前言

本書不僅是一本實用的 FastAPI：現代化 Python 網站框架，同時也是一個講述我們隨手使用的小工具，卻意外地成為不可或缺利器的故事。畢竟，當遭遇狼人時，手上有顆銀彈可是再好不過了；而在本書後續章節中，你也確實會遇到一些「狼人」。

我於 1970 年代中期開始撰寫科學應用程式。1977 年，當我在 PDP-11 電腦上首次接觸 Unix 和 C 語言時，就有預感這個叫 Unix 的系統可能會大受歡迎。

在 80 年代和 90 年代初期，網際網路尚未商業化，但已是免費軟體和技術資訊的絕佳來源。1993 年，當名為 Mosaic 的網頁瀏覽器在新興開放網路上發布時，我又有種預感，這個網路可能會大放異彩。

幾年後，當我創立自己的網站開發公司時，我採用當時的主流工具：PHP、HTML 和 Perl。再過幾年，我在一份約聘工作中終於嘗試 Python，驚訝於自己能如此迅速地存取、處理和呈現資料，不過兩週的閒暇時間，我就複製大部分原本要耗費 4 名開發人員一年時間才能完成的 C 語言應用程式。這時我的預感又出現了，這個叫 Python 的程式語言可能會大受歡迎。

自那時起，我的大部分工作都與 Python 及其網站框架有關，主要是 Flask 和 Django，我特別欣賞 Flask 的簡潔性，在許多專案中都偏好使用它。但就在幾年前，我瞥見一個閃閃發光的新事物：由 Sebastián Ramírez 開發的嶄新 Python 網站框架，名為 FastAPI。

閱讀他那優秀無比的文件（*https://fastapi.tiangolo.com*）時，我對其設計理念印象深刻，特別是他的歷史（*https://oreil.ly/Ds-xM*）頁面，展現他在評估替代方案時投入的心力。這不是一個只想自己試試的專案或有趣的實驗，而是一個為實際開發而生的正式框架。這時我又有種預感，這個叫 FastAPI 的框架可能會成為主流。

我使用 FastAPI 開發一個生物醫學 API 網站，效果相當出色，以致於本團隊在接下來的一年裡，用 FastAPI 重構我們的舊核心 API，而這個 API 至今仍在生產環境中穩定運作。本團隊學習你將在本書中接觸到的基礎知識，眾人一致認為我們寫出更優質、更高效、錯誤率更低的程式碼；值得一提的是，我們中有些人之前從未接觸過 Python，而且只有我使用過 FastAPI。

因此，當我有機會向 O'Reilly 提議《精通 Python》（歐萊禮）的後續作品時，FastAPI 自然成為我的首選。在我看來，FastAPI 至少會發揮與 Flask 和 Django 同等的影響力，甚至可能更甚前兩者。

如我所提到，FastAPI 官網本身就提供頂尖水準的文件，涵蓋許多常見網站主題的細節：資料庫、驗證、部署等等；所以，為什麼還要再撰寫一本書呢？

本書並不致力於面面俱到，因為，嗯，那實在太耗時費力了。本書目的在求實用，幫助讀者快速掌握 FastAPI 的核心概念並應用，我會指出一些需要一點偵探工作的技巧，並就日常最佳實踐提供建議。

每章開始時都會有一個預覽，介紹即將涵蓋的內容，接著，會盡量不忘記剛才承諾的內容，提供詳細資訊和一些題外話；最後，還有一個簡明扼要的回顧。

俗話說：「這些事實來自我個人的看法。」你的經驗必定獨一無二，但我希望你能在這裡找到堪稱有價值的內容，成為一個更高效的網站開發者。

本書編排慣例

本書使用以下編排慣例：

斜體字（*Italic*）

表示新術語、URL、電子郵件地址、檔案名稱和檔案副檔名。中文以楷體表示。

定寬字（`Constant width`）

用於程式碼列表，以及在段落中指稱程式元素，如變數或函式名稱、資料庫、資料型別、環境變數、陳述式和關鍵字。

定寬粗體字（**`Constant width bold`**）

顯示使用者應逐字輸入的命令或其他文字。

話題扯遠了[2]。這本書主要是關於以下幾個方面：

網路

一項極為高效的技術，它的演變過程，以及今日為它開發軟體的辦法

Python

一種極為高效的網站開發語言

FastAPI

一個極為高效的 Python 網站框架

第一部分將著重探討網路技術和 Python 程式語言中的新興議題，包括服務與 API、並行處理、分層架構，以及巨量資料的處理方法。

第二部分提供 FastAPI 的全面概覽。FastAPI 是一個新興的 Python 網站框架，其設計理念正好呼應第一部分所提出的各項挑戰。

第三部分深入剖析與 FastAPI 相關的工具組合，並分享在實際開發過程中累積的寶貴經驗與技巧。

第四部分呈現一系列 FastAPI 的實作範例。這些範例均採用同一個資料源——虛構生物，這樣的設計比起一般零散範例更能吸引讀者，也更具連貫性。這些範例可作為特定應用開發的理想起點。

2 不會是最後一次。

第三章

FastAPI 導覽

> FastAPI 是一個現代、快速（高性能）的網站框架，使用 Python3.6+，並基於標準 Python 型別提示建立 API。
>
> —Sebastián Ramírez，FastAPI 的創造者

預覽

FastAPI（*https://fastapi.tiangolo.com*）由 Sebastián Ramírez（*https://fastapi.tiangolo.com*）於 2018 年發布，相較於多數 Python 網站框架而言更為現代化，且充分利用 Python 3 近年來新增的功能。本章旨在快速概述 FastAPI 的核心特性，聚焦於開發者最關心的議題：處理網路請求和回應的方法。

FastAPI 簡介

作為一款網站框架，FastAPI 致力於簡化網站應用程式的開發流程。每個框架都有獨特設計理念，透過特定功能、適度省略和預設配置，來優化某些操作，顧名思義，FastAPI 主要針對網路 API 的開發而設計，但同時也支援傳統的網頁內容應用程式開發。

根據 FastAPI 官方網站，該框架具備以下優勢：

效能

在某些情況下，速度可與 Node.js 和 Go 相媲美，這對 Python 框架來說相當罕見。

更快的開發速度

沒有棘手的問題或怪異之處。

更好的程式碼品質

型別提示和模型有助於減少錯誤。

自動產生的文件和測試頁面

比手動編輯 OpenAPI 描述容易得多。

FastAPI 使用以下技術：

- Python 型別提示
- Starlette 作為網站機制，包括非同步支援
- Pydantic 用於資料定義和驗證
- 特殊整合以利用和擴充其他功能

這種組合為網站應用程式，尤其是 RESTful 網路服務，提供令人愉悅的開發環境。

FastAPI 應用程式

以下將開發一個簡單的 FastAPI 應用程式，是一個只有單一端點的網路服務。現在專注於所謂的「網站層」，主要處理網路請求和回應，首先，需要安裝幾個基本的 Python 套件：

- FastAPI（*https://fastapi.tiangolo.com*）框架：`pip install fastapi`
- Uvicorn（*https://www.uvicorn.org*）網站伺服器：`pip install uvicorn`
- HTTPie（*https://httpie.io*）文字網站用戶端：`pip install httpie`
- Requests（*https://requests.readthedocs.io*）同步網站用戶端套件：`pip install requests`
- HTTPX（*https://www.python-httpx.org*）同步 / 非同步網站用戶端套件：`pip install httpx`

儘管 curl（*https://curl.se*）是最廣為人知的命令列網路工具，但我認為 HTTPie 在使用上更為直覺，更重要的是，HTTPie 預設支援 JSON 編碼和解碼，這與 FastAPI 的設計理念更加契合。本章稍後會顯示一張螢幕截圖，說明使用 curl 存取特定端點所需的命令列語法。

在範例 3-1 中跟隨一位內向的網頁開發者，並將這段程式碼儲存為 *hello.py* 檔案。

在標準程式設計中，一個變數通常與同一個物件相關聯，如果將型別提示與該變數相關聯，可以避免某些程式設計錯誤。因此，Python 在標準 typing 模組中為語言加入了型別提示功能。Python 直譯器會忽略型別提示語法，執行程式時就如同它不存在一樣。所以，型別提示的意義何在？

你可能在某一行程式碼中將變數視為字串，而稍後不經意地賦予它不同型別的物件，雖然其他語言的編譯器會對此提出警告，但 Python 不會。標準 Python 直譯器只會捕捉一般的語法錯誤，和執行時期例外，但不會檢查變數型別的混用情況，像 mypy 這樣的輔助工具會注意型別提示，並在任何型別不匹配的情況下發出警告。

此外，這些提示對 Python 開發者而言是很有價值的資訊，他們可以據此開發出比單純型別錯誤檢查更進階的工具，以下各節將描述 Pydantic 套件為解決一些較不明顯需求的開發。稍後，會看到它與 FastAPI 的整合，而使得許多網站開發問題變得更容易處理。

順帶一提，型別提示是什麼樣子呢？變數宣告和函式回傳值各有不同的語法。

變數型別提示可能只包含型別：

```
name: type
```

亦可使用初始值來宣告變數：

```
name: type = value
```

型別（*type*）可以是標準 Python 簡單型別之一，如 `int` 或 `str`；或者集合型別如 `tuple`、`list` 或 `dict`：

```
thing: str = "yeti"
```

在 Python 3.9 之前，需要從 typing 模組匯入這些標準型別名稱的大寫版本：

```
from typing import Str
thing: Str = "yeti"
```

含初始值的變數宣告範例：

```
physics_magic_number: float = 1.0/137.03599913
hp_lovecraft_noun: str = "ichor"
exploding_sheep: tuple = "sis", "boom", bah!"
responses: dict = {"Marco": "Polo", "answer": 42}
```

也可以包含集合的子型別：

```
name: dict[keytype, valtype] = {key1: val1, key2: val2}
```

typing 模組有一些有用的子型別擴充功能，最常見的如下：

`Any`

任何型別

`Union`

指定的任何型別之一，例如 `Union[str, int]`

在 Python 3.10 及以上版本，可以使用 `type1 | type2` 來代替 `Union[type1, type2]`。

Pydantic 定義 Python `dict` 的例子包括以下幾種：

```
from typing import Any
responses: dict[str, Any] = {"Marco": "Polo", "answer": 42}
```

或者，更具體一點：

```
from typing import Union
responses: dict[str, Union[str, int]] = {"Marco": "Polo", "answer": 42}
```

或者（Python 3.10 及以上版本）：

```
responses: dict[str, str | int] = {"Marco": "Polo", "answer": 42}
```

注意，帶有型別提示的變數行是合法的 Python 語法，但單獨一行變數不是：

```
$ python
...
>>> thing0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name thing0 is not defined
>>> thing0: str
```

此外，正常的 Python 直譯器不會捕捉到不正確的型別使用：

```
$ python
...
>>> thing1: str = "yeti"
>>> thing1 = 47
```

第三部分

建立網站

第二部分提供 FastAPI 的快速概覽，讓你能迅速掌握其基本概念，接下來，將更深入且全面地探討其細節。在這一部分中，我們將著手開發一個中型網路服務，用於存取和管理有關虛構神祕生物（cryptids）及虛構探險者的資料。

如先前所討論，一個完整的服務架構通常包含三個層級：

網站

網站介面

服務

商業邏輯

資料

整個系統的寶貴 DNA

此外，這個網路服務還將有以下跨層元件：

模型

Pydantic 資料定義

測試

單元、整合和端對端測試

第十三章

正式環境

如果建築工人像軟體工程師寫程式那樣蓋房子，
恐怕第一隻來的啄木鳥就足以摧毀整個文明了。

—Gerald Weinberg，電腦科學家

預覽

假設你有一個在本機上執行的應用程式，而你想與他人分享，本章將介紹將應用程式部署到正式環境，並確保它能夠正確且高效地執行的多種情境。由於某些細節可能非常複雜，在某些情況下，我會適當地引用外部文件，而不是在此塞入過多冗長的內容。

部署

到目前為止，本書中的所有程式碼範例都使用單一實例的 `uvicorn`，在 `localhost` 的 `8000` 連接埠上執行。然而，為了處理大量流量，使用多個多核心伺服器執行在效能足夠的硬體上不可或缺，且還需要在這些伺服器上執行以下工作工作：

- 確保服務持續運作（監督程式（*supervisor*））
- 蒐集並分發外部請求（反向代理（*reverse proxy*））
- 回傳回應
- 提供 HTTPS 終止（SSL 解密）

多個工作行程

你可能聽過另一個叫做 Gunicorn 的 Python 伺服器（*https://gunicorn.org*），Gunicorn 能夠管理多個工作行程，但它是一個 WSGI 伺服器；而 FastAPI 則是基於 ASGI 規範所開發。幸好有一個特殊的 Uvicorn 工作行程類別 Gunicorn 可以管理。

範例 13-1 在 `localhost` 的 `8000` 連接埠上設定這些 Uvicorn 工作行程，改編自官方文件（*https://oreil.ly/Svdhx*）。引號是為了保護 shell 不會對命令進行特殊解釋。

範例 13-1　使用 Gunicorn 搭配 Uvicorn 工作行程

```
$ pip install "uvicorn[standard]" gunicorn
$ gunicorn main:app --workers 4 --worker-class \
uvicorn.workers.UvicornWorker --bind 0.0.0.0:8000
```

你會看到很多行輸出，顯示 Gunicorn 正在執行指令。它會啟動一個頂層的 Gunicorn 行程，與 4 個 Uvicorn 工作子行程通訊，所有程式共享 `localhost`（`0.0.0.0`）的 `8000` 埠，如果想要其他設定，可以更改主機、連接埠或工作行程數量。`main:app` 指的是 *main.py* 和名為 `app` 的 FastAPI 物件。Gunicorn 文件（*https://oreil.ly/TxYIy*）聲稱：

> *Gunicorn 應該只需要 4-12 個工作行程，就能處理每秒數百或數千個請求。*

事實證明，Uvicorn 本身也可以啟動多個 Uvicorn 工作行程，如範例 13-2 所示。

範例 13-2　使用 Uvicorn 搭配 Uvicorn 工作行程

```
$ uvicorn main:app --host 0.0.0.0 --port 8000 --workers 4
```

但這種方法不進行行程管理，所以通常還是偏好使用 gunicorn 方法。Uvicorn 還有其他行程管理器：請參見其官方文件（*https://www.uvicorn.org/deployment*）。

這解決了前一節提到的 4 項工作中的 3 項，但不包括 HTTPS 加密。

HTTPS

FastAPI 官方的 HTTPS 文件（*https://oreil.ly/HYRW7*）內容全面且深入，一貫保持 FastAPI 官方文件的高水準。我建議先閱讀這些文件，再參考 Ramírez 的說明（*https://oreil.ly/zcUWS*），了解使用 Traefik（*https://traefik.io*）為 FastAPI 加入 HTTPS 支援的方法。Traefik 位於網站伺服器「外層」，類似於 nginx 作為反向代理和負載平衡器的角色，但額外提供 HTTPS 的功能。

API 讀取所有探險家資料的時間遠遠超過資料層的讀取時間，這部分可能是因為 FastAPI 將回應轉換為 JSON 的額外開銷所致。此外，初次寫入資料庫的速度也不盡理想，由於資料層 API 只有單一的 `create()` 函式，而沒有 `create_many()` 函式，因此每次只能寫入一個探險家的資料。在讀取方面，API 可以透過 `get_one()` 回傳單一資料，或透過 `get_all()` 回傳所有資料。因此，若要批量載入，可能需要新增一個專門的資料載入函式，和一個具有受限權限的新網站端點。

此外，如果預期資料庫中的任何表格可能增長至超過 10 萬筆資料，就不應該允許一般使用者在單次 API 呼叫中獲取所有資料。碰到這種情況，實作分頁功能會很有幫助，或者提供一種可從表格下載單一 CSV 檔案的方法。

資料科學和 AI

Python 已成為資料科學領域，特別是機器學習領域中最受歡迎的程式語言，因為它在處理大量資料方面表現出色，這正是這個領域所需要的。

開發人員有時會使用像 pandas 這樣的外部工具（*https://oreil.ly/WFHo9*），來協助 SQL 中難以處理的資料操作。

PyTorch（*https://pytorch.org*）是目前最熱門的機器學習工具之一，它充分利用了 Python 在資料處理方面的優勢。雖然底層計算可能會使用 C 或 C++ 來提高速度，但 Python 或 Go 非常適合處理「較高層級」的資料整合工作。Mojo 語言（*https://www.modular.com/mojo*）是 Python 的超集，如果能按計畫成功發展，可能會同時處理高層和低層工作。儘管它是一種通用語言，但特別能解決目前 AI 開發中的一些複雜問題。

名為 Chroma（*https://www.trychroma.com*）的新興 Python 工具是一個類似於 SQLite 的資料庫，但專為機器學習，特別是大型語言模型（LLMs）量身打造。建議閱讀其入門頁面（*https://oreil.ly/W59nn*）以開始使用。

儘管 AI 開發複雜且發展迅速，你仍然可以在自己的電腦上使用 Python 來嘗試一些 AI 應用程式，而無需投入像 GPT-4 和 ChatGPT 背後那樣的巨額資金。以下為一個小型 AI 模型建立簡單的 FastAPI 網頁介面。

值得注意的是，在 AI 和 Pydantic/FastAPI 中，模型這個詞有不同的含義。在 Pydantic 中，模型是一個將相關資料欄位組合在一起的 Python 類別；而 AI 模型則涵蓋了一系列用於辨識資料模式的廣泛技術。

Hugging Face（*https://huggingface.co*）提供免費的 AI 模型、資料集和使用它們的 Python 程式碼。首先，需要安裝 PyTorch 和 Hugging Face 的程式碼庫：

```
$ pip install torch torchvision
$ pip install transformers
```

範例 14-5 是一個 FastAPI 應用程式，它使用 Hugging Face 的 transformers 模組，來存取預先訓練的中型開源機器語言模型，並嘗試回答你的提問。（根據 YouTube 頻道「CodeToTheMoon」的命令列範例改編而來。）

範例 14-5　頂層 LLM 測試（ai.py）

```
from fastapi import FastAPI

app = FastAPI()

from transformers import (AutoTokenizer,
    AutoModelForSeq2SeqLM, GenerationConfig)
model_name = "google/flan-t5-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
config = GenerationConfig(max_new_tokens=200)

@app.get("/ai")
def prompt(line: str) -> str:
    tokens = tokenizer(line, return_tensors="pt")
    outputs = model.generate(**tokens,
        generator_config=config)
    result = tokenizer.batch_decode(outputs,
        skip_special_tokens=True)
    return result[0]
```

使用 `uvicorn ai:app` 執行此程式（如往常一樣，首先確保沒有其他網站伺服器在 `localhost` 的 `8000` 連接埠上執行）。向 */ai* 端點提出問題並獲得回答，如下，要注意 HTTPie 查詢參數使用雙等號 `==`：

```
$ http -b localhost:8000/ai line=="What are you?"
"a sailor"
```

這的確是一個相當小型的模型，從結果可以看出，模型回答問題的表現並不特別突出。我嘗試其他提示，如 `line` 引數，得到的回答也大致相同：

- 問：貓比狗好嗎？
- 答：不

資料探索與視覺化

預覽

雖然 FastAPI 的名字中包含 *API*，但它能提供的不僅僅是 API。本章將示範使用來自世界各地虛構生物的小型資料庫，產生表格、圖表、圖形和地圖的方法。

Python 和資料

近年來，Python 因多種原因變得非常熱門：

- 易於學習
- 簡潔的語法
- 豐富的標準庫
- 大量高品質的第三方套件
- 特別強調資料操作、轉換和內省

這個概念與傳統 ETL 工作有著密切的關聯，這些工作在資料庫建立中扮演著重要角色。值得一提的是，有一個名為 PyData（*https://pydata.org*）的非營利組織，專門舉辦研討會並開發工具，致力於運用 Python 進行開源資料分析。Python 的普及程度也反映近年來人工智慧（AI）的快速發展，以及為準備餵入 AI 模型所需資料而開發工具的迫切需求。

本章將嘗試幾個 Python 資料處理套件，並深入了解它們與現代 Python 網站開發以及 FastAPI 框架之間的關聯性。

PSV 文字輸出

本章節將使用附錄 B 中列舉的生物資料作為範例，這些資料儲存在本書的 GitHub 儲存庫中，包括管線分隔檔案 *cryptid.psv* 和 SQLite 資料庫檔案 *cryptid.db*。雖然逗號分隔（*.csv*）和定位分隔（*.tsv*）檔案格式在資料處理領域中相當普遍，但它們各有一些限制，逗號經常出現在資料儲存格的內容中；而定位字元（tab）有時會難以與其他空白字元明確區分。相較之下，管線字元（|）作為分隔符號具有明顯優勢：它在一般文字中較為罕見，因此能更有效地區隔資料欄位。

以下首先會探討 *.psv* 文字檔的處理方法。為了簡化示範，會先使用純文字輸出作為範例，接著，會進行一個更完整的網站開發範例，使用 SQLite 資料庫作為資料來源。

.psv 檔案的初始標頭行包含欄位名稱：

- `name`（名稱）
- `country`（國家）（`*` 表示多個國家）
- `area`（可選，美國各州或其他國家地區）
- `description`（描述）
- `aka`（又稱）

檔案的其餘行依序描述一種生物，欄位以 | 字元分隔。

CSV

範例 17-1 將生物資料讀入 Python 資料結構。首先，可以使用標準 Python csv 套件讀取管線分隔檔案 *cryptids.psv*，產生一個元組列表，每個元組代表檔案中的一行資料。（csv 套件還包括一個 `DictReader`，可以回傳字典列表。）此檔案的第一行是標頭，包含欄位名稱；如果沒有這個標頭，仍可以透過 csv 函式的引數提供標頭。

我在範例中加入型別提示，但如果你使用較舊版本的 Python，可以省略這些提示，程式碼依然能夠正常運作。這裡只列印標頭和前五行，好節省一些紙張[1]。

1 如果真有像托爾金筆下樹人那樣的樹木，誰都不希望他們某天晚上蹣跚地走到我們家門口來說教。