔細比較，可以發現變數 `current_time` 的使用方式與之前不同，在這裡用的是兩個大括號。

`{{ <variable_name> }}` 是在 Django Template 中顯示變數的語法。

其它 Django Template 語法，我們會在後面的章節陸續練習到。

使用 render function

最後，將 view function `hello_world` 修改如下：

```
# trips/views.py

from datetime import datetime
from django.shortcuts import render

def hello_world(request):
    return render(request, 'hello_world.html', {
        'current_time': datetime.now(),
    })
```

我們改成用 `render` 這個 function 產生要回傳的 `HttpResponse` 物件。

這次傳入的參數有：

- **request** -- `HttpRequest` 物件
 - **template_name** -- 要使用的 template
 - **dictionary** -- 包含要新增至 template 的變數
-

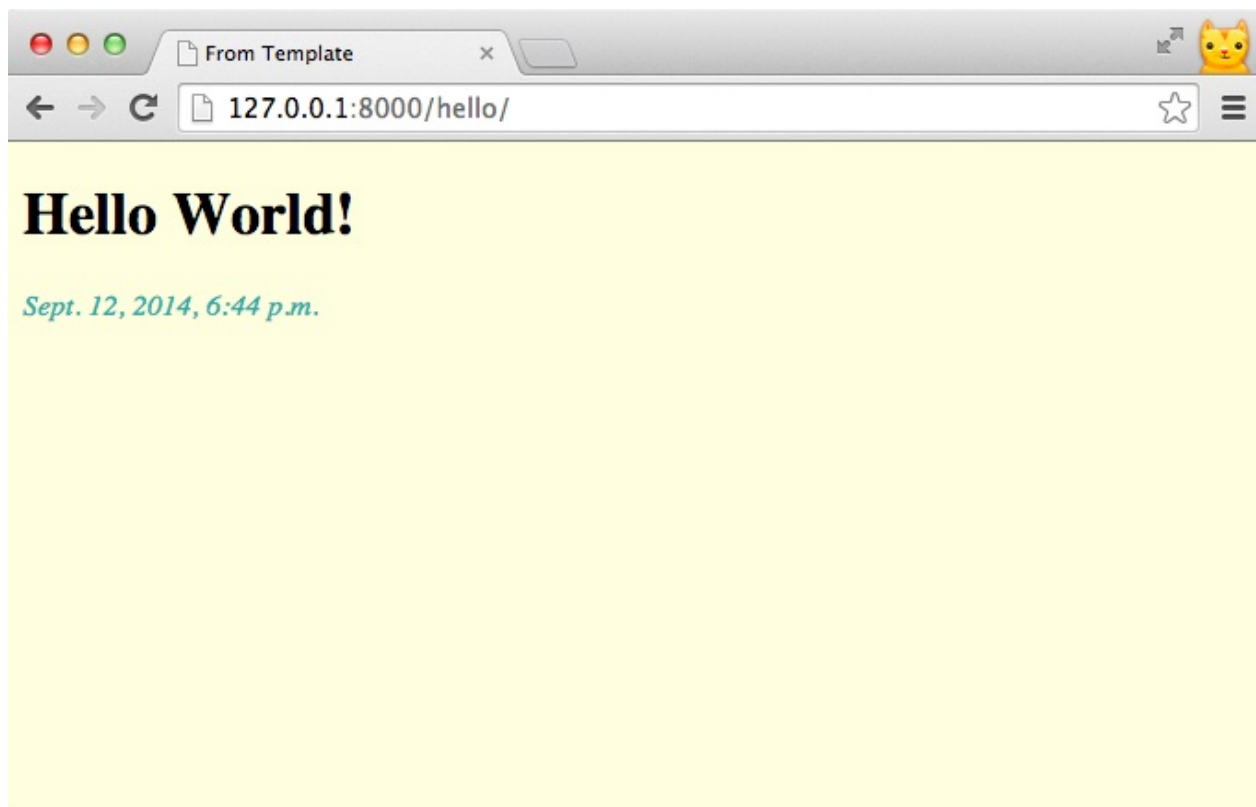
`render`：產生 `HttpResponse` 物件。

`render(request, template_name, dictionary)`

大功告成

HTML 程式碼獨立成 template 後，程式也變得簡潔許多了。

重新載入 <http://127.0.0.1:8000/hello/>，你會發現畫面有了小小的改變：



Models

現今的網站，都不再只是僅單純展示網頁內容的靜態網頁。大多數網站，都會加上一些與使用者互動的功能，如留言版、討論區、投票等等。而這些使用者產生的資料，往往會儲存於資料庫中。

這一章，你會學到如何利用 Django Model 定義資料庫的結構（schema），並透過 Django 指令創建資料庫、資料表及欄位。

使用 Django Model 的好處

雖然資料庫的語法有其標準，但是各家資料庫還是或多或少有差異。使用 Django Model 的來操作資料庫的優點之一，就是資料庫轉換相當方便。

在大部份情況下，不再需要為不同的資料庫，使用不同語法來撰寫程式。只要修改設定，就可以輕易地從 SQLite 轉換到 MySQL、PostgreSQL、或是 Oracle 等等。

設定資料庫

為了開發方便，我們使用 Python 預設的資料庫引擎 - SQLite。打開

`mysite/settings.py`，看看 `DATABASES` 的設定。它應該長得像下面這樣：

```
# mysite/settings.py

...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```


在這裡我們設定了資料庫連線的預設值：

- **ENGINE** -- 你要使用的資料庫引擎。例如：
 - MySQL: `django.db.backends.mysql`
 - SQLite 3: `django.db.backends.sqlite3`
 - PostgreSQL: `django.db.backends.postgresql_psycopg2`
- **NAME** -- 你的資料庫名稱

如果你使用 MySQL 或 PostgreSQL 等等資料庫，可能還要設定它的位置、名稱、使用者等等。不過我們這裡使用的 SQLite 3 不需要這些性質，所以可以省略。

Django Models

我們在 `trips/models.py` 宣告一個 `Post` 類別，並定義裡面的屬性，而 Django 會依據這個建立資料表，以及資料表裡的欄位設定：

```
# trips/models.py

from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField(blank=True)
    photo = models.URLField(blank=True)
    location = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
```

- Django 預設會為每一個 Model 加上 `id` 欄位，並將這個欄位設成 primary key（主鍵），簡稱 **pk**，讓每一筆資料都會有一個獨一無二的 ID。
- 為 Post 定義以下屬性：

屬性	資料型態	說明	參數
title	CharField	標題	<code>max_length=100</code> -- 標題不可以超過 100 個字元
content	TextField	內文	<code>blank=True</code> -- 非必填欄位（表單驗證時使用），預設所有欄位都是 <code>blank=False</code>
photo	URLField	照片網址	同 content，非必填欄位
location	CharField	地點	同 title
created_at	DateTimeField	建立時間	<code>auto_now_add=True</code> -- 物件新增的時間。若想設成物件修改時間，則用 <code>auto_now=True</code>

Model fields 可為 Django Model 定義不同型態的屬性。

- **CharField** -- 字串欄位，適合像 title、location 這種有長度限制的字串。
- **TextField** -- 合放大量文字的欄位
- **URLField** -- URL 設計的欄位
- **DateTimeField** -- 日期與時間的欄位，使用時會轉成 Python `datetime` 型別。

更多 Model Field 與其參數，請參考 [Django 文件](#)

同步資料庫

首先執行 `makemigrations` 指令：

```
(djangogirls_venv) ~/djangogirls/mysite$ python manage.py makemigrations
Migrations for 'trips':
  0001_initial.py:
    - Create model Post
```

這個指令會根據你對 Model 的修改刪除建立一個新的 `migration` 檔案，讓 `migrate` 指令執行時，可以照著這份紀錄更新資料庫。

接著用以下的指令，讓 Django 根據上面的紀錄，把 `models.py` 中的欄位寫入資料庫：

```
(djangogirls_venv) ~/djangogirls/mysite$ python manage.py migrate
```

結果應該類似下面這樣：

```
Operations to perform:
  Synchronize unmigrated apps: staticfiles, messages
  Apply all migrations: sessions, admin, auth, contenttypes
Synchronizing apps without migrations:
  Creating tables...
  Running deferred SQL...
  Installing custom SQL...
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying sessions.0001_initial... OK
  Applying trips.0001_initial... OK
```

`migrate` 指令會根據 `INSTALLED_APPS` 的設定，按照 app 順序建立或更新資料表，將你在 `models.py` 裡的更新跟資料庫同步。

Admin

大部份網站都設計有管理後台，讓管理者方便新增或異動網站內容。

而這樣的管理後台，Django 也有內建一個 App -- **Django Admin**。只需要稍微設定，網站就能擁有管理後台功能。

前一章，我們學到如何使用 Django Model 抽象地表達資料庫結構。現在，我們要透過 **Django Admin** 看到實際的資料，並跟資料庫進行互動。

完成本章後，你會瞭解如何設定 Django Admin，並使用 Django 管理後台，完成 Post 的新增、修改及刪除。

設定管理後台

將 **Django Admin** 加入 `INSTALLED_APPS`

後台管理的功能 Django 已預設開啟。因此，設定檔中的 `INSTALLED_APPS` 裡，已經有 `django.contrib.admin` 這個 app：

```
# mysite/settings.py

INSTALLED_APPS = (
    'django.contrib.admin',
    ...
)
```

當你在同步資料庫時，也會建立需要的資料表及欄位。

設定管理後台的 URL

為了讓你可以從瀏覽器進入管理後台，我們需要設定對應的 urls。

我們將管理後台的網址設定為 `/admin/`。確認 `mysite/urls.py` 中的 `urlpatterns` 包含下面這行：

```
url(r'^admin/', include(admin.site.urls)),
```

建立 superuser

要使用 Django 的管理後台，需要一個管理員帳號。

使用 `createsuperuser` 這個指令，建立一個 superuser：

```
(djangoenv) ~/djangoenv$ python manage.py createsuperuser
Username (leave blank to use 'YOUR_NAME'):
Email address: your_name@yourmail.com
Password:
Password (again):
Superuser created successfully.
```

輸入帳號、Email、密碼等資訊，就完成 superuser 的新增了。

註冊 Model class

最後，我們需要在讓 Django 知道，有哪些 Model 需要管理後台。

修改 `trips app` 裡的 `admin.py`，並註冊 `Post` 這個 Model：

```
# trips/admin.py

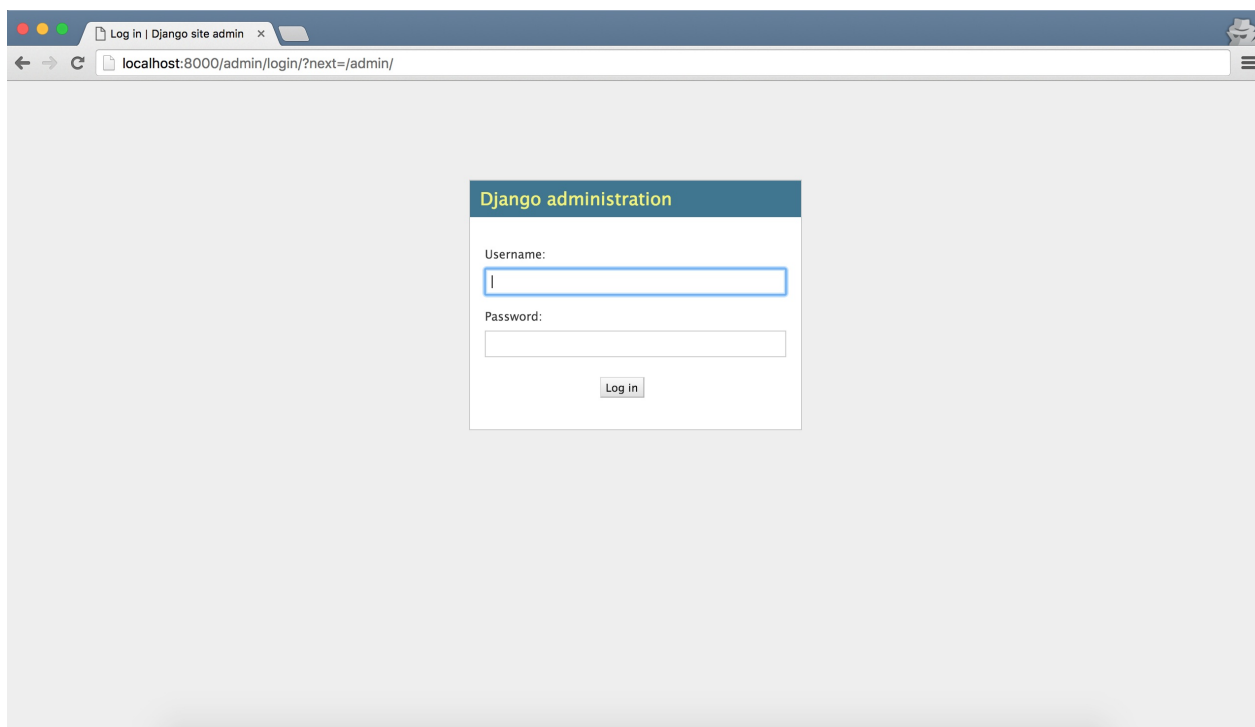
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

使用管理後台

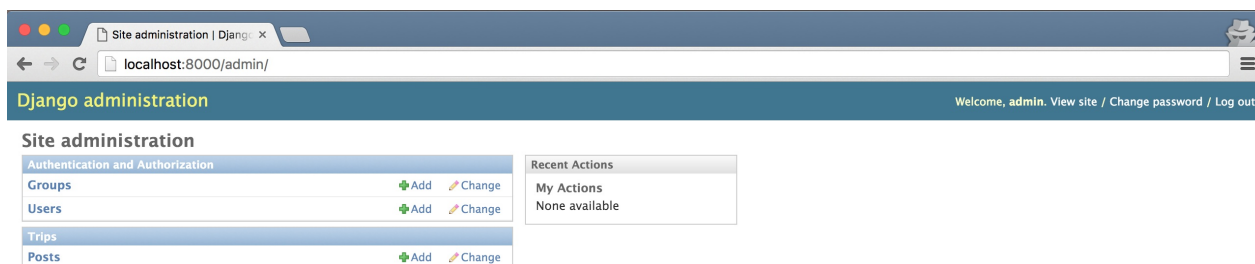
進入管理後台

連至 <http://127.0.0.1:8000/admin>，可以看到管理後台的登入頁面：



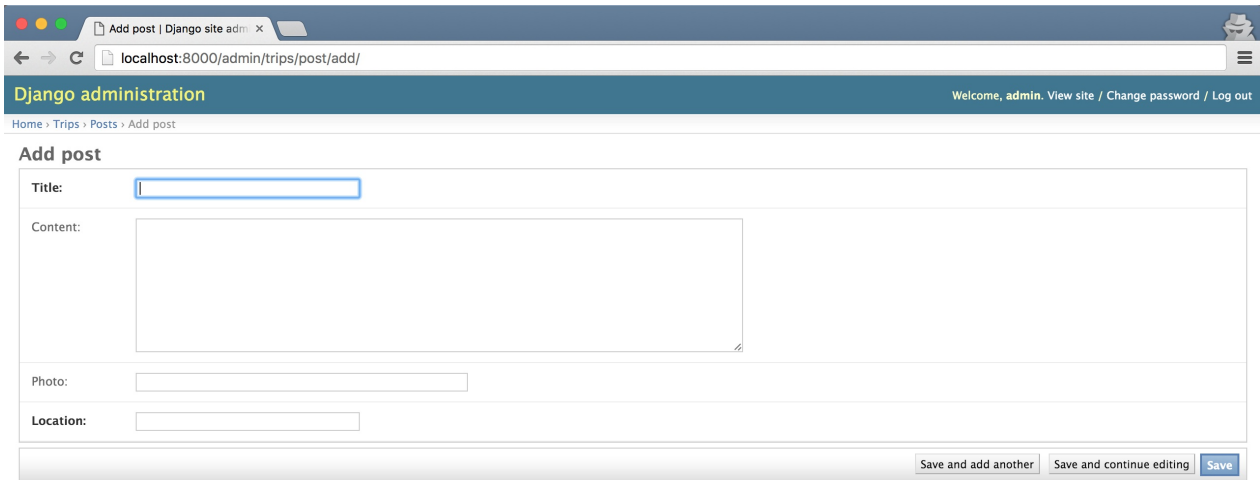
請輸入你剛創立的 superuser 帳號密碼，進入管理後台：

第一個區塊 **Authentication and Authorization**，可以管理使用者（User）和群組（Group）；第二個 **Trips** 區塊裡，則可以看到剛剛設定的 Post model。在這裡可以執行 Post 的新增、修改、刪除等功能。



新增一個 Post

現在試著建立一個新的 Post 看看：

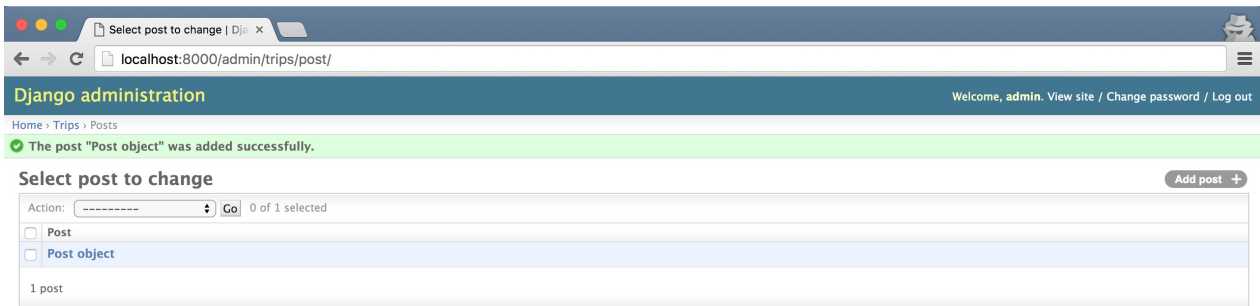


The screenshot shows the 'Add post' form in the Django administration interface. The browser address bar shows 'localhost:8000/admin/trips/post/add/'. The form has the following fields:

- Title:** A text input field.
- Content:** A large text area.
- Photo:** A text input field.
- Location:** A text input field.

At the bottom right of the form, there are three buttons: 'Save and add another', 'Save and continue editing', and 'Save'.

建立成功後會回到 Posts 頁面，你會發現有一筆資料顯示為 `Post object`：



The screenshot shows the 'Select post to change' page in the Django administration interface. The browser address bar shows 'localhost:8000/admin/trips/post/'. A green message at the top states: 'The post "Post object" was added successfully.' Below this, the 'Select post to change' section shows a table with one row:

Action:	Go	0 of 1 selected
<input type="checkbox"/> Post		
<input checked="" type="checkbox"/> Post object		

Below the table, it says '1 post'.

Django 通常以 `Post` object 來表示 `Post` 物件，但此種顯示不易辨別。我們可以透過 `def __str__` 更改 `Post` 的表示方式。

修改 `trips/models.py`：

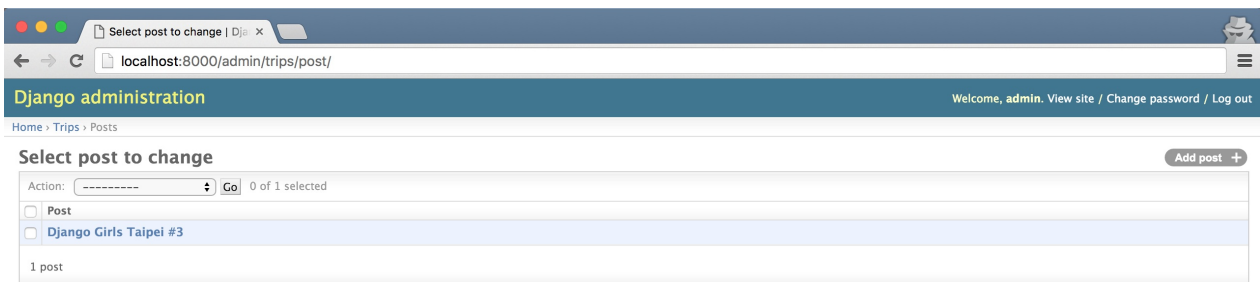
```
# trips/models.py

from django.db import models

class Post(models.Model):
    ...
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

重新整理 `Posts` 頁面後，`Post` 已經被定義成顯示標題：



小結

你現在已經學會：

- 設定 Django Admin
 - 建立 superuser
 - 註冊 Model 至 Admin
-

本章新學到的指令

指令	說明
<code>python manage.py createsuperuser</code>	新增 Django 管理者帳號

使用 Django ORM 操作資料庫

在前一章，我們利用 **Django Admin** 新增、修改及刪除 Post。而實際在寫程式時，我們會使用 Django 提供的 QuerySet API，來達成類似的資料庫操作。

本章你會學到：如何使用 Django QuerySet API 與資料庫互動 (CRUD)。

CRUD 指的是，**Create** (新增)、**Read** (讀取)、**Update** (修改)、**Delete** (刪除) 等常見的資料庫操作。

使用 Django Shell

與先前不同的是，在這裡我們不使用 Python Shell，而是 **Django Shell**。

使用 `shell` 指令，進入 Django Shell：

```
(djangogirls_venv) ~/djangogirls/mysite$ python manage.py shell
```

這個 shell 和我們之前輸入 `python` 執行的 shell 長得一樣，只是它會預先為我們設定 Django 需要的環境，方便我們執行 Django 相關的程式。

IPython

IPython 是強化版的 Python 互動式命令列介面，它比預設的命令列介面多了許多進階功能，例如：

- 按 `tab` 鍵可以補齊未輸入完的指令、檔案及資料夾名稱。
- 按 `↑` 鍵和 `↓` 鍵可以瀏覽輸入過的程式碼，便於微調先前的程式碼（修改參數等等）。
- 在套件、模組或函數名稱後加上 `?` 可查看與之相關的資訊。
- `history` 指令可查看所有輸入過的指令。
- 可以使用 `shell` 指令，如：`ls`、`cd`。

我們一樣可以用 `pip` 來安裝這個強大的套件：

```
(djangogirls_venv) ~/djangogirls/mysite$ pip install ipython
```

QuerySet API

Create

首先，讓我們來試著新增幾筆資料：

```
>>> from trips.models import Post

>>> Post.objects.create(title='My First Trip', content='肚子好餓，吃！')
<Post: My First Trip>

>>> Post.objects.create(title='My Second Trip', content='去散散步吧')
<Post: My Second Trip>

>>> Post.objects.create(title='Django 大冒險', content='從靜態到動態',
                        content_type='text/html')
<Post: Django 大冒險>
```

Read

若想顯示所有的 `Post`，可以使用 `all()`：

```
>>> from trips.models import Post
>>> Post.objects.all()
[<Post: My First Trip>, <Post: My Second Trip>, <Post: Django 大冒險>]
```

只想顯示部分資料時，則可以使用 `get` 或 `filter`：

```
>>> Post.objects.get(pk=1)
<Post: My First Trip>

>>> Post.objects.filter(pk__gt=1)
[<Post: My Second Trip>, <Post: Django 大冒險>]
```

- **get** : 返回符合條件的唯一一筆資料。（注意：如果找不到符合條件的資料、或是有多筆資料符合條件，都會產生 exception）
- **filter** : 返回符合條件的陣列。如果找不到任何資料則會返回空陣列。

Update

當想修改資料時，可以使用 **update** 更新一筆或多筆資料：

首先，先取得欲更新的 Post。這裡使用 `pk < 3` 的條件篩選

```
>>> posts = Post.objects.filter(pk__lt=3)
```

共有 2 個 Post 符合上面的條件

```
>>> posts
[<Post: My First Trip>, <Post: My Second Trip>]
```

我們將 location 的值印出

```
>>> posts[0].location
'台北火車站'

>>> posts[1].location
'台北火車站'

>>>
```

印出後發現，兩個 Post 的 location 都是台北火車站。現在我們試試用 **update** 指令，把它改成 '捷運大安站'

```
>>> posts.update(location='捷運大安站')
2
```

回傳的數字 `2` 指的是已被更新的資料筆數。我們可以驗證一下 `location` 是否皆已被正確更新

```
>>> posts[0].location
'捷運大安站'

>>> posts[1].location
'捷運大安站'
```

Delete

我們也可以使用 `delete` 刪除資料：

我們試著使用 `delete`，將剛剛的那兩筆 Post 刪除。

```
>>> posts.delete()
```

確認一下，資料是否刪除

```
>>> Post.objects.all()
[<Post: Django 大冒險>]
```

Template tags

在先前的 Templates 章節中，我們已經學會基礎的 Django Template 用法 (在 Template 裡呈現變數內容)。但為了產生完整的網頁，我們會需要能在 Template 裡執行一些簡單的 Python 語法，例如：

- 邏輯判斷 (if-else) -- 若使用者已經登入，則顯示使用者的暱稱；若未登入，則顯示登入按鈕
- 重覆 **HTML** 片段 (for loop) -- 列出所有好友的帳號和顯示圖片
- 格式化 **Template** 中的變數 -- 日期的格式化等等

[Django template tags](#) 讓你可以在 HTML 檔案裡使用類似 Python 的語法，動態存取從 view function 傳過來的變數，或是在顯示到瀏覽器之前幫你做簡單的資料判斷、轉換、計算等等。

在這一章，我們將使用 Django ORM 存取資料庫，撈出旅遊日記全部的 posts 傳入 template，並使用 Django 的 template tags 與 filters，一步步產生旅遊日記的首頁。

建立旅遊日記的首頁

確認首頁需求

在開始動工之前，我們先確認需求。

旅遊日記的首頁應該會有：

1. 標題
2. 照片
3. 發佈日期
4. 部份的遊記內文

建立首頁的 **View**

首先我們建立一個新的 view function - `home()` :

```
# trips/views.py

...

from .models import Post

def home(request):
    post_list = Post.objects.all()
    return render(request, 'home.html', {
        'post_list': post_list,
    })
```

- 匯入所需的 **model** -- 記得 import 需要用到的 Model `Post`
- 取得所有 **posts** -- 透過 `Post.objects.all()` 從資料庫取得全部的 posts, 並傳入 `home.html` 這個 template。

設定首頁的 URL

接下來, 我們修改 `urls.py`, 將首頁 (正規表達式 `^$`) 指向 `home()` 這個 view function :

```
# mysite/urls.py
from trips.views import hello_world, home

urlpatterns = [
    ...
    url(r'^$', home),
]
```

Template Tags

建立首頁的 **Template** 並印出 `post_list`

首先，在 `templates` 資料夾底下新增 `home.html`：

```
<!-- home.html -->

{{ post_list }}
```

打開瀏覽器進入首頁 <http://127.0.0.1:8000/>，可以看到 `post_list` 已呈現至網頁上了。



顯示 **Post** 中的資料

仔細觀察印出的 `post_list`，會發現是以 `list` 的形式顯示。但我們希望的則是：存取每個 **Post** 中的資料，並印出來。

為了達成這個功能，我們會用到 `for` 這個 `template tag`。

`for` 迴圈

在寫 Python 時，若想存取 `list` 裡的每一個元素，我們會使用 `for` 迴圈。而在 Django Template 中，也提供了類似的 `template tags` -- `{% for %}`。

`{% for %}`

在 `template` 中使用類似 Python 的 `for` 迴圈，使用方法如下：

```
{% for <element> in <list> %}  
    ...  
{% endfor %}
```

瞭解了 **for** 的用法後，我們試著印出首頁所需的資訊。修改 `home.html` 如下：

```
<!-- home.html -->  
  
{% for post in post_list %}  
    <div>  
        {{ post.title }}  
        {{ post.created_at }}  
        {{ post.photo }}  
        {{ post.content }}  
    </div>  
{% endfor %}
```

- 開始標籤為 `{% for %}` 開始；結束標籤為 `{% endfor %}`
- `post_list` 中有 3 個元素，所以 for 區塊中的內容會執行 3 次
- 迴圈中，使用標籤 `{{ var }}`，反覆印出每個 `post` 中的標題、建立時間、照片網址和文章內容

重新整理瀏覽器，網頁上會有首頁所需的 `post` 資訊：



顯示照片

現在網頁已經有照片網址，我們稍微修改 `template`，讓照片以圖片方式呈現。

把 `home.html` 的下面這一行：

```
{{ post.photo }}
```

換成下面這樣：

```
<div class="thumbnail">
    
</div>
```

處理沒有照片的遊記

`if ... else`

另一個常用的 `template tags` 是 `{% if %}` 判斷式，用法如下：

```
{% if post.photo %}
<div class="thumbnail">
    
</div>
{% else %}
<div class="thumbnail thumbnail-default"></div>
{% endif %}
```

- 符合條件所想要顯示的 HTML 放在 `{% if <condition> %}` 區塊裡
- 不符合的則放在 `{% else %}` 區塊裡面
- 最後跟 **for** 一樣，要加上 `{% endif %}` 作為判斷式結尾。

在這裡，我們判斷如果 `post.photo` 有值就顯示照片，否則就多加上一個 CSS class `photo-default` 另外處理。

Template Filter

除了 template tags，Django 也內建也許多好用的 [template filters](#)。它能在變數顯示之前幫你做計算、設定預設值，置中、或是截斷過長的内容等等。使用方法如下：

```
{{<variable_name>|<filter_name>:<filter_arguments>}}
```

- `<variable_name>` -- 變數名稱
- `<filter_name>` -- filter 名稱，例如 `add`、`cut` 等等
- `<filter_arguments>` -- 要傳入 filter 的參數

變更時間的顯示格式

在這裡，我們只練習一種很常用的 filter [date](#)。它可以將 `datetime` 型別的物件，以指定的時間格式輸出。

我們試著將 `created_at` 時間顯示成年 / 月 / 日：

```
{{ post.created_at|date:"Y / m / d" }}
```

完整的 HTML 與 CSS

接著，補上完整的 HTML 標籤，並加上 CSS 樣式後，旅遊日記首頁就完成了。

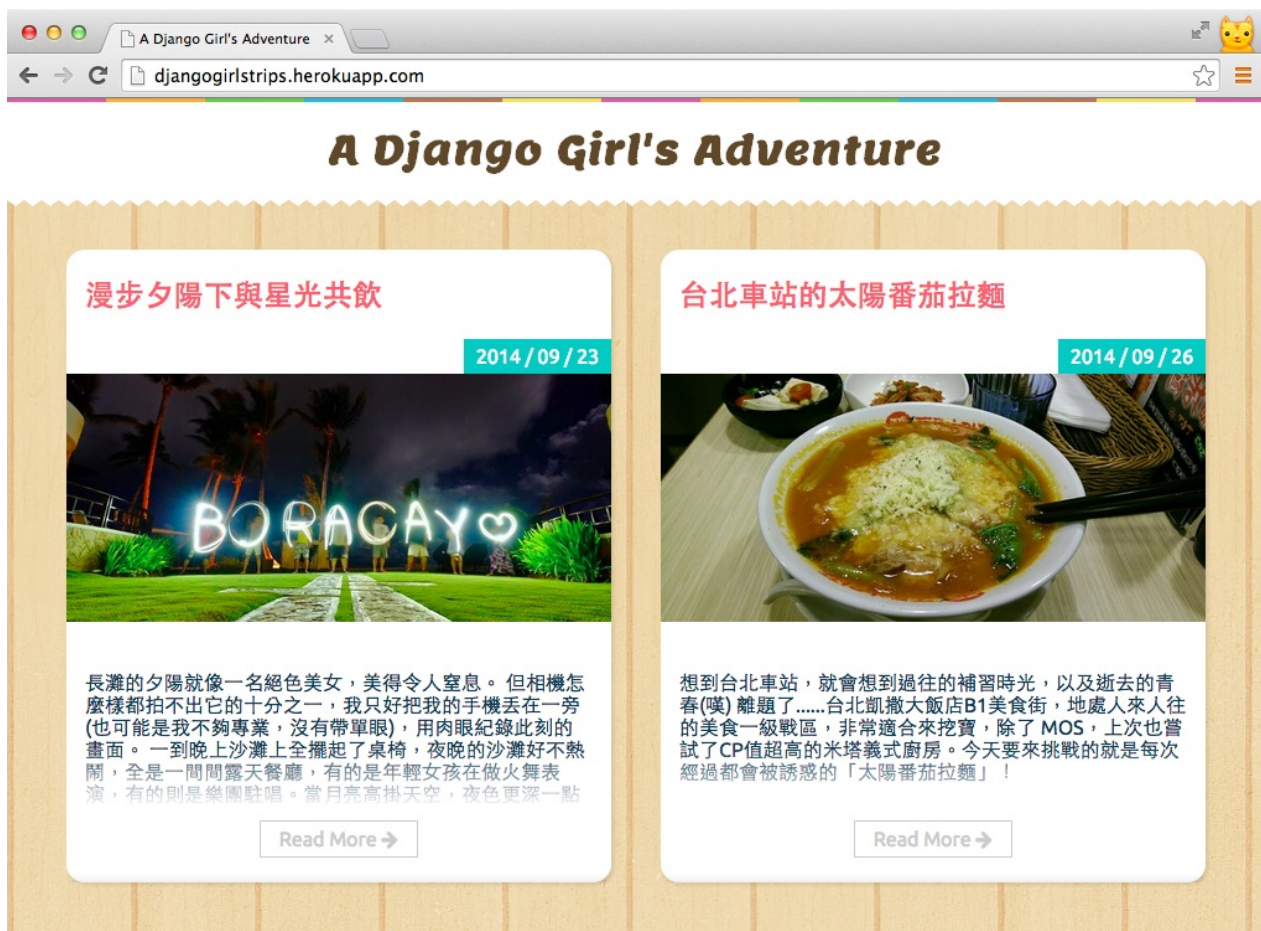
最終版 `home.html` 程式碼如下：

```
<!-- home.html -->

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>A Django Girl's Adventure</title>
  <link href="//fonts.googleapis.com/css?family=Lemon" rel="stylesheet">
  <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css" rel="stylesheet">
  <link href="//djangogirlstapei.github.io/assets/css/style.css" rel="stylesheet">
</head>
<body>
  <div class="header">
    <h1 class="site-title text-center">
```

```
        <a href="/">A Django Girl's Adventure</a>
    </h1>
</div>
<div class="container">
    {% for post in post_list %}
    <div class="post-wrapper">
        <div class="post">
            <div class="post-heading">
                <h2 class="title">
                    <a href="#">{{ post.title }}</a>
                </h2>
                <div class="date">{{ post.created_at|date:"Y /
            </div>
            {% if post.photo %}
            <div class="thumbnail">
                
            </div>
            {% else %}
            <div class="thumbnail thumbnail-default"></div>
            {% endif %}
            <div class="post-content read-more-block">
                {{ post.content }}
            </div>
            <div class="post-footer">
                <a class="read-more" href="#">
                    Read More <i class="fa fa-arrow-right"></i>
                </a>
            </div>
        </div>
    </div>
    {% endfor %}
</div>
</body>
</html>
```

打開 <http://127.0.0.1:8000/> 看看你的成果吧！



小結

最後，我們複習一下本章學到的 **Template Tag** 與 **Template Filter**：

Template Tags

語法	說明
<code>{% for ... in ... %}...{% endfor %}</code>	類似 Python 的 for 迴圈，反覆執行 for 區塊中的內容
<code>{% if %} ... {% else %} ... {% endif %}</code>	在 Template Tags 中進行 if/else 的邏輯判斷

Template Filters

語法	說明
<code>{{ value date:<date_format> }}</code>	可以將 `datetime` 型別的物件，以指定的時間格式 Date Format 輸出

Dynamic URL

除了在首頁顯示文章的摘要外，通常也會希望每篇文章能有獨立的網址與頁面。例如，我們可能會希望 `http://127.0.0.1/post/3/` 能夠是 pk 為 3 那篇文章的網址，而頁面內容則是此篇日記的詳細資訊，而非摘要。

在這個章節，我們會學到如何設定動態網址的 URL conf，讓每篇旅遊日記，擁有獨一無二的網址與頁面。

建立單篇文章的 View

首先建立單篇文章所使用的 view function。在 `trips/views.py` 中，新增 `post_detail` 這個 view 如下：

```
# trips/views.py

def post_detail(request, pk):
    post = Post.objects.get(pk=pk)
    return render(request, 'post.html', {'post': post})
```

以訪客瀏覽 `http://127.0.0.1:8000/post/3/` 的例子，來解釋以上程式：

- 目前瀏覽文章的 **pk** 會傳入 **view** 中：當訪客瀏覽 `http://127.0.0.1/post/3/` 時，傳入 view 的 pk 會是 3。
 - URL 與 pk 的對應，會在稍後設定。這裡只需知道 view 中傳入的，會是當前瀏覽文章 pk 即可。
- 取得傳入 **pk** 的那篇 **Post** 資料：當傳入的 `pk=3`，代表訪客想看到 `pk=3` 那篇文章。我們可以利用之前學過的 ORM 語法 `get`，取得該篇日記的 **Post** 物件：

```
post = Post.objects.get(pk=pk) # 此時 pk = 3
```

- 回傳 **HttpResponse**：將取得的 post (pk=3) 傳入 Template *post.html*，並呈現 render 後的結果。

設定動態網址的對應

日記單頁的 view function 完成後，我們來設定網址與 view 的對應。修改 *mysite/urls.py*，加入以下內容：

```
# mysite/urls.py
from trips.views import hello_world, home, post_detail

urlpatterns = [
    ...
    url(r'^post/(?P<pk>\d+)/$', post_detail, name='post_detail'),
]
```

上面的修改完成後，只要連至 `http://127.0.0.1/post/3/` 就會對應到 `post_detail()` 這個 view，並且傳入的 **pk=3**。

使用 **Regex** 提取部份 URL 為參數

我們前面提過，Django 的 URL 是一個 *regular expression (regex)*。Regular expression 語法可用來描述一個字串的樣式。除了可以表示固定字串之外，還可以用來表示不確定的內容。我們一步一步解釋文章單頁所使用的 URL 設定：

```
(?P<pk>\d+)
```

1. `\d` 代表一個阿拉伯數字。
2. `+` 代表「一個以上」。

所以 `\d+` 代表一個以上的阿拉伯數字，例如「0」、「99」、「12345」。可是像「8a」就不符合，因為「a」不是數字。

3. `(?P<pk>)` 代表「把這一串東西抓出來，命名為 pk」。

所以 `(?P<pk>\d+)` 代表：抓出一個以上阿拉伯數字，並把抓出來的東西取名為 `pk`。

綜合以上的規則，`r'^post/(?P<pk>\d+)/$'` 會達成以下的效果：

URL	符合結果
http://127.0.0.1/posts/	不符合，因為前面不是 post/ 開頭。
http://127.0.0.1/post/	不符合，因為後面抓不到數字。
http://127.0.0.1/post/1/	符合，抓到的 <code>pk</code> 是 1。
http://127.0.0.1/post/1234/	符合，抓到的 <code>pk</code> 是 1234。
http://127.0.0.1/post/12ab/	不符合，因為後面有不是數字的東西。

建立單篇日記頁的 Template

回顧一下前面寫的 view function (`post_detail`) 內容

```
return render(request, 'post.html', {'post': post})
```

我們取得所需 `post` 物件後，傳入 `post.html` 這個 template 中 `render`。現在我們就來完成這個 template。建立 `post.html` 如下：

```

<!-- templates/post.html -->

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>{{ post.title }} | A Django Girl's Adventure</title>
    <link href="//fonts.googleapis.com/css?family=Lemon" rel="stylesheet">
    <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css" rel="stylesheet">
    <link href="//djangogirlstaipei.github.io/assets/css/style.css" rel="stylesheet">
</head>
<body>
    <div class="header">
        <h1 class="site-title text-center">
            <a href="/">A Django Girl's Adventure</a>
        </h1>
    </div>
    <div class="container post post-detail">
        <div class="post-heading">
            <h1 class="title"><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a>
            </h1>
            <div class="date">{{ post.created_at|date:'Y / m / d' }}</div>
        </div>
        <div class="location">
            <i class="fa fa-map-marker"></i>
            <span id="location-content">{{ post.location }}</span>
        </div>
        <div id="map-canvas" class="map"></div>
        <div class="post-content">
            {{ post.content }}
        </div>
        <hr class="fancy-line">
        
    </div>
    <script src="//maps.googleapis.com/maps/api/js?v=3.exp&libraries=places" type="text/javascript"></script>
    <script src="//djangogirlstaipei.github.io/assets/js/map.js"></script>
</body>
</html>

```

這個 template 將 post 物件的屬性 (e.g. 標題、內文、時間.....等), 利用 `{{ var }}` 與 template filter 顯示並格式化於 HTML 中。若資料庫裡有 pk=3 的 Post, 現在連至 <http://127.0.0.1:8000/post/3/> 即可看到此日記的單頁。

{% url %}

連結到特定 view 的 template tag

使用方法：

語法	說明
<code>{% url '<view_name>' %}</code>	根據在 urls.py 中設定的「name」值，找到對應的 URL

也可以傳入參數，如：

```
{% url '<view_name>' arg1=<var1> arg2=<var2> ...%}
```

其餘用法可參考[官方文件](#)。

加入到單篇日記頁的連結

最後，我們還需在首頁加上單篇日記的連結。我們可以使用 `{% url %}` 這個 template tag 達成。需要加入的地方有：

1. 每篇日記
2. 每篇日記的 Read More 按鈕

設定標題連結

打開 *home.html*, 找到下面的內容：

```
<!-- home.html -->

<h2 class="title">
    <a href="#">{{ post.title }}</a>
</h2>
```

將它改成

```
<!-- home.html -->

<h2 class="title">
    <a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a>
</h2>
```

設定 **Read More** 按鈕的連結

在 *home.html* 中找到以下內容：

```
<!-- home.html -->

<a class="read-more" href="#">
    Read More <i class="fa fa-arrow-right"></i>
</a>
```

修改如下：

```
<!-- home.html -->

<a class="read-more" href="{% url 'post_detail' pk=post.pk %}">
    Read More <i class="fa fa-arrow-right"></i>
</a>
```

驗收成果

連至 <http://127.0.0.1:8000/>。現在只要點擊各個日記的標題或 *Read More* 按鈕，就會顯示那該篇日記的詳細頁面。

A Django Girl's Adventure

台北車站的太陽番茄拉麵

2014 / 09 / 26

📍 台北車站



想到台北車站，就會想到過往的補習時光，以及逝去的青春(嘆) 離題了.....台北凱撒大飯店B1美食街，地處人來人往的美食一級戰區，非常適合來挖寶，除了 MOS，上次也嘗試了CP值超高的米塔義式廚房。今天要來挑戰的就是每次經過都會被誘惑的「太陽番茄拉麵」！



Deploy

目前為止，我們所有的工作都是在自己的電腦完成，你可以在自己的瀏覽器上看到成果。但是，如果我們想要讓其他使用者使用這個網站，就必須將它部署（deploy）到穩定的伺服器上，才能隨時瀏覽。

我們選擇 [Heroku](#) 作為這次的範例，它的免費額度足夠經營一個小型網站，並擁有完善的開發者教學資源。

本章根據 [Heroku 的官方教學 “Getting Started with Django on Heroku”](#) 稍作調整，教你如何準備部署，並在 Heroku 上發佈你的網站。

安裝部署工具

首先利用 `pip` 安裝一些部署時需要用到的套件：

```
(djangogirls_venv) ~/djangogirls$ pip install dj-database-url gunicorn
```

當終端機顯示 *Successfully installed...* 時，表示必要的套件都已經安裝好了。

部署準備

為了讓 server 了解部署時所需要的安裝環境，我們需要調整和準備一些設定檔案。

requirements.txt

在 `djangogirls` 專案目錄底下，利用以下的指令將此虛擬環境裡的 Python 套件全部條列出來，包括套件名稱與版本資訊，並儲存於 [requirements.txt](#)：

```
(djangogirls_venv) ~/djangogirls$ pip freeze > requirements.txt
```

由於 Heroku 使用 [PostgreSQL](#) 資料庫，我們還需要手動在 `requirements.txt` 最後面加上 `psycopg2==2.6.1`（Python 的 PostgreSQL 模組）。最終的檔案內容範例如下，版本可能會稍有不同：

```
#.djangogirls/requirements.txt

Django==1.8.6
appnope==0.1.0
decorator==4.0.4
dj-database-url==0.3.0
dj-static==0.0.6
gnureadline==6.3.3
gunicorn==19.3.0
ipython==4.0.0
ipython-genutils==0.1.0
path.py==8.1.2
pexpect==4.0.1
pickleshare==0.5
ptyprocess==0.5
simplegeneric==0.8.1
static3==0.6.1
traitlets==4.0.0
psycopg2==2.6.1
```

Procfile

建立一個 [Procfile](#) 檔案，告訴 Heroku 要如何啟動我們的應用：

```
web: gunicorn --pythonpath mysite mysite.wsgi
```

這一行指令分成兩個部分，其格式 `<process_type>: <command>` 表示：

- -- 啟用 `web` 應用
- -- [Gunicorn](#) 是一個用 Python 開發的 WSGI 工具，可以用來執行 Django 的網站。我們透過指令下列指令來啟動網站：

```
gunicorn --pythonpath <directory_path> <project_name>.wsgi
```

runtime.txt

為了讓 Heroku 知道要用哪一個版本的 Python，新增 `runtime.txt` 輸入：

```
python-3.4.3
```

production_settings.py

在前面的章節中，我們透過修改 `settings.py` 來調整 Django project 的設定，但是通常正式上線（production）的環境會和開發/本機（development/local）環境有所不同。所以我們在 `mysite/mysite/` 底下新建一個 `production_settings.py`，專門放部署時所需要的設定：

```
# mysite/mysite/production_settings.py

# Import all default settings.
from .settings import *

import dj_database_url
DATABASES = {
    'default': dj_database_url.config()
}

# Static asset configuration.
STATIC_ROOT = 'staticfiles'

# Honor the 'X-Forwarded-Proto' header for request.is_secure().
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

# Allow all host headers.
ALLOWED_HOSTS = ['*']

# Turn off DEBUG mode.
DEBUG = False
```


wsgi.py

[WSGI - Web Server Gateway Interface](#) 是 Python 定義網頁程式和伺服器溝通的介面。為了讓 Heroku 的服務能夠透過這個介面與我們的網站溝通，修改

`mysite/mysite/wsgi.py` 如下：

```
# mysite/mysite/wsgi.py

import os

from django.core.wsgi import get_wsgi_application

from dj_static import Cling

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.settings")

application = Cling(get_wsgi_application())
```

我們將 `dj_static` 引入，並在 `application` 上使用它，以協助幫我們部署 static 檔案（例如圖片、CSS、JavaScript 檔案等等）。

.gitignore

我們不希望把有些開發時使用的檔案，例如虛擬環境、本機資料庫等等，都一股腦放到網路上。因此，接下來需要建立一個 `.gitignore` 檔案，排除這些資料：

```
#.djangogirls/.gitignore

djangogirls_venv
*.pyc
__pycache__
staticfiles
db.sqlite3
```

小結

最後的檔案結構如下：

```
djangoirls
├──mysite
│   ├──mysite
│   │   ├──__init__.py
│   │   ├──production_settings.py
│   │   ├──settings.py
│   │   ├──urls.py
│   │   └──wsgi.py
│   ├──templates
│   ├──trips
│   └──manage.py
├──djangoirls_venv
├──.gitignore
├──Procfile
├──requirements.txt
└──runtime.txt
```

Deploy to Heroku

在開始部署（deploy）之前，請先確定你已經按照[教學手冊](#)：

1. 註冊 Heroku 帳號：<https://id.heroku.com/signup>
2. 安裝 Heroku 工具箱：<https://toolbelt.heroku.com/>

Step 1: 登入 Heroku

安裝完工具箱裡的 Heroku client 後，就可以使用 `heroku` 指令，首先讓我們登入：

```
$ heroku login
```

輸入註冊時的 Email 帳號和密碼，當你看到 *Authentication successful.* 時，表示認證成功。

Step 2: 新增一個新的 git repository

在 `djangogirls` 資料夾底下新增一個 git repository :

```
~/djangogirls$ git init
~/djangogirls$ git add .
~/djangogirls$ git commit -m "my djangogirls app"
```

Step 3-1: 新增新的 Heroku app

接下來，我們需要新增一個可以上傳 repository 的地方，如果你之前已經新增過 app，請跳到 **Step 3-2** :

```
~/djangogirls$ heroku create
```

預設 `create` 後面不放名字時，會自動產生隨機名稱的 Heroku app，如果想要命名自己的 app，如下：

```
~/djangogirls$ heroku create djangogirlsdiary
```

注意：

- Heroku app 是不能重名的，所以如果你也輸入 `djangogirlsdiary`，會得到 `! Name is already taken` 的警告。
- Heroku app 名稱會顯示在 deploy 成功後的網址上，例如：<https://djangogirlsdiary.herokuapp.com>

Step 3-2: 指定已經存在的 app

如果你之前已經新增過 app，並且想發佈在已經存在的 app 上時，可以先用指令 `heroku apps` 查看 app 的名稱：

```
$ heroku apps
=== My Apps
djangogirlsdiary
```

然後設定成你想要上傳的 app：

```
$ heroku git:remote -a djangogirlsdiary
Git remote heroku added.
```

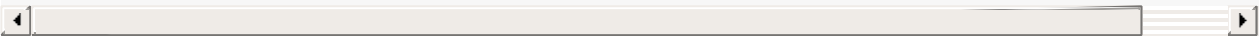
最後透過 `git remote -v` 檢查一下是否設定到正確的位置：

```
$ git remote -v
heroku      https://git.heroku.com/djangogirlsdiary.git (fetch)
heroku      https://git.heroku.com/djangogirlsdiary.git (push)
```

Step 4: 設定環境變數

我們利用 `heroku config:set` 指令設置 [環境變數](#)，以確保未來在 Heroku 執行任何指令時，都是使用到部署專用的設定檔：

```
$ heroku config:set DJANGO_SETTINGS_MODULE=mysite.production_settin
```



Step 5: 利用 `git push` 上傳到 Heroku

使用 `git push` 指令上傳 git repository 後，你會發現它按照 **runtime.txt** 安裝 python-3.4.3，也透過 pip 安裝我們在 **requirements.txt** 上列出的所有套件：

```
~/djangogirls$ git push heroku master
...
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: -----> Installing runtime (python-3.4.3)
remote: -----> Installing dependencies with pip
...
remote: -----> Compressing... done, 50.8MB
remote: -----> Launching... done, v1
remote:          https://djangogirlsdiary.herokuapp.com/ deployed to
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/djangogirlsdiary.git
 * [new branch]      master -> master
```

如果你遇到下列的錯誤訊息：

```
Permission denied (publickey).
fatal: The remote end hung up unexpectedly
```

請透過下列指令新增 public key，然後再重新 `git push`。

```
~/djangogirls$ heroku keys:add
```

Step 6: 啟動 web process

先前建立了 **Procfile** 檔案告訴 Heroku 啟動時要執行的指令，現在我們使用指令啟動 web process，並指定只需要 1 個 instance：

```
~/djangogirls$ heroku ps:scale web=1
```


Step 7: Django project 初始化

Django 已經成功啟動了，但是我們還需要進行資料庫初始化，利用 `heroku run` 可以在 Heroku 執行指令：

```
~/djangogirls$ heroku run python mysite/manage.py migrate
```

並為新資料庫建立一個 superuser：

```
~/djangogirls$ heroku run python mysite/manage.py createsuperuser
```



Step 8: 開啟瀏覽器觀看你的網站

最後，透過 `open` 指令會自動在瀏覽器打開你的網站：

```
~/djangogirls$ heroku open
```

恭喜你成功地把網站發佈到網路上了！

因為資料庫是不同的，之前在本機端的日記都需要再重新輸入喔。

你可以分享網址給任何人：<https://djangogirlsdiary.herokuapp.com/>。記得前面要替換成你自己的 Heroku app 名稱！

未來如果對網站進行任何修改並想更新到 Heroku，只要先確定 **git commit** 完成後再 **push** 到 Heroku 即可。

```
$ git push heroku master
```

What's next?

恭喜! 你已經懂得如何使用 Django 寫出自己的網站，並發佈到網路上了。

接下來，我們希望你能試著：

- 修改 HTML 與 CSS，調整成你喜歡的樣子
- 為旅遊日記添加新的欄位（例如旅遊日期），並使用 `makemigrations` 和 `migrate` 更新資料庫。
- 為旅遊日記加入作者（提示：你可能會需要修改 Model，並與 [Django 使用者認證](#) 功能整合）
- 將 HTML 重複的部分獨立出來共用（提示：使用 [Template 繼承](#)）
- 為每一篇日記加上留言板（提示：[Django Forms](#) 可以幫助你更快速地完成）

其他學習資源

- [Codecademy](#) -- 透過闖關遊戲方式學習 Python, HTML/CSS, JavaScript
- [Writing your first Django app](#) -- Django 1.8 官方學習指南
- [Getting Started With Django](#) -- 影片課程
- [The Django Book](#) -- 雖然 Django 版本不是最新，但相當適合初學者的一本書
- [Two Scoops of Django: Best Practices for Django](#) -- 非常推薦，曾經在 [Taipei.py](#) 隔週二聚會指定書籍
- [Django Packages](#) -- Django 相關套件彙整平台，提供搜尋和評比

關注我們的最新消息請至：

- Django Girls Taipei 網站：<http://djangogirls.org/taipei>
- Django Girls Taiwan 臉書社團：<https://www.facebook.com/groups/djangogirls.taiwan/>
- 如果有任何問題，歡迎來信與我們聯繫：taipei@djangogirls.org