序

Flask 是 Python Web 微型框架，是 Armin Ronacher 於 2010 年 4 月 1 日作為愚人節玩笑所發表的，之後在 Python 使用者之間廣受歡迎。根據 2018 年 Python 開發人員的調查，Flask 獲選為最受歡迎的網路框架，至今依舊受到歡迎。

本書的目的是幫助你藉由透過 Flask 實作網路應用程式（下稱應用程式）的過程，**學會自行製作應用程式**。

從建立**最簡單的應用程式**開始，逐步製作**諮詢表單**、**資料庫應用程式**、**驗證功能**，學習 Flask 開發應用程式的基礎知識。接著，建立由圖片資料（照片）識別物體的**物件偵測應用程式**，學習如何製作實際可用的應用程式後，再講解如何將該功能**轉為網路 API**。

Flask 是一種微型框架，不同於其他受限於重重規範的大型框架，可以相當靈活的運用。而且，由於框架本身內建的功能不多，具備自行思考實作的餘裕、自由度，是適合用來學習應用程式開發的網路框架。

在商務領域上，模擬實證試驗、開發概念展示產品等小規模專案，框架部分經常採用 Flask 微型框架。此外，在開發機器學習等運用資料的產品時，往往也是採用 Flask 將機器學習的實作程式碼嵌入產品，作成通用的網路 API 提供服務。由於運用資料的產品開發歷史尚淺，如何將機器學習嵌入產品發布成應用程式的範本並不多。有鑑於此，本書的分析腳本題材採用易於瞭解機器學習運作的手寫文字辨識，詳細解說**如何將機器學習嵌入應用程式**。

對於今後想用 Flask 開發應用程式、欲將機器學習嵌入應用程式的各位讀者，期望本書能夠帶來幫助。

佐藤 昌基、平田 哲也

第 1 篇

Flask 入門

第 1 章

建立最基礎的應用程式——Flask 的基礎知識

本章內容

1.1 MVT（Model、View、Template）模型

Flask 採用 **MVT**（Model、View、Template）設計模式，實作具有使用者介面的應用程式（圖 1.1）。

Model、View、Template 分別負責下述任務：

1. **Model**　：負責商務邏輯
2. **View**　：根據輸入控制 Model 和 Template
3. **Template**：負責輸出入

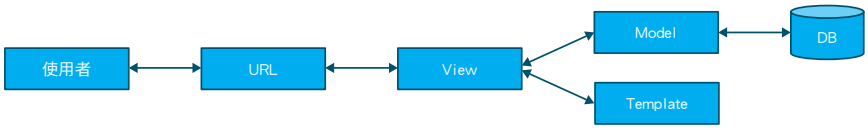

圖 1.1　MVT 模型

一般來說，MVC 模型更為有名，MVT 的 View 相當於 MVC 的 Controller；MVT 的 Template 相當於 MVC 的 View（圖 1.2）。彼此有細微上的差異，但整體而言差異不大。

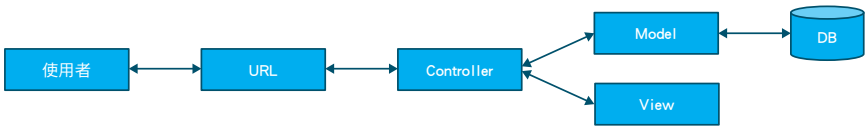

圖 1.2　MVC 模型

下一節將會解說如何使用 Flask 開發（建立）網路應用程式，若尚未完成虛擬環境、工作目錄、安裝 Flask 等的環境架設，請先參閱 0.4 節（p.9）的內容操作。

1.2 建立最基礎的應用程式

那麼，來建立具備最基礎功能的應用程式 minimalapp。

首先，執行下述指令，確認當前狀態：

```
(venv) $ pwd  ————顯示當前的工作目錄
/path/to/flaskbook
(venv) $ ls  ————顯示目錄內部的資訊
venv
```

在 flaskbook 工作目錄啟用虛擬環境，裡頭僅有 venv 目錄。

建立工作目錄

準備應用程式 minimalapp 的工作目錄。建立 apps/minimalapp 目錄，以便後續在 flaskbook 工作目錄新增多組應用程式。

```
(venv) $ mkdir -p apps/minimalapp
```

以如圖 1.3 的目錄架構建立應用程式。

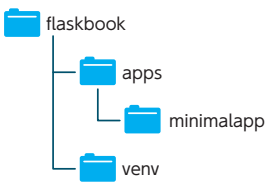

圖 1.3　目前的目錄架構

啟動應用程式

啟動 Flask 應用程式的步驟，如下所示：

1. 編寫 Python 腳本（程式碼）
2. 設定環境變數
3. 執行 `flask run` 指令

❶編寫 Python 腳本（程式碼）

使用 VSCode，在 `apps/minimalapp` 底下建立 `app.py`（圖 1.4），再於 `app.py` 裡頭編寫範例 1.1 的程式碼（圖 1.5）[※1]。

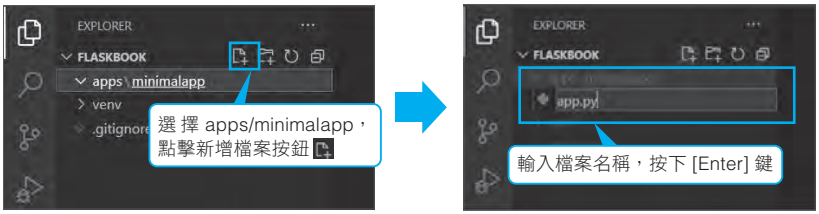

圖 1.4 建立檔案

圖 1.5 在 app.py 中編寫程式碼

※1 若檔案名稱取為 `flask.py`，會與Flask本體相衝而無法啟動，需要小心留意。

範例 1.1　apps/minimalapp/app.py

```
# 匯入 Flask 類別
from flask import Flask

# 建立 Flask 類別的實體（instance）
app = Flask(__name__)

# 配對網址和執行的函數
@app.route("/")
def index():
    return "Hello, Flaskbook!"
```

❷設定環境變數

啟動應用程式前，得先設定環境變數 **FLASK_APP** 和 **FLASK_ENV**（表 1.1）。

表 1.1　設定環境變數

環境變數	設定的值
FLASK_APP	應用程式的位置
FLASK_ENV	指定 development 或者 production。指定 development 後，會開啟除錯模式

由控制台（終端機或者 PowerShell）設定環境變數。

首先，執行下述指令，移動至 app.py 所在的目錄。

```
(venv) $ cd apps/minimalapp
```

完成後，執行下述指令，設定環境變數。

● **Mac/Linux 的情況**

```
(venv) $ export FLASK_APP=app.py
(venv) $ export FLASK_ENV=development
```

● **Windows（PowerShell）的情況**

```
> (venv) $env:FLASK_APP="app.py"
> (venv) $env:FLASK_ENV="development"
```

❸執行 flask run 指令

設定環境變數後，在 apps/minimalapp 目錄下，執行 **flask run** 指令，啟動應用程式。若 FLASK_ENV 為 development，則自動開啟除錯模式。在開發 Flask 應用程式時，需要開啟除錯模式。

```
(venv) $ pwd
/path/to/flaskbook/apps/minimalapp

(venv) $ flask run
 * Serving Flask app "app.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
```

使用瀏覽器開啟下述網址，會顯示「Hello, Flaskbook!」（圖 1.6）。

http://127.0.0.1:5000/

圖 1.6 flask tun 的執行結果

Column

未指定 FLASK_ENV 時的警告

若未指定 FLASK_ENV，執行 flask run 指令時，會跳出如下的警告內容。這是 Flask 應用程式嘗試以正式模式執行時所顯示的內容，附屬 Flask 的嵌入式伺服器主要用來開發，不適用於正式環境。

```
 * Serving Flask app "app.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a ↩
production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
Usage: flask run [OPTIONS]
```

何謂除錯模式？

不使用 Flask 的嵌入式伺服器，而將環境變數 FLASK_ENV 指定為 production，則也可於正式環境開發。不過，指定 development 模式並開啟除錯模式，具有下述優點：

- 網頁會顯示錯誤訊息（圖 1.7）
- 自動載入會開啟，編輯程式碼時自動套用至應用程式（不需要手動重新啟動）

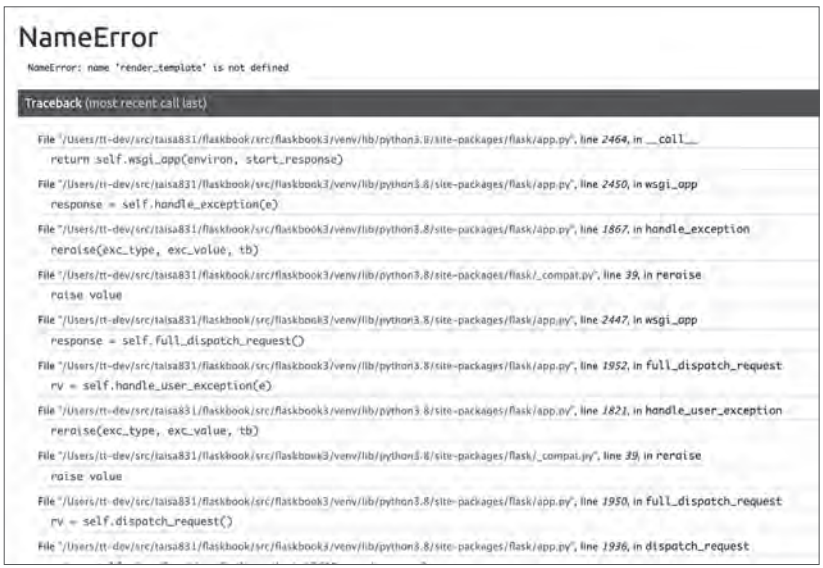

圖 1.7　除錯時的錯誤資訊

在編輯器（這裡使用 VSCode）修改 app.py 的程式碼，Flask 的嵌入式伺服器會自動載入，於控制台輸出下述內容：

```
 * Detected change in '/path/to/flaskbook/apps/minimalapp/app.py', ↩
reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 140-347-940
```

使用 .env 設定環境變數

前面使用 export 指令設定環境變數，不過以該指令設定的變數，登出控制台後會跟著消失。雖然也可永久更改控制台的環境變數，但每次切換應用程式都得再更改環境變數。

有鑑於此，這裡建議使用 **.env** 檔案，以應用程式為單位更改環境變數。事先建立 .env 檔案並編寫環境變數的值，再由應用程式加載環境變數。另外，**.flaskenv** 檔案同樣也可加載環境變數，但兩者是不一樣的檔案，選擇一種利用即可。

加載 .env 或者 .flaskenv 時，需要 **python-dotenv** 套件，一般得先編寫使用 python-dotenv 的程式碼，才能夠利用該套件加載檔案。然而，Flask 的核心（Flask 內部）已有載入 python-dotenv，僅需以 pip install 安裝就可利用。

安裝 python-dotenv 與建立 .env 檔案

那麼，嘗試安裝 python-dotenv，加載 .env 檔案的值當作環境變數。

執行下述指令，安裝 python-dotenv。

```
(venv) $ pip install python-dotenv
```

在 minimalapp 底下建立 .env 檔案並設定環境變數（範例 1.2、圖 1.8）。

範例 1.2　apps/minimalapp/.env

```
FLASK_APP=app.py
FLASK_ENV=development
```

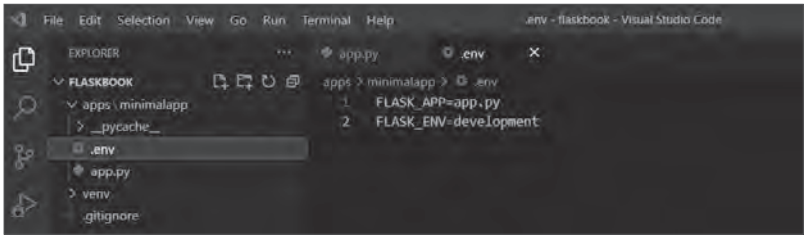

圖 1.8　建立 apps/minimalapp/.env

為了確認是否成功加載 .env 檔案，請在未使用 export、$env: 指定環境變數的控制台，或者重新啟動的控制台，執行 flask run 指令。

執行 flask run 後，確認應用程式的啟動細節。

```
(venv) $ flask run
 * Serving Flask app "app.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 619-621-965
```

應用程式路由

應用程式路由（application root）是指執行該程式的目錄，決定加載模組、套件的路徑[※2]。

Flask 是在內部呼叫 python-dotenv 套件、安裝 python-dotenv，應用程式路由會因有無 .env 檔案而異。

※2　模組是指集結函數、類別的.py檔案，而套件是指集結模組的模組集。

- 無 .env 檔案的情況 ➡ 應用程式路由是**執行 flask run 指令的目錄**
- 有 .env 檔案的情況 ➡ 應用程式路由是**含有 .env 檔案的目錄**

雖然前面列出兩種情況，但有 .env 檔案的時候，是**在 minimalapp 目錄底下配置 .env 檔案**，執行 flask run 指令。因此，兩種情況的應用程式路由相同（apps/minimalapp）。

下面來看無 .env 檔案和有 .env 檔案的情況吧。

無 .env 檔案的情況

執行 flask run 指令後，取得環境變數 FLASK_APP 的值，檢測工作目錄中是否有與該值同樣名稱的模組（圖 1.9）。若有該模組的話，從中取得並執行 Flask 應用程式。

圖 1.9 無 .env 檔案的情況

有 .env 檔案的情況（已安裝 python-dotenv）

執行 flask run 指令後，搜尋 .env 檔案並以該檔案位置為應用程式路由（圖 1.10）。

取得 .env 檔案的 FLASK_APP 值，檢測工作目錄中是否有與該值同樣名稱的模組。若有該模組的話，從中取得並執行 Flask 應用程式。

圖 1.10 有 .env 檔案的情況

更改應用程式路由

首先，先確認當前的應用程式路由，如圖 1.11 讓 minimalapp 變成應用程式路由。

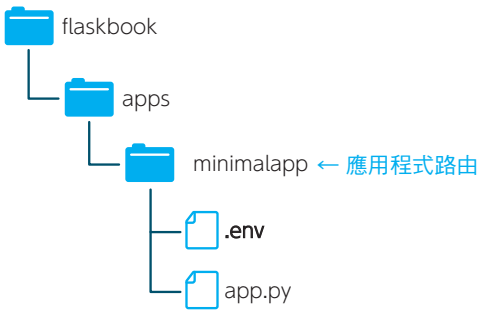

圖 1.11 當前的應用程式路由

將應用程式路由更改為 flaskbook，這樣在開發其他的應用程式時，不需要每次都切換 VSCode 視窗。為此，需要將 apps/minimalapp 中的 .env 檔案，移動到 flaskbook 目錄（圖 1.12）。

圖 1.12 更改應用程式

移動 .env 檔案的時候，得於 .env 檔案中的 FLASK_APP 將 minimalapp 應用程式的位置，

```
apps.minimalapp.app.py
```

以「.」連起來指定（範例 1.3）：

範例 1.3 更改應用程式路由（.env 或者 flaskbook/.env）

```
FLASK_APP=apps.minimalapp.app.py          修改
FLASK_ENV=development
```

如此一來，應用程式路由就變成 flaskbook。請移動至 flaskbook 目錄，重新執行 flask run 指令，使用瀏覽器訪問下述網址：

http://127.0.0.1:5000/

跟前面一樣，畫面會顯示 Hello, Flaskbook!。

這樣就準備好於本地電腦啟動並開發 Flask 應用程式，接著解說 Flask 的網路開發基礎。

使用路由建置

路由建置是指，綁定請求目的地的網址[※3]和實際處理的函數。

藉由在前面加上**裝飾器（decorator）**函數**@app.route()**，Flask 能夠增加路由。

在 app.py 增加輸出「Hello, World!」的路由（範例 1.4）。

範例 1.4　增加路由（apps/minimalapp/app.py）

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello, Flaskbook!"

@app.route("/hello")
def hello():
    return "Hello, World!"
```

1 增加

如此一來，使用瀏覽器訪問下述網址，畫面會顯示「Hello, World！」（圖 1.13）。

http://127.0.0.1:5000/hello

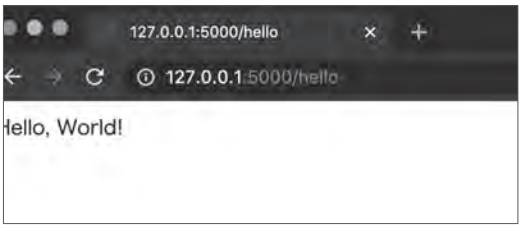

圖 1.13　http://127.0.0.1:5000/hello 的執行結果

※3　http://127.0.0.1:5000/ ～的「～」部分表示瀏覽器（客戶端）對伺服器端請求的目標資源。

以 flask routes 指令確認路由資訊

使用 **flask routes** 指令，確認路由資訊。flask routes 非常便利，後續也會頻繁用來確認資訊。

```
(venv) $ flask routes
Endpoint  Methods  Rule
--------  -------  -----------------------
hello     GET      /hello
index     GET      /
static    GET      /static/<path:filename>
```

在 HTML 表單使用的 HTTP 方法

HTTP 方法（method）是客戶端對伺服器發送請求時，傳達希望伺服器執行的操作。

雖然 HTTP 方法有好幾種，但 HTML 表單[※4] 僅會用到 **GET 方法**、**POST 方法**。

GET 方法用於獲取資源，如搜尋等操作。除了使用表單外，平常的網路瀏覽也是 GET 方法。

POST 方法用於登錄或者修改表單的值，如登入、提交查詢等操作。

然後，除了 GET 和 POST 外，網路 API 還會用到 PUT、DELETE 等 HTTP 方法，細節留到第 12 章的「HTTP 方法的 CRUD 資源操作」解說。

Flask 的端點命名

Endpoint（端點）一般是指訪問 API 的網址，但**在 Flask 是指已綁定網址的函數名稱，或者函數本身的名稱**。後面將會講解 Flask 中的端點。

端點名稱預設為裝飾 @app.route 的函數名稱，但也可如下取任意名稱：

```
@app.route("/", endpoint="endpoint-name")
```

※4 HTML 表單的規格是，由客戶端輸入、選擇資料，再傳送給處理 HTML 表單的網路伺服器。

例如，假設端點名稱取為 hello-endpoint，如範例 1.5 修改剛才 app.py 新增的部分（範例 1.4 ❶），端點當作 Flask 內部的設定值，使用 p.46 說明的 url_for 等名稱。

範例 1.5　hello 端點名稱取為 hello-endpoint（apps/minimalapp/app.py）

```
... 省略 ...

@app.route("/hello",
  methods=["GET"],
  endpoint="hello-endpoint")
def hello():
    return "Hello, World!"
```

增加

執行 flask routes 指令，確認端點名稱是否改變（圖 1.14）。

```
(venv) $ flask routes
Endpoint        Methods    Rule
--------------  ---------  ----------------------
hello-endpoint  GET        /hello
index           GET        /
static          GET        /static/<path:filename>
```

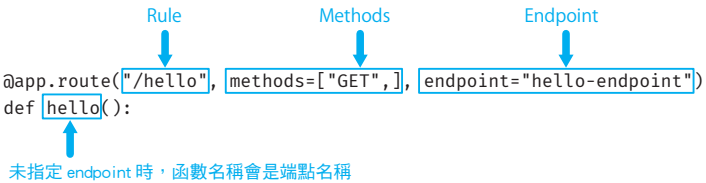

圖 1.14　flask routes 的端點、方法與規則

指定允許的 HTTP 方法

@app.route 裝飾器可指定允許的 HTTP 方法，如下在 methods 指定 HTTP 方法名稱（範例 1.6）：

```
@app.route("/", methods=["GET", "POST"])
```

未做任何指定時，預設的方法為 GET。另外，Flask 版本 2.0 後，可直接省略 route() 寫成：

```
@app.get("/hello")
@app.post("/hello")
```

範例 1.6　HTTP 方法指定 GET 和 POST 的情況

```
@app.route("/hello", methods=["GET", "POST"])
def hello():
    return "Hello, World!"

# 自 Flask 2 版本後，可寫成 @app.get("/hello")、@app.post("/hello")
# @app.get("/hello")
# @app.post("/hello")
# def hello():
#     return "Hello, World!"
```

如上在 methods 指定 HTTP 方法後，該函數可接受 GET 和 POST 方法的請求。

在 Rule 指定變數

在 @app.route 裝飾器中的 Rule（規則），可以 < **變數名稱** > 的形式指定變數。

將 app.py 範例 1.5 的一部分，如範例 1.7 修改程式碼。

範例 1.7　在網址 Rule 指定 <name> 變數（apps/minimalapp/app.py）

```
... 省略 ...

@app.route("/hello/<name>",                                    修改
  methods=["GET", "POST"],
  endpoint="hello-endpoint")
def hello(name):                                               修改
    # 自 Python 3.6 導入以 f-string 定義字串
    return f"Hello, {name}!"                                   修改
```

在瀏覽器網址列，如下將 /hello/<name> 的 <name> 部分指定任意字串（例如 ichiro），執行後會顯示該字串（圖 1.15）。

http://127.0.0.1:5000/hello/ichiro

圖 1.15　http://127.0.0.1:5000/hello/ichiro 的執行結果

選項的部分使用轉換器（converter）的類型定義（表 1.2），寫成 **< 轉換器：變數名稱 >** 來指定變數的資料類型。藉由轉換器檢測類型，當類型不相符時會顯示錯誤。

表 1.2　轉換器

轉換器的類別	說明
string	沒有斜線的內文
int	正的整數
float	正的浮點小數
path	允許帶有斜線的內文
uuid※5	UUID 字串

使用模板引擎

模板引擎是指，合成名為模板的雛形和資料，並輸出成果文件的軟體（圖 1.16）。Flask 的預設模板引擎是 Jinja2，安裝 Flask 時會一併安裝 Jinja2。藉由 render_template，使用模板引擎產生（成像）HTML。

※5　UUID（Universally Unique Identifier）是指，軟體上唯一辨別物件的識別碼。

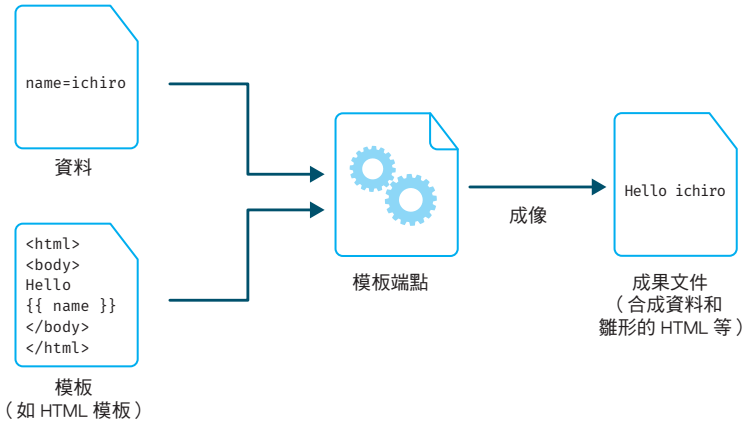

圖 1.16 模板引擎的規格

建立模板後，僅需將模板名稱和關鍵字引數，當作變數傳給 `render_template` 函數，即可於應用程式端利用。嘗試使用 `render_template` 函數使用模板吧。

首先，建立 HTML 檔案當作所需的成像模板。啟動 VSCode 軟體，在 `minimalapp` 底下建立 `templates` 目錄，並於該目錄製作 `index.html`（範例 1.8）。

模板中編寫 `{{ 變數名稱 }}` 的地方，Jinja2 會展開變數進行成像。在 `index.html` 增加 `{{ name }}`（❶），以便顯示變數的值。

範例 1.8 模板（apps/minimalapp/template/index.html）

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="UTF-8" />
    <title>Name</title>
  </head>

  <body>
    <h1>Name: {{ name }}</h1>          ❶ 編寫{{ name }}
  </body>
</html>
```

然後，將 `apps/minimalapp/app.py` 如範例 1.9 修改程式碼：