# Visual Studio 2017 Python 程式設計實務

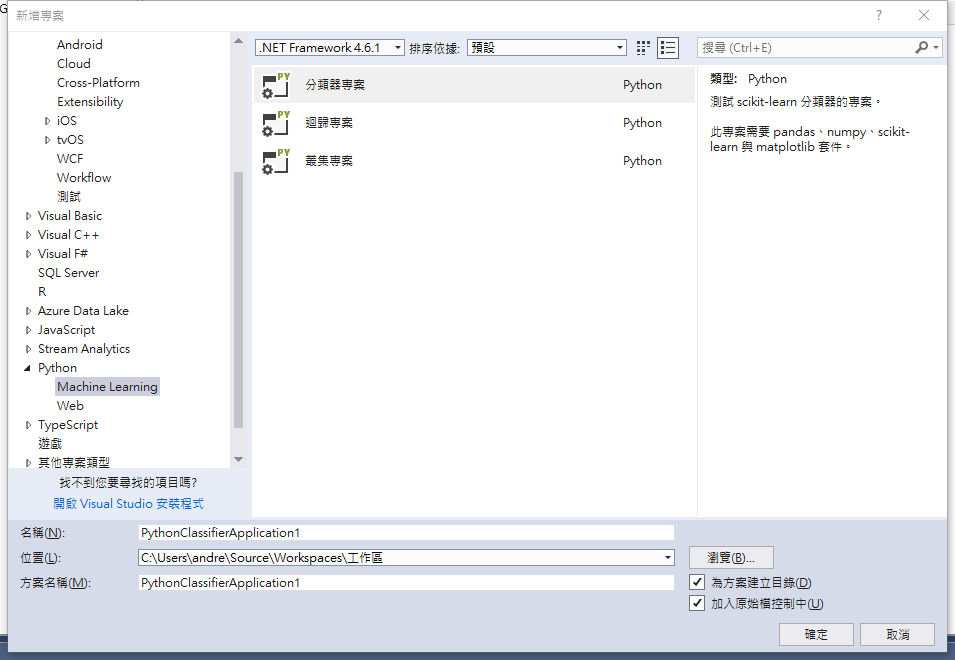
作者：王昌元

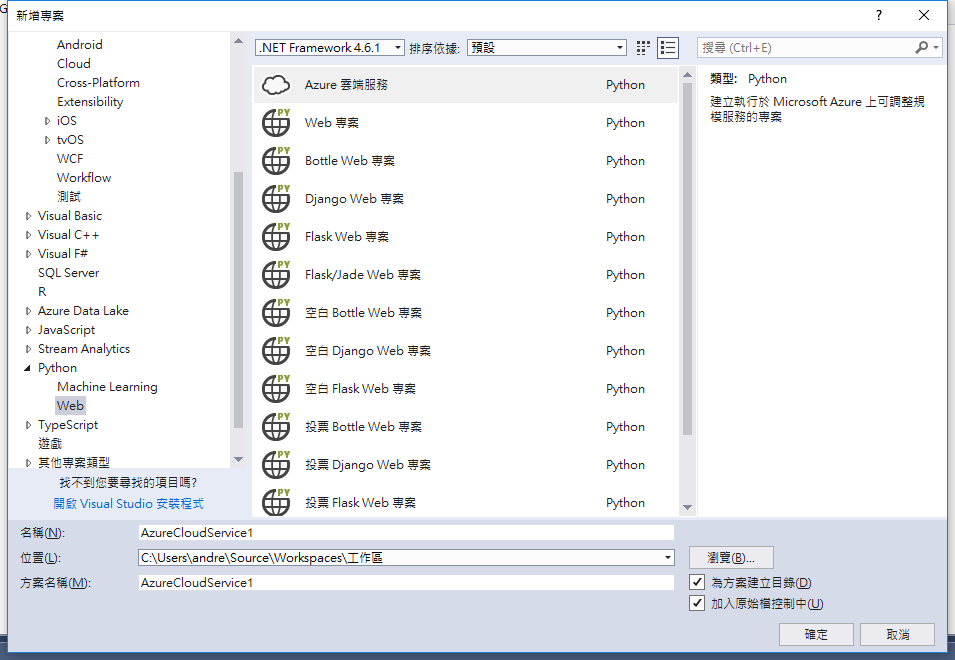
## 熟悉Visual Studio Python 使用介面

### Python 開發環境

IDLE 是 Python 標準的官方開發環境，不過，我兩年前使用IDLE之後的感覺，就是有些簡陋，後來乾脆使用 command line 的方式開發。除了IDLE之外，也有其他的開發環境可以使用，例如： wingIDE、PyCharm、PythonWin、Eclipse、IPython、Komodo等。我在坊間找了很久，完全看不到 Visual Studio Python 的相關書籍，我的習慣是，盡量使用現行的開發環境，所以，Visual Studio 是我唯一的選擇。

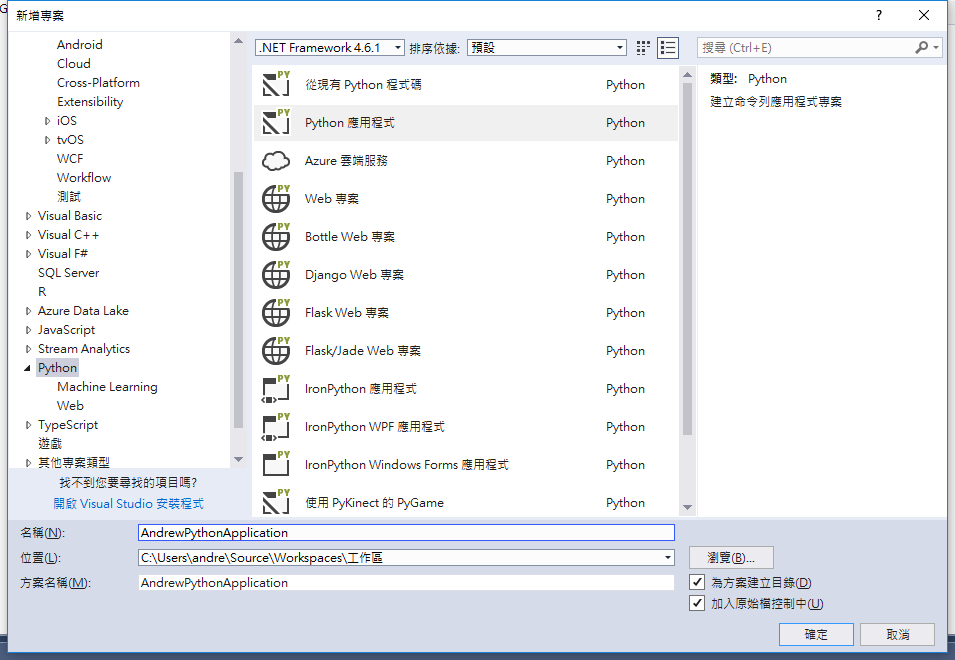
### 新增Python 方案



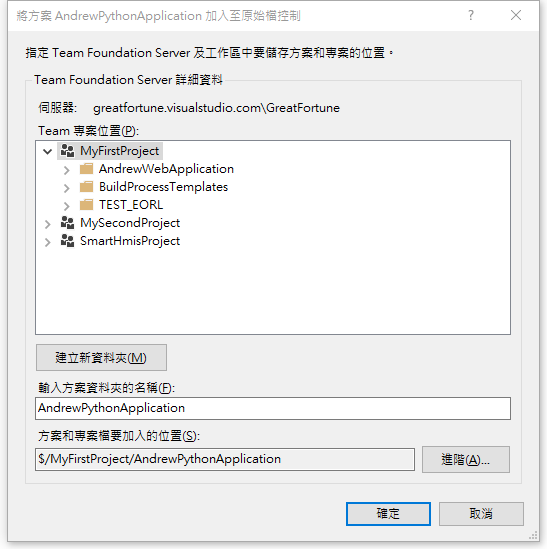


對我而言，Python 就是兩種用途，一種是編寫 Machine Learning 的演算法則、另一種則是用來設計Web程式(事實上現在單機的程式已經完全不用考慮了，即使是寫給自己用，也建議以網頁的模式開發)。在我們新增 Python 方案的時候，剛好分成這兩類。

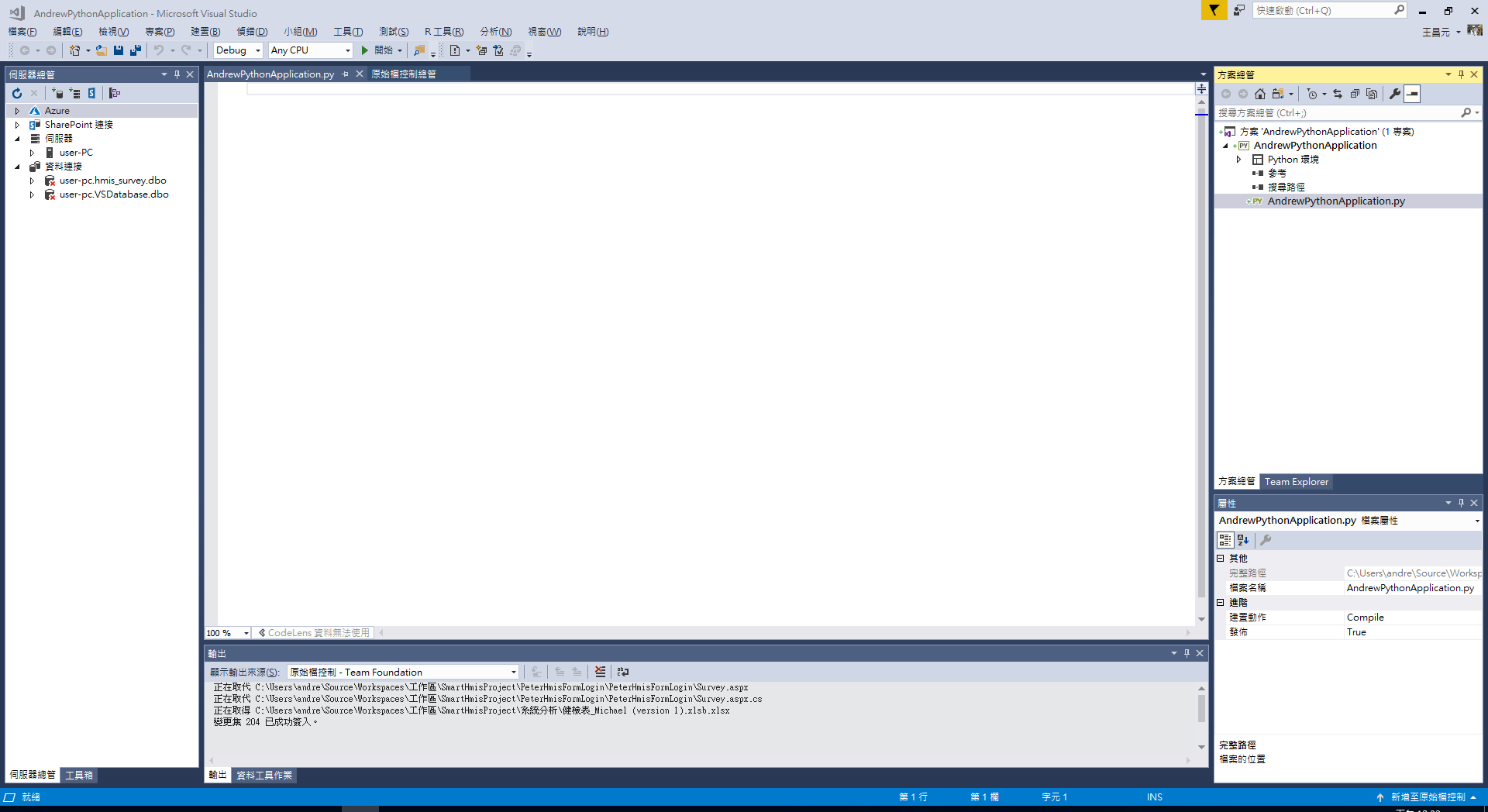
不過，如果要按部就班的學習，我還是建議從最基本的開始。我們先新增一個Python Application，如下圖：



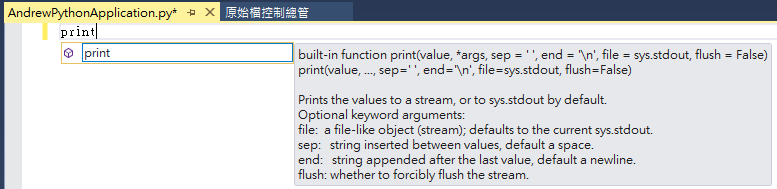
接下來系統會問你要新增的位置，直接按確定，如下圖：



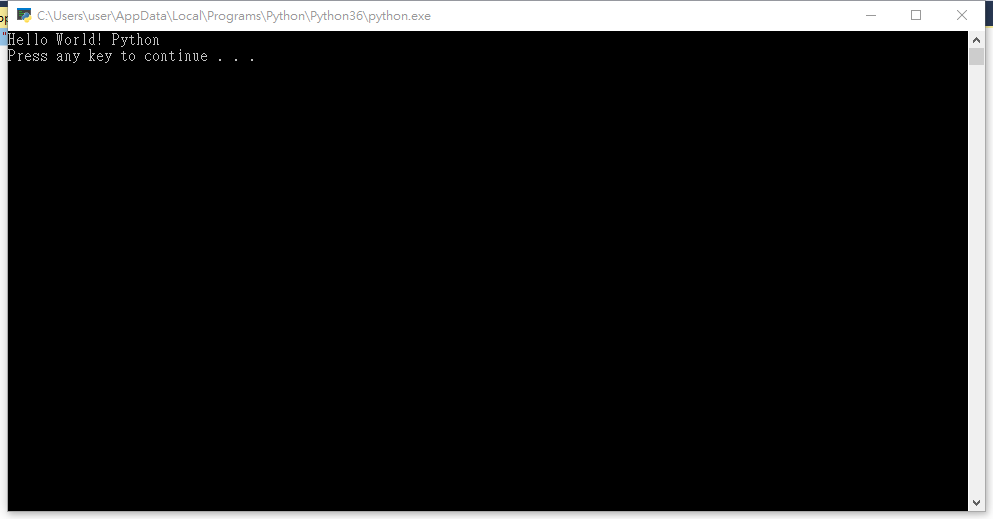
我們的 Python Application 已經建置完成了，系統自動產生了一個與方案名稱相同的 .py 程式，如下圖：



### Hello World! Python



當然，第一個 Python 程式當然是 Hello World!，由於 Visual Studio 有智慧選字功能，當我們輸入 Print 的時候，螢幕上會顯示該函數的語法，有興趣可以看一下。輸入 print("Hello World! Python") 之後，我們就可以看看執行結果。

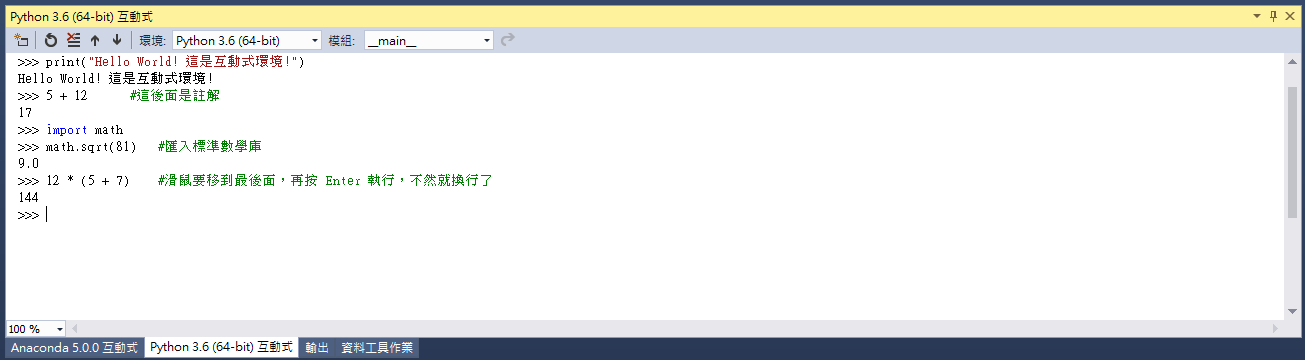


如我所預期的，這個程式會在 “DOS” 畫面執行。執行完畢之後，請將這個 .py 程式更名為 HelloWorld.py。

回想當初第一次使用 Python 的時候，用的是 Python Shell ，也就是把 Python 當成直譯語言來使用，Visual Studio 也提供 Python Shell，選擇[工具]/[Python]/[Python 互動式視窗] 就會出現下面的畫面，有沒有很熟悉：



這個互動式視窗對我們學習 Python 有很大的功用，在初期我們會大量使用此直譯環境。如下圖：



太基礎的東西我們就不提，變數、運算子等都式基礎的程式語言概念。由於Python 的數學能力很強，就順帶提一下，Python 也支援複數運算：

>>> x = 3 + 4j

>>> y = 5 + 6j #j 或 J表示虛部

>>> x + y

(8+10j)

>>> x - y

(-2-2j)

>>> x \* y

(-9+38j)

>>> x / y #複數的加減乘除

(0.6393442622950819+0.03278688524590165j)

>>> abs(x)

5.0

>>> x.imag #複數的虛部

4.0

>>> x.real #複數的實部

3.0

>>> x.conjugate() #共軛複數

(3-4j)

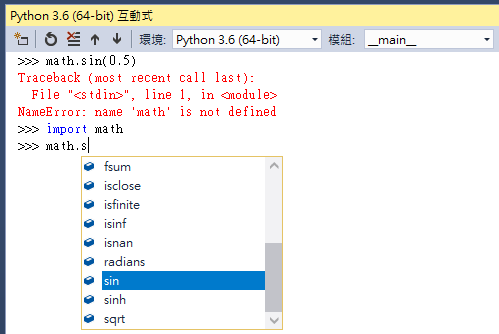
>>>

### 模組的匯入與使用

Visual Studio 中的 Python 預設安裝僅包含基本或是核心模組，啟動時也只載入基本模組，所以有時候我們會在需要時，明確的載入其他模組。

#### Import 模組名稱 [as 別名]

使用這種方式載入後，使用時需要在物件前面加上模組名稱，請注意，為載入的時候會出現錯誤訊息，載入後便會支援智慧選字，如下圖：



範例如下：

>>> math.sin(0.5)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'math' is not defined

>>> import math

>>> math.sin(0.5)

0.479425538604203

>>> import random

>>> x = random.random()

>>> x

0.8977542716703938

>>> y = random.randint(1,100)

>>> y

27

>>> import numpy as np

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ModuleNotFoundError: No module named 'numpy'

>>>

當 math 模組沒有被載入，直接執行 math.sin() 會發生錯誤，載入後，執行就沒有問題，但請注意最後一行，當我們想載入 numpy 模組的時候，卻發現找不到，那是因為 numpy 模組根本沒有安裝在整合環境中。要如何安裝呢? 我們後面在說明。

#### From 模組名稱 import 物件名稱 [as 別名]

如果每次在寫程式的時候，都要寫成 math.sin()，程式碼會變得比較冗長，Python 提供另外一種方式

>>> from math import sin

>>> sin(3)

0.1411200080598672

>>> from math import sin as f

>>> f(3)

0.1411200080598672

>>> gcd(36,18)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'gcd' is not defined

>>> from math import \*

>>> gcd(36,18)

18

>>>

第一行只把 sin 匯入，所以即使 gcd 也在 math 裡面，仍然無法執行，可以用一個方便的方法，萬用字元 \*，這樣所有 math 裡面的物件都可以使用了。這種方式省事歸省事，但是有被覆蓋的危險，我們透過下一個主題來說明。

#### 撰寫自己的模組和套件

先解釋三個名詞，套件是一群模組的集合，模組則是 .py 檔案，物件則是 .py 裡面定義的函數。前面談到的 math 模組存在於 Python 的基本套件之中，而 numpy 模組無法載入，它勢必存在於其他套件之中，如果要使用，必須安裝該套件，sin() 物件則是存在於 math.py 裡面定義的一個函數。

我們就來設計自己的模組吧，首先新增一個 a.py 檔案，在專案名稱上面按滑鼠右鍵，選擇加入/新增項目，再選擇 [空白Python檔案] 並在下面輸入檔案名稱，接著輸入內容如下：

def test():

print("在 a.py 中的 test() 物件")

同樣的也新增一個 b.py，內容如下：

def test():

print("在 b.py 中的 test() 物件")

這語法是 Python 用來定義函數的語法，因為 Python 沒有 C 或 Java 的大括號，所以是用縮排來表示下一層，而冒號就代表要縮排。上面的語法，就等於C的：

void test(){

print("在 b.py 中的 test() 物件\n");

}

由於 Python 沒有大括號，所以當函數結束的時候，只需要結束縮排，回到上一層就可以了。因此，如果縮排不正確，程式就會發生錯誤。這點要特別注意。

然後我們回歸正統，在程式裡面引用，請將 HelloWorld.py 改寫：

print("Hello World! Python")

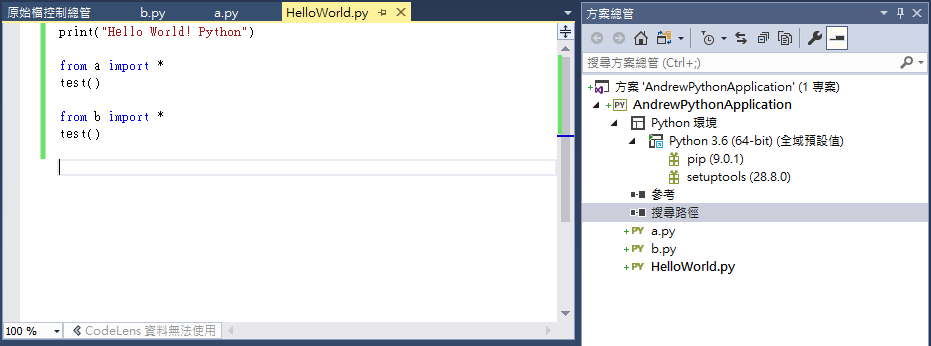
from a import \*

test()

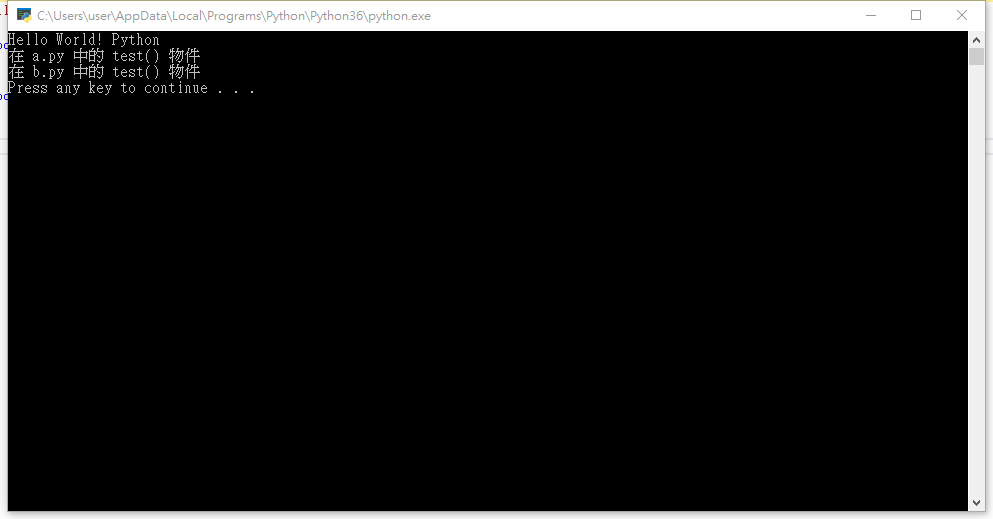
from b import \*

test()

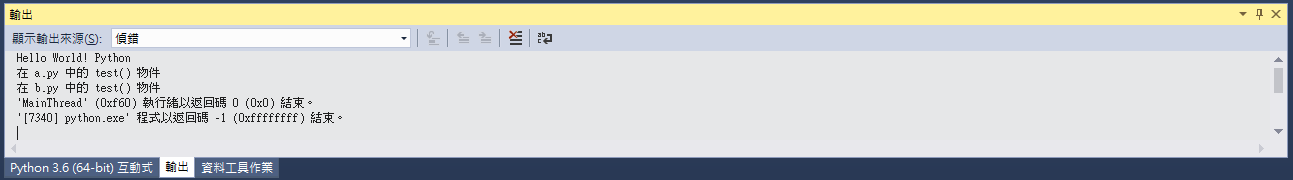
在 Visaul Studio 整合環境中如下圖：



執行結果如下：



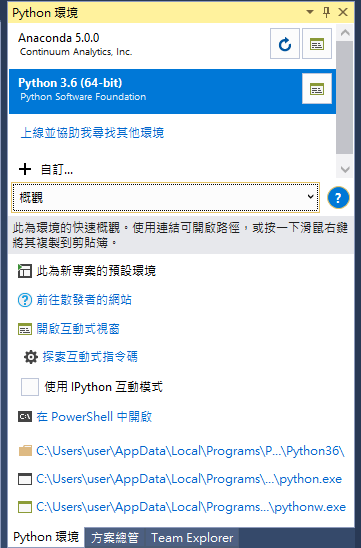
當然，也可以在輸出框中看見結果：



#### 在 Visual Studio 中安裝套件

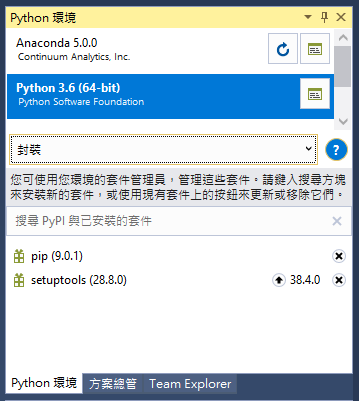
還記得剛才的 numpy 無法載入嗎? 那是因為我們沒有安裝正確的套件，Visual Studio 提供視覺化的方式讓我們可以輕鬆的安裝套件(package)。

我們先進入 Python 環境的視窗 (可以從 [視窗]/[其他視窗] 中找到)：



在[概觀]裡面我們可以快速的開啟我們要使用的工具，例如 “開啟互動式視窗”，點下去之後，Python Shell 就會出現。

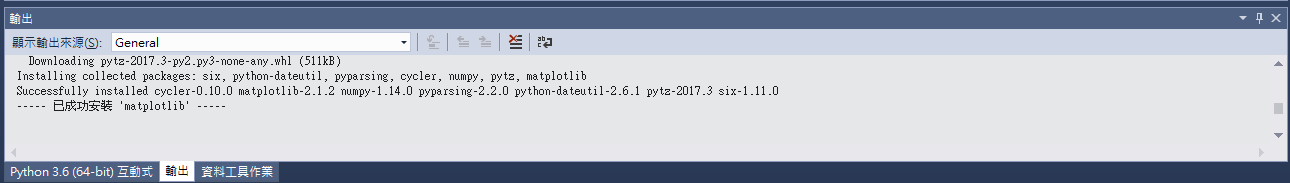
選擇[封裝]就會看到現在我們所安裝的套件，如下圖：



在中間有一個 “搜尋 PyPl 與已安裝的套件” 中，我們嘗試搜尋我們要安裝的套件 “matplotlib”，然後選擇第一個選項，用pip 安裝，如下圖：



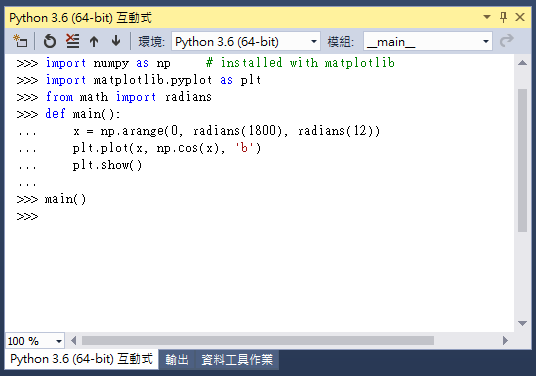
最後，輸出畫面會出現安裝成功的訊息：



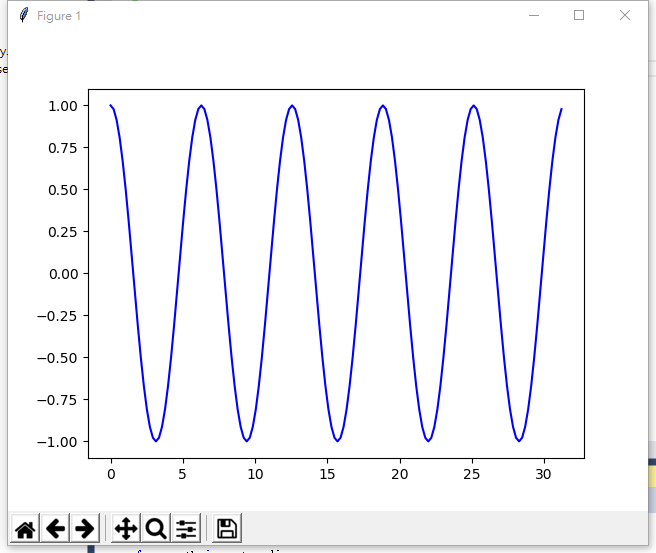
而 Python 環境視窗中，就會出現已經安裝好的東西(等待綠線跑完)：



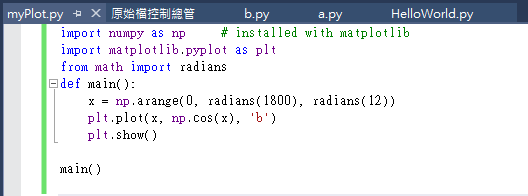
由於 numpy 模組包含在 matplotlib套件之中，安裝完成之後，我們來試試 numpy 是否可以使用，直接在互動式視窗中輸入下列程式碼：



這樣的寫法有沒有覺得很熟悉？Python 其實也跟C一樣有 main()，在這裡我們把程式碼寫在 main() 模組裡面，然後執行 main()，得到的執行結果如下：



如果我想把剛才在互動式視窗裡面的程式碼存到 myPlot.py，只要在互動式視窗裡面選取全部程式碼，按滑鼠右鍵，然後選取 [複製程式碼] ，接著新增一個檔案貼入即可。如下圖：

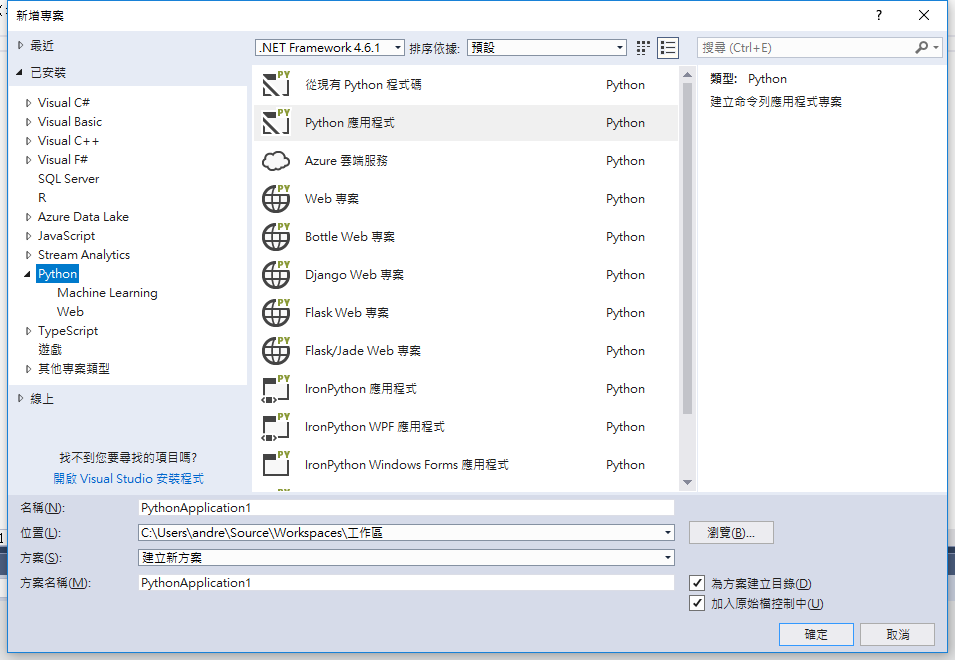


然後打開專案屬性視窗，修改啟動檔案，然後存檔，如下圖：



按下VS整合開發環境的[執行]按鈕，就會出現相同的執行結果了。

## Visual Studio 中的 Python 專案



當我們新增一個 Python 專案的時候，首先映入眼簾的，是一大堆的專案範本，我們第一個使用的模板，是 Python應用程式，它很像一個簡單的DOS程式，就我們現在的需求而言，似乎並不符合。當然，我們可以將 Python 程式寫在任何其他例如 WebForm 的 Application 之中，做多語言的開發，本來 WebForm 網頁之中就包含了 aspx, c#, JavaScript, jQuery, html, BootStrap 等各種程式語言，多一個 Python 也不為過。

但是如果我們要開發獨立的 Python 專案，那麼就必須了解這些範本的用途，解釋如下：

| 範本 | 說明 |
| --- | --- |
| [From existing Python code](https://docs.microsoft.com/zh-tw/visualstudio/python/managing-python-projects-in-visual-studio#creating-a-project-from-existing-files) | 從一個已存在的目錄結構的 Python 程式碼建造一個專案 |
| Python Application | 建在一個基本的 Python 應用程式專案，裡面只有一個空白的程式檔案。內定的執行環境是console interpreter。 |
| [Azure cloud service](https://docs.microsoft.com/zh-tw/visualstudio/python/template-azure-cloud-service) | 建立在 Azure cloud service 之中的Python 專案 |
| [Web projects](https://docs.microsoft.com/zh-tw/visualstudio/python/template-web) | 空白的網路服務專案，支援不同的 frameworks 包含 Bottle, Django, Flask, 以及Flask/Jade。 |
| IronPython Application | 與 Python Application template 類似，但使用IronPython 環境。可以與 .NET 語言混和。 |
| IronPython WPF Application | 使用IronPython 的專案結構，藉由 Windows Presentation Foundation XAML 的使用者介面， Visual Studio 提供 XAML UI 設計工具，背後的程式碼可以用Python 編寫，執行時不會顯示畫面。 |
| IronPython Silverlight Web Page | 是一個可以在使用Silverlight的瀏覽器中執行的 IronPython 專案，程式中的 Python 程式碼會報包裹在網頁之中當成script 來執行。借助在Silverlight 中的 JavaScript 程式碼讓Python 程式碼可以跟 DOM 互動。 |
| IronPython Windows Forms Application | 使用 IronPython 建置 Windows Forms 形式的專案 |
| Background Application (IoT) | 支援Python 專案成為設備的背景服務程式。IoT 相關應用。詳細資料請參考 [Windows IoT Dev Center](https://dev.windows.com/en-us/iot) 。 |
| Python Extension Module | 安裝 Visual C++時如果有選取 **Python native development tools** 就會出現此選項。可編寫 C++ extension DLL。 |

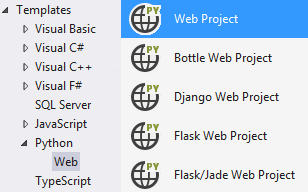
以上所介紹的都是傳統程式開發的專案範本，如果要開發 Web Application，就必須進一步了解其他Web 相關的專案範本

### Web 專案範本

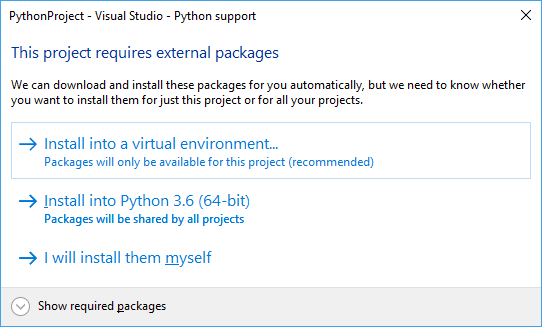
Visual Studio 中的 Python 支援透過專案範本以及偵錯啟動器 (其可針對處理各種架構進行設定)，在 Bottle、Flask 和 Django 架構中開發 Web 專案。 針對 Pyramid 這類其他架構，您也可以使用一般的 [Web 專案] 範本。

Visual Studio 不包含這些架構本身。 您必須以滑鼠右鍵按一下專案，並選取 [Python] > [安裝/升級架構] 來個別安裝架構。

執行時，從範本 (透過 [檔案] > [新增] > [專案] 存取) 建立的專案會搭配隨機選取的本機連接埠啟動網頁伺服器，於偵錯期間開啟您的預設瀏覽器，並允許直接發佈至 Microsoft Azure。



Bottle、Flask 和 Django 範本各自都包含一個含有一些頁面及靜態檔案的起始網站。 從架構特定的範本建立專案時，系統會顯示對話方塊來協助您使用 pip 安裝必要的套件。 另外也建議您針對 Web 專案使用[虛擬環境](https://docs.microsoft.com/zh-tw/visualstudio/python/managing-python-environments-in-visual-studio#global-and-virtual-environments)，如此當您發佈網站時，就會包含正確的相依性：



雖然在大陸擁有眾多使用者的另一大腳本語言PHP也有不少框架，但遠沒有Python這麼誇張，也正是因為Python Web Framework（Python Web開發框架，以下簡稱Python框架）太多，所以在Python社區總有關於Python框架孰優孰劣的話題，討論的時間跨度甚至長達3-5年。

Python這麼多框架，能每個都玩遍的人不多，坦白的說我也只用過其中的兩個開發過測試項目，另外一些稍微接觸過，所以這裡只能淺談一下，歡迎懂的夥伴們補充。

在 Visual Studio 裡面，已經存在三種最流行的框架了，分別是 Bottle, Flask 與 Django，所以我就介紹一下這三種框架。

Django

Python框架雖然說是百花齊放，但仍然有那麼一家是最大的，它就是Django。要說Django是Python框架里最好的，有人同意也有人堅決反對，但說Django的文檔最完善、市場占有率最高、招聘職位最多估計大家都沒什麼意見。Django為人所稱道的地方主要有：

完美的文檔，Django的成功，我覺得很大一部分原因要歸功於Django近乎完美的官方文檔（包括Django book）。

全套的解決方案，Django就像Rails一樣，提供全套的解決方案（full-stack framework + batteries included），基本要什麼有什麼（比如：cache、session、feed、orm、geo、auth），而且全部Django自己造，開發網站必要的工具Django基本都給你做好了，因此開發效率是不用說的，出了問題也容易找到解答。

強大的URL路由配置，Django讓你可以設計出非常優雅的URL，在Django里你基本可以跟醜陋的GET參數說拜拜。

自助管理後台，admin interface是Django里比較吸引眼球的一項contrib，讓你幾乎不用寫一行程式碼就能擁有一個完整的後台管理介面，這讓我想到 WordPress 強大的後台管理，其實就是差不多的意思。

而Django的缺點主要源自Django堅持自己造所有的輪子，整個系統相對封閉，Django最為人詬病的地方有：

系統緊耦合，如果你覺得Django內置的某項功能不是很好，想用喜歡的第三方程式庫來代替是很難的，比如下面將要說的ORM、Template。要在Django裡用SQLAlchemy或Mako幾乎是不可能。重點是 Django自帶的ORM遠不如SQLAlchemy強大，SQLAlchemy是Python世界裡事實上的 ORM標準，其它框架都支持SQLAlchemy了，唯獨Django仍然堅持自己的那一套。

Template功能比較弱，不能插入Python代碼，要寫複雜一點的邏輯需要另外用Python實現Tag或Filter。關於模板這一點，一直以來爭論比較多。

Bottle & Flask

Bottle和Flask作為新生一代Python框架的代表，挺有意思的是都採用了decorator的方式配置URL路由，Bottle、Flask跟web.py一樣，都非常精簡，Bottle甚至所有的代碼都在那一個兩千來行的.py文件里。另外Flask和Pylons一樣，可以跟Jinja2、SQLAlchemy之類結合的很好。

說實在的，這兩個 Framework 由於都是微型，性質很像，所以通常會有互斥性。當然，它們各有各的擁護者，在網路上面，你也可以輕易的查到很多比較的資料。在我看來，不一定要爭個你死我活，兩個並存是一種良性競爭。

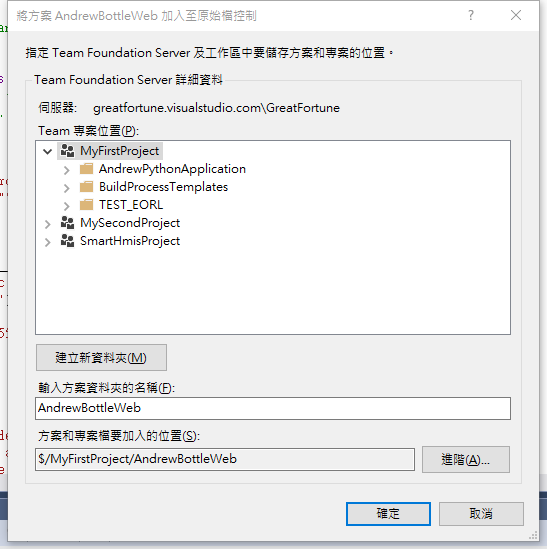
### Bottle Web 專案

不囉嗦，我們就從 Visual Studio 所提供的第一個 Web 專案開始，按照字母順序，就是 Bottle Web 專案。(我不打算建立一個空白專案，對於那些對 Python 一片空白的初學者，空白專案只會雪上加霜)

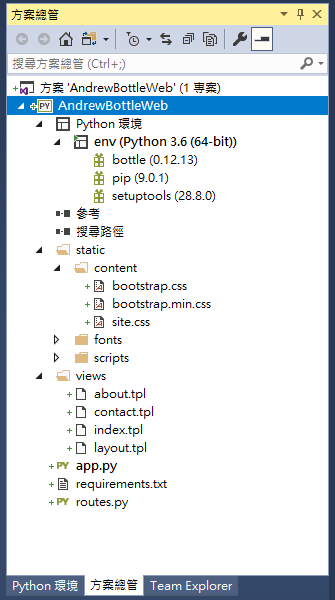
選取新增專案之後，馬上就會出現需要外部下載的提示，請選擇安裝置虛擬環境。接著出現下面的視窗，直接按建立就好：



然後選擇 Team Foundation Server 的位置：

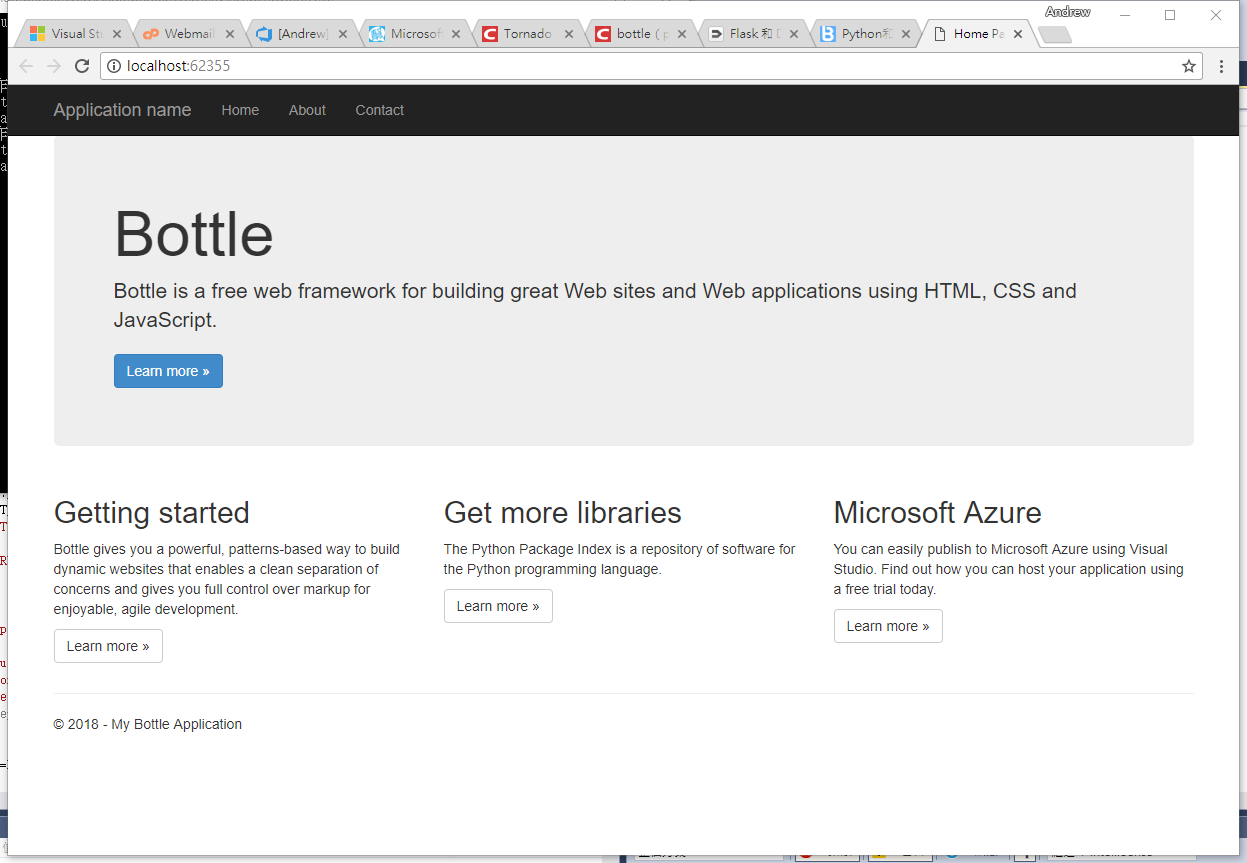


專案建立完成之後，我們先看一下方案總管裡面的架構：



主程式是 app.py，網頁 views 有我們熟悉的 about, contact, index 等，同樣也使用 BootStrap 的技術，裡面包含了該有 css 以及 js 檔案。

我們先不做任何的改變，直接按下執行按鈕，這樣這個簡單的 Bottle 框架就會在瀏覽器裡面執行了：



這個框架包含了三個網頁，Home (index) 、About、以及 Contact。Visual Studio 提供的介面式樣都很像，有沒有覺得這個畫面似曾相識呢?

執行程式的時候，其實是執行 app.py 這個主程式，程式碼如下：

"""

This script runs the application using a development server.

"""

import bottle

import os

import sys

# routes contains the HTTP handlers for our server and must be imported.

import routes

if '--debug' in sys.argv[1:] or 'SERVER\_DEBUG' in os.environ:

# Debug mode will enable more verbose output in the console window.

# It must be set at the beginning of the script.

bottle.debug(True)

def wsgi\_app():

"""Returns the application to make available through wfastcgi. This is used

when the site is published to Microsoft Azure."""

return bottle.default\_app()

if \_\_name\_\_ == '\_\_main\_\_':

PROJECT\_ROOT = os.path.abspath(os.path.dirname(\_\_file\_\_))

STATIC\_ROOT = os.path.join(PROJECT\_ROOT, 'static').replace('\\', '/')

HOST = os.environ.get('SERVER\_HOST', 'localhost')

try:

PORT = int(os.environ.get('SERVER\_PORT', '5555'))

except ValueError:

PORT = 5555

@bottle.route('/static/<filepath:path>')

def server\_static(filepath):

"""Handler for static files, used with the development server.

When running under a production server such as IIS or Apache,

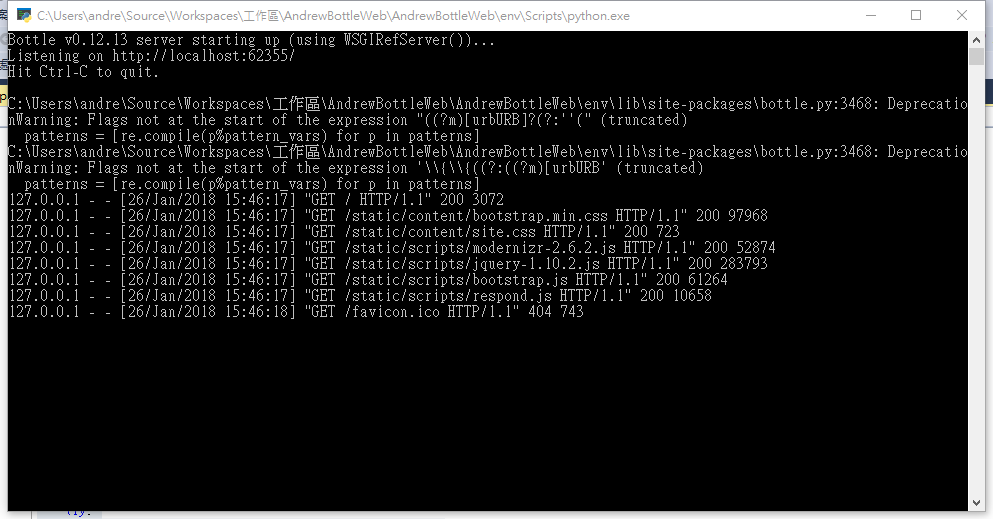
the server should be configured to serve the static files."""

return bottle.static\_file(filepath, root=STATIC\_ROOT)

# Starts a local test server.

bottle.run(server='wsgiref', host=HOST, port=PORT)

其實這段程式碼是在背景執行的，如下圖：



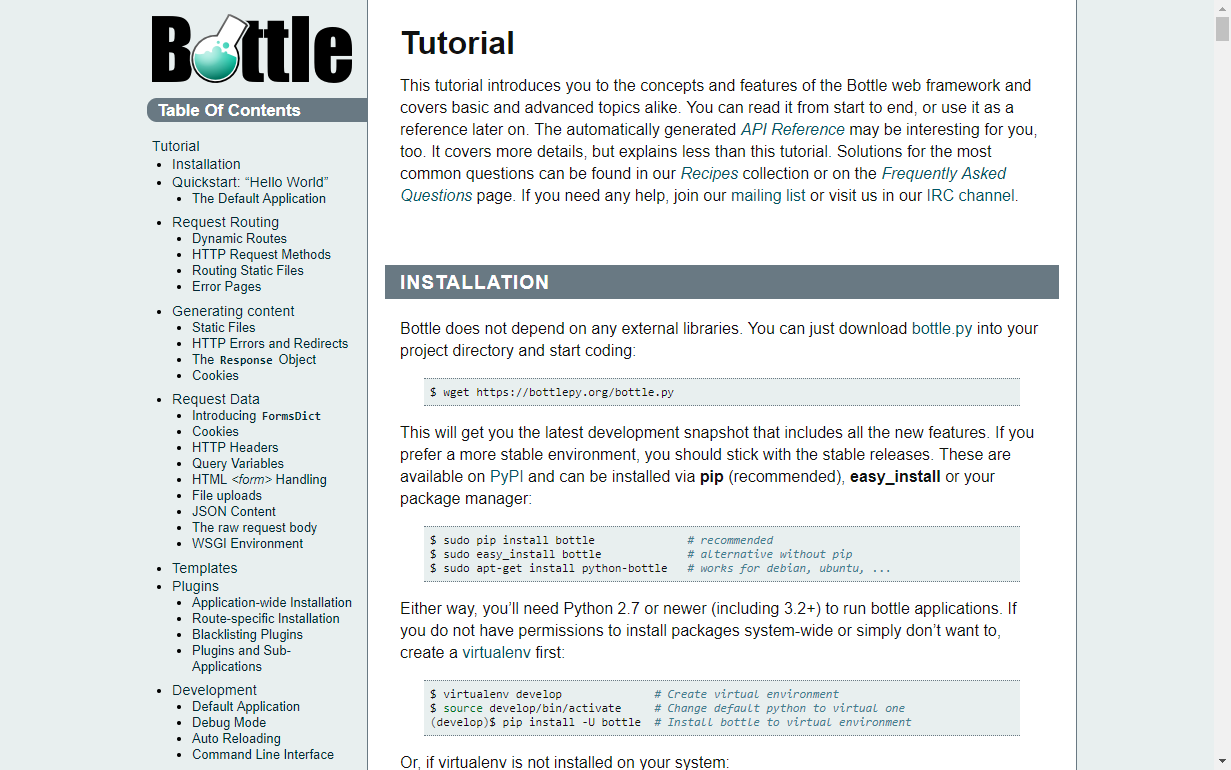
想進一步了解 Bottle，可以在執行出來的網頁點選[learn more…]，就會進入 Bottle 的官網：



裡面介紹了 Bottle 的四個特性：

* Routing: Requests to function-call mapping with support for clean and dynamic URLs.
* Templates: Fast and pythonic built-in template engine and support for mako, jinja2 and cheetah templates.
* Utilities: Convenient access to form data, file uploads, cookies, headers and other HTTP-related metadata.
* Server: Built-in HTTP development server and support for paste, fapws3, bjoern, gae, cherrypy or any other WSGI capable HTTP server.

裡面也有教學網頁：



在我們還不了解 Python 語言的時候，建議都不要去變動 Python 程式碼，由於這是一個網頁框架，我們可以從網頁結構開始。

如同 WebForm Application 裡面的 Site.Master，Bottle 的網頁布局是寫在 layout.tpl 裡面：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{{ title }} - My Bottle Application</title>

<link rel="stylesheet" type="text/css" href="/static/content/bootstrap.min.css" />

<link rel="stylesheet" type="text/css" href="/static/content/site.css" />

<script src="/static/scripts/modernizr-2.6.2.js"></script>

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a href="/" class="navbar-brand">Application name</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="/home">Home</a></li>

<li><a href="/about">About</a></li>

<li><a href="/contact">Contact</a></li>

</ul>

</div>

</div>

</div>

<div class="container body-content">

{{!base}}

<hr />

<footer>

<p>&copy; {{ year }} - My Bottle Application</p>

</footer>

</div>

<script src="/static/scripts/jquery-1.10.2.js"></script>

<script src="/static/scripts/bootstrap.js"></script>

<script src="/static/scripts/respond.js"></script>

</body>

</html>

現在可以試著將網站做一些中文化，至少把 Menu 部分改成中文。接下來可以打開 index.tpl，程式碼如下：

% rebase('layout.tpl', title='Home Page', year=year)

<div class="jumbotron">

<h1>Bottle</h1>

<p class="lead">Bottle is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>

<p><a href="http://bottlepy.org/docs/dev/index.html" class="btn btn-primary btn-large">Learn more &raquo;</a></p>

</div>

<div class="row">

<div class="col-md-4">

<h2>Getting started</h2>

<p>

Bottle gives you a powerful, patterns-based way to build dynamic websites that

enables a clean separation of concerns and gives you full control over markup

for enjoyable, agile development.

</p>

<p><a class="btn btn-default" href="http://bottlepy.org/docs/dev/index.html">Learn more &raquo;</a></p>

</div>

<div class="col-md-4">

<h2>Get more libraries</h2>

<p>The Python Package Index is a repository of software for the Python programming language.</p>

<p><a class="btn btn-default" href="https://pypi.python.org/pypi">Learn more &raquo;</a></p>

</div>

<div class="col-md-4">

<h2>Microsoft Azure</h2>

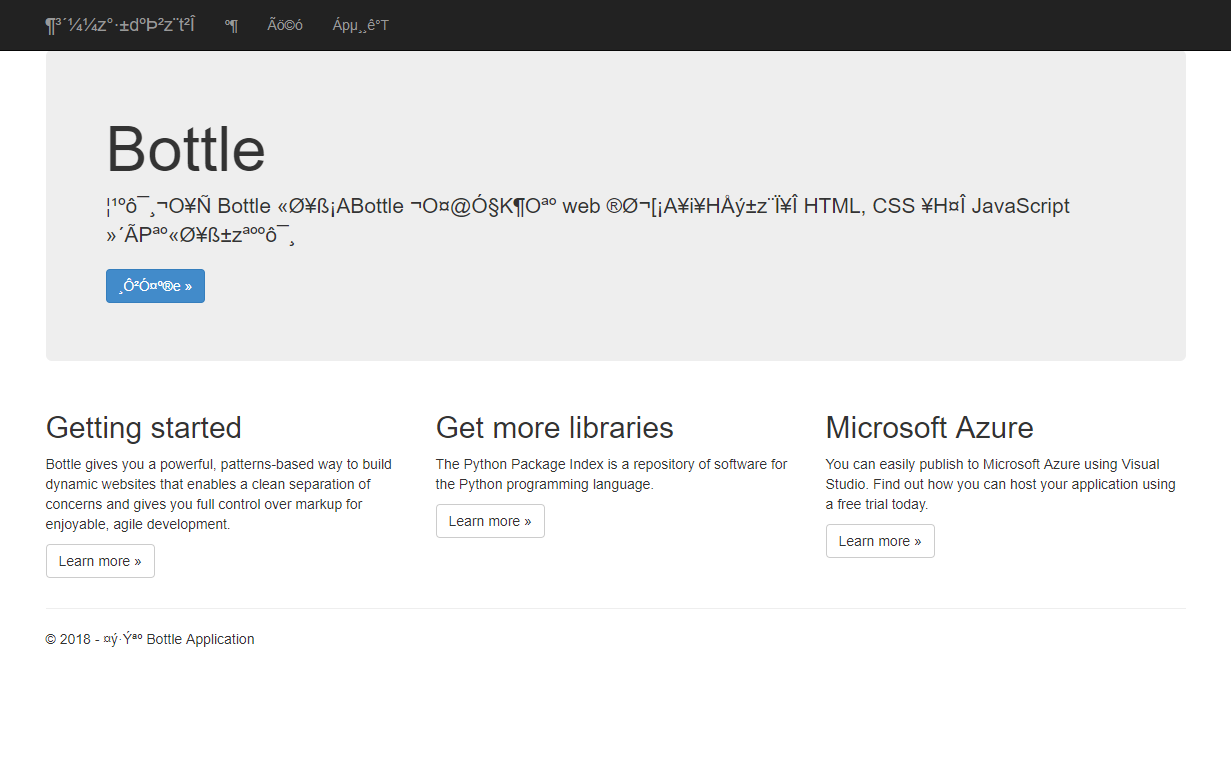
<p>You can easily publish to Microsoft Azure using Visual Studio. Find out how you can host your application using a free trial today.</p>

<p><a class="btn btn-default" href="http://azure.microsoft.com">Learn more &raquo;</a></p>

</div>

</div>

第一行就說明了 layout.tpl 的用途， 同樣的我們也做一些中文化，看看對中文的支援：



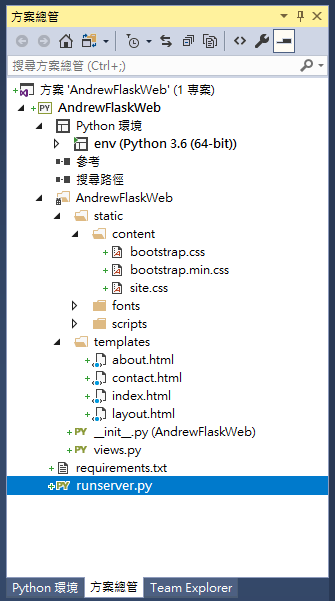
很不幸的，網頁裡面的中文都變成亂碼了，我們看一下這網頁的原始程式碼：

|  |
| --- |
| <!DOCTYPE html> |
|  | <html> |
|  | <head> |
|  | <meta charset="utf-8" /> |
|  | <meta name="viewport" content="width=device-width, initial-scale=1.0"> |
|  | <title>§ÚªºBottle­º­¶ - ¤ý·Ýªº Bottle Application</title> |
|  | <link rel="stylesheet" type="text/css" href="[/static/content/bootstrap.min.css](http://localhost:64734/static/content/bootstrap.min.css)" /> |
|  | <link rel="stylesheet" type="text/css" href="[/static/content/site.css](http://localhost:64734/static/content/site.css)" /> |
|  | <script src="[/static/scripts/modernizr-2.6.2.js](http://localhost:64734/static/scripts/modernizr-2.6.2.js)"></script> |
|  | </head> |
|  |  |

在原始碼中，中文字串的處理就出問題了，所以問題不在瀏覽器，而在 Python。我們先擱置這個問題，先看下一個主題。

### Flask Web 專案

Flask Web 的架構，與 Bottle 十分類似：



這裡的 Templates 是 html 檔案，看起來會比較習慣，程式的進入點是 runserver.py，程式碼更為簡潔：

"""

This script runs the AndrewFlaskWeb application using a development server.

"""

from os import environ

from AndrewFlaskWeb import app

if \_\_name\_\_ == '\_\_main\_\_':

HOST = environ.get('SERVER\_HOST', 'localhost')

try:

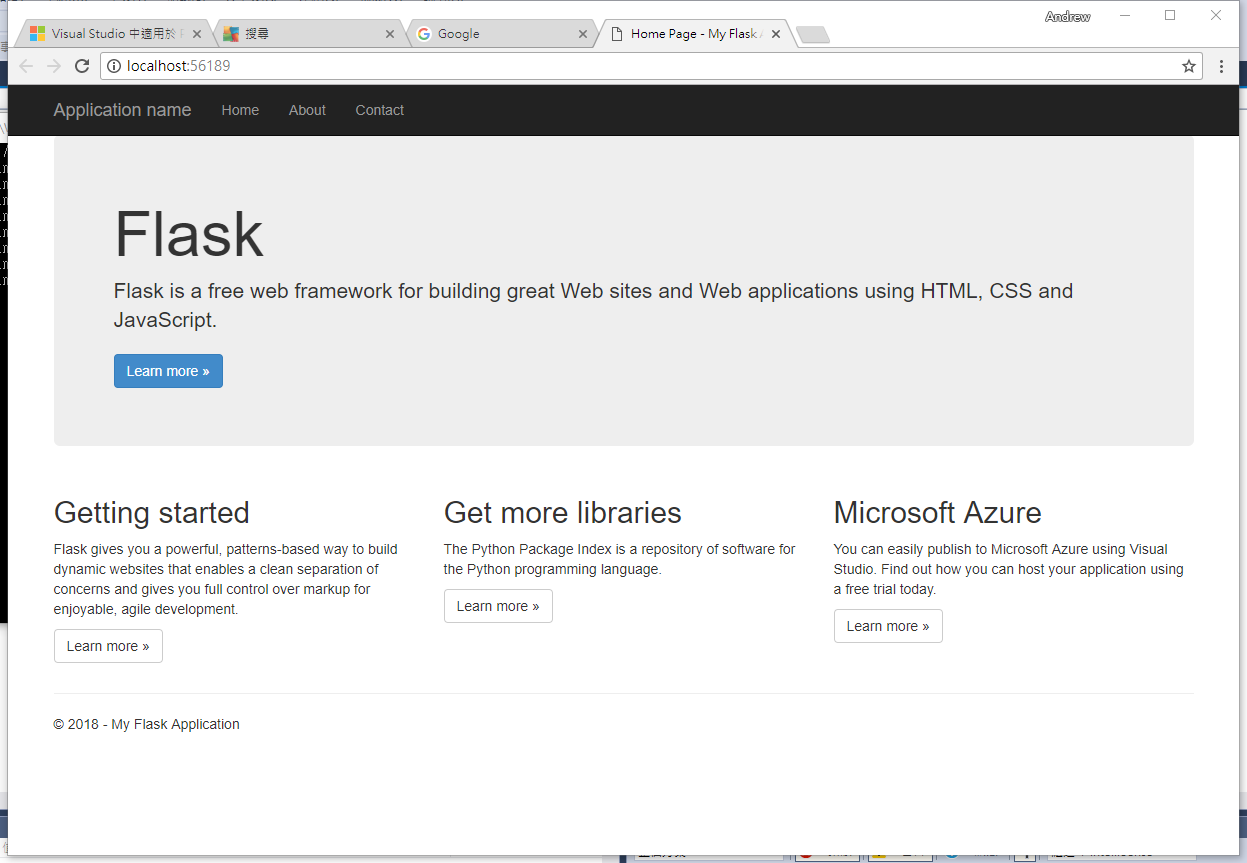
PORT = int(environ.get('SERVER\_PORT', '5555'))

except ValueError:

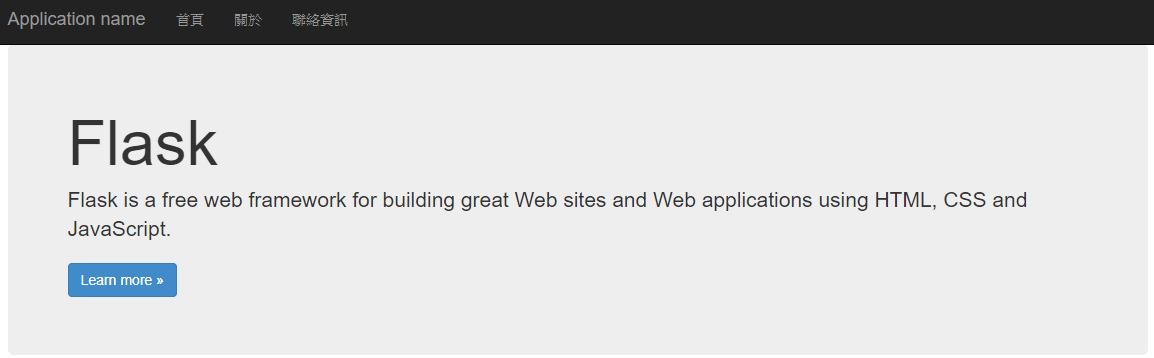
PORT = 5555

app.run(HOST, PORT)

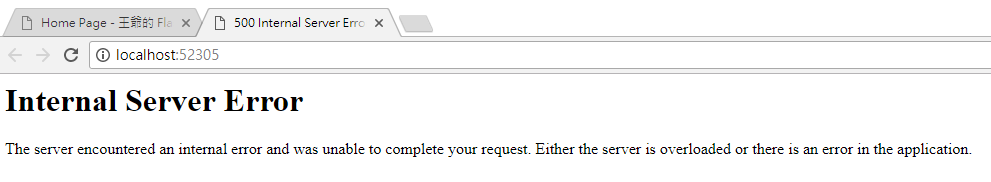
快速看一下執行結果，完全一模一樣：



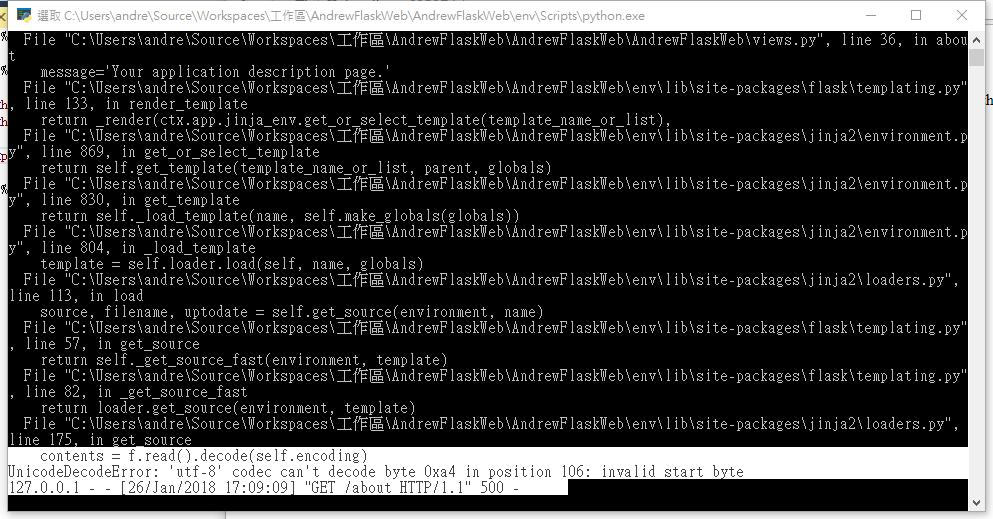
好消息是，當我修改 layout.html 的內容為中文時，是可以執行的，請看下圖中的 Menu 文字，首頁、關於、聯絡資訊：



但是真的不能高興得太早，當我修改 index.html 的時候，就出現問題了，只要在裡面加入任何中文，馬上就會有錯誤訊息：



回到執行視窗看看錯誤訊息：



這應該是無法處理 Unicode 的問題，看來 Bottle 與 Flask 都有中文相容性的BUG。我們還是先擱置，繼續我們最大的 Django 架構。

### Django Web 專案

前面新增方案的步驟都一樣，跳過不談，專案建立好之後，第一個畫面就是執行前的說明：



One More Thing…

千萬不要急著執行，先看一下說明。第一件重要的事情就寫在這裡，在執行這個專案之前，請先建置一個 super user 的帳號。請按照下面的步驟做，首先開啟方案總管，在方案名稱上面點滑鼠右鍵，選取 Python -> Django Create Superuser，接下來就會進入 Django 管理主控台，按照指示完成帳號建置，請注意，帳號與密碼不可以相同：

正在執行 manage.py createsuperuser

You have 1 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): auth.

Run 'python manage.py migrate' to apply them.

Username (leave blank to use 'user'):

andrew

Email address:

andrewwang@great-fortune.net

C:\Users\user\AppData\Local\Programs\Python\Python36\lib\getpass.py:100: GetPassWarning: Can not control echo on the terminal.

return fallback\_getpass(prompt, stream)

Warning: Password input may be echoed.

Password:

Andrew0xxx2

Warning: Password input may be echoed.

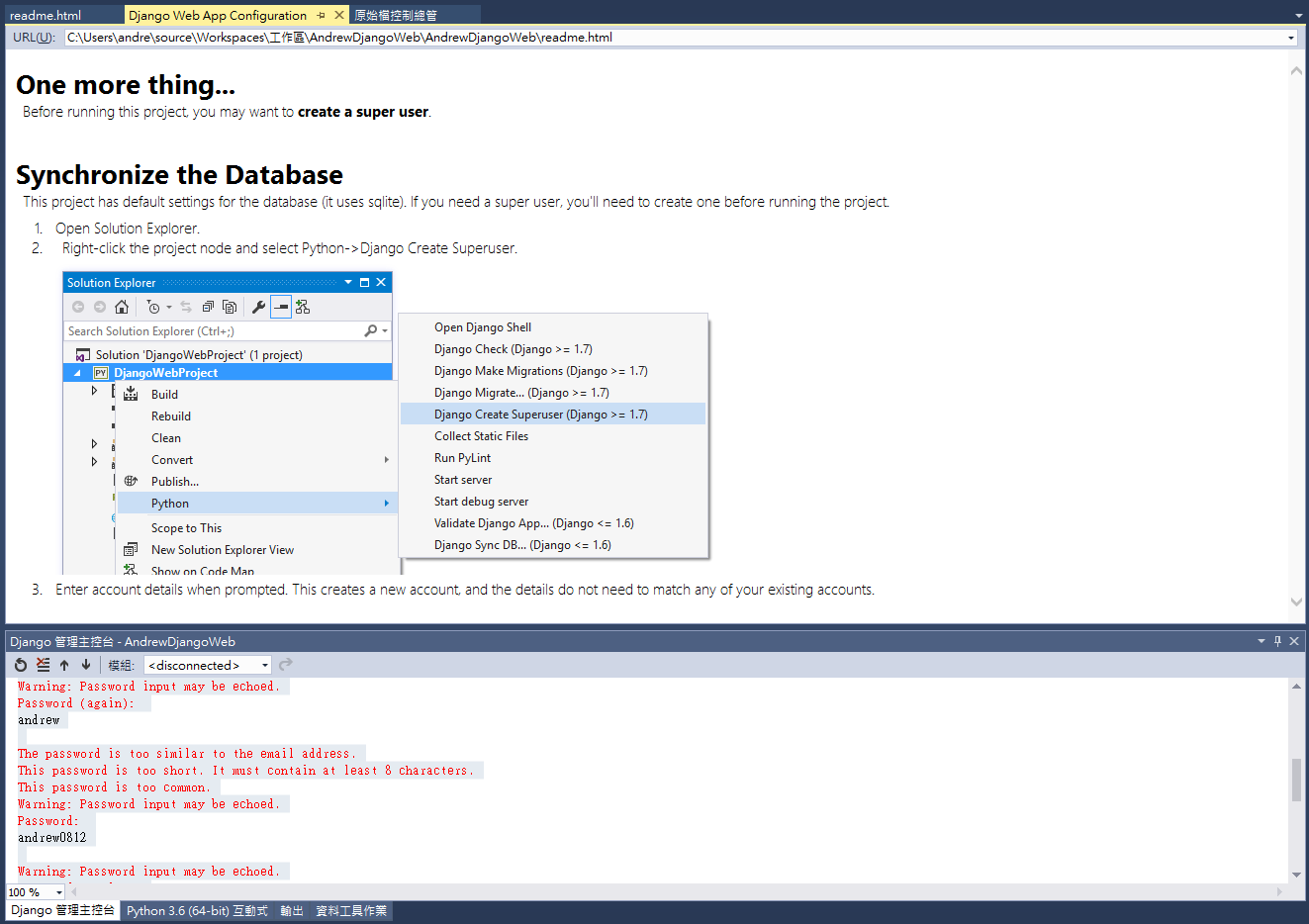
Password (again):

Andrew0xxx2

Superuser created successfully.

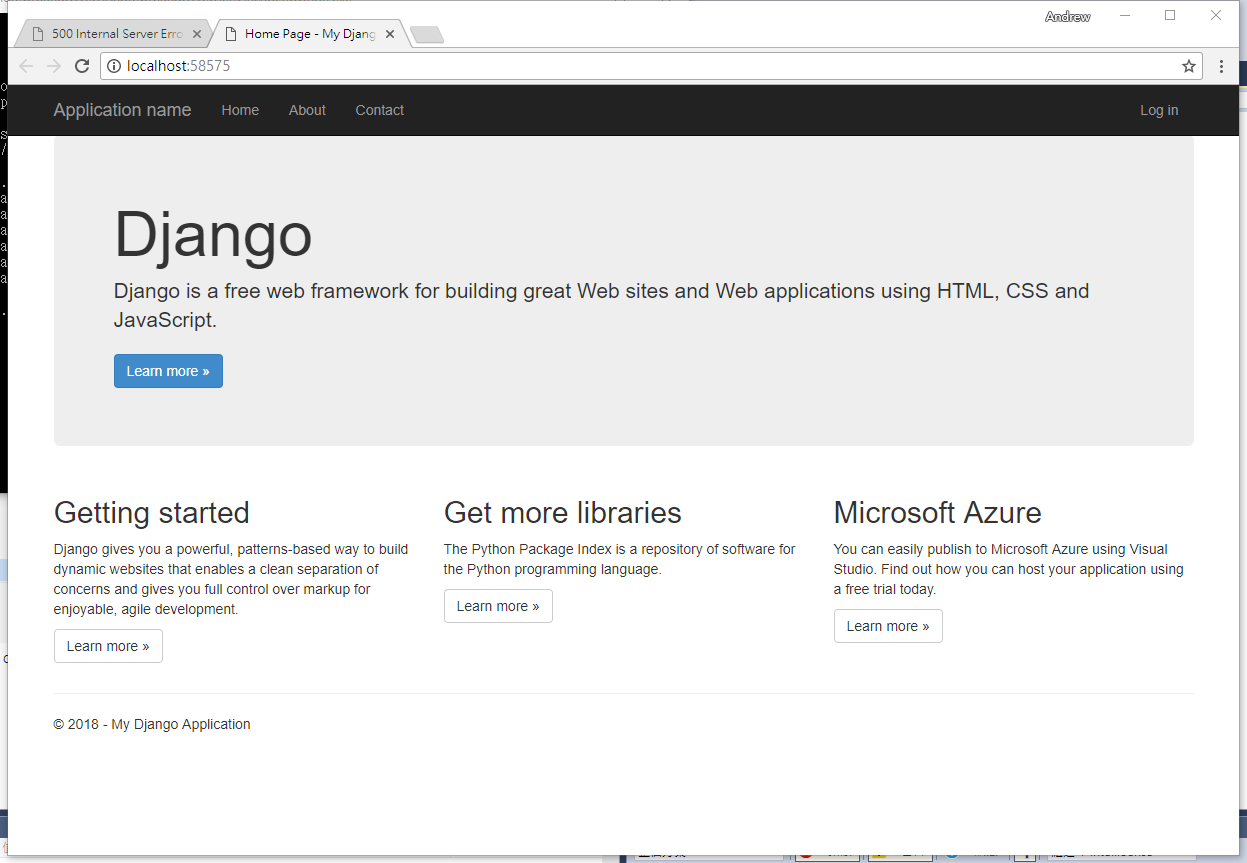
互動式 Python 處理序已結束。

互動式 Python 處理序已結束。

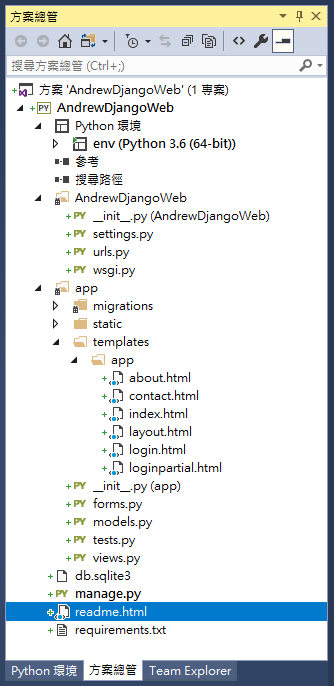


(測試用密碼備忘截圖，勿貼)

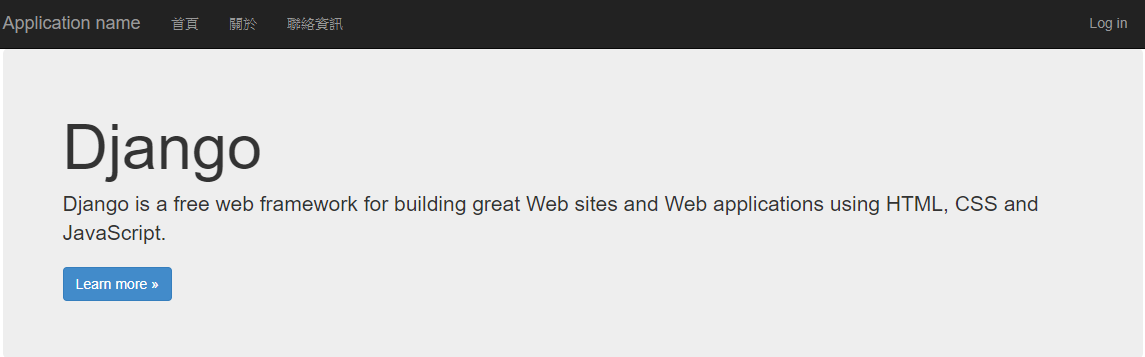
接著，我們就可以執行了，架構也是相同的：



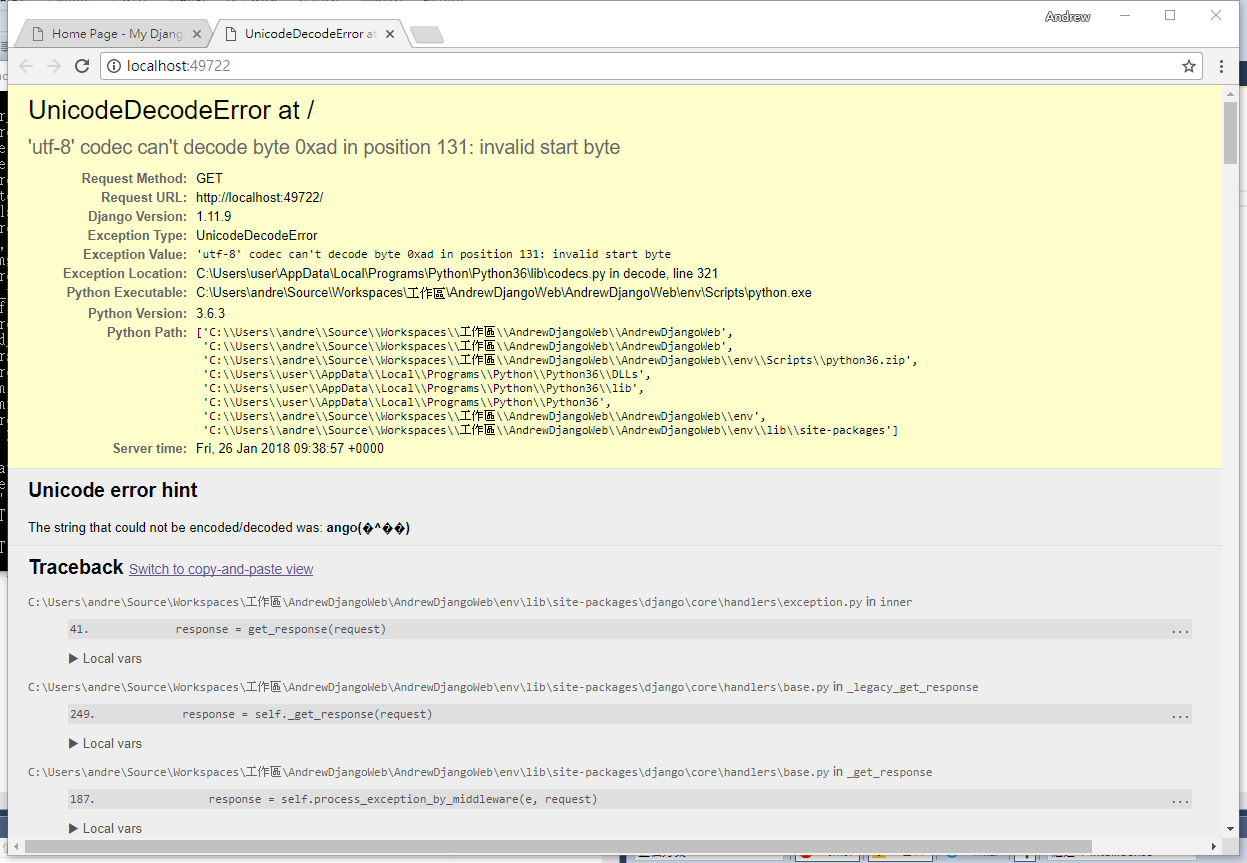
Django 的專案架構就複雜很多，但是我想如果已經有 WebForm 或是 MVC 的設計經驗的人，應該很容易看懂：



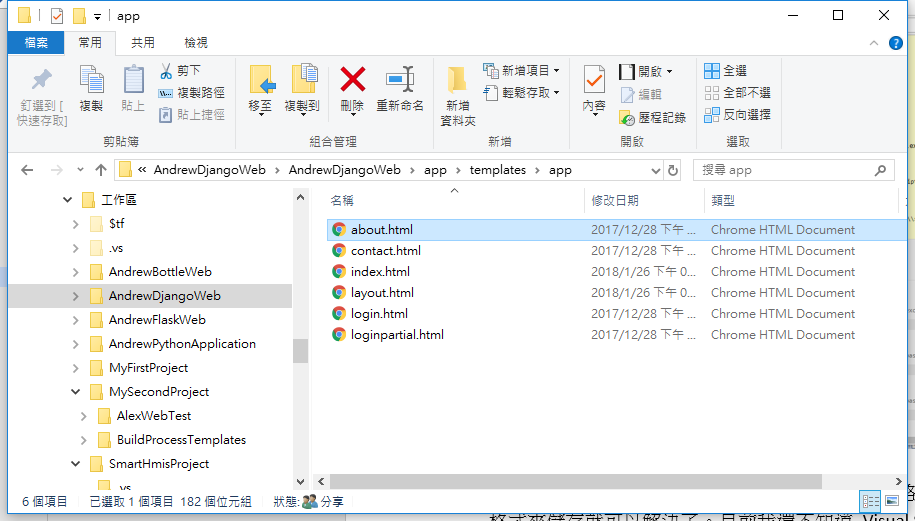
我們就依照之前一樣的方式，修改 layout.html 與 index.html 檢查中文相容性問題。很不幸的，Django 的情況跟 Flask 一模一樣，在 layout.html 裡面是可以修改成中文的，如下圖：



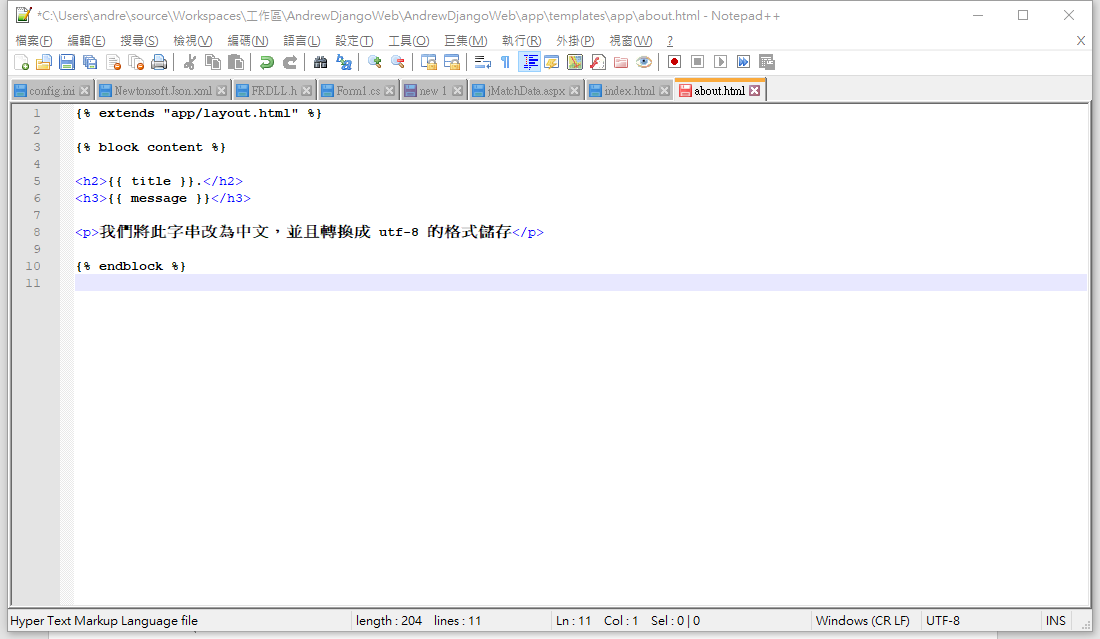
但是在 index.html 裡面加入中文的話，就會出現 Unicode 的錯誤：



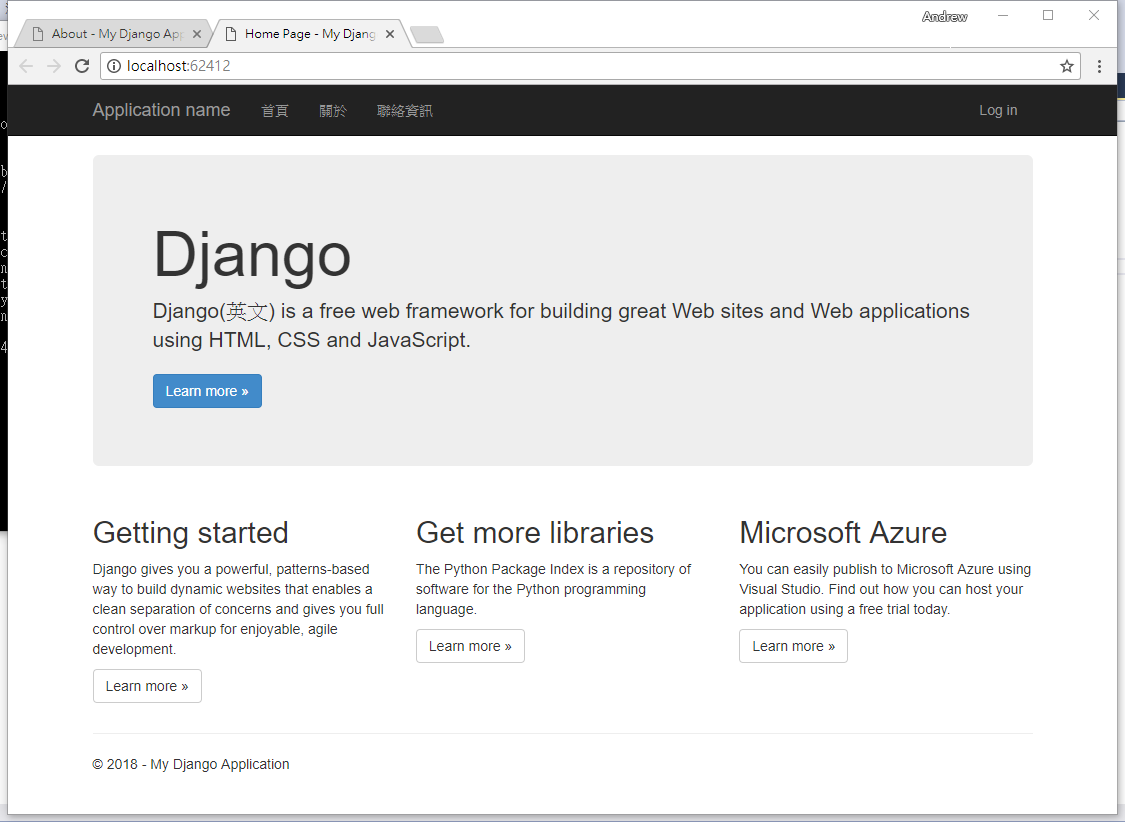
這個問題主要來自於 index.html 檔案本身的編碼格式，Visual Studio 的編碼格式為 utf-8 (無BOM)，但是我們必須使用 utf-8 的編碼格式來儲存就可以解決問題了。目前我還不知道 Visual Studio 整合開發工具中的 Visual Studio Code用什麼方式將檔案轉成 utf-8，所以我透過 Notepad++ 來做，先開啟專案的目錄：



在 index.html (或其他要改的檔案) 上面按滑鼠右鍵選擇 Edit with Notepad++，然後修改檔案內容如下：



存檔前，請選擇 [編碼] [轉換成 utf-8格式]，然後存檔即可。再執行一次：成功！

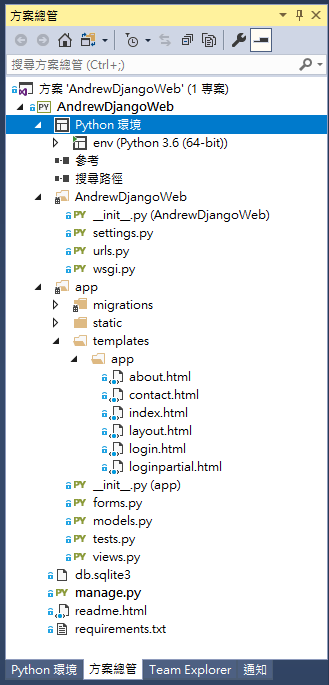


用同樣的方式，就可以解決Flask 的問題了。經過實際測試，Bottle 還是會有奇怪的現象，或許是因為 .tpl 為特別的格式，在存成 utf-8 之後，執行就會不正常，而 Django 與 Flask 都是使用 .html 存檔，就沒有問題，一切正常。

到目前為止，有兩個框架是確實可以用的，只是要注意出現 utf-8 的問題的時候，借助 Notepad++來解決。

## Django 資料夾與檔案解析

如果我們不是透過 Visual Studio 來建立網站，而是在其他的環境，通常會透過執行 Django-admin stratproject mblog 這樣的指令來建立，建立完成之後，我們必須了解 Django 幫我們建立了哪些東西，這點很重要。這樣當我們要替網站加入功能的時候，才知道要如何著手。



現在我就來說明透過 Visual Studio 標準方法所建立的專案架構。上圖是一個新增的專案，首先，在我們的專案檔裡面，我們會看到一個 Python 環境，這裡面放著 Pyhton 3.6 (64-bit) 檔案，這代表這個 Web Site 的虛擬環境是 Python 3.6版本，這種做法可以讓我們所建立的每一個專案獨立執行而不互相干擾，這點對於開發環境很重要，因為有先專案可能是使用 Python 2.7 來開發的。

接下來在專案的根目錄中會看見一個天字第一號的檔案 manage.py，內容如下：

#!/usr/bin/env python

"""

Command-line utility for administrative tasks.

"""

import os

import sys

if \_\_name\_\_ == "\_\_main\_\_":

os.environ.setdefault(

"DJANGO\_SETTINGS\_MODULE",

"AndrewDjangoWeb.settings"

)

from django.core.management import execute\_from\_command\_line

execute\_from\_command\_line(sys.argv)

這是 Django 用來管理網站組態的檔案，算是一個接收命令列指令的工具程式，Django 所有的命令都是執行此程式，平時我們並不會去修改它，只要確保能夠執行就好了。

而第一個建好的，和專案同名的資料夾(這裡是AndrewDjangoWeb 資料夾)，這裡面放的就是這個專案中最重要的一些設定檔案，例如 settings.py, urls.py 以及 wsgi.py，其中 wsgi.py 是和虛擬主機中的網頁伺服器(IIS)溝通的介面，中間的設定等要把網站上線使用時才會用到，內容如下：

"""

WSGI config for AndrewDjangoWeb project.

This module contains the WSGI application used by Django's development server

and any production WSGI deployments. It should expose a module-level variable

named ``application``. Django's ``runserver`` and ``runfcgi`` commands discover

this application via the ``WSGI\_APPLICATION`` setting.

Usually you will have the standard Django WSGI application here, but it also

might make sense to replace the whole Django WSGI application with a custom one

that later delegates to the Django one. For example, you could introduce WSGI

middleware here, or combine a Django application with an application of another

framework.

"""

import os

os.environ.setdefault("DJANGO\_SETTINGS\_MODULE", "AndrewDjangoWeb.settings")

# This application object is used by any WSGI server configured to use this

# file. This includes Django's development server, if the WSGI\_APPLICATION

# setting points here.

from django.core.wsgi import get\_wsgi\_application

application = get\_wsgi\_application()

# Apply WSGI middleware here.

# from helloworld.wsgi import HelloWorldApplication

# application = HelloWorldApplication(application)

Urls.py 用來對應每一個URL的網址要對應的函數以及對應的方式，通常都是在建立新的網頁時要先編輯的檔案。內容如下，裡面有說明，當我們需要增加某些功能的時候，必須 uncomment 哪幾行，當要加入 admin 功能的時候就要注意了。另外當我們要新增網頁的時候，也會修改到此檔案：

"""

Definition of urls for AndrewDjangoWeb.

"""

from datetime import datetime

from django.conf.urls import url

import django.contrib.auth.views

import app.forms

import app.views

# Uncomment the next lines to enable the admin:

# from django.conf.urls import include

# from django.contrib import admin

# admin.autodiscover()

urlpatterns = [

# Examples:

url(r'^$', app.views.home, name='home'),

url(r'^contact$', app.views.contact, name='contact'),

url(r'^about', app.views.about, name='about'),

url(r'^login/$',

django.contrib.auth.views.login,

{

'template\_name': 'app/login.html',

'authentication\_form': app.forms.BootstrapAuthenticationForm,

'extra\_context':

{

'title': 'Log in',

'year': datetime.now().year,

}

},

name='login'),

url(r'^logout$',

django.contrib.auth.views.logout,

{

'next\_page': '/',

},

name='logout'),

# Uncomment the admin/doc line below to enable admin documentation:

# url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

# Uncomment the next line to enable the admin:

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

]

Settings.py 則是此網站的系統設計所在的位置，新建立的網站，都要先開啟此檔案，做些編輯設定的動作。如下圖：

"""

Django settings for AndrewDjangoWeb project.

Generated by 'django-admin startproject' using Django 1.9.1.

For more information on this file, see

https://docs.djangoproject.com/en/1.9/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/1.9/ref/settings/

"""

import os

import posixpath

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/1.9/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'd8c77380-d00e-4915-8ffe-7bc55f237809'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

'app',

# Add your apps here to enable them

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

MIDDLEWARE\_CLASSES = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.auth.middleware.SessionAuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'AndrewDjangoWeb.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [],

'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',

],

},

},

]

WSGI\_APPLICATION = 'AndrewDjangoWeb.wsgi.application'

# Database

# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

# Password validation

# https://docs.djangoproject.com/en/1.9/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/1.9/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/1.9/howto/static-files/

STATIC\_URL = '/static/'

STATIC\_ROOT = posixpath.join(\*(BASE\_DIR.split(os.path.sep) + ['static']))

在 Application Definition 中定義了我們這個網頁的 app 資料夾，而真正的網站邏輯，就在這個資料夾之中。會使用這樣的方式，主要的目的是讓網站的每一個主要功能都能成為一個單獨的模組，方便網站的開發者可以在不同的網站中重複使用，這也是 Python 程式語言中 reuse 觀念的應用。

網站的APP目錄定義在此處的第一行(如果你的目錄名稱不是app，可以在此修改)：

# Application definition

INSTALLED\_APPS = [

'app',

# Add your apps here to enable them

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

我們先練習編輯這檔案的時區設定，如下：

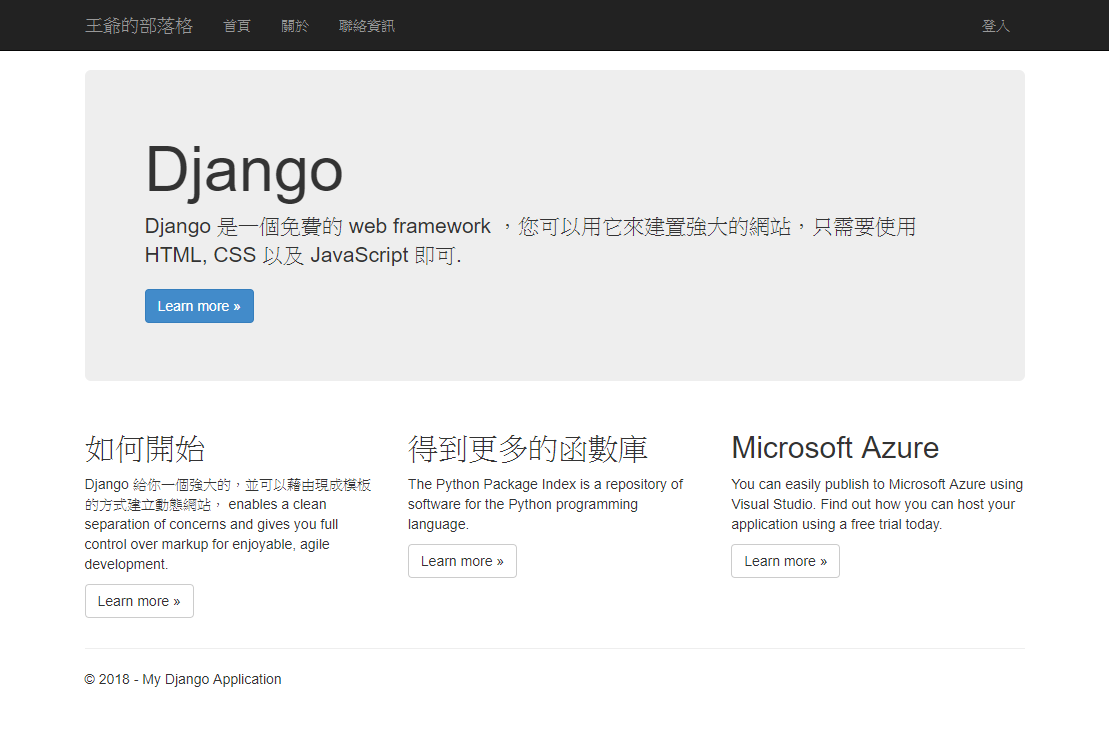
LANGUAGE\_CODE = 'zh-TW'

TIME\_ZONE = 'Asia/Taipei'

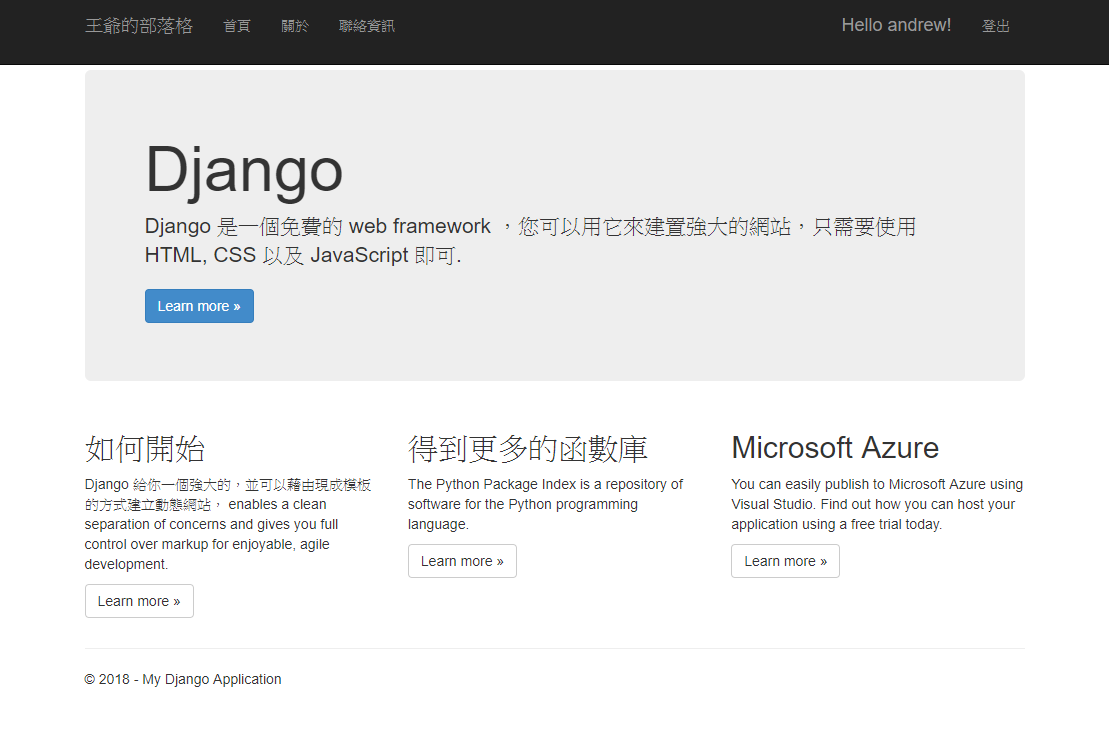
PS: 建議先不要修改，否則會無法執行。等到我們後面載入新的模組之後再談。

另外，在預設的情況下，Django 會使用 SQLite 來儲存資料庫的內容，在我們建立專案的時候，根目錄中即會產生一個叫做 db.sqlite3 的檔案。之後，所有在此網站中新增到資料庫的資料，都會被放在這裡。如果要搬移網站的時候，也記得要把這個檔案帶上。

接著我們把注意力集中到 app 的資料夾中，首先你會在 templates/app 目錄底下看到所有網站裡面使用到的網頁 html 檔，前一個章節我已經分享過如何讓這些網頁顯示中文，你可以修改這些 html 的內容。經過簡單的修改，我已經將首頁的大部分內容改成中文了，包含上面的選單。如下圖：



還記得一開始新增網站時我們建立了一個 super user 嗎? 在這裡可以點 “登入”，以 Andrew 帳號登入，登入之後如下圖，右上角就會看到歡迎字樣：



由於 Django 是一種 MVC 架構的網站，所以在 app 目錄底下還可以看到 models.py, views.py 以及 forms.py (與 controller 相關程式碼)，範例程式只有 login 有使用者互動功能，所以 forms.py 裡面定義的 login 的方法：

"""

Definition of forms.

"""

from django import forms

from django.contrib.auth.forms import AuthenticationForm

from django.utils.translation import ugettext\_lazy as \_

class BootstrapAuthenticationForm(AuthenticationForm):

"""Authentication form which uses boostrap CSS."""

username = forms.CharField(max\_length=254,

widget=forms.TextInput({

'class': 'form-control',

'placeholder': 'User name'}))

password = forms.CharField(label=\_("Password"),

widget=forms.PasswordInput({

'class': 'form-control',

'placeholder':'Password'}))

View.py 裡面則是定義了網站裡面的 Views：

"""

Definition of views.

"""

from django.shortcuts import render

from django.http import HttpRequest

from django.template import RequestContext

from datetime import datetime

def home(request):

"""Renders the home page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/index.html',

{

'title':'Home Page',

'year':datetime.now().year,

}

)

def contact(request):

"""Renders the contact page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/contact.html',

{

'title':'Contact',

'message':'Your contact page.',

'year':datetime.now().year,

}

)

def about(request):

"""Renders the about page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/about.html',

{

'title':'About',

'message':'Your application description page.',

'year':datetime.now().year,

}

)

最後 models.py 則是定義資料庫，目前是空白，後面我們會特別說明 Django 是如何設計資料庫：

"""

Definition of models.

"""

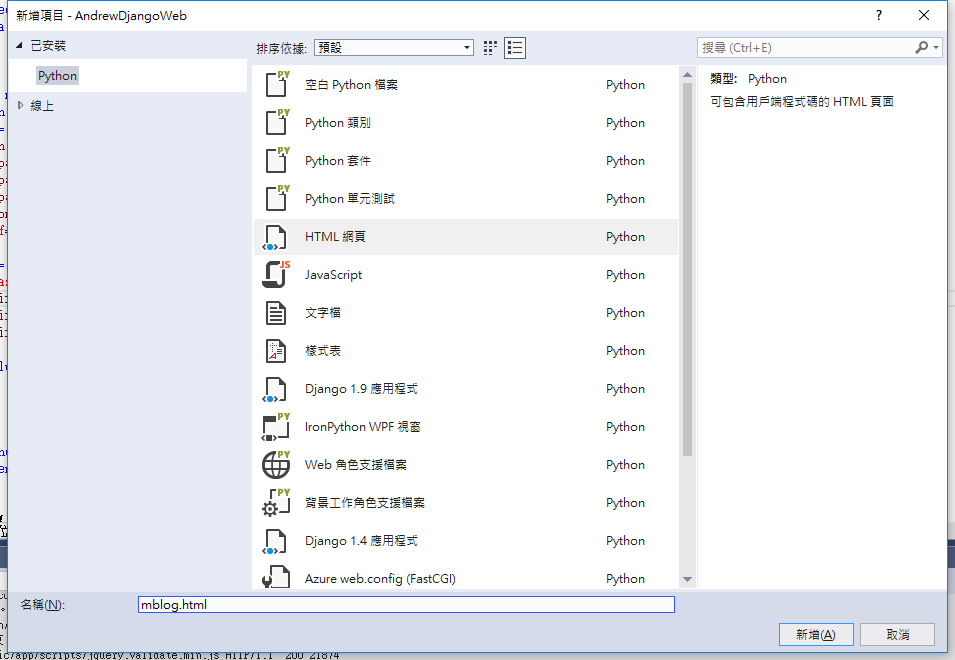
from django.db import models

# Create your models here.

### 在Django 主選單中增加網頁

目前我們的網頁架構很簡單，就是 home, contact, about 三個 html，他們的架構關係則是放在 layout.html 裡面，如果我們要建立一個新的網也叫做 mblog.html 該如何做呢?

最簡單的方式，就是在 app 目錄底下，按滑鼠右鍵選擇[加入]，如下圖：



直接加入一個 html 網頁，內容如下，由於前面我們已經知道 Visual Studio 會有編碼的問題，請將此檔案用 Notepad++打開，將原本的 UTF-8 碼(檔首無BOM) 編碼轉換為UTF-8 碼之後存檔：

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

</body>

</html>

以上內容完全不能用，我們比照 about.html 的寫法，將其改為：

{% extends "app/layout.html" %}

{% block content %}

<h2>**{{** title }}</h2>

<h3>**{{** message }}</h3>

<p>這是我的部落格</p>

{% endblock %}

裡面的 title 與 message 定義在哪裡呢? 請在 views.py 裡面加入定義：

def mblog(request):

"""Renders the home page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/mblog.html',

{

'title':'我的部落格',

'message':'部落格內容.',

'year':datetime.now().year,

}

)

並且在 urls.py 裡面加入一行：

urlpatterns = [

# Examples:

url(r'^$', app.views.home, name='home'),

url(r'^mblog', app.views.mblog, name='mblog'),

url(r'^contact$', app.views.contact, name='contact'),

url(r'^about', app.views.about, name='about'),

最後，為了讓我們可以在主選單中選到，修該 layout.html 如下：

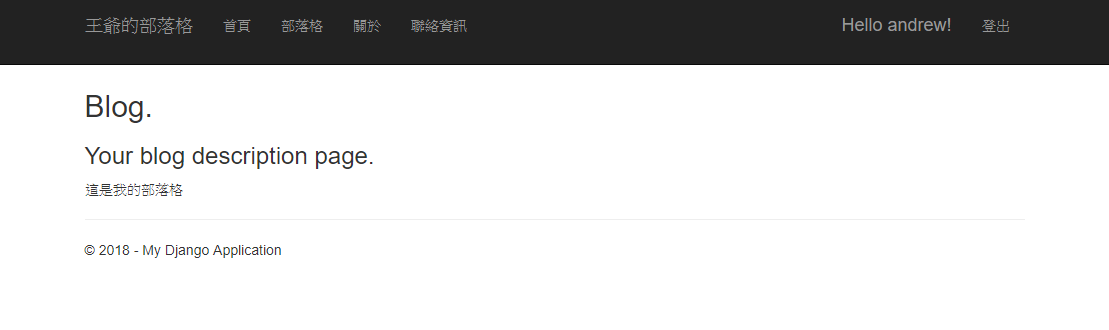
<li><a href="{% url 'home' %}">首頁</a></li>

<li><a href="{% url 'mblog' %}">部落格</a></li>

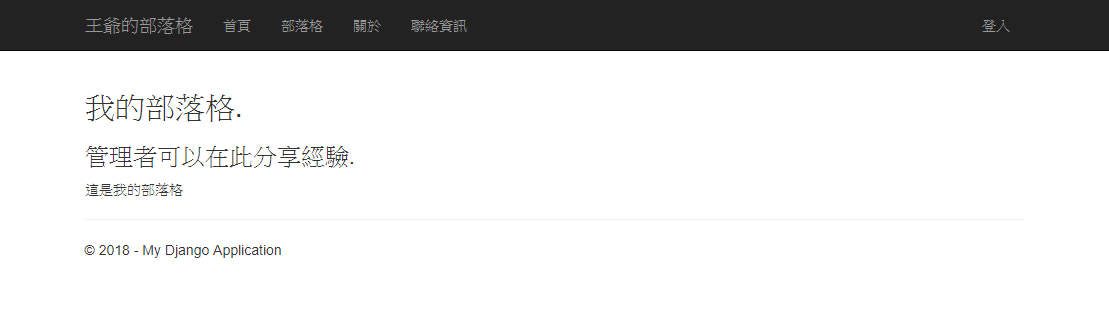
<li><a href="{% url 'about' %}">關於</a></li>

<li><a href="{% url 'contact' %}">聯絡資訊</a></li>

這樣就完成了，結果如下：



中文也完全支援：



## 建立部落格資料庫

上一個章節我們建立了一個簡單可以運行的網站架構，但是卻只有一些基本的內容，我們在使用 C# 建立健康網站的時候，使用了資料庫與動態網頁的技巧，當然 Django 也同樣可以做到。

在這個章節，我們要設計一個簡單的資料表，提供使用者可以建立自己的部落格中最重要的文章內容。

### 資料庫與 Django 的關係

在預設的情況下，Django 的資料庫是以 Model 的方式來操作，也就是在程式中不直接面對資料庫以及資料表，而是以 class 類別的方式先建立出 Model，然後再透過對 Model 的操作，達到操作資料庫的目的。這樣的好處是把程式和資料庫之間的關係以一層中介層來做為連接的介面，日後如果需要更換資料庫系統，就可以不需要更動到程式內容。

也是因為這樣，對於第一次接觸到此種方式的網站開發者而言會不太直覺，不去直接定義資料庫中的資料表，而是以定義一個資料類別來當作是資料表，在定義資料類別之後，還要再執行一些指令讓這個資料表的每一個資料欄位的名稱、格式、屬性可以和資料類別中的內容同步，的確是有些麻煩。

簡單的看，Django 要使用資料庫時，有以下幾個步驟：

1. 在 models.py 中定義所需要使用的類別
2. 詳細地設定每一個在類別中的變數，亦即資料表中的欄位
3. 使用 python manage.py makemigrations app 來建立資料庫與 Django 間的中介檔案
4. 使用 python manage.py migrate 同步更新資料庫內容
5. 在程式中使用 python 的語法操作所定義的資料類別，等於是在操作資料庫中的資料表

### 定義資料模型

一開始使用 Visual Studio 建立出來的 models.py 的內容是空白的，我們要修改其內容如下：

"""

Definition of models.

"""

from django.db import models

from django.utils import timezone

# Create your models here.

class Post(models.Model):

title = models.CharField(max\_length=200)

slug = models.CharField(max\_length=200)

body = models.TextField()

pub\_date = models.DateTimeField(default=timezone.now)

class Meta:

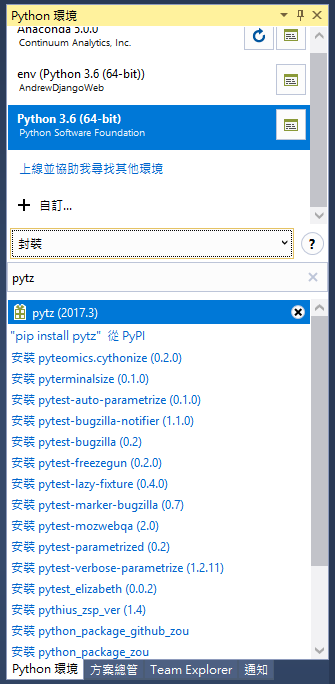
ordering = ('-pub\_date',)

def \_\_unicode\_\_(self):

return self.title

這裡我們首先建立了一個類別叫做 Post，其中 title 是用來顯示文章的標題，而 slug 是用來顯示文章的網址，body 則是文章的內容，最後 pub\_date 則是本文發表的時間。至於 class Meta 中的設定，則是指定文章要顯示的順序，是以 pub\_date 為依據。最後 \_\_unicode\_\_ 則是提供此類別所產生的資料項目，以文章標題當作是顯示內容，增加在操作時的可讀性，使用 Unicode 而不是 str，讓這個標題可以正確的支援中文。

Pub\_date 是以 timezone.now 的方式讓它資動產生，這還需要一個 pytz模組才行。請安裝：



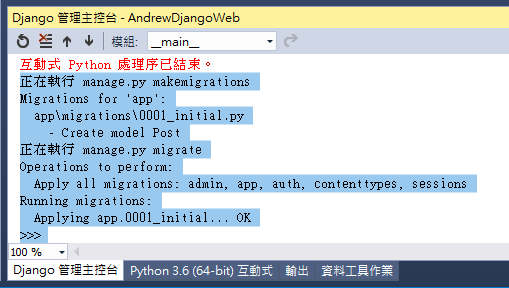
找到安裝 pytz，點擊安裝，輸入結果如下：

----- 正在安裝 'pytz' -----

Requirement already satisfied: pytz in c:\users\user\appdata\local\programs\python\python36\lib\site-packages

----- 已成功安裝 'pytz' -----

要讓此 models.py 生效，需要執行下面的指令：在Shell中使用 python manage.py makemigrations app 來建立資料庫與 Django 間的中介檔案，在 Visual Studio 整合環境中，我們不需要下指令，請選取[專案][Django建立轉移]。然後使用 python manage.py migrate 同步更新資料庫內容，同樣的，請選取[專案][Django遷移]，此兩個動作執行結果如下圖：



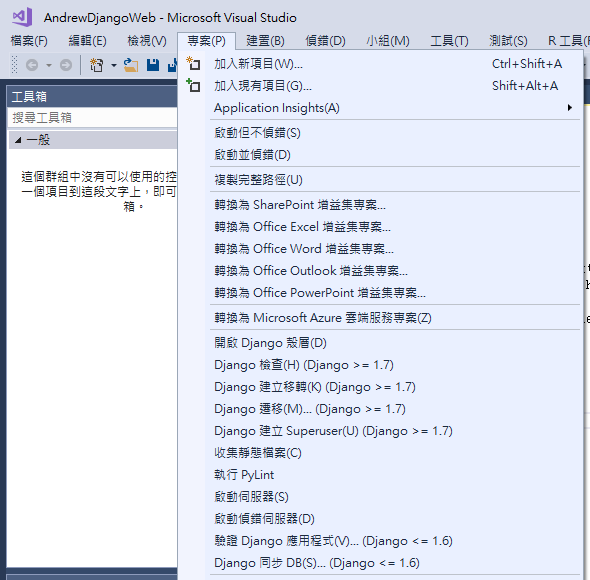
### 啟用 Admin 管理介面

這個部分十分重要，請務必耐心看完，由於使用 Python 管理資料庫與之前使用 C# 管理資料庫完全不同，所以我們必須忘記使用 Visual Studio 的伺服器管理功能，而必須使用 Python 所提供的管理功能。

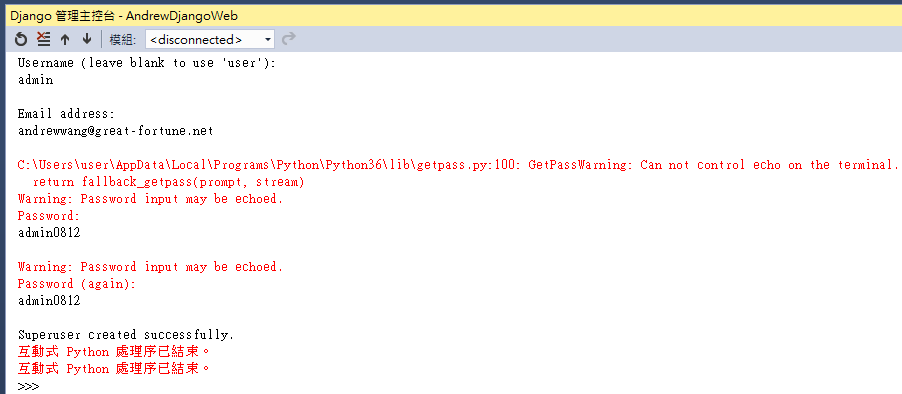
admin 是 Django 預設的資料庫內容管理介面，在使用之前，有幾個設定的步驟，第一個步驟，就是建立一個管理者的帳號以及密碼，還記得我們在一開始建立第一個網站的時候，就建立了一個名為 Andrew 的 superuser，現在我們多建立一個 admin，你可以使用下面的命令來達成：

$ python manage.py createsuperuser

當然，在GUI環境，我們可以不需要下命令，如下圖：



選取[專案]選單，就可以看到所有 Django 裡面常用的命令，前一個章節我們用了[Django 建立移轉] 與 [Django 遷移]，現在我們要用 [Django 建立 Superuser] 來建立 admin。完成之後如下圖：



接著，把上一章節定義的 Post 納入管理，請修改 mainsite/admin.py，一開始 admin.py 的內容如下：

from django.contrib import admin

# Register your models here.

不過在此處我們並沒有這個檔案，所以先把範例放到 app 目錄底下(這裡所指的範例程式，是 Django 架站的16堂課這本書中所提供的範例，此處的範例是在 mainsite 目錄裡面)。

內容修改如下：

from django.contrib import admin

from .models import Post

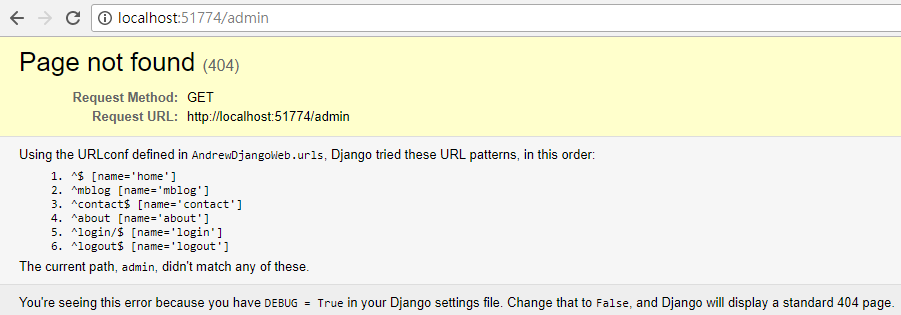
# Register your models here.

class PostAdmin(admin.ModelAdmin):

list\_display = ('title', 'slug', 'pub\_date')

admin.site.register(Post, PostAdmin)

在這裡，我們引入了之前在 models 裡面定義的 Post 類別，然後再透過 admin.site.register(Post, PostAdmin) 註冊即可。完成以上設定之後，再次啟動此網頁，透過瀏覽器連結到 <http://localhost:8000/admin>，結果如下：



問題出現了，系統看不懂的原因在於 urls 裡面並沒有定義 admin 網址，所以無法顯示，我們看一下範例程式中 urls.py 的定義：

from django.conf.urls import include, url

from django.contrib import admin

from mainsite.views import homepage, showpost

urlpatterns = [

url(r'^$', homepage),

url(r'^post/(\w+)$', showpost),

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

]

再比較我們自己的 urls.py 定義：

"""

Definition of urls for AndrewDjangoWeb.

"""

from datetime import datetime

from django.conf.urls import url

import django.contrib.auth.views

import app.forms

import app.views

# Uncomment the next lines to enable the admin:

# from django.conf.urls import include

# from django.contrib import admin

# admin.autodiscover()

urlpatterns = [

# Examples:

url(r'^$', app.views.home, name='home'),

url(r'^mblog', app.views.mblog, name='mblog'),

url(r'^contact$', app.views.contact, name='contact'),

url(r'^about', app.views.about, name='about'),

url(r'^login/$',

django.contrib.auth.views.login,

{

'template\_name': 'app/login.html',

'authentication\_form': app.forms.BootstrapAuthenticationForm,

'extra\_context':

{

'title': 'Log in',

'year': datetime.now().year,

}

},

name='login'),

url(r'^logout$',

django.contrib.auth.views.logout,

{

'next\_page': '/',

},

name='logout'),

# Uncomment the admin/doc line below to enable admin documentation:

# url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

# Uncomment the next line to enable the admin:

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

]

這裡的寫法跟範例程式有很大的不同，但是我們的目的是學習如何在 Visual Studio環境中寫程式，所以我們必須使用 Visual Studio 內帶的 admin 寫法來測試。請看這段程式碼裡面有兩個提示需要 uncomment 的說明：

# Uncomment the next lines to enable the admin:

# from django.conf.urls import include

# from django.contrib import admin

# admin.autodiscover()

我們直接改成：

# Uncomment the next lines to enable the admin:

from django.conf.urls import include

from django.contrib import admin

admin.autodiscover()

後面的也改成：

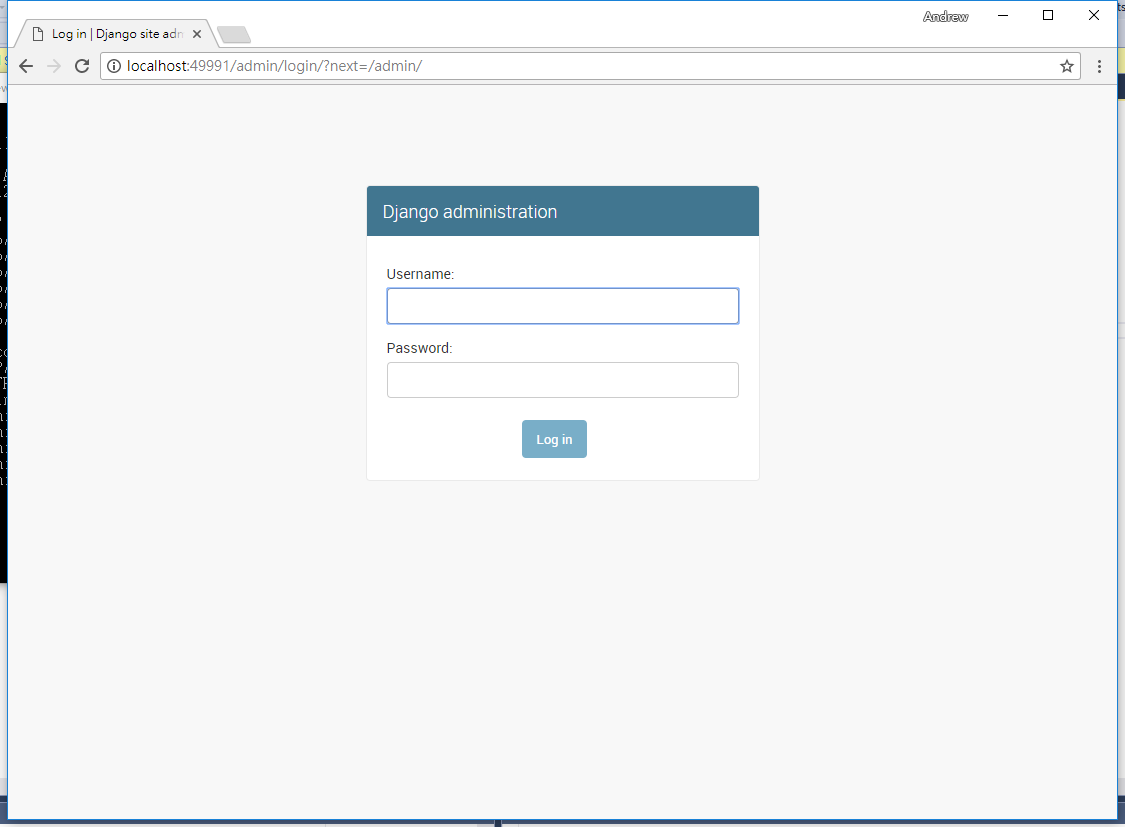
# Uncomment the admin/doc line below to enable admin documentation:

url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

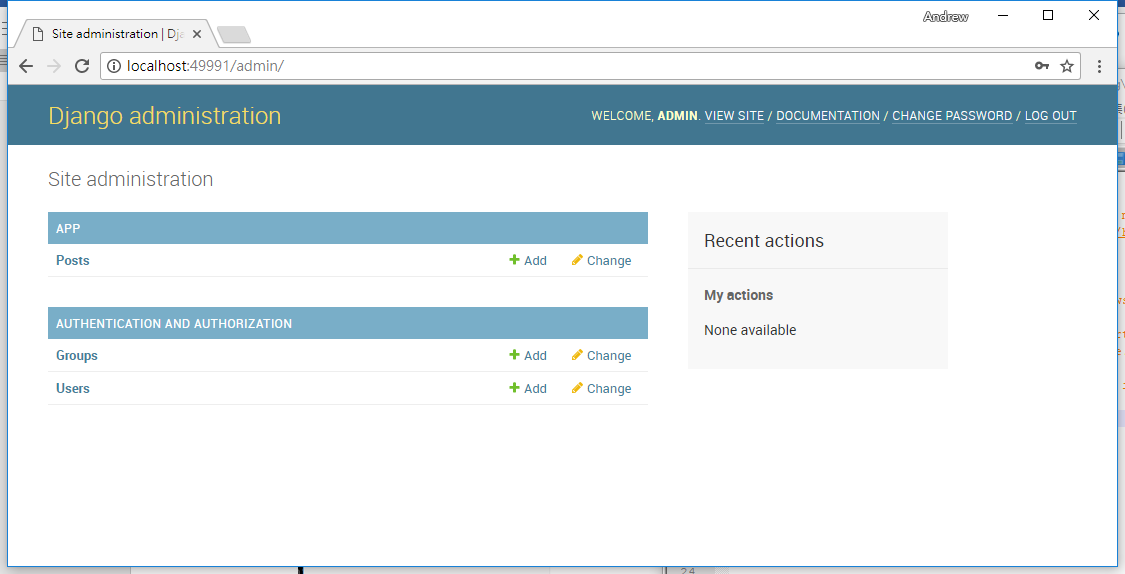
# Uncomment the next line to enable the admin:

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

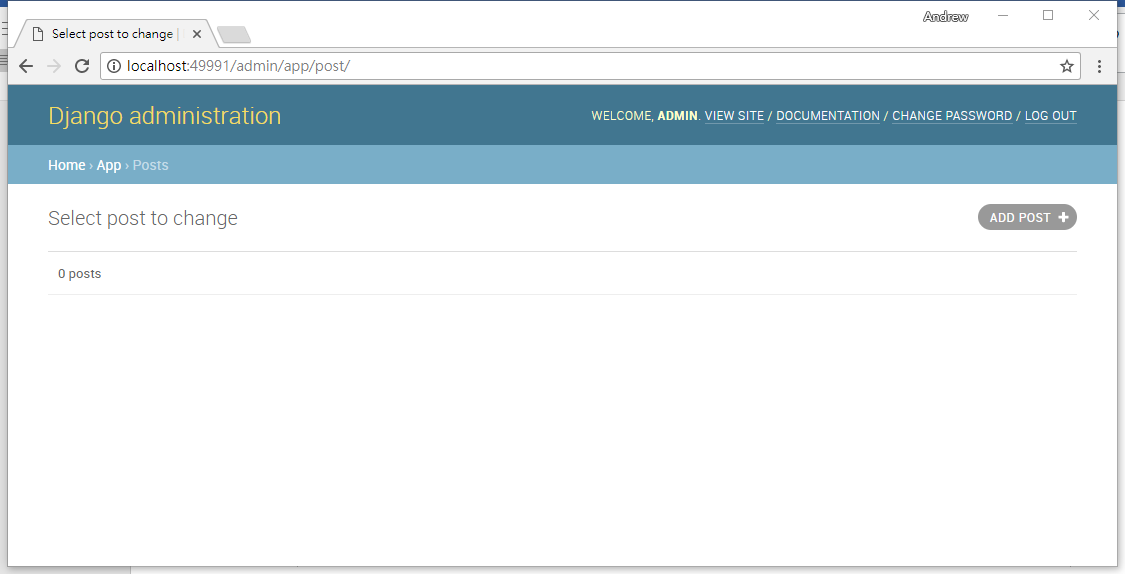
這樣該有的東西都有了，我們直接測試，完美，成功了：



輸入之前新增的 superuser 帳號以及密碼之後，即可以看到一個美觀的資料庫管理介面，順帶說明一下，到目前為止我們做的動作，跟我們設計的網頁沒有關係，而是利用 Django 的 admin 來管理自己的資料庫而已，透過 admin.py 裡面的程式，我們將自己所定義的 Posts 順利地納入管理介面了，如下圖：



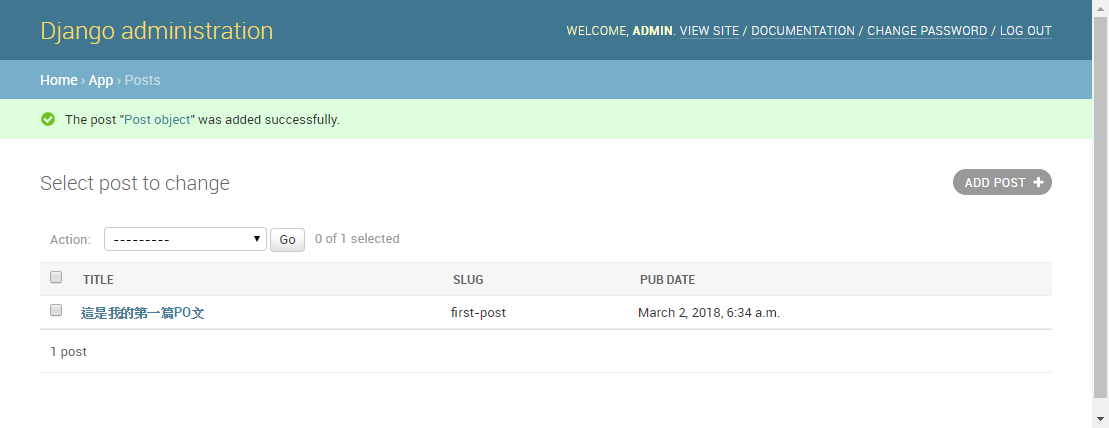
這個管理介面並不需要任何的設計，因為所有的功能都是 Django 自帶的。上方有一個 [VIEW/SITE] 按鈕，點選之後才會進入我們的網站。點擊 APP底下的 Posts，目前裡面還沒有任何內容，如下圖：



我們就來新增一筆資料吧，按下右上角的 [ADD POSTS +]：

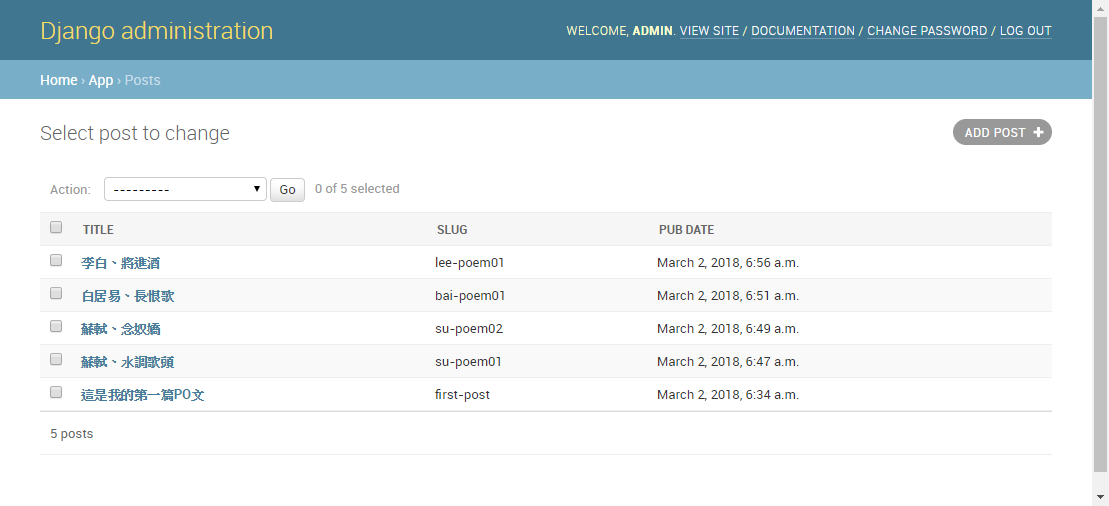


按下[SAVE]按鈕儲存。很好，第一篇PO文出現了，如下圖：



由於我們在 admin.py 裡面定義了 PostAdmin，繼承 ModelAdmin，其中又定義了 list\_display顯示三個欄位，所以在這裡我們看到的除了 title 之外，還有 slug 與 pub date 兩個欄位(我們並沒有修改時區，所以這裡的時間還是格凌威志標準時間，並不是台北時間)，要修改時區，請在 settings.py 裡面修改此行: TIME\_ZONE = 'Asia/Taipei'。

為了測試方便，請自行輸入至少五篇PO文，中英文皆可，但是 slug 請輸入英文或數字，中間不要有任何空白，如範例所示。輸入完成之後，如下圖：



最後我們做個結論，坊間有很多 Django 架設網站的書籍，但是目前我還沒有看到任何一本是使用 Visual Studio 來架站的書，這就是我為何要自己寫的原因，將 Python Command 改成使用 Visual Studio 整合環境介面、並將其他環境的範例程式改成 Visual Studio 可以執行的寫法，其實是有難度的。這裡的分享希望可以加速了解專屬於 Visual Studio 內建的 Django 的操作與寫作方式。

## 建立部落格網頁

資料庫中有了文章內容之後，接下來是要讀取這些資料，然後在網站的部落格網頁中把它們顯示出來。在做這些事情之前，先簡單說明一下，Django 的 MTV (大約可以對應到 MVC) 架構。為了把資料抽象化，Django 把資料的存取和顯示區分為 Model、Template、以及View，分別對應到 models.py、template 目錄、以及 views.py 這些檔案。

### 讀取資料庫中的內容

前一個章節所做的事情中，models.py 主要負責定義要存取的資料模型，以 Python 的 class 類別方式來定義，在後端 Django 自動會為我們把這個類別中的設定對應到資料庫系統中，不管是使用哪種資料庫。

如果要把這些資料拿出來，或是要如何進去的這些程式邏輯，則是在 views.py 中處理，而這也是我們在這個章節要寫程式的地方。至於如何把這些取得的資料用美觀有彈性的方式輸出，則是在 template 目錄中加以處理。

在此，先開啟 views.py，一開始的預設內容應該只有一行，如下：

from django.shortcuts import render

不過由於我們的網站架構是來自於 Visual Studio 的模板，再加上我們在之前的章節中已經加入了一個部落格(mblog)所以內容會如下：

"""

Definition of views.

"""

from django.shortcuts import render

from django.http import HttpRequest

from django.template import RequestContext

from datetime import datetime

def home(request):

"""Renders the home page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/index.html',

{

'title':'Home Page',

'year':datetime.now().year,

}

)

def mblog(request):

"""Renders the mblog page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/mblog.html',

{

'title':'我的部落格',

'message':'管理者可以在此分享經驗.',

'year':datetime.now().year,

}

)

def contact(request):

"""Renders the contact page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/contact.html',

{

'title':'Contact',

'message':'Your contact page.',

'year':datetime.now().year,

}

)

def about(request):

"""Renders the about page."""

assert isinstance(request, HttpRequest)

return render(

request,

'app/about.html',

{

'title':'About',

'message':'Your application description page.',

'year':datetime.now().year,

}

)

其中 mblog 只是簡單的顯示一個 mblog.html 網頁而已，並沒有任何內容，我們就修改我們自己的 views.py 程式碼，首先，程式前面加上 from .models import Post，如下：

from django.shortcuts import render

from django.http import HttpRequest, HttpResponse

from django.template import RequestContext

from datetime import datetime

from .models import Post

然後修改 def mblog(request): 的內容如下：

def mblog(request):

posts = Post.objects.all()

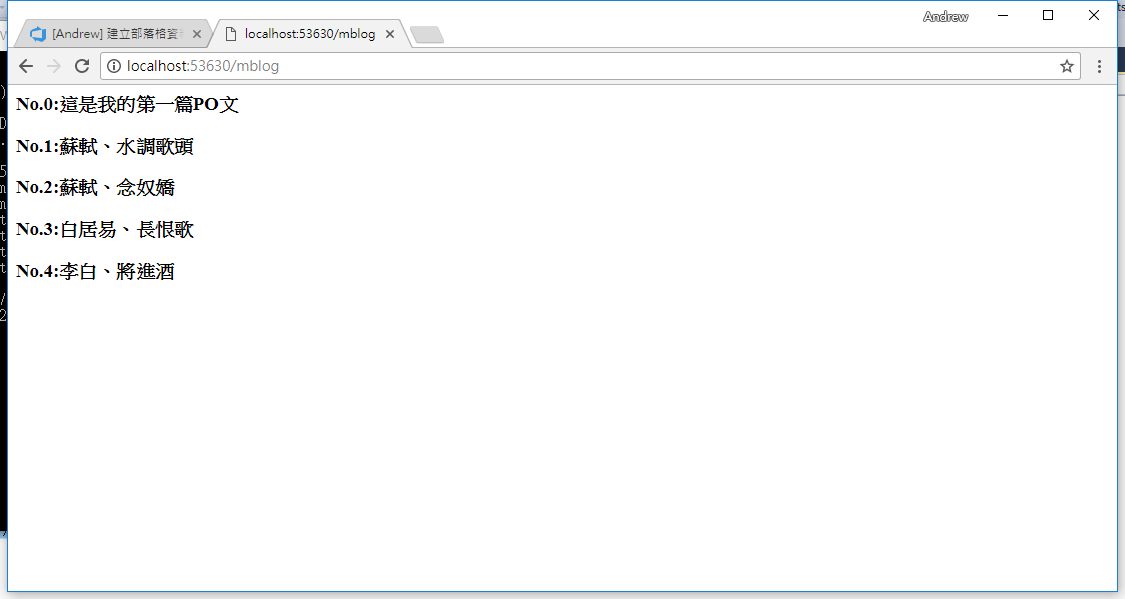
post\_lists = list()

for count, post in enumerate(posts):

post\_lists.append("<h3>No.{}:".format(str(count)) + str(post.title)+"</h3>")

return HttpResponse(post\_lists)

執行之後，在首頁的主選單中點選[部落格]，便會執行 mblog，結果如下：



原本漂亮的 header 和 footer 都不見了，沒關係，我們後面再慢慢往上加。如果我們想一下子看到所有內容，可修改程式為：

def mblog(request):

posts = Post.objects.all()

post\_lists = list()

for count, post in enumerate(posts):

post\_lists.append("<h3>No.{}:".format(str(count)) + str(post.title) + "</h3>")

post\_lists.append("<small>" + str(post.body) + "</small><hr>")

return HttpResponse(post\_lists)

執行結果如下：



依此類推，我們就可以在 mblog 中編寫程式，讓網頁的白板更加美麗。不過，這並不是明智的做法，因為顯示的樣式，和如何存取資料內容，這兩者必須分開，比較容易維護，而且在大型的合作專案中，這兩個部份的工作通常是由不同的人員負責的，放在一起會造成開發上的困難。

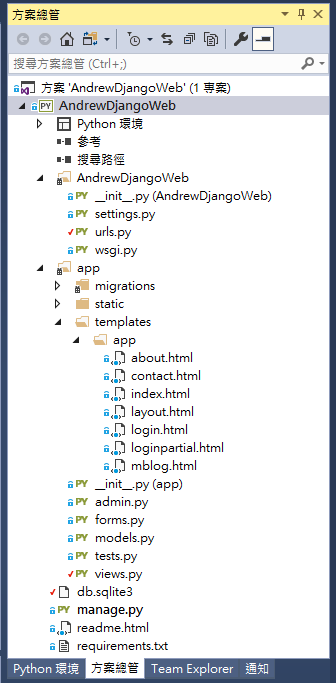
因此，正確的做法是，在 views.py 中把資料準備好，然後丟到 template 中，讓 template 中的 .html 檔案來負責真正的顯示工作，我們接下來就會介紹。

### 建立網頁輸出模板 template

前一個章節中我們示範了如何建立資料模型、在 admin 介面中輸入及編輯資料，而在本章節一開始，則是介紹如何使用 Post.objects.all() 取出所有的資料並透過 HttpResponse 輸出到瀏覽器。那如何把從資料庫中拿到的資料，經過排版，以比較美觀的方式呈現呢？就是利用 template。每一個輸出的網頁都可以準備一個或一個以上對應的模板，而這模板是以 .html 的檔案形式儲存在指定的資料夾中，一般都會命名為 templates，當網站有資料需要輸出的時候，在透過 render 函數把資料放到模板指定的位置中，得到結果後再交給 HttpResponse 輸出給瀏覽器。基本的步驟如下：

1. 在 settings.py 中設定模板資料夾的位置
2. 在 urls.py 建立網址和 views.py 中的函數對應關係
3. 建立 .html 檔案，做好排版並安裝資料要放置的位置
4. 執行程式，以 objects.all() 在 views.py 取得資料，並放入變數中
5. 以 render 函數，把資料送到指定的模板檔案中

在開始這些步驟之前，我們再來更深入的了解 Visual Studio 建立的 Django Python 專案的基本架構，請參考下圖：



首先我們在Visual Studio 中新增的是方案(Solution)，而不是專案(Project)，一個方案底下，可以有多個專案，這是 Visual Studio 的概念，要先弄清楚，所以最上層是一個名為 AndrewDjangoWeb 的方案名稱，一般而言，在此方案名稱底下，則是一個同名的 AndrewDjangoWeb 的專案根目錄。在專案根目錄中，則放了資料庫檔案與 manage.py 兩個檔案以及兩個子目錄。

這兩個子目錄分別是與專案同名的 AndrewDjangoWeb子目錄與 app 子目錄，AndrewDjangoWeb子目錄存放著與此專案相關的重要設定程式，前面所提到的 settings.py 與 urls.py 就在這裡。

而 app 子目錄(預設的名稱)中則是存放這個 Web Application 專案(當初新增時選的) 的 MTV 結構，所以，models.py、views.py、與 templates 目錄都存在於此。另外，前一張所談到的 admin.py 我們也會放在這個目錄底下，還有 forms.py 和 tests.py 也在這裡(尚未使用到)。

再打開 templates 目錄，裡面還有一個 app 子目錄，這是代表在這裡存放的 .html 是屬於 app 這個 Web Application 所有，也是內定的，這是 Visual Studio 的習慣，很多 Django 書籍可能會將 .html 直接放在 templates 目錄裡面，少了 app 這層，也是沒有問題的。

最後，跟 templates 同層還有一個重要的 static 目錄， 裡面又有 app 與 images 兩個子目錄，其中 app 裡面放的都是 jQuery 模組，千萬不要去更動它，而 images 目錄則可以放置網頁中所使用到的圖檔。

弄清楚這些目錄的關係之後，我們就可以來看看 settings.py 中如何設定 templates 的位置，首先開啟 settings.py，templates 的相關內容如下：

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [],

'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',

],

},

},

]

主要修改的地方是 DIRS 那行，由於 Visual Studio 在新增專案的時候已經將 templates 加入 BASE\_DIR 了，因此在這裡我們不需要做任何動作。

但是，如果你是自己從無到有編寫，就必須修改下面這行：

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

整個網站最外層的框架，是 layout.html，其部分內容如下：

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="{% url 'home' %}">首頁</a></li>

<li><a href="{% url 'mblog' %}">部落格</a></li>

<li><a href="{% url 'about' %}">關於</a></li>

<li><a href="{% url 'contact' %}">聯絡資訊</a></li>

</ul>

{% include 'app/loginpartial.html' %}

</div>

當我們在首頁按下 [部落格] 選單之後，就會去找 “mblog” 這個 url，再看 urls.py 的結構如下：

urlpatterns = [

# Examples:

url(r'^$', app.views.home, name='home'),

url(r'^mblog', app.views.mblog, name='mblog'),

url(r'^contact$', app.views.contact, name='contact'),

url(r'^about', app.views.about, name='about'),

這時候，就會去執行 app.views.mblog ，顯而易見的，mblog 是在 views.py 裡面定義的：

def mblog(request):

posts = Post.objects.all()

post\_lists = list()

for count, post in enumerate(posts):

post\_lists.append("<h3>No.{}:".format(str(count)) + str(post.title) + "</h3>")

post\_lists.append("<small>" + str(post.body) + "</small><hr>")

return HttpResponse(post\_lists)

這是原本我們在前一個小傑裡面寫的程式，先做要做一些修改，因為我們希望進入部落格網頁之後，不要直接看到所有的內容，而是先看到所有PO文的標題。所以程式碼要修改一下，首先加入下面的 import，並修改程式如下：

from django.template.loader import get\_template

def mblog(request):

template = get\_template('app/mblog.html')

posts = Post.objects.all()

now = datetime.now()

html = template.render(locals())

return HttpResponse(html)

上面的程式我們用了一個小技巧把變數丟到模板，就是使用 locals() 這個函數。這個函數會把目前記憶體中所有區域變數使用字典型態打包起來，在模塊中因未接受到了所有的區域變數，當然可以把 posts和 now都拿來使用。請注意，這裡包含了新增與修改，程式碼其他的部分則不要更動。最後，我們就來修改 mblog.html並在裡面使用區域變數，程式碼如下：

{% extends "app/layout.html" %}

{% block content %}

<h1>這是我的部落格</h1>

<hr />

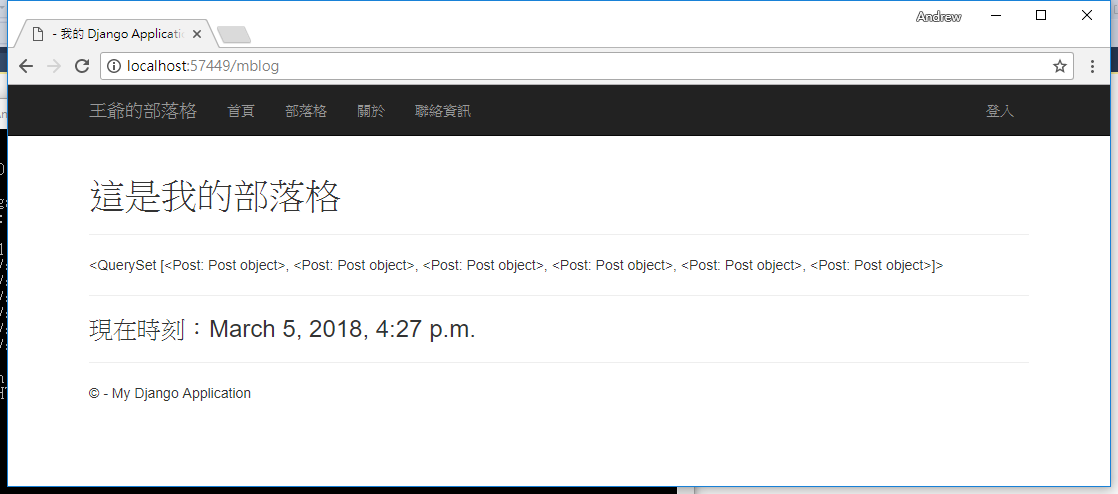
**{{** posts }}

<hr />

<h3>現在時刻：**{{** now }}</h3>

{% endblock %}

程式碼中我們以 {% … %} 呼叫 Python 語法，這種做法跟 JavaScript 很像，我們可以把它當成 PythonScript，其中使用了 extends 把 layout.html 包進來，另外用 block包裝網頁的內容({% block content %}是作用在一個區塊，必定有{% endblock %}作為結束)，在內容中我們使用 {{ … }} 呼叫區域變數，執行結果如下：



這裡看到的內容是整個 Post object，並無法看到內容，如果要看到內容的話，我們可以在 html 裡面同樣以 {% … %} 使用 Python 的 for 迴圈來完成，還記得我們之後是把 for 迴圈寫在 views.py 裡面吧，現在要寫在 mblog.html 裡面。為何要這樣做？就是為了設計上符合 MTV 的設計架構。程式碼如下：

{% extends "app/layout.html" %}

{% block content %}

<h1>這是我的部落格</h1>

<hr />

{% for post in posts %}

<p style='font-family:微軟正黑體;font-size:16pt;font-weight:bold;'>

**{{** post.title }}

</p>

<p style='font-family:微軟正黑體;font-size:12pt;letter-spacing:1pt;'>

**{{** post.body }}

</p>

<p>

發佈時間：**{{** post.pub\_date | date:"Y M d, h:m:s"}}

</p>

<hr>

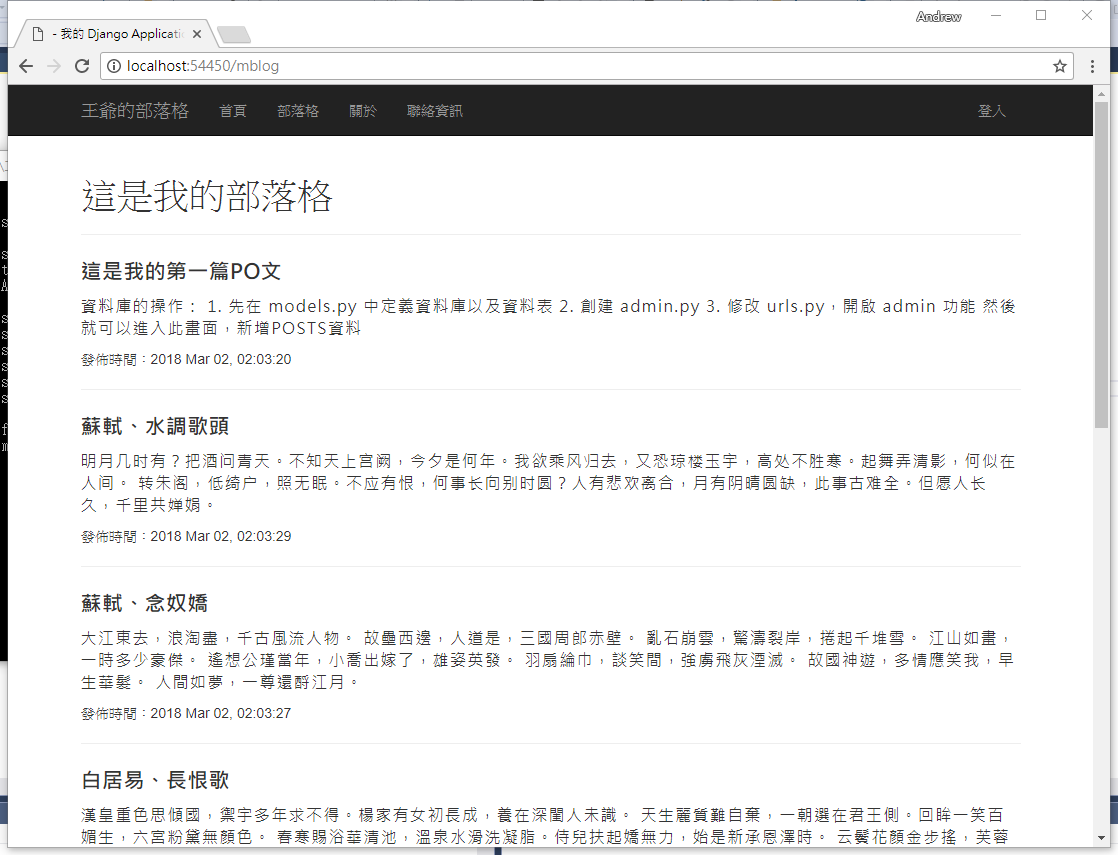
{% endfor %}

<hr />

<h3>現在時刻：**{{** now }}</h3>

{% endblock %}

結果如下：



### 兩層式的瀏覽網頁

對於上面的顯示結果，當PO文很多而且文章內容也很多的時候，這個網頁就會變得非常龐大，一般來說，部落格的首頁是不會把所有的內容都顯示出來的，而是當瀏覽者點擊連結的時候才會開啟另一個頁面顯示出該篇文章。因此，我們接著把 mblog.html 進一步修改如下：

{% extends "app/layout.html" %}

{% block content %}

<h1>這是我的部落格</h1>

<hr />

{% for post in posts %}

<p style='font-family:微軟正黑體;font-size:16pt;font-weight:bold;'>

<a href='/post/**{{**post.slug}}'>**{{** post.title }}</a>

</p>

<p>

發佈時間：**{{** post.pub\_date | date:"Y M d, h:m:s"}}

</p>

<br>

{% endfor %}

<hr />

<h3>現在時刻：**{{** now }}</h3>

{% endblock %}

透過 <a href> 這個 html 標籤，取出 post.slug 鏈結網址，並放在 post/ 的後面，執行結果如下圖：



這個程式還沒有寫完，請細心觀察左下角，出現 …/post/first-post 字樣，我們會利用這個值，來查詢得到文章的內容。接下來，我們就要討論如何使用另外一個網頁來顯示單篇文章的內容。

### 網址對應 urls.py

從前面的例子中，我們發現，/post/ 是我們加在 mblog.html 中為了顯示單篇文章的前置詞，而後面的 first-post 則是在建立文章時，我們自行設定的自訂網址。也就是說，要辨識出這些網址以對應到要顯示的單篇文章，應該有下面幾個步驟：

1. 在 urls.py 設定，只要是 /post/ 開頭的網址，就把後面接著的文字當作是參數傳遞 slug 給 post\_detail 這個顯示單篇文章的函數
2. 在 views.py 中新增一個 post\_detail 函數，除了接收 request 參數之外，亦接收 slug 這個參數
3. 在 templates/app 資料夾中建立一個用來顯示單篇文章用的 post.html
4. 在 post\_detail 函數中，以 slug 為關鍵字，搜尋資料庫，找出是否有符合的項目
5. 如果有符合，則把找到的資料項目傳遞給 render 函數，找出 post.html 這個模板頁面出來渲染，再把結果交給 HttpResponse 回傳給瀏覽器
6. 如果沒有符合的項目，再把網頁轉回部落格首頁

這些步驟就是在 Django 中新增一個網頁的標準步驟，其實我們在前面的章節已經有提過，當初 mblog.html 就是這樣新增出來的。這裡只是多了變數的操作而已。

請開啟 urls.py，在網址對應方面，需做如下的修改：

urlpatterns = [

# Examples:

url(r'^$', app.views.home, name='home'),

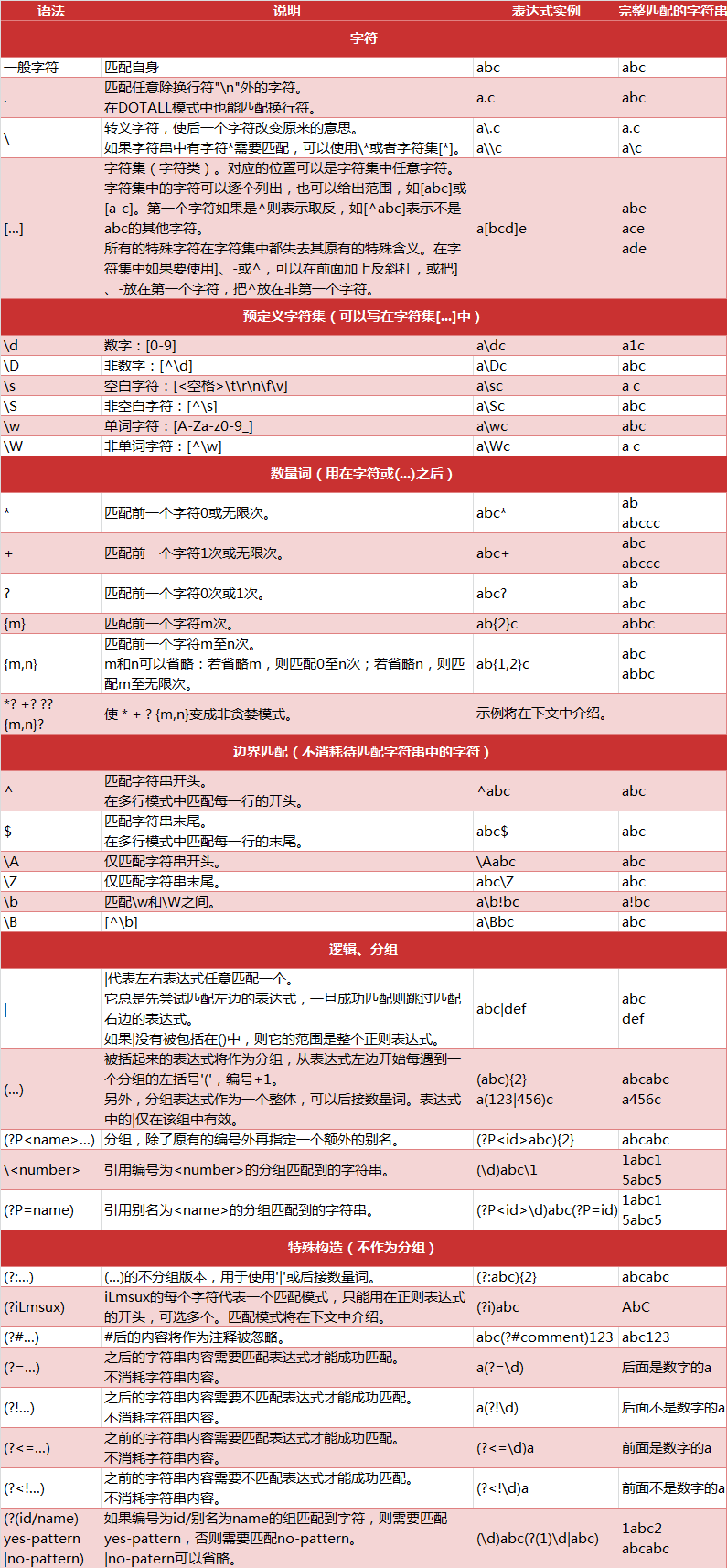
url(r'^mblog', app.views.mblog, name='mblog'),

# 增加單獨顯示文章網頁

url(r'^post/(\w+)$', app.views.showpost, name='showpost'),

其中 (\w+) 是 Python 的正則表達式的語法， “\” 是轉譯字符，使後面的字符改變原來的意思，也就是說w不再是字母w，而變成 [A-Za-z0-9\_] 大小寫英文字母、數字、以及底線， “+” 則代表可以出現無限次 [A-Za-z0-9\_]。(\w+) 會被當成第二個參數傳遞給後面的 showpost 函數(第一個參數是預設的 request)。

下圖是正則表達式語法的說明：



接著請開啟 views.py，我們要建立 showpost 這個函數來處理接收到的參數，如下：

from django.shortcuts import render, redirect

… 省略 …

def showpost(request, slug):

template = get\_template('app/post.html')

try:

post = Post.objects.get(slug=slug)

if post != None:

html = template.render(locals())

return HttpResponse(html)

except:

return redirect('app/mblog.html')

最後，建議用複製貼上再重新命名的方式，建立一個 post.html，內容如下：

{% extends "app/layout.html" %}

{% block content %}

<h1> **{{** post.title }} </h1>

<hr />

<p style='font-family:微軟正黑體;font-size:12pt;letter-spacing:2pt;'>

**{{** post.body }}

</p>

<p>

發佈時間：**{{** post.pub\_date | date:"Y M d, h:m:s"}}

</p>

<hr />

<h3><a href='/mblog.html'>回首頁</a></h3>

{% endblock %}

PS: 因為我們用了 /w+ 這個正則表達式，要得到正確的結果，請使用 admin 工具，將之前輸入的文章中 slug 欄位所有的 “-” 都改成 “\_”，修改方式如下：



執行結果如下：



回首頁：



### 共用模板的使用

幾乎所有的商業網站在每一樣都會有一些共同的元素以強調網站的風格，這是我們會把網頁共同的部分獨立出來做成另外一個檔案，來達成這樣的需求。

Visual Studio 使用了 layout.html 來做為主要的基礎模板，我們來看一下內容：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>**{{** title }} - 我的 Django Application</title>

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'app/content/bootstrap.min.css' %}" />

<link rel="stylesheet" type="text/css" href="{% static 'app/content/site.css' %}" />

<script src="{% static 'app/scripts/modernizr-2.6.2.js' %}"></script>

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a href="/" class="navbar-brand">王爺的部落格</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="{% url 'home' %}">首頁</a></li>

<li><a href="{% url 'mblog' %}">部落格</a></li>

<li><a href="{% url 'about' %}">關於</a></li>

<li><a href="{% url 'contact' %}">聯絡資訊</a></li>

</ul>

{% include 'app/loginpartial.html' %}

</div>

</div>

</div>

<div class="container body-content">

{% block content %}{% endblock %}

<hr/>

<footer>

<p>&copy; **{{** year }} - My Django Application</p>

</footer>

</div>

<script src="{% static 'app/scripts/jquery-1.10.2.js' %}"></script>

<script src="{% static 'app/scripts/bootstrap.js' %}"></script>

<script src="{% static 'app/scripts/respond.js' %}"></script>

{% block scripts %}{% endblock %}

</body>

</html>

在 layout.html 基礎模板中，包含了完整的 html 網頁寫法，包含了 head, body 等，這裡最重要的設計方式，就是區塊，就是下列兩行：

{% block content %}{% endblock %}

{% block scripts %}{% endblock %}

這裡有兩個完全沒有內容的 block，分別是 content 與 script ，這是讓未來所有繼承 layout.html 的 html 檔案，自行 implement 此 block。

所以接下來，我們來看 mblog.html 如何繼承 layout.html，請參考程式碼：

{% extends "app/layout.html" %}

{% block content %}

<h1>歡迎光臨我的部落格</h1>

<hr />

{% for post in posts %}

<p style='font-family:微軟正黑體;font-size:16pt;font-weight:bold;'>

<a href='/post/**{{**post.slug}}'>**{{** post.title }}</a>

</p>

<p>

發佈時間：**{{** post.pub\_date | date:"Y M d, h:m:s"}}

</p>

<br>

{% endfor %}

<hr />

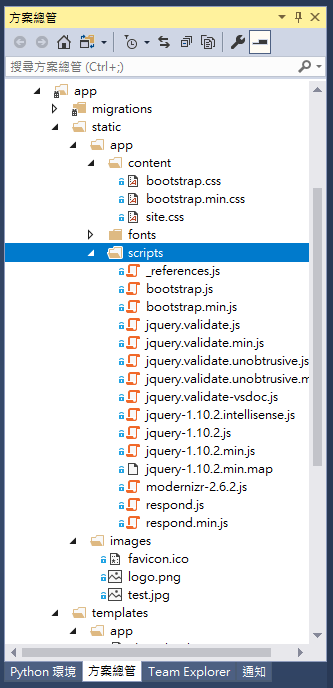
<h3>現在時刻：**{{** now }}</h3>

{% endblock %}

其中 {% extends "app/layout.html" %} 就是繼承的語法，後面一定會出現 {% block content %}{% endblock %} ，只不過這裡面就必須有實做的內容了。

### 圖形檔的應用

接下來，如何在網站中使用圖形檔或是其他如 .css 或是 .js 的檔案呢? 檔案中的圖形檔，大部分會被放置在 images 資料夾底下，.css 檔案會放在 app/content 底下，而 .js 檔案會放在 app/scripts 裡面。如下圖：



上述所有的檔案，對於 Django 而言，都是屬於不需要被另外處理的靜態檔案，所以它們都被放置在 static 這個目錄底下。而利用 Visual Studio Django 樣板產生的專案，都已經具備了這些目錄以及會使用到的檔案，我們不太需要去動它，只有圖形檔的部份我們可以用來美化我們的網頁。

Django 如何知道這些靜態檔案的位置，首先要看 settings.py：

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/1.9/howto/static-files/

STATIC\_URL = '/static/'

STATIC\_ROOT = posixpath.join(\*(BASE\_DIR.split(os.path.sep) + ['static']))

其實我們無須修改，基本上 static 資料夾跟 templates 資料夾是平行的。定義好位置之後，Django 會在 layout.html 裡面使用 {% load staticfiles %} 將其載入，在同一個檔案中它之需要出現一次，所以在不同的檔案中，如果使用到 images ，都必須寫這行，請參考 layout.html 部分程式碼如下：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>**{{** title }} - 我的 Django Application</title>

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'app/content/bootstrap.min.css' %}" />

<link rel="stylesheet" type="text/css" href="{% static 'app/content/site.css' %}" />

<script src="{% static 'app/scripts/modernizr-2.6.2.js' %}"></script>

</head>

接著，我們將 logo.png 檔案放到 images 目錄底下，我們就可以在 post.html 檔案中引用它了。

{% extends "app/layout.html" %}

{% block content %}

{% load staticfiles %}

<img src="{% static "images/logo.png" %}" />

<h1> **{{** post.title }} </h1>

<hr />

<p style='font-family:微軟正黑體;font-size:12pt;letter-spacing:2pt;'>

**{{** post.body }}

</p>

<p>

發佈時間：**{{** post.pub\_date | date:"Y M d, h:m:s"}}

</p>

<hr />

<h3><a href='/mblog.html'>回首頁</a></h3>

{% endblock %}

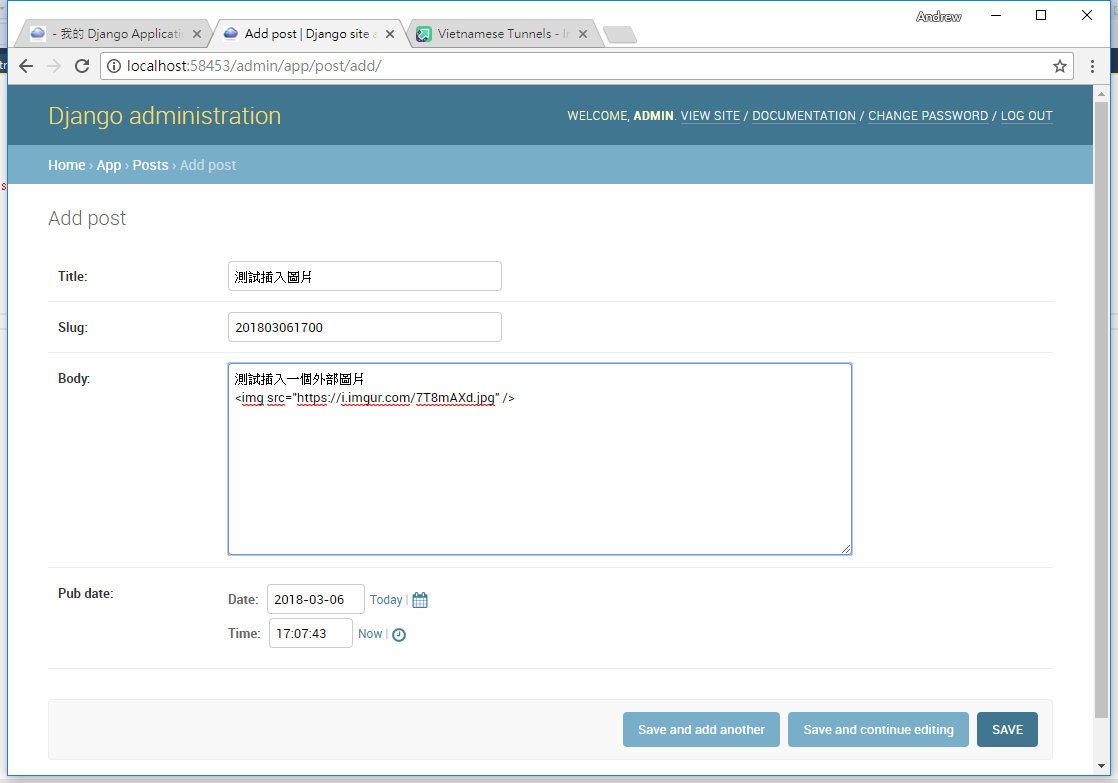
執行結果如下，左上角多了一個文學天地：



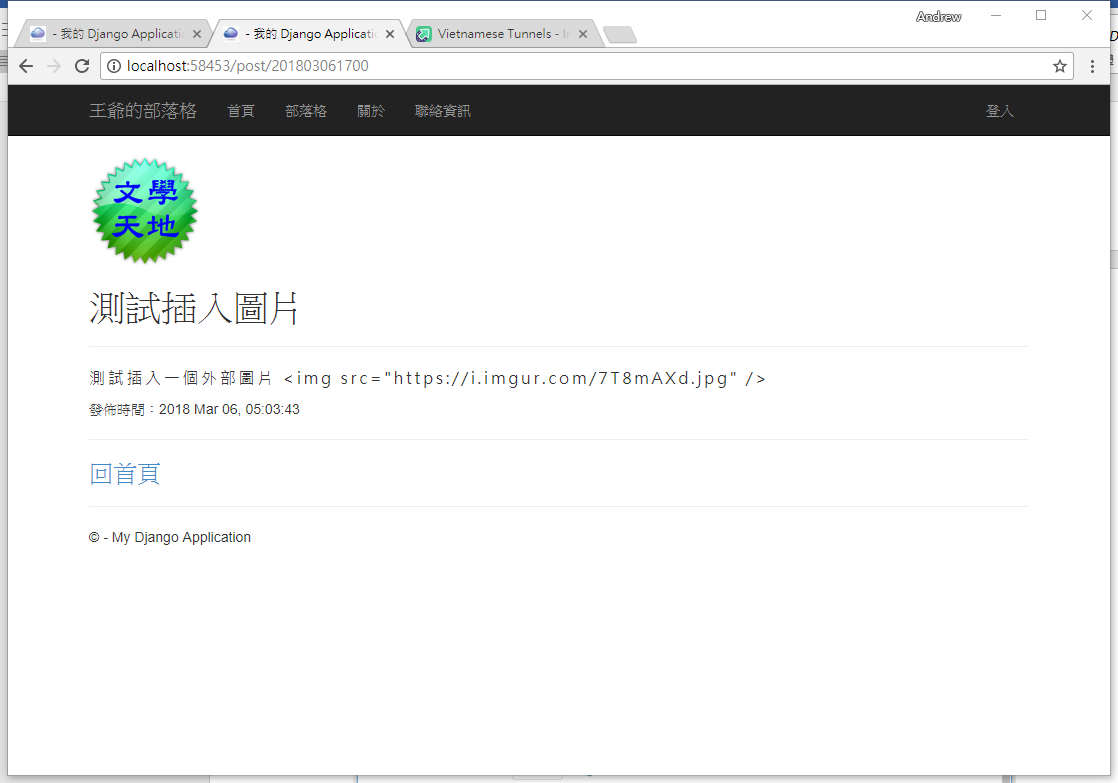
### 部落格文章的 HTML 內容處理

我們都知道在網頁之中可以利用 <img …> 來插入圖片，如果我們將這個 html element 加入我們的部落格本文之中會如何呢?

馬上進入 admin 做測試。直接在網頁上面寫 html 語法，如下圖：



存檔後，按下 [VIEW SITE] 進入部落格，結果如下：



這裡只看得見原始 html 程式碼，主要是因為 Django 在預設的情況下，是不隨便解讀 html 碼的，是擔心網路安全性的問題。但由於這是我們自己的部落格網站，並不開放其他人新增資料，所以可以讓圖片顯示出來，只要在 post.html 裡面修改一下 **{{** post.body | safe }} 即可，如下圖：

{% extends "app/layout.html" %}

{% block content %}

{% load staticfiles %}

<img src="{% static "images/logo.png" %}" />

<h1> **{{** post.title }} </h1>

<hr />

<p style='font-family:微軟正黑體;font-size:12pt;letter-spacing:2pt;'>

**{{** post.body | safe }}

</p>

<p>

發佈時間：**{{** post.pub\_date | date:"Y M d, h:m:s"}}

</p>

<hr />

<h3><a href='/mblog.html'>回首頁</a></h3>

{% endblock %}

結果如下：



## 發行網站

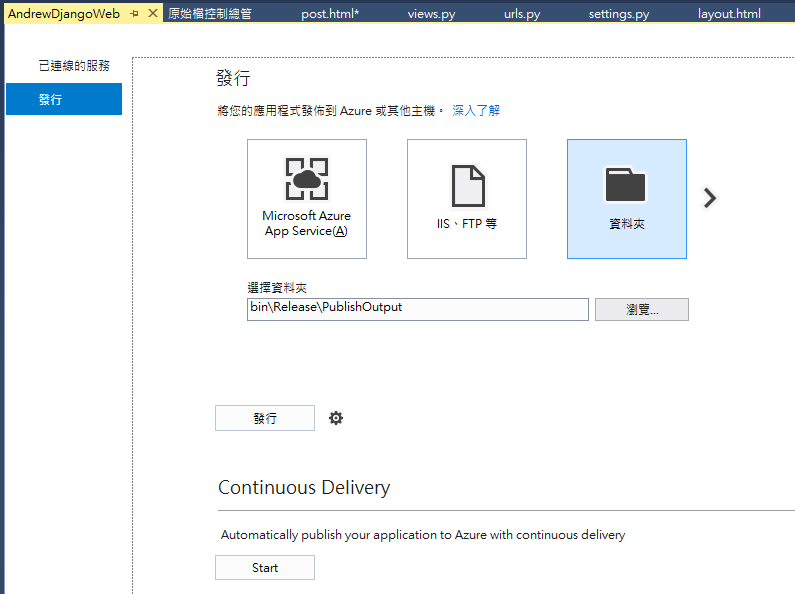
在 Visual Studio 中按下執行按鈕，是以動態的方式執行，並不是在網站中執行，如果要在網站中執行，無論是 localhost 還是公有雲虛擬伺服器，都必須透過發行功能來佈署。

### 在 Visual Studio 中的部署概觀

透過部署(或者稱為發行，每個版本的翻譯都不同)應用程式、服務或元件，就可以將它散發到其他電腦、裝置、伺服器或雲端上進行安裝。 請在 Visual Studio 中針對您需要的部署類型選擇適當的方法。

我們先從最簡單的方式談起，就是將應用程式佈署到本機的 IIS，如果我們是用 C# 編寫的網頁，用這種方式馬上就可以執行了，我們來看一下用 Python Django 來做會遇到那些情況，我會依照遇到的情況一個一個說明並提供解決方式。

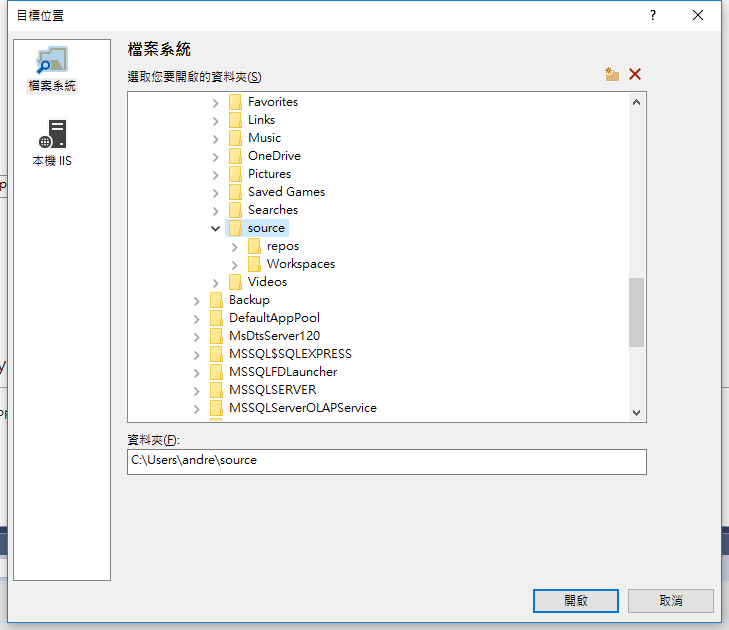
在方案總管中，以滑鼠右鍵按一下專案，然後選擇 [發行]。則出現下圖：



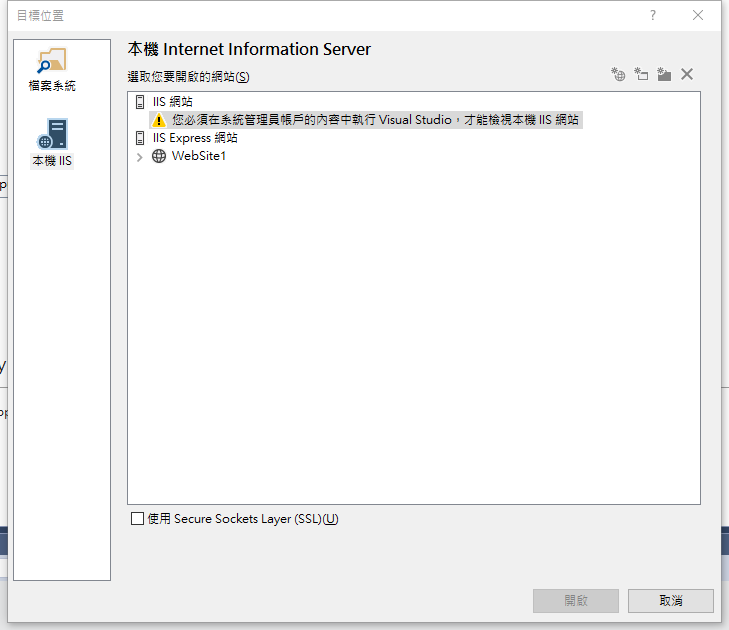
光是這裡，很多第一次使用發行功能的人就會開始搞不清我了，在這裡有很多種發行的方式，按下 “>” 按鈕，還可以看到其他的方式，如下圖：



想要發行到 localhost 的 IIS，很多人會選錯，並不是選擇 “IIS、FTP等” 這個選項，那是要發行到公司伺服器、Google、AWS 或阿里雲等雲端服務的IIS伺服器上面用的，一開始我們先發行到自己電腦上面的IIS上面，所以我們必須選擇 “資料夾”，選取之後，再按下下面的 “瀏覽” 按鈕，就會出現下圖：



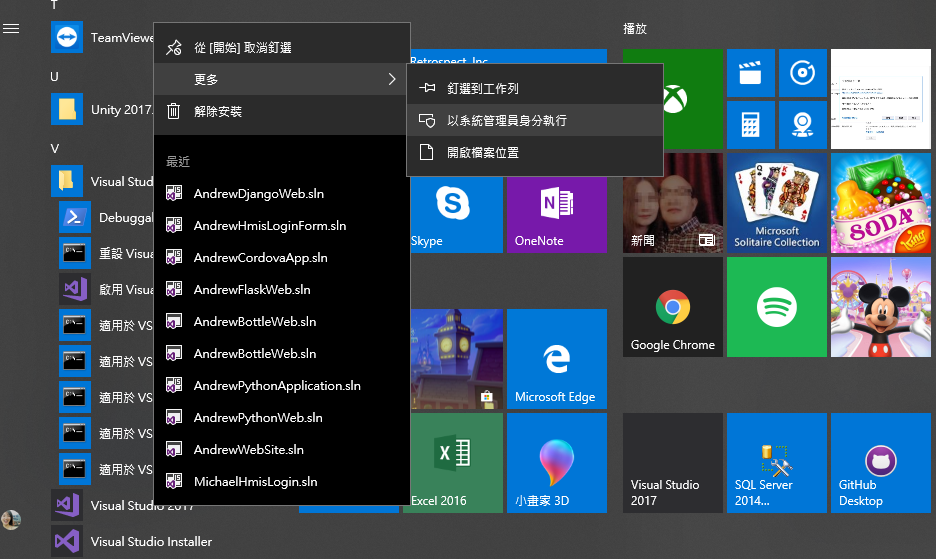
這裡很重要，請特別注意左邊會出現兩個選項，很多第一次使用的人會選成檔案系統，那就錯了，請選取 “本機 IIS”，就會看見下面的錯誤訊息：



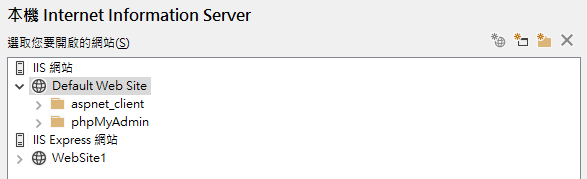
我故意保留所有會遇到的錯誤或警告訊息，就如同我也是第一次使用一樣，這樣可以幫助大家在遇到同樣問題的時候不會那麼緊張，順便也可以練習解決問題的能力。

這是權限的問題，代表使用者並沒有以管理者的身分來開啟Visual Studio工具，接下來會用一種標準做法，務必記住此做法，因為很多地方都會用到。

請退出 Visual Studio，然後從開始選取 Visual Studio 按下滑鼠右鍵選擇 “以系統管理員身分執行”，如下圖：



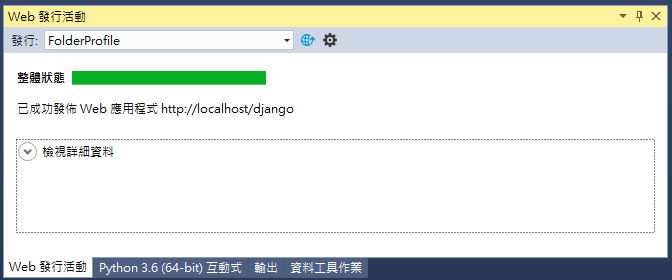
接著再來一次，就可以看到本機的 IIS 網站了，如下圖：



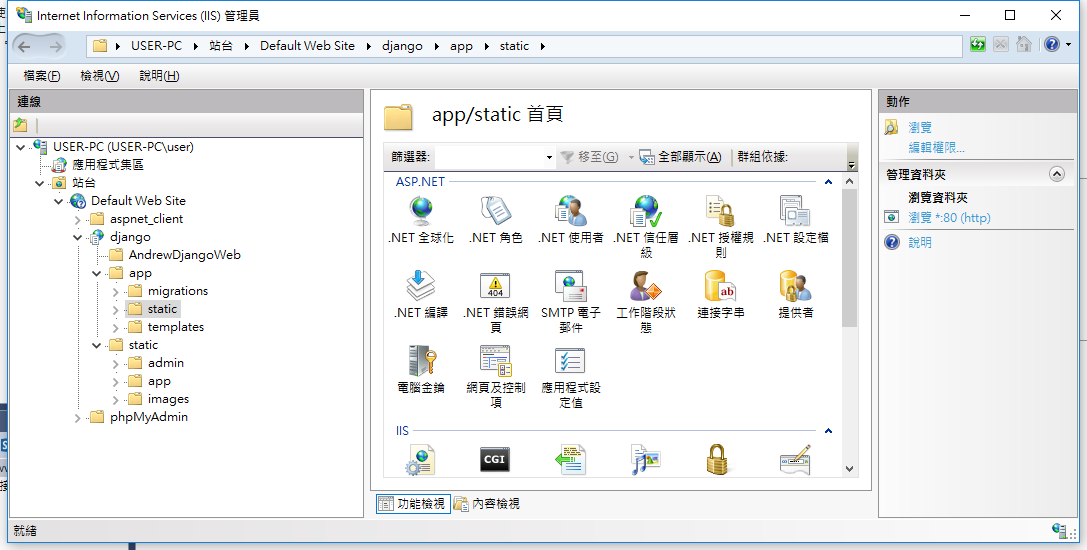
先注意右上角有三個小按鈕，分別是新增網站、應用程式、與虛擬目錄。首先，我不開網站，因為我並不想把我的網路應用程式放在根目錄，因為如果放在根目錄，就會跟 Default Web Site 衝突，必須改變 port 才能執行。當然，也不是新增一個虛擬目錄，通常這是在整個網站要擴充空間，原本的硬碟放不下的時候使用。因此，我就使用一般我們用 C# 開發網頁時最常用的方式，開一個在 Default Web Site 底下的應用程式。所以從右上角選取 [建立新 Web 應用程式]，取名 django，如下圖 (後來我發現這種方式在 Django 上面會有問題，讓我們繼續看下去)：



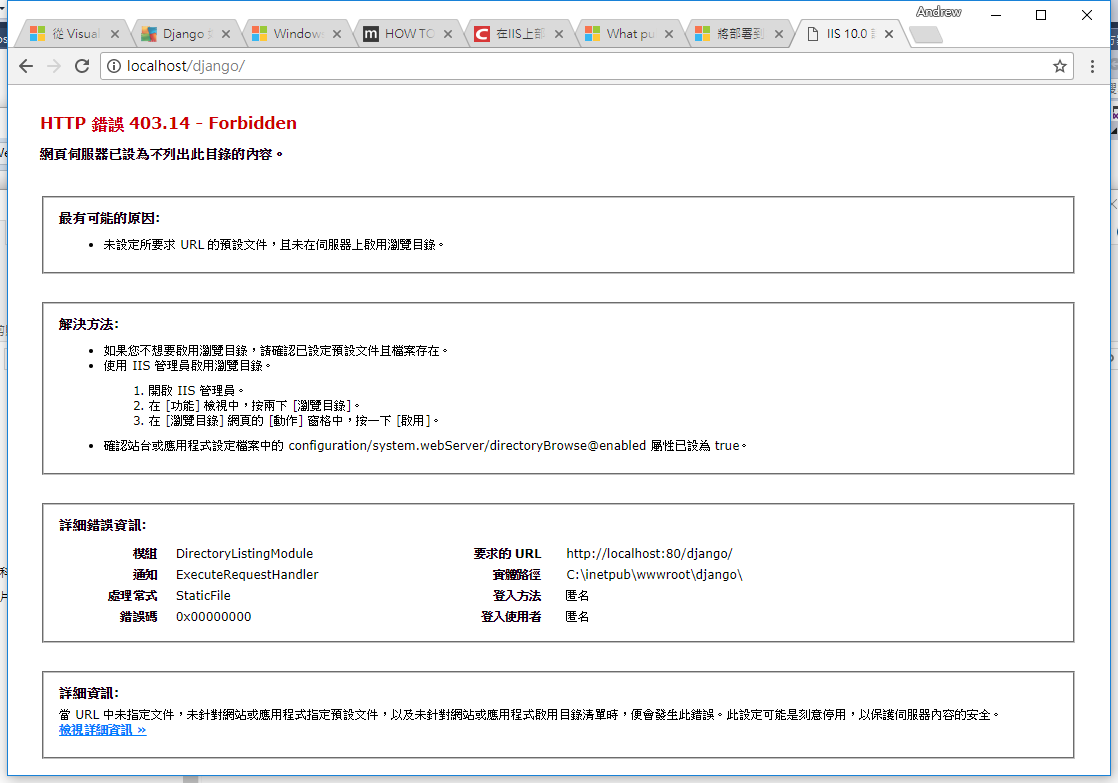
按下 [開啟]，回到上層畫面，接著就可以按下 [發行]，很快的，我們就可以在下面的視窗看到 Web 發行活動的結果，如下圖：



打開 IIS管理員，可以看到網站的架構，如下圖：



前面有說過，如果我們發行的是一個 C# Web Application，到這裡通常就成功了。不過我們現在是用 Python Django，當我們執行的時候，就會出現問題，如下圖：



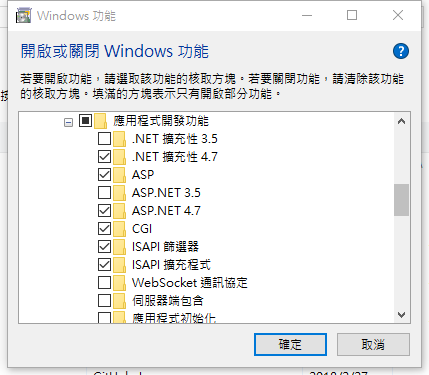
這個錯誤是一定會遇到的，因為IIS 的標準的執行環境是 .NET Framework，Python 必須進一步的處理才能執行。

如果那麼麻煩，為何還要使用 Python 來建網站，用 C# 不就行了？主要是因為，伴隨著人工智慧和深度學習的火熱發展，Python也成為了一門非常熱門的語言。我們可以看到熱門的深度學習框架都提供Python的介面，有些甚至只有Python的介面，這一定程度上推動了Python的普及。當然，在我們完成模型的訓練之後，我們總是要搭建一個演示的平臺，要麼開放給用戶使用，要麼是內部進行測試。

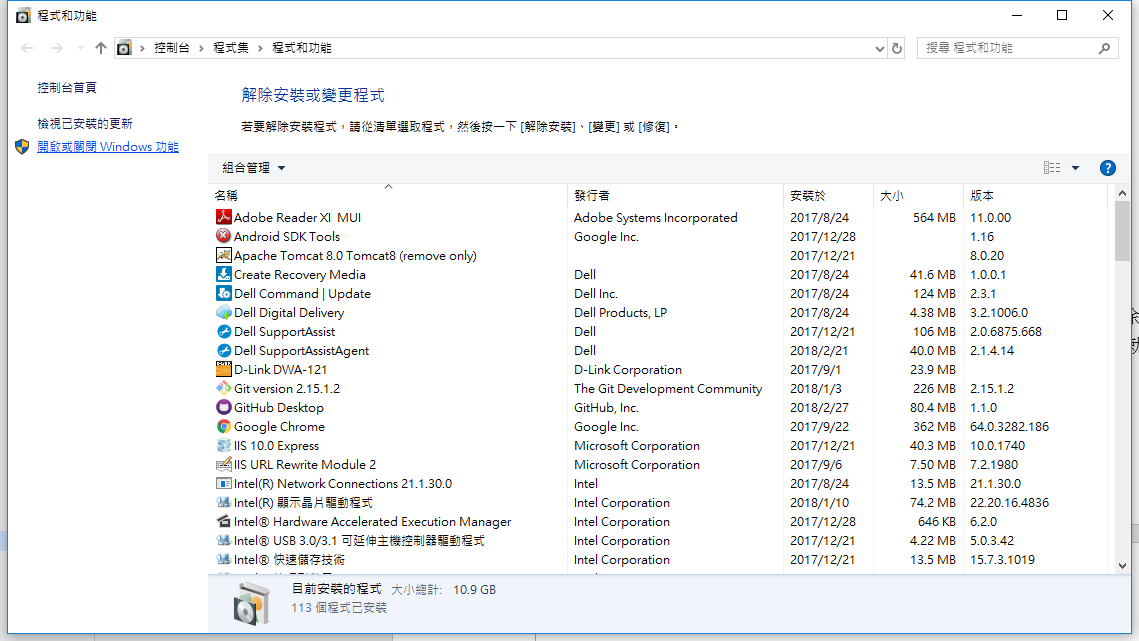
雖然說Python是著名的膠水語言，可以和很多語言和框架進行交互操作，但是要是能直接使用Python編寫的框架進行伺服器的部署的話，則可以簡化程式的編寫過程。Python下最著名的網站框架就是Django了，因此今天就介紹下新版本Django的部署方法，主要以Windows為例。為何要強調是新版呢？因為Django這幾年也變化了很多，然而網上有很多陳舊的部署教程，會對我們的部署過程進行干擾，所以我覺得有這個必要整理一下部署的經驗。

我們先看看Windows下怎麼進行Django的部署，我借助的是Azure中的Django範本中所使用wfastcgi庫以及微軟自己的伺服器平臺IIS。過程如下：

### 步驟一、安装IIS，请务必勾选服务器技术中的CGI。



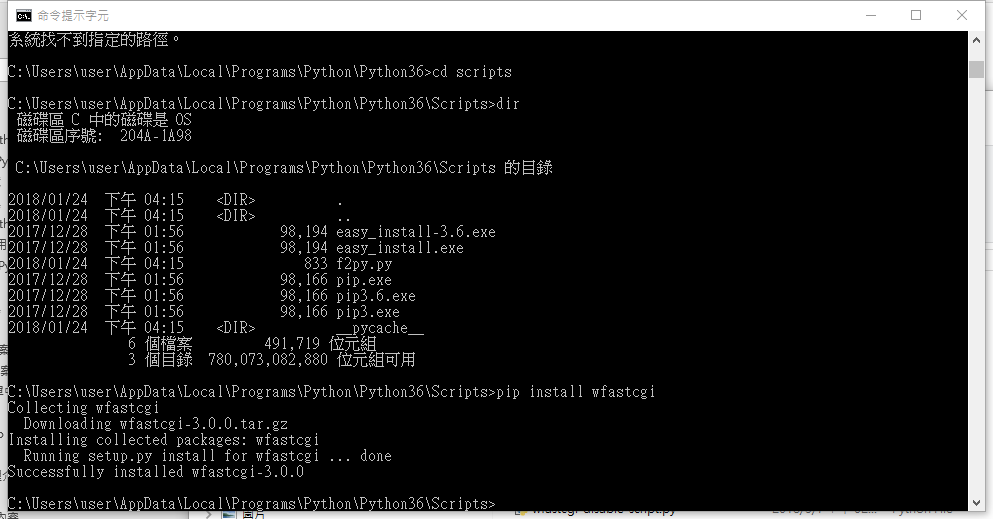
如果已經安裝了IIS，那就要檢查一下，請開啟 “設定”，選擇 “App 解除安裝、預設值、選擇性功能”，然後點選右邊相關設定裡面的 “程式和功能”，就會跳出下面的視窗：



在點選 [開啟或關閉Windows功能]，就會看到剛才的視窗了。

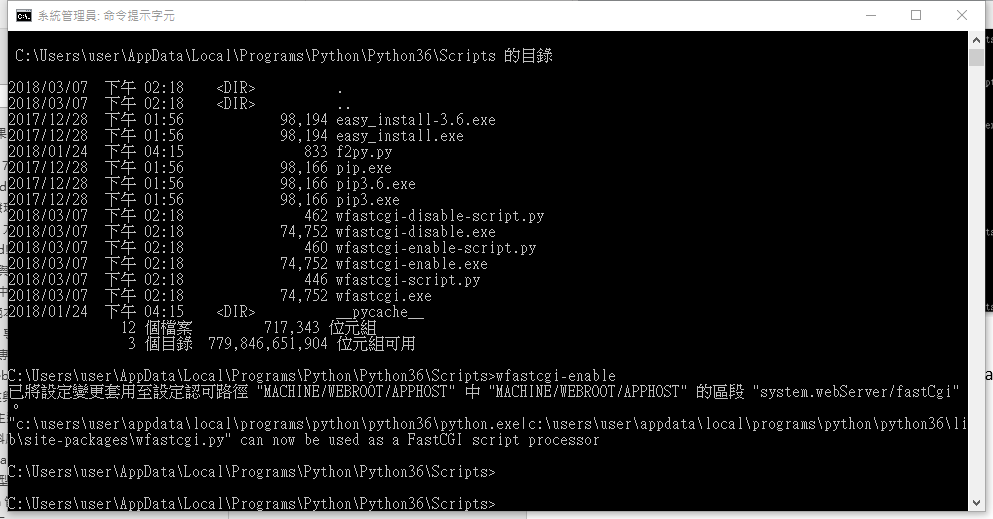
### 步驟二、安装wfastcgi

這個步驟是最主要的步驟，因為 Python 靠的就是這個 CGI。打開 cmd 視窗，進入 “C:\Users\user\AppData\Local\Programs\Python\Python36\Scripts” 目錄，執行 “pip install wfastcgi” 命令，成功之後的畫面如下：



### 步驟三、执行指令，获得FastCGI的Handler参数。

請記得用 admin 的身分開啟 cmd，在同樣目錄下面執行 “wfastcgi-enable”，結果如下：



這裡會獲得一串類似 “c:\users\user\appdata\local\programs\python\python36\python.exe|c:\users\user\appdata\local\programs\python\python36\lib\site-packages\wfastcgi.py” 的字串，其實就是python的路徑和handler的路徑用|進行連接而成的字串，記錄下來，後面會用到。

### 步驟四、在網站根目錄下新建一個web.config

Web.config 的官方範例如下：

<configuration>

<system.webServer>

<handlers>

<add name="Python FastCGI"

path="\*"

verb="\*"

modules="FastCgiModule"

scriptProcessor="C:\Python36\python.exe|C:\Python36\Lib\site-packages\wfastcgi.py"

resourceType="Unspecified"

requireAccess="Script" />

</handlers>

</system.webServer>

<appSettings>

<!-- Required settings -->

<add key="WSGI\_HANDLER" value="django.core.wsgi.get\_wsgi\_application()" />

<add key="PYTHONPATH" value="C:\MyApp" />

<!-- Optional settings -->

<add key="WSGI\_LOG" value="C:\Logs\my\_app.log" />

<add key="WSGI\_RESTART\_FILE\_REGEX" value=".\*((\.py)|(\.config))$" />

<add key="APPINSIGHTS\_INSTRUMENTATIONKEY" value="\_\_instrumentation\_key\_\_" />

<add key="DJANGO\_SETTINGS\_MODULE" value="my\_app.settings" />

<add key="WSGI\_PTVSD\_SECRET" value="\_\_secret\_code\_\_" />

<add key="WSGI\_PTVSD\_ADDRESS" value="ipaddress:port" />

</appSettings>

</configuration>

依照官方範例的web.config檔，我們需要做一些更改，首先scriptProcessor改為上面我們記錄的那段字串，然後是一些必要的參數，像PYTHONPATH改為網站的根目錄的路徑，最後就是一些可選的參數，像WSGI\_LOG是log檔的路徑，還有DJANGO\_SETTINGS\_MODULE需要改為自己專案的名稱，像APPINSIGHTS\_INSTRUMENTATIONKEY，WSGI\_PTVSD\_SECRET, WSGI\_PTVSD\_ADDRESS這些都是Azure上的參數，我們可以將其刪除。修改之後的內容如下：

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<system.webServer>

<handlers>

<add name="Python FastCGI"

path="\*"

verb="\*"

modules="FastCgiModule"

scriptProcessor="c:\users\user\appdata\local\programs\python\python36\python.exe|c:\users\user\appdata\local\programs\python\python36\lib\site-packages\wfastcgi.py"

resourceType="Unspecified"

requireAccess="Script" />

</handlers>

</system.webServer>

<appSettings>

<!-- Required settings -->

<add key="WSGI\_HANDLER" value="django.core.wsgi.get\_wsgi\_application()" />

<add key="PYTHONPATH" value="C:\inetpub\wwwroot\django" />

<!-- Optional settings -->

<add key="WSGI\_LOG" value="C:\Users\Logs\AndrewDjangoWeb.log" />

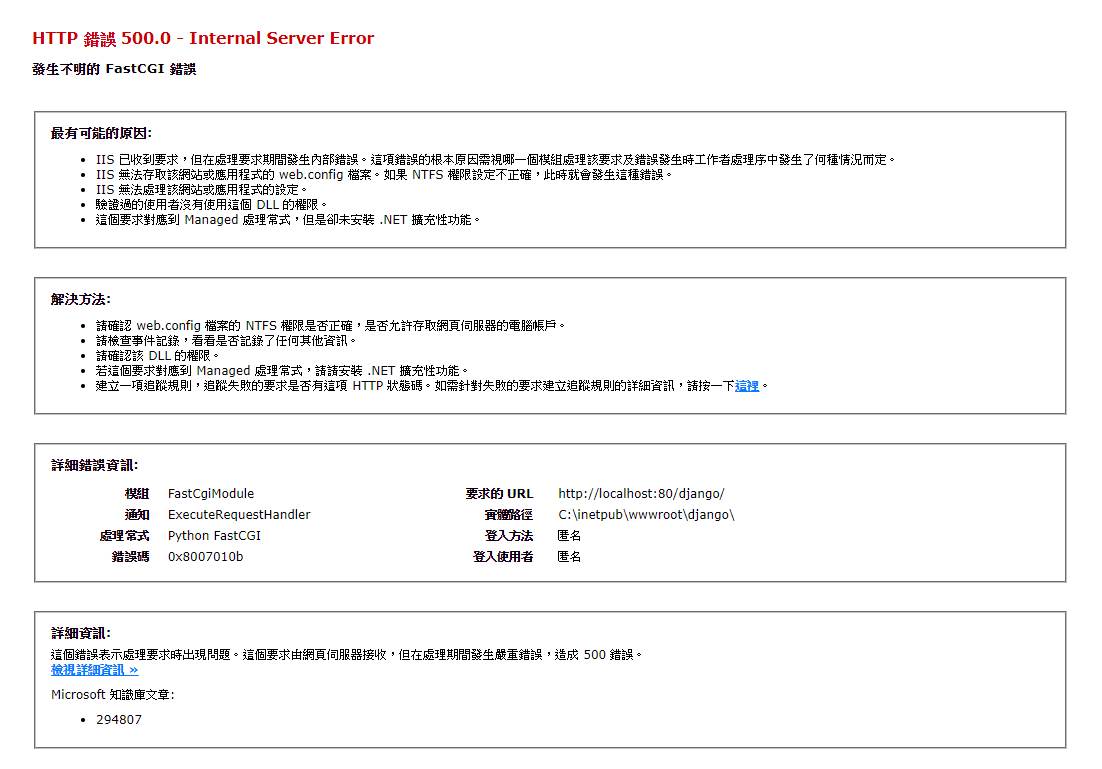
<add key="WSGI\_RESTART\_FILE\_REGEX" value=".\*((\.py)|(\.config))$" />

<add key="DJANGO\_SETTINGS\_MODULE" value="AndrewDjangoWeb.settings" />

</appSettings>

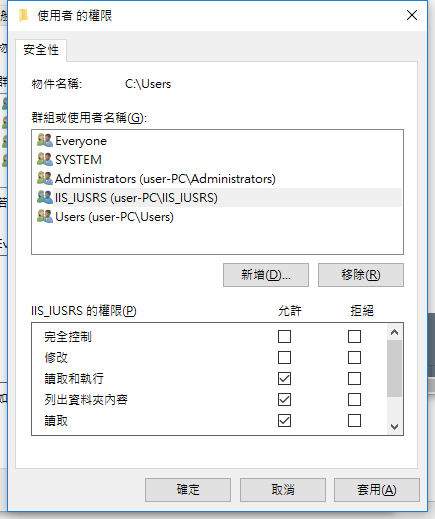
</configuration>

由於站點我們已經建立了，理論上可以執行，不過由於 Python 的執行環境相關檔案，是存在於自己的目錄，因此，我們會遇到額外的問題，就是執行權限的問題，如下圖所示：

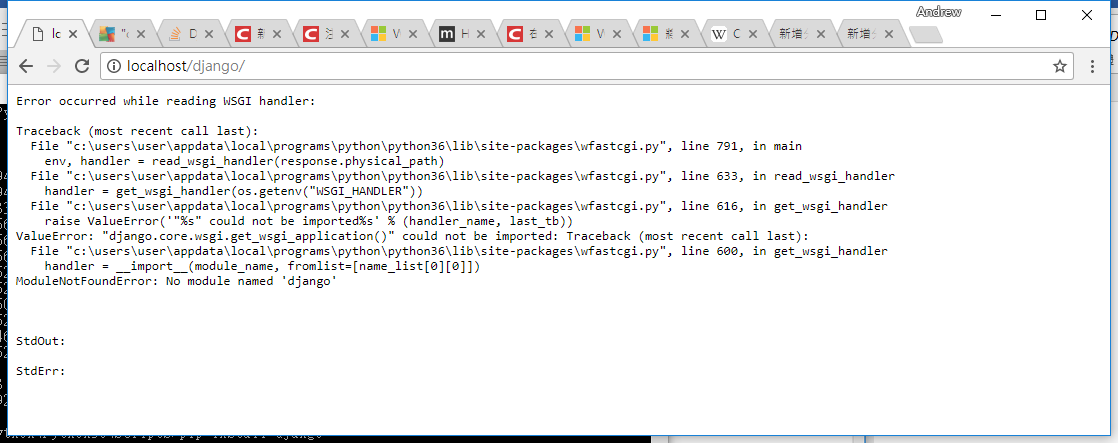


### 步驟五、開IIS\_IUSRS權限

我們把 C:\Users 的權限開給 IIS\_IUSRS，基本上只要開三個權限給 IIS\_IUSRS就好，如下圖都勾的三個，一直開到最底層為止，或者只把最底層的權限打開：

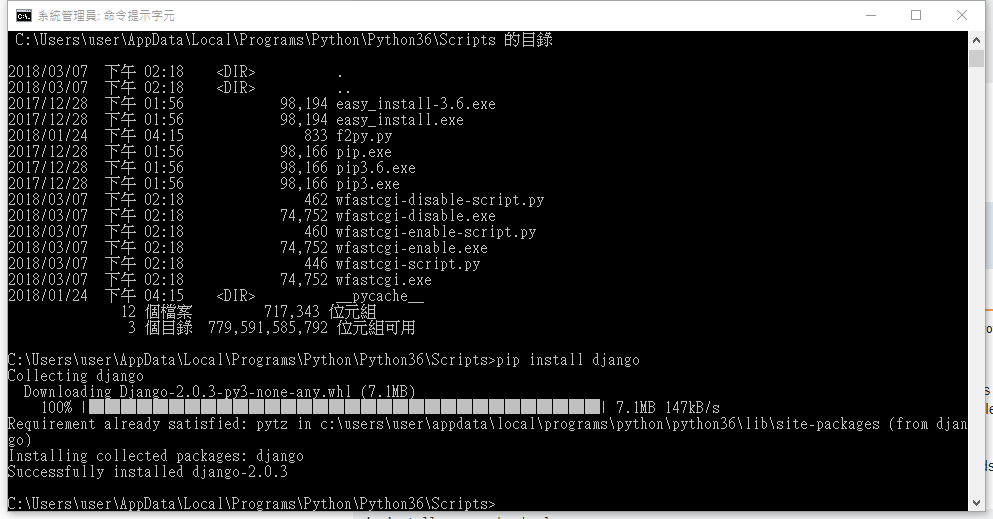


這一步完成之後，理論上應該就可以執行了，很多網路上的討論大概就到這一步為止，不幸的，執行的結果如下圖：



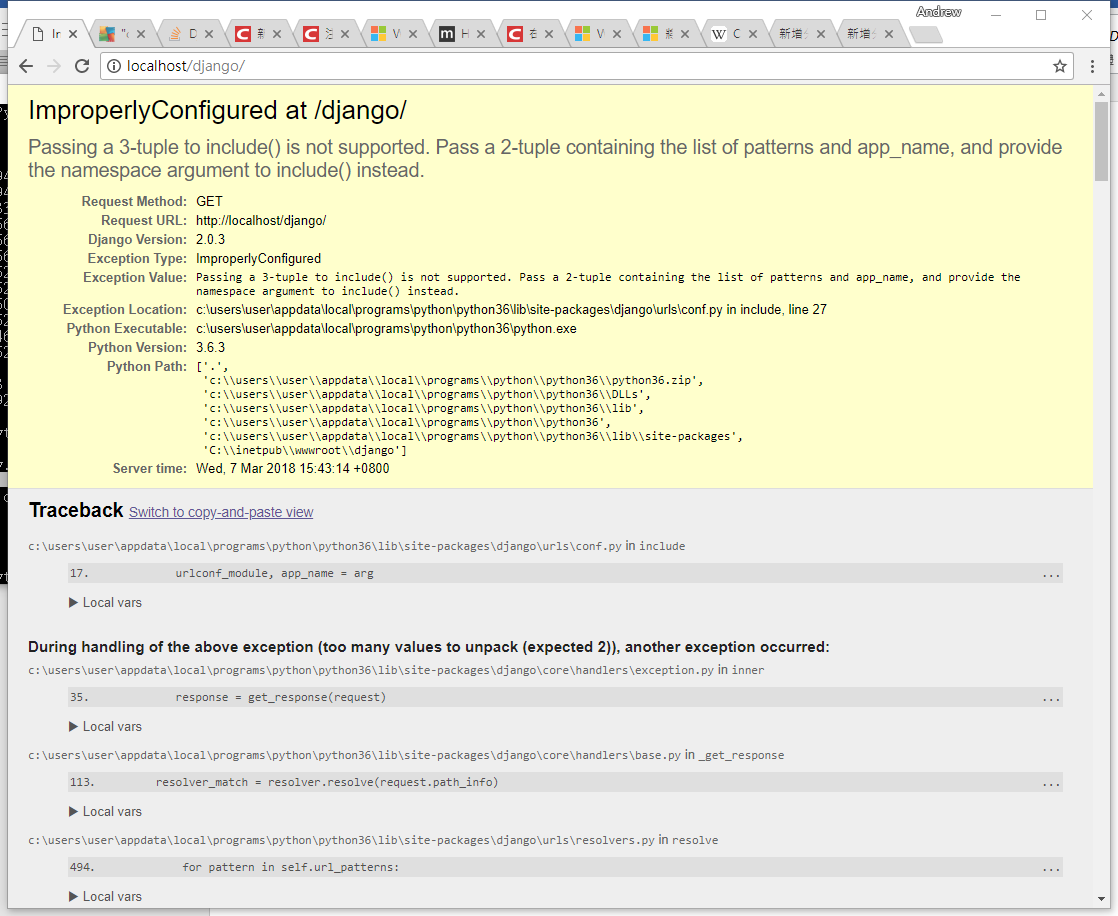
這個問題比之前更為嚴重，這代表我們剛剛裝的 wfastcgi 雖然經過修改權限而成功執行了，但是卻不認識 Django 這個東西！回想一下我們在 Visual Studio 開發的時候，會在 NuGet 套件管理的的方下載 django，明明已經安裝了，為何這裡卻找不到？這是因為 Visual Studio 的開發環境設計的比較複雜，它把每一個方案(solution)都當成一個獨立的虛擬 container，這個好處是不會互相影響。但是實際在主機上面執行的時候，就必須替這台主機安裝元件。

### 步驟六、利用 pip 安裝 django



安裝好之後，會發生一件慘事，那就是原本方案中的 NuGet管理工具無法執行了，會顯示不支援。不過先不要氣餒，因為後面接踵而來會發生的慘事還很多，相形之下這只是一件小事。

當我們使用 pip 安裝了 django 之後，請觀察一下安裝畫面，我們裝的是 2.0.3版，感覺一切順利，但是執行的結果又要讓人氣餒了，如下圖：



產生這個問題的根源在於，Visual Studio 開發環境中安裝的 Django 是 1.9版，而我們在系統上安裝的是 2.0 版，兩個版本的寫法上有差異，差異如下：

In Django 2.0 you can no longer use include(admin.site.urls) (release notes). Just use admin.site.urls instead.

urlpatterns = [

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

...

]

urlpatterns = [

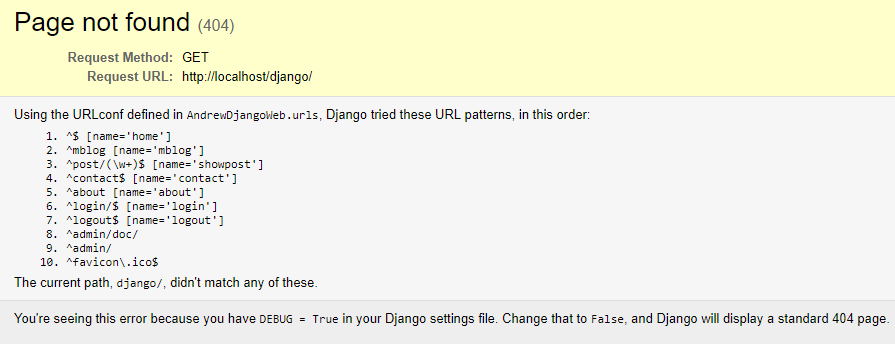
url(r'^admin/', admin.site.urls),

...

]

這下子必須去改原始程式碼了，請打開發行出來的 urls.py 檔案，找到該行，按照上面的方式加以修改，如果你是使用 NotePad++，請同樣用管理者的身分執行。

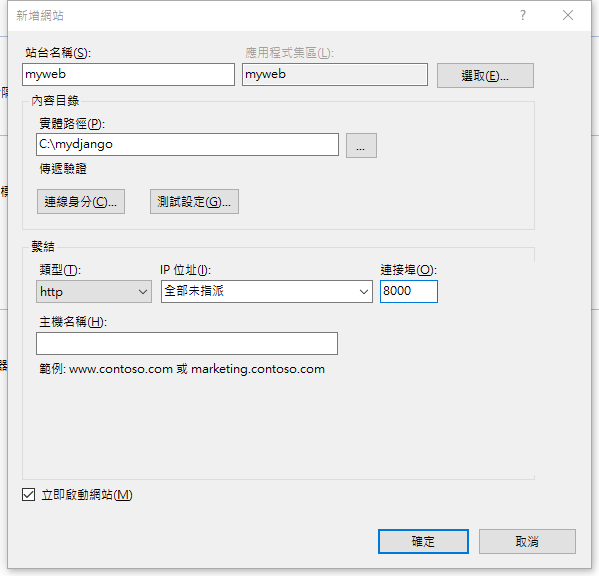
我們在看看執行結果，如下圖：



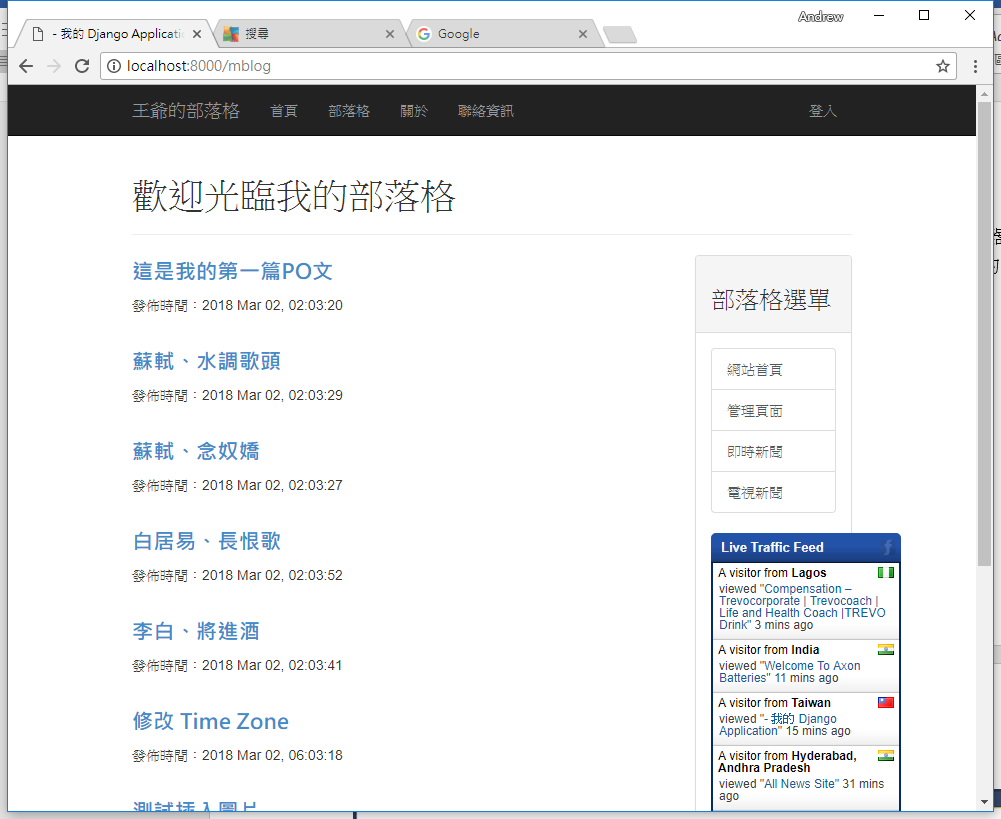
這個問題是所有問題裡面最嚴重的，因為程式碼與環境其實我們都修正過了，理論上應該沒有任何問題，也找不到任何相關的文件，所以我做了一個大膽的假設，Python Django 並沒有辦法依附在 Default Web Site 底下執行，因為可能會和 .NET Framework 衝突。

### 步驟七、建立網站

如果是這樣，我們就在 IIS 裡面建立一個新的網站，如下圖：

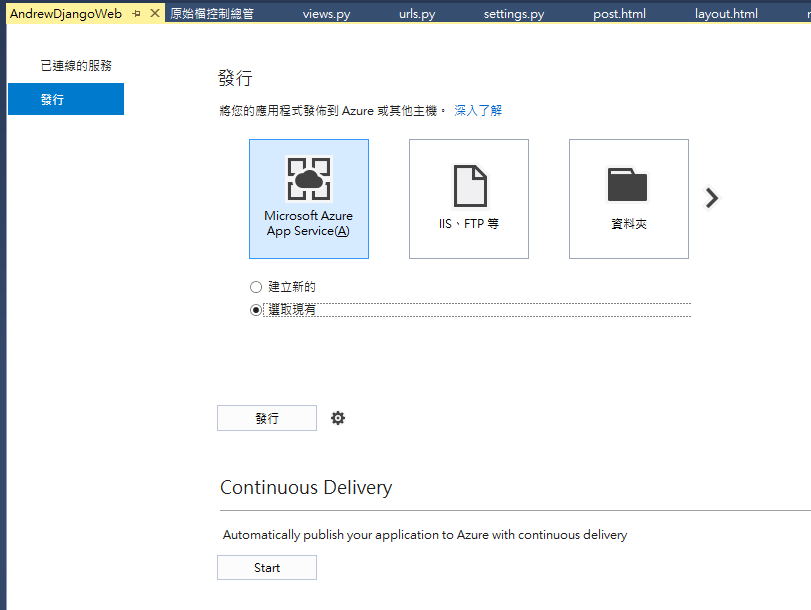


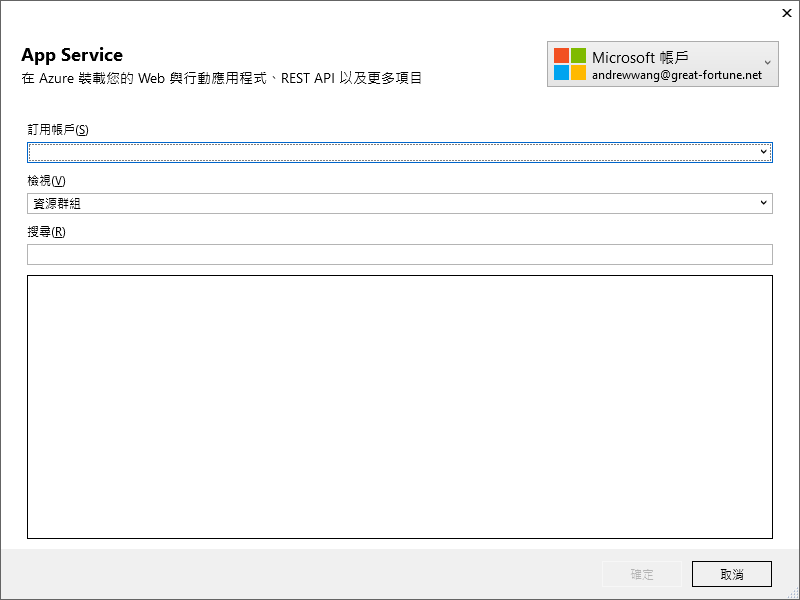
請特別注意，port 的部分建議用 8000，因為這是 Django 的內定的使用數值，我試過用 81 或是其他的值，執行結果會無法顯示 BootStrap 的效果，最後的執行結果如下圖：



歷經了千辛萬苦總算看到結果了，但是必須用 Chrome才能看到此結果。最後，只剩下一個小問題，admin管理畫面無法進入，就留給讀者自己研究吧。

## 適用於 Python 的 Azure 雲端服務專案





## Python 的資料結構(序列)