前言

電腦程式設計對尚未掌握技巧的人來說如同一種魔法。如果說程式設計就像魔法，那麼**網路爬取**就像巫術了，所謂的巫術，就是輕鬆寫意地施展魔法來實現令人印象深刻且實用的壯舉。

在我擔任軟體工程師的時期，很少程式設計實務可以像網路爬取一樣，讓程式設計師和外行人都備感期待。寫一個簡單的自主程式來蒐集資料，並將那些資料顯示在終端機上或儲存在資料庫裡並非難事，但無論你做過幾次，那種令人振奮的體驗和無限可能的感受都不會消失。

遺憾的是，當我和其他程式設計師討論網路爬取時，他們依然對這一門技術有一些誤解與困惑。有些人不確定這樣做是否合法（它是合法的），或不知道如何處理大量使用 JavaScript 的網頁，或需要登入網站才能爬取的情況。很多人不明白如何啟動大型的網路爬蟲專案，甚至不知道該去哪裡尋找資料。本書的目標是解答這些常見的疑問與誤解，並提供大多數網路爬取任務的完整指南。

網路爬取是一個多樣且變化迅速的領域，本書試著提供高階的概念和具體的範例，希望涵蓋你可能遇到的各種資料蒐集專案。在整本書中，我提供許多程式來示範這些概念，並讓你實際操作它們。這些範例程式可以自由使用和修改，無論你是否註明來源（當然，若能註明來源，我將感激不盡）。所有程式範例都可以在 GitHub 上閱讀和下載（*https://github.com/REMitchell/python-scraping*）。

什麼是網路爬取？

從網際網路自動蒐集資料的技巧幾乎與網路本身一樣古老。雖然**網路爬取**不是新詞，但這種做法以前通常稱為**畫面爬取**（*screen scraping*）、**資料挖掘**（*data mining*）、**網頁收割**（*web harvesting*）或類似的變體。現在大家傾向使用**網路爬取**（*web scraping*）一詞，因此本書將使用此名稱，儘管我本人也將專門用來瀏覽多個網頁的程式稱為**網路爬蟲**（*web crawler*），或乾脆將網路爬取程式稱為**機器人**（*bot*）。

理論上，網路爬取就是不透過 API 蒐集資料（顯然也不能依靠人力在網頁瀏覽器裡蒐集資料），通常是藉著編寫自動化的程式來完成，這種程式能查詢網頁伺服器、請求資料（通常是 HTML，和構成網頁的其他檔案），然後解析資料，以提取所需的資訊。

在實務上，網路爬取涵蓋各種程式設計技巧和技術領域，例如資料分析、自然語言解析，以及資訊安全。由於這個領域範圍如此廣泛，本書的第一部分將介紹網路爬取和爬蟲的基本原理，第二部分則深入探討進階主題。建議所有讀者仔細研讀第一部分，並根據需要，深入瞭解第二部分中較具體的內容。

為什麼要使用網路爬取？

如果你只會透過瀏覽器來使用網際網路，那麼你可能錯過非常多的可能性。雖然瀏覽器能夠方便地執行 JavaScript、顯示圖片，並以容易閱讀的格式來排列物件（以及其他功能），但網路爬蟲非常擅長快速蒐集和處理大量資料。它可以讓你同時查看涵蓋了數千頁甚至數百萬頁的資料庫，而不是僅僅透過狹窄的螢幕視窗一次瀏覽一個網頁。

此外，網路爬蟲可以到達傳統搜尋引擎無法觸及之處。當你在 Google 搜尋「飛到波士頓最便宜的機票」時，你會看到充斥著廣告和一些熱門機票搜尋網站的結果。Google 只認識那些網站在它們的網頁上呈現的資訊，卻無法知道在機票搜尋應用程式中輸入各種查詢的具體結果。然而，精心開發的網路爬蟲可以畫出每一個時間點飛往波士頓的機票價格（這些價格來自各式各樣的網站），並告訴你何時購買機票最划算。

你可能想問：「蒐集資料不就是 API 的功能嗎？」（如果你不熟悉 API，請參見第 15 章）。沒錯，API 確實很棒，但你得先找到符合需求的 API。API 的目的，是讓一個程式能夠輕鬆地從另一個程式接收格式整齊的資料。網路上有許多提供所需資料類型的 API，例如 Twitter 貼文或維基百科網頁。一般來說，如果有 API 可用，那麼使用它是比較好的選擇，而不是製作一個自主程式來抓取相同的資料。不過，由於諸多原因，你想要的 API 可能不存在，或不符合需求：

- 你需要從大量的網站蒐集相對較少且有限的資料，但那些網站並沒有統一的 API 可用。
- 你想要的資料相對較少或不常見，且創作者認為沒必要為它建立 API。
- 資料來源沒有基礎設施或技術能力可以製作 API。
- 資料很有價值而且（或）受到保護，不希望被廣泛傳播。

即使有 API 可用，但它的請求數量、速率限制、資料類型，或它提供的格式可能無法滿足你的需求。

這就是使用網路爬蟲的時機。除了少數的例外，瀏覽器顯示的資料幾乎都可以用 Python 程式來抓取。如果你知道如何寫程式來抓取資料，你就知道如何將它存入資料庫，如果你能夠將資料存入資料庫，你就幾乎可以使用那些資料來做任何事情。

近乎無限的資料顯然有許多極有幫助的用法，例如市場預測、機器語言翻譯，甚至醫學診斷，分別因為能夠從新聞網站、翻譯文本和健康論壇取得和分析資料而受益匪淺。

即使在藝術領域，網路爬取也開闢了創作的新天地。Jonathan Harris 和 Sep Kamvar 於 2006 年的專案「We Feel Fine」（*http://wefeelfine.org/*）中，從許多英語部落格網站抓取以「I feel」或「I am feeling」開頭的句子，產生一個極具人氣的資料視覺化專案，展示全球每一天、每分鐘的情感狀態。

無論你來自哪一個領域，網路爬蟲幾乎都可以幫助你更有效地指引商務實踐、提升生產力，甚至進入全新的領域。

關於本書

本書不僅是網路爬取的入門指引，也是一本全面的指南，包含從難以爬取的資料來源蒐集、轉換和使用資料的做法。儘管本書使用 Python 程式語言，且包含許多 Python 的基本內容，但請勿將它視為 Python 語言的入門書籍。

如果你完全不懂 Python，看懂這本書可能有點困難。請勿將此書當成 Python 入門教材。儘管如此，為了幫助廣泛的讀者吸收內容，我試著讓所有的概念和範例程式都維持在初階到中階 Python 程式設計的水準。此外，我會在適當的地方簡要地解釋一些較進階的 Python 程式設計和一般計算機科學主題。如果你是進階讀者，請放心地略過這些部分！

如果你正在尋找更全面性的 Python 資源，Bill Lubanovic 所著的《*Introducing Python*》（O'Reilly 出版）是一本不錯的指南，儘管篇幅較長。如果你的注意力較難長期集中，Jessica McKellar 的影片系列《*Introduction to Python*》（O'Reilly 出版）是極佳的資源。我也很喜歡我的教授 Allen Downey 所著的《*Think Python*》（O'Reilly 出版），這本書特別適合程式設計新手，它除了教導 Python 語言之外，也傳授計算機科學和軟體工程概念。

技術書籍往往專注於某一項語言或技術，但網路爬蟲是一種相對廣泛的主題，需要運用資料庫、網頁伺服器、HTTP、HTML、網路安全、圖片處理、資料科學，及其他工具的知識。本書試圖從「資料蒐集」的角度來涵蓋以上及其他相關主題。儘管如此，本書並非專門且完整地探討這些主題的著作，但我相信，我的解說足以幫助你開始編寫網路爬蟲！

第一部分將深入探討網路爬蟲和網路爬取主題，特別關注書中使用的少數幾個程式庫，可以當成這些程式庫和技術的全面參考（在一些例外的情況下，將提供額外的參考資料）。這個部分教導的技術對每一位網路爬蟲創作者來說都有幫助，無論他們的具體目標或應用是什麼。

第二部分則介紹編寫網路爬蟲時可能有用的其他主題，但這些主題不一定對所有爬蟲而言都隨時有用。不幸的是，由於這些主題太廣泛了，我很難在有限的篇幅之中明確地總結，因此，我會經常指出其他的參考資源，讓你可以獲得更多的資訊。

本書的編排方式可以讓你輕鬆地在各章之間翻閱，尋找你需要的網路爬取技術或資訊。如果有某個概念或程式是基於之前的章節介紹的內容，我會明確地指出那些章節。

本書編排方式

本書使用下列的編排方式：

斜體字（*Italic*）

代表新術語、URL、email 地址、檔名，與副檔名。中文以楷體表示。

定寬字（`Constant width`）

在長程式或文章中，代表變數、函式名稱、資料庫、資料型態、環境變數、陳述式、關鍵字…等程式元素。

第 6 章

編寫網路爬蟲

到目前為止，你看到的網頁都是刻意設計的靜態頁面。在本章，我們要開始探索實際的問題，使用網路爬蟲來遍歷多個網頁，甚至多個網站。

網路爬蟲之所以稱為「爬蟲」，是因為它們會在網路上爬行。爬蟲的核心元素是遞迴，它們必須先讀取一個 URL 的網頁內容，檢查該網頁上的其他 URL，並繼續抓取**那些**網頁，如此反覆操作。

但請注意：有能力在網路上爬行，並不意味著你隨時都該這樣做。之前的範例展示的爬蟲適合在你要的資料都在同一個網頁裡時使用。但是在使用網路爬蟲時，你必須關注你使用了多少頻寬，並且盡量設法減輕目標伺服器的負擔。

遍歷單一網域

即使你沒聽過 Six Degrees of Wikipedia（維基百科六度分隔），你也可能聽過同名的 Six Degrees of Kevin Bacon（凱文貝肯六度分隔）[1]。這兩個遊戲的目標，都是將兩個看似不相關的主題串連起來（在前者中，你要串連互相連結的維基百科文章；在後者中，你要串連參演了同一部電影的演員），那一個關係鏈最多不能超過六個環節（包括兩個原始主題在內）。

1　這是一種在 1990 年代創做出來的流行遊戲，*https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon*。

例如，Eric Idle 參演了《*Dudley Do-Right*》，Brendan Fraser 也參演了該片；Brendan Fraser 又與 Kevin Bacon 一起參演了《*The Air I Breathe*》[2]。在這個例子中，從 Eric Idle 到 Kevin Bacon 的關係鍵只有三個主題長。

這一節要開始進行一項專案，它最終會成為「維基百科六度分隔」的解答搜尋器，能夠讓你從 Eric Idle 的網頁（*https://en.wikipedia.org/wiki/Eric_Idle*) 開始，找出到達 Kevin Bacon 的網頁（*https://en.wikipedia.org/wiki/Kevin_Bacon*）的最少連結。

但要不要考慮維基百科的伺服器負擔啊？

根據 Wikimedia Foundation（維基百科的母組織）的資料，該網站每秒接收大約 2,500 次的訪問，其中超過 99% 的訪問量前往維基百科網域（參見「Wikimedia in Figures」網頁中的「Traffic Volume」部分（*https://meta.wikimedia.org/wiki/Wikimedia_in_figures_-_Wikipedia#Traffic_volume*））。由於流量本來就很大了，你的爬蟲應該不會對維基百科的伺服器負載造成顯著的影響。然而，如果你大量執行本書的範例程式，或建立自己的專案來抓取維基百科，你或許可以捐一筆可抵稅的捐款給 Wikimedia Foundation（*https://wikimediafoundation.org/wiki/Ways_to_Give*），這不僅能夠彌補你對伺服器造成的負擔，也可以幫助別人獲得教育資源。

同時記住，如果你打算進行一個涉及維基百科資料的大型專案，你應該檢查一下那些資料是不是可以透過 Wikipedia API 取得（*https://www.mediawiki.org/wiki/API:Main_page*）。維基百科經常被用來展示爬蟲，因為它的 HTML 結構簡單且相對穩定。然而，透過它的 API 來提取相同的資料通常更有效率。

你應該知道如何編寫一個 Python 程式來抓取任意的維基百科網頁，並產生該網頁的連結列表了：

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://en.wikipedia.org/wiki/Kevin_Bacon')
```

2 感謝 The Oracle of Bacon（*http://oracleofbacon.org*）滿足了我對這個關係鍵的好奇心。

```
bs = BeautifulSoup(html, 'html.parser')
for link in bs.find_all('a'):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

觀察程式產生的連結列表可以發現，你想得到的電影文章都在裡面，包括《*Apollo 13*》、《*Philadelphia*》、《*Primetime Emmy Award*》，以及 Kevin Bacon 參演的其他電影。然而，你也會發現一些應該不想要的內容：

```
//foundation.wikimedia.org/wiki/Privacy_policy
//en.wikipedia.org/wiki/Wikipedia:Contact_us
```

事實上，維基百科的每一個網頁都充斥著側邊欄、頁腳和頁首的連結，也有連至分類網頁、討論網頁，和不包含其他文章的網頁的連結：

```
/wiki/Category:All_articles_with_unsourced_statements
/wiki/Talk:Kevin_Bacon
```

最近，我有一位朋友進行了類似的維基百科抓取專案，他提到，他寫了一個超過 100 行程式的大型過濾函式來判斷內部的維基百科連結是不是文章網頁。遺憾的是，他沒有事先尋找「文章連結」與「其他連結」之間的模式，否則他應該會發現竅門。檢查連往文章網頁（而不是其他的內部網頁）的連結，可以發現它們都有三個共同點：

- 它們位於 `id` 為 `bodyContent` 的 `div` 內。
- URL 裡面沒有冒號。
- URL 以 */wiki/* 開頭。

你可以利用這些規則，稍微修改程式碼，使用正規表達式 `^(/wiki/)((?!:).)*$` 即可抓取所需的文章：

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen('http://en.wikipedia.org/wiki/Kevin_Bacon')
bs = BeautifulSoup(html, 'html.parser')
for link in bs.find('div', {'id':'bodyContent'}).find_all(
    'a', href=re.compile('^(/wiki/)((?!:).)*$')):
    print(link.attrs['href'])
```

執行這段程式後，你會看到從 Kevin Bacon 的維基百科文章連出去的所有文章 URL 列表。

雖然這個可在寫死的維基百科文章裡找到所有文章連結的程式很有趣，卻沒有太大的實際用途。我們應該將這段程式轉換成這樣的程式：

- 一個函式 `getLinks`，讓它接收一個 `/wiki/<Article_Name>` 格式的維基百科文章 URL，並回傳一個包含它連接的所有文章的 URL 列表，讓列表裡的 URL 也有相同的格式。
- 一個主函式，讓它使用起始文章來呼叫 `getLinks`，從回傳的列表中隨機選擇一個文章連結，再次呼叫 `getLinks`，如此反覆執行，直到停止程式為止，或是直到新網頁裡找不到任何文章連結為止。

以下是實現這些功能的完整程式碼：

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import re

random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen('http://en.wikipedia.org{}'.format(articleUrl))
    bs = BeautifulSoup(html, 'html.parser')
    return bs.find('div', {'id':'bodyContent'}).find_all('a',
         href=re.compile('^(/wiki/)((?!:).)*$'))

links = getLinks('/wiki/Kevin_Bacon')
while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs['href']
    print(newArticle)
    links = getLinks(newArticle)
```

在匯入所需的程式庫之後，我們的程式先使用當下的時間來設置隨機數產生器的種子。這可以確保程式每次執行時，都會產生一條新的、有趣的維基百科文章隨機路徑。

偽隨機數與隨機種子

前面的範例使用 Python 的隨機數產生器在每一個網頁中隨機選擇一篇文章，進而繼續隨機遍歷維基百科。但是在使用隨機數時要很謹慎。

儘管電腦擅長計算正確的答案，但它們實在拙於捏造內容。因此，叫它們產生隨機數是一項挑戰。大多數的隨機數演算法都致力於產生均勻分布且難以預測

的數字序列，但這些演算法需要一個「種子」數來運作。使用同一個種子數來執行它們時，每次都會出現相同的「隨機」數字序列，所以我在此使用系統時鐘作為產生新隨機數序列的起點，從而產生新的隨機文章序列。這可讓程式跑起來更有趣一些。

如果你好奇的話，Python 的偽隨機數產生器使用的是 *Mersenne Twister 演算法*。雖然它能夠產生難以預測且均勻分布的隨機數，但這種演算法占用的處理器資源有點多。產生好的隨機數可不便宜！

接下來，程式定義了 `getLinks` 函式，該函式接收一個 `/wiki/...` 格式的文章 URL，並在前面加上維基百科的網域名稱 `http://en.wikipedia.org`，然後抓取該網域的 HTML 的 `BeautifulSoup` 物件。接著，它使用之前討論過的參數來取出一個文章連結標籤串列，並回傳它們。

程式的主體先將文章連結標籤串列（`links` 變數）設為初始網頁（*https://en.wikipedia.org/wiki/Kevin_Bacon*）裡面的連結串列。然後進入一個迴圈，在網頁中隨機找到一個文章連結標籤，從中提取 `href` 屬性，印出該網頁，然後從所提取的 URL 中取得一個新的連結串列。

要解決「維基百科六度分隔」問題，僅僅製作一個能夠從一個網頁爬到另一個網頁的爬蟲當然還不夠，你還要儲存並分析結果資料。第 9 章會提供這個問題的後續解決方案。

處理你的例外！

為了簡潔起見，這些範例程式省略了大多數的例外處理程式，但注意，你可能會遇到許多陷阱。例如，如果維基百科更改 `bodyContent` 標籤的名稱呢？當程式嘗試從該標籤提取文本時，它會丟出 `AttributeError`。

因此，雖然這些程式可以當成受到密切觀察的範例來執行，但自動化的程式還要加入其他的例外處理程序，受限於篇幅，本書的範例無法展示它們。詳情請參閱第 4 章。

第 11 章

處理髒資料

到目前為止，我都使用格式整齊的資料來源，或直接刪除格式混亂的資料，以避免格式混亂的資料造成的問題。但是在爬網時，你通常沒有選擇資料的來源或格式的自由。

由於錯誤的標點符號、不一致的大小寫、換行符號及拼寫錯誤，髒資料在網路上可能是個大問題。本章介紹的工具和技術，將幫助你在問題的根源預防它們，藉著改變程式的寫法，以及在將資料存入資料庫後清理它們。

這一章是網頁爬取領域和資料科學重疊的領域。雖然「資料科學家」這個職位可能讓人想到頂尖的程式設計技術和高深的數學，但事實上，他們的工作有很大一部分相當枯燥乏味。在建立機器學習模型之前，得要有人清理並標準化數百萬條用來訓練它的紀錄，那個人就是資料科學家。

提取、轉換、載入

提取、轉換、載入（Extract, Transform, Load，ETL）是程式設計經常使用的概念，尤其是在業界。許多公司都非常重視如何將資料從一個地方移動到另一個地方，並在過程中進行資料轉換。他們會從一個來源抽取資料，透過一些業務流程來轉換它們，然後將它們填入另一個來源。

本書大部分的內容討論的是 ETL 中的「E」，也就是從網路上的來源提取資料。第 9 章「儲存資料」涵蓋「L」，亦即將資料填入資料庫。

本章和第 12 章「讀寫自然語言」專注於 ETL 中的「T」，即資料的轉換。

將資料應用程式想成提取、轉換與載入三個階段可以幫助你組織流程與工具，建立乾淨的軟體架構。

清理文字

Python 是一種非常適合處理文字的程式語言。即使是面對複雜的文字處理專案，你也可以用 Python 來寫出乾淨、能執行功能、且模組化的程式碼。我們可以使用下面的程式碼來抓取維基百科上關於 Python 的文章，其網址為：*http://en.wikipedia.org/wiki/Python_(programming_language)*：

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

url = 'http://en.wikipedia.org/wiki/Python_(programming_language)'
html = urlopen(url)
bs = BeautifulSoup(html, 'html.parser')
content = bs.find('div', {'id':'mw-content-text'}).find_all('p')
content = [p.get_text() for p in content]
```

這段內容的開頭是：

```
Python is a high-level, general-purpose programming language. Its
design philosophy emphasizes code readability with the use of
significant indentation via the off-side rule.[33]
```

我們將對這段文字執行以下操作：

- 移除 [123] 這種格式的引用代號
- 移除換行符號
- 將文本拆成短句
- 移除在句子之中，放在小括號裡面的插入式說明文字
- 移除未被提取至文本中的範例的敘述
- 將文字改成全小寫
- 移除所有標點符號

第 15 章

透過 API 來爬取資料

JavaScript 一直以來都是網路爬蟲的天敵。在網際網路的蠻荒時代，當你向網頁伺服器要求取得一個 HTML 網頁時，保證會在瀏覽器中看到同一個 HTML 網頁。

隨著 JavaScript 和 Ajax 內容生成 / 載入程式越來越普遍，這種情況已經不常見了。在第 14 章，你已經學會一種解決這種情況的方法：使用 Selenium 來自動化瀏覽器並抓取資料。這件事很容易做，而且幾乎一定有效。

問題在於，當你握著一把 Selenium 這種強大且有效的「錘子」時，每一個網頁爬取問題看起來都像是「釘子」。

在本章，你將學習如何完全繞過 JavaScript（不需要執行它，甚至載入它！），直達資料的來源，也就是產生這些資料的 API。

API 簡介

儘管目前已經有無數的書籍、演講和指南專門討論 REST、GraphQL、JSON 和 XML API 的各種細節，但它們的核心概念相當簡單。*API*（Application Programming Interface，應用程式介面）定義了一種標準化的語法，可讓一個軟體與另一個軟體溝通，即使它們是用不同的語言寫成的，或是有不同的結構。

本節專門探討 web API（尤其是讓網頁伺服器與瀏覽器溝通的 API），本節的 *API* 一詞專指這一種類型。不過，你要記得，在其他情境下，API 也是一個泛用術語，（舉例）

可以讓一個 Java 程式與同一台機器上的 Python 程式溝通。API 並非都「跨越網際網路」，也不一定涉及任何 web 技術。

使用 web API 的人通常是使用公共服務的開發者，那些公共服務都被廣泛宣傳並具備完整說明文件。例如，美國國家氣象局製作了一個氣象 API（*https://www.weather.gov/documentation/services-web-api*），它可以提供任何地點的即時天氣資料和預測。Google 在它的 Developers 區域（*https://console.developers.google.com*）提供了數十個 API，用於語言翻譯、分析及地理定位。

這些 API 的文件通常將路由或**端點**描述成可以請求的 URL，而且在 URL 的路徑裡，或是在 `GET` 的參數中，使用一些變數參數。

例如，以下這段程式透過路由路徑的參數來提供 `pathparam`：

```
http://example.com/the-api-route/pathparam
```

下面這段程式則將 `pathparam` 設成參數 `param1` 的值來提供：

```
http://example.com/the-api-route?param1=pathparam
```

這兩種將變數資料傳給 API 的做法都很常見。不過，就像電腦科學的許多主題一樣，關於「應該在什麼時候，以及應該在哪裡，透過路徑或參數來傳遞變數」的哲學辯論一直沒有停止。

API 的回應通常採用 JSON 或 XML 格式。目前 JSON 比 XML 更受歡迎，但你應該還會看到一些 XML 回應。許多 API 允許你使用另一個參數來定義你想要獲得哪一種類型的回應，從而改變回應的格式。

以下是 JSON 格式的 API 回應：

```
{"user":{"id": 123, "name": "Ryan Mitchell", "city": "Boston"}}
```

這是 XML 格式的 API 回應：

```
<user><id>123</id><name>Ryan Mitchell</name><city>Boston</city></user>
```

ip-api.com 有一個簡單易用的 API 可以將 IP 位址轉換為實際的物理地址。你可以在瀏覽器輸入以下內容來試著發出簡單的 API 請求[1]：

1 這個 API 可將 IP 位址解析為地理位置，在本章稍後，你還會使用這個 API。

第 20 章

網路爬蟲代理

這一章很適合當成本書的最後一章。到目前為止，你都是在自己電腦上的命令列執行所有的 Python 應用程式。有道是：「如果你真的愛它，那就讓它自由吧！」

也許你認為現在還不需要這樣做，等以後再說，但是當你再也不使用筆電來執行 Python 爬蟲時，你可能會意外地發現，生活怎麼變得如此輕鬆。

更重要的是，自從本書的第一版於 2015 年出版以來，整個網路爬蟲代理產業已經迅速地崛起並蓬勃發展。以前，如果你想要請人幫你運行網路爬蟲，你只要支付雲端伺服器費用，然後像運行其他軟體一樣，在伺服器上運行爬蟲。但現在，你可以發出一個 API 請求，實質上指示「抓取這個網站」，遠端程式就會處理所有細節、任何安全問題，並將資料回傳給你（當然要收錢！）。

本章將探討一些透過遠端 IP 位址來路由請求、在其他地方託管並運行你的軟體，甚至將工作完全交給網路爬蟲代理來處理的方法。

為什麼要使用遠端伺服器？

當你推出一個想讓廣大用戶使用的網頁應用程式時，使用遠端伺服器看起來是個再明顯不過的做法，但程式設計師為了自用而開發的工具通常是在本地執行的。如果沒有將程式移到其他地方的動機，為什麼要這麼做？移動它的原因可以歸類為兩種情況：需要使用另一個 IP 位址（也許你的 IP 位址被封鎖了，或是為了防止它被封鎖），以及需要更強大的運算能力和靈活性。