前言

程式設計對不具備此技能的人是某種魔術。若程式設計是魔術，那麼**網站擷取**是一種魔法：特別實用的應用程式 - 且不花什麼功夫 - 的功能。

我在擔任軟體工程師的日子中發現少有像網站擷取一樣有趣。無論做個多少次，撰寫簡單的程式將搜集到的資料輸出到螢幕或儲存進資料庫總是很有趣。

不幸的是，很多程式設計師在討論到網站擷取時都有誤解。有些人不確定它是否合法（確實合法），或不知道如何處理很多 JavaScript 的網頁。許多人搞不清楚如何進行網站擷取專案或從哪裡找資料。這本書嘗試解決網站擷取的常見問題與誤解並提供常見網站擷取任務的指南。

網站擷取經常發生變化，我嘗試提供高階概念與你可能會遇到的資料搜集實務範例。本書程式範例展示這些概念並能讓你自己嘗試。範例程式可自由使用（最好能提及出處）。程式範例可從 GitHub（*http://www.pythonscraping.com/code/*）檢視與下載。

網站擷取是什麼？

從網際網路自動搜集資料的歷史幾乎與網際網路同齡。雖然**網站擷取**不是新名詞，但這種動作過去有時被稱為**螢幕擷取**、**資料挖礦**、**網路收割**等。現在似乎都偏好使用**網站擷取**，因此本書也採用這一詞，但我也偏向將遍歷多個網頁的程式稱為**網站爬行程序**或**機器人**。

理論上，網站擷取是除透過 API（或人透過瀏覽器）外以各種方法搜集資料。這通常由自動化程式查詢網站伺服器、請求資料（HTML 形式以及其他組成網頁的檔案）、然後解析資料以取出所需資訊。

實務上，網站擷取由各種程式設計技巧與技術組成，例如資料分析、自然語言解析與資訊安全等。由於範圍很大，這本書在第一部涵蓋基本網站擷取與爬行並在第二部深入進階主題。建議所有讀者仔細研讀第一部並在有需要時深入第二部特定內容。

為什麼要擷取網站？

如果只能透過瀏覽器存取網際網路，則你少了很多可能。雖然瀏覽器能執行 JavaScript、顯示圖像、以人類易讀的方式排列內容，網站擷取擅長搜集與處理大量資料。相較於從電腦螢幕檢視一頁，你可以瀏覽包含萬千網頁的資料庫。

此外，網站擷取程序可做傳統搜尋引擎做不到的事。Google 搜尋“波士頓便宜機票”會顯示很多廣告與旅行社網站。Google 只知道這些網站上網頁的內容而不是各種航班查詢結果。但一個寫好的網站擷取程序可顯示多個網站上各種時段到波士頓的航班，並告訴你最好的時段。

你可能會問：“API 不就是做這個嗎？”（API 見第 12 章）。是的，API 很棒，如果剛好符合你的需求的話。它們提供電腦程式間格式化資料的交換。你可以找到各種資料的 API，例如 Twitter 發文或 Wikipedia 網頁。一般來說，有 API（如果有）比建構機器人去取得相同資料更好。但 API 可能不存在或不符你的需求：

- 你想從大量沒有相應 API 的網站中搜集特定資料。
- 你需要的資料量很獨特且對方沒有想到要提供 API。
- 資料來源沒有能力提供 API。
- 有價值的資料被保護且不打算大量傳播。

就算有 API，但可能資料量受限或格式不符你的需求。

這個時候就需要網站擷取。除了少數例外，你可以用瀏覽器看到的資料都可以透過 Python 腳本存取。若可以透過腳本存取，則可以將它儲存在資料庫中。若能夠存在資料庫，可以對資料做任何事。

有很多特定應用程式可存取各種資料：市場預測、語言翻譯甚至是醫療也可以從分析新聞網站、翻譯文字與健康論壇的資料中獲益。

就算是藝術，網站擷取也開啟新的創作方式。“We Feel Fine”（*http://wefeelfine.org/*）計劃從各種英文部落格網站擷取“I feel”或“I am feeling”開頭的句子產生資料圖表來描述每時每刻全世界的感受。

無論是什麼領域，網站擷取都能提供某種生意指引、改善生產力、甚或產生全新的領域。

關於本書

這本書不只介紹網站擷取，還包含從不合作來源搜集、轉換與使用資料的指南。雖然使用的是 Python 程式設計語言並包含基本介紹，但不應作為此語言的入門。

若你完全不懂 Python，這本書可能很難消化。不要用這本書學 Python。我盡量保持概念與範例的簡單以讓各種程度的讀者都能夠吸收。厲害的讀者可以跳過進階 Python 程式設計與一般電腦科學主題的解釋部分！

更詳盡的 Python 資源可見 Bill Lubanovic 所著的《**精通** *Python*》（歐萊禮）。時間不多的人可參考 Jessica McKellar 的 Introduction to Python 系列影片（歐萊禮）。我也喜歡 Allen Downey 的《*Think Python*：**學習程式設計的思考概念**》（歐萊禮），這本書特別適合程式設計新人，其內容是以 Python 語言介紹電腦科學與軟體工程概念。

專業書籍通常專注於單一語言或技術，但網站擷取是範圍很廣的題目，需要碰到資料庫、網頁伺服器、HTTP、HTML、網際網路安全、圖像處理、資料科學與其他工具。本書嘗試從“資料搜集”方面涵蓋所有內容。本書不應作為個別主題的完整教材，但我認為已經涵蓋足夠讓你起步的細節！

第一部包含網站擷取與網路爬行並深入本書用到的幾個函式庫。第一部可作為這些函式庫與技術的詳細參考（特定部分會提供其他參考），在這部分討論的技巧對撰寫網站擷取程序很實用。

第二部包含撰寫網站擷取程序時可能會有用的其他主題。這些主題太廣泛，因此會列出其他資源供參考。

本書結構讓你容易在章節中找尋所需的資訊。當某個概念或程式需要依賴前面的內容時，我會不厭其煩的特別標示出來。

第一部

建構擷取程序

第一部專注於網站擷取的基本機制：使用 Python 從網頁伺服器請求資訊、處理伺服器回應，以及與網站互動的自動化。你將會學習如何建置可跨越網域搜集並儲存資料的擷取程序。

老實說，擷取網站是小投資大回報的領域。90% 的擷取專案可能只需動用到接下來六章所討論的技術。第一部涵蓋大部分人（懂技術的人）所知道的“網站擷取程序”：

- 從域名取得 HTML 資料
- 解析所需資料
- 儲存資訊
- 或許移動到其他頁重複此程序

這會在前進到第二部之前為你打好基礎。不要以為第一部的主題就比第二部的進階主題較不重要。撰寫網站擷取程序時你會需要第一部的所有資訊！

第一章

你的第一個擷取程序

開始寫網站擷取程序時就會知道瀏覽器幫你做了多少事情。沒有了 HTML 的格式、CSS 的樣式、JavaScript 的功能，網站乍看之下很可怕，但這一章與下一章會討論如何在沒有瀏覽器的幫助下解析與排列資料。

這一章從基本的發送 `GET` 請求（請求取得網頁內容）開始，然後執行簡單的資料擷取以抽取你所需的內容。

連線

如果沒有研究過網路或網路安全，你可能看不懂網際網路的機制。你未曾想要知道打開瀏覽器輸入 *http://google.com* 時網路如何運作，你也無需知道。事實上，我認為網路界面很棒的一點是，大部分網路使用者不需要知道它是如何運作的。

但網站擷取必須要深入這個界面——不只是瀏覽器的層次（如何解析 HTML、CSS、JavaScript），還要深入網路連線。

為理解瀏覽器取得資訊所需的基礎建設，讓我們看個例子。Alice 有個網頁伺服器。Bob 使用電腦嘗試連線 Alice 的伺服器。一台電腦想要與另一台電腦交談時會發生下列動作：

1. Bob 的電腦發送一系列的 0 與 1，實際上是網路線上的電壓起伏。這些位元組成某種資訊，含有標頭（header）與內文（body）。標頭帶有本地（local）路由器（router）的 MAC 地址與 Alice 的 IP 地址。內文帶有向 Alice 的伺服器發出的請求（request）。

2. Bob 的本地路由器接收這些 0 與 1 並解譯成帶有 Bob 自己的 MAC 地址與 Alice 的 IP 地址的封包（packet）。他的路由器在封包加上自己的 IP 地址作為“from”的 IP 地址並從網路發送出去。
3. Bob 的封包經由實體線路穿越網路上的伺服器到達 Alice 的伺服器。
4. Alice 的伺服器以她的 IP 地址接收到該封包。
5. Alice 的伺服器讀取封包標頭的埠（port）位址並傳給對應的應用程式——網頁伺服器應用程式（網頁伺服器的封包埠號通常是 80；它可以視為封包資料的門牌號碼，而 IP 地址是街道名稱）。
6. 網頁伺服器應用程式接收伺服器處理程序的資料流（stream）。資料帶有類似下列訊息：

   - 這是個 GET 請求。

   - 請求的是這個檔案：index.html。
7. 網頁伺服器找出正確的 HTML 檔案，將它包裝成封包透過本地的路由器以相同程序寄回 Bob 的電腦。

好了！這就是網際網路。

這個過程中瀏覽器參與什麼工作？完全沒有。事實上，以 1990 年代出現的 Nexus 來說，瀏覽器是網際網路上相對新的發明。

沒錯，瀏覽器是建構這些資訊封包、告訴作業系統送出封包、將取得的資料解譯成圖像、聲音、影片、文字的應用程式，但瀏覽器只是程式碼，而此程式碼能執行你的要求。瀏覽器能要求處理器發送資料給處理無線（或有線）界面的應用程式，但你也能以幾行 Python 程式做到同樣的事情：

```
from urllib.request import urlopen

html = urlopen('http://pythonscraping.com/pages/page1.html')
print(html.read())
```

你可以使用 GitHub 程式庫（*https://github.com/REMitchell/python-scraping/blob/master/Chapter01_BeginningToScrape.ipynb*）上的 iPython 執行，或儲存成 scrapetest.py 並在命令列中以下列命令執行：

```
$ python scrapetest.py
```

請注意，若電腦上還有安裝 Python 2.x 版本，就必須明確的以下列命令呼叫 Python 3.x 版本：

```
$ python3 scrapetest.py
```

這個命令輸出 *http://pythonscraping.com/pages/page1.html* 的完整 HTML。更精確的說法是輸出 *http://pythonscraping.com* 網域上的伺服器 *<web root>/pages* 目錄下的 *page1.html* 這個 HTML 檔案。

為何將這些地址視為“檔案”而非“網頁”很重要？大部分網頁都有很多相關的資源檔案，包括圖像檔案、JavaScript 檔案、CSS 檔案與請求網頁內連結的其他內容。瀏覽器遇到 `<img src="cuteKitten.jpg">` 這樣的標籤（tag）時，瀏覽器知道要發出另一個請求給伺服器以取得 *cuteKitten.jpg* 檔案的資料以完整繪製網頁。

當然，你的 Python 腳本（目前）沒有再取得其他檔案的功能；它只能讀取你請求的單一 HTML 檔案。

```
from urllib.request import urlopen
```

這一行的意思是：找出 request 這個 Python 模組（在 *urllib* 函式庫中）並只匯入 `urlopen` 這個功能。

urllib 是 Python 標準函式庫（表示你無需另行安裝其他東西），內含從網路請求資料、處理 cookie、處理標頭與 user agent 等 metadata 的功能。本書大量使用 urllib，因此建議閱讀此 Python 函式庫的文件（*https://docs.python.org/3/library/urllib.html*）。

`urlopen` 用於打開跨網路的遠端物件並讀取。由於是通用的功能（可讀取 HTML 檔案、影像檔案、或其他檔案），本書會經常用到它。

BeautifulSoup 簡介

> Beautiful Soup, so rich and green,
> Waiting in a hot tureen!
> Who for such dainties would not stoop?
> Soup of the evening, beautiful Soup!

BeautifulSoup 函式庫以**愛麗絲夢遊仙境**中的一首詩命名，故事中此詩由 Mock Turtle 吟誦（與維多利亞時代以牛而非龜作為材料的 Mock Turtle Soup 雙關語）。

如同夢遊仙境，*BeautifulSoup* 試著讓無厘頭有道理；它矯正不良 HTML 的一團亂以產生 XML 結構的 Python 物件。

安裝 BeautifulSoup

由於 *BeautifulSoup* 不是 Python 預設的函式庫，必須要另外安裝。如果你會安裝 Python 函式庫，可以略過這一段到下一節“安裝 BeautifulSoup”。

不會（或忘記如何）安裝 Python 函式庫的人，以下的方法可安裝多個函式庫，因此可供後續參考。

本書使用 BeautifulSoup 4 函式庫（又稱為 BS4）。完整 BeautifulSoup 4 安裝說明見 Crummy.com（*http://www.crummy.com/software/BeautifulSoup/bs4/doc/*）；Linux 上的基本安裝方法如下：

```
$ sudo apt-get install python-bs4
```

Mac 的安裝：

```
$ sudo easy_install pip
```

這會安裝 Python 套件管理員 *pip*。然後執行安裝命令：

```
$ pip install beautifulsoup4
```

再重複一次，若電腦上同時安裝了 Python 2.x 與 3.x，你必須明確的呼叫 `python3`：

```
$ python3 myScript.py
```

安裝套件時也要這麼做，不然套件可能會裝在 Python 2.x 而非 Python 3.x 下：

```
$ sudo python3 setup.py install
```

如果使用 pip，你也可以呼叫 `pip3` 來安裝 Python 3.x 的套件：

```
$ pip3 install beautifulsoup4
```

Windows 上的安裝與 Mac 以及 Linux 差不多。從下載網頁（*http://www.crummy.com/software/BeautifulSoup/#Download*）下載最新版 BeautifulSoup 4，解壓縮後執行：

```
> python setup.py install
```

BeautifulSoup 會在電腦上被識別為 Python 函式庫。你可以從 Python 命令列開啟並匯入以進行測試：

```
$ python
> from bs4 import BeautifulSoup
```

匯入動作應該會完成且沒有錯誤。

此外，Windows 上有個 *.exe* 版本的 pip 安裝程式（*https://pypi.python.org/pypi/setuptools*），所以你可以輕鬆的安裝與管理套件：

```
> pip install beautifulsoup4
```

以虛擬環境保持函式庫的整潔

若同時進行多個 Python 專案，或需要包裝專案與相關函式庫，或擔心函式庫的安裝衝突，你可以安裝 Python 虛擬環境以方便管理。

沒有虛擬環境下安裝 Python 函式庫是**全域**安裝，通常必須由管理員或 root 進行，且 Python 函式庫可供該電腦上的每個使用者與專案使用。虛擬環境的安裝很

簡單：

```
$ virtualenv scrapingEnv
```

啟動並使用稱為 scrapingEnv 的新環境：

```
$ cd scrapingEnv/
$ source bin/activate
```

啟動此環境後，你會在命令列提示看到環境名稱以提醒你目前的工作環境。接下來安裝的函式庫或執行的腳本都只在這個環境下。

在新啟用的 scrapingEnv 環境下，你可以安裝與使用 BeautifulSoup；例如：

```
(scrapingEnv)ryan$ pip install beautifulsoup4
(scrapingEnv)ryan$ python
> from bs4 import BeautifulSoup
>
```

你也可以使用 deactivate 命令離開環境，於是你不再能存取此虛擬環境下安裝的任何函式庫：

```
(scrapingEnv)ryan$ deactivate
ryan$ python
> from bs4 import BeautifulSoup
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ImportError: No module named 'bs4'
```

分離專案的函式庫也方便將整個環境目錄壓縮寄給別人。只要他們也在電腦上安裝相同版本的 Python，你的程式就可以在此環境中運行而無需他們另外安裝任何函式庫。

雖然本書的範例沒有指導你使用虛擬環境，但你可以事先啟用虛擬環境。

執行 BeautifulSoup

BeautifulSoup 函式庫中最常用的物件是 BeautifulSoup 物件。讓我們修改前面的範例來看看它的運作：

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

html = urlopen('http://www.pythonscraping.com/pages/page1.html')
bs = BeautifulSoup(html.read(), 'html.parser')
print(bs.h1)
```

其輸出如下：

```
<h1>An Interesting Title</h1>
```

請注意，它只回傳網頁中的第一個 h1 標籤。習慣上，一頁應該只有一個 h1 標籤，但網路經常打破習慣，因此應該要注意它只會讀取第一個標籤，而不一定是你要的那一個。

前面的網站擷取範例匯入 urlopen 函式並呼叫 html.read() 以取得網頁的 HTML 內容。除了文字外，BeautifulSoup 也可以直接使用 urlopen 回傳的檔案物件而無需先呼叫 .read()：

```
bs = BeautifulSoup(html, 'html.parser')
```

然後 HTML 內容被轉換成具有以下結構的 BeautifulSoup 物件：

- **html** → <html><head>...</head><body>...</body></html>
  - — **head** → *<head><title>A Useful Page<title></head>*
    - — **title** → *<title>A Useful Page</title>*
  - — **body** → *<body><h1>An Int...</h1> Lorem ip... </body>*
    - — **h1** → *<h1>An Interesting Title</h1>*
    - — **div** → * Lorem Ipsum dolor... *

請注意，你從網頁擷取出的 h1 標籤是埋在 BeautifulSoup 物件兩層套疊內（html → body → h1），但要從物件中取得時是直接呼叫此 h1 標籤：

```
bs.h1
```

事實上，下列函式的呼叫都會產生相同的輸出：

```
bs.html.body.h1
bs.body.h1
bs.html.h1
```

建構 BeautifulSoup 物件時要傳入兩個參數：

```
bs = BeautifulSoup(html.read(), 'html.parser')
```

第一個是物件的 HTML 文字，第二個是你指定 BeautifulSoup 用於建構物件的解析程序。大部分情況下，用什麼解析程序沒差。

html.parser 是 Python 3 內含的解析程序，無需另行安裝。除非有另外要求，否則本書使用這個解析程序。

另一個常見的解析程序是 lxml（*http://lxml.de/parsing.html*）。它可以透過 pip 安裝：

```
$ pip3 install lxml
```

lxml 可透過更改解析程序參數而用於 BeautifulSoup：

```
bs = BeautifulSoup(html.read(), 'lxml')
```

lxml 比 html.parser 更好的一點是比較能處理“雜亂”或格式錯亂的 HTML。它能應付沒有關閉的標籤、套疊順序錯誤的標籤與沒有頭或身體的標籤。它比 html.parser 稍快，但速度快對網站擷取不一定有好處，因為網路速度通常才是瓶頸。

lxml 的缺點是必須另行安裝並相依第三方的 C 函式庫，這會導致可攜性與易用性的問題。

另一個常見的 HTML 解析程序是 html5lib。如同 lxml，html5lib 非常擅長處理糟糕的 HTML。它也有相依性問題，且較 lxml 與 html.parser 慢。除此之外，對付雜亂或手工打造的 HTML 網站時它也是個好選擇。

它可用於 BeautifulSoup 物件：

```
bs = BeautifulSoup(html.read(), 'html5lib')
```

希望以上說明能讓你了解 *BeautifulSoup* 函式庫的能力。只要有標籤在前後包圍，幾乎所有東西都可以從任何 HTML（或 XML）檔案中擷取。第 2 章會深入討論 BeautifulSoup 的函式與正規表示式以及如何用於 BeautifulSoup。

連線的可靠性與例外處理

網路亂七八糟，資料格式不良、網站當機、標籤沒有關閉。網站擷取最糟糕的體驗是執行擷取程序希望隔天都存到資料庫中——然後發現擷取程序因為遇到資料格式錯誤而很早的停止執行。在這種情況下，你可能會問候該網站開發者的長輩，但你應該怪的是你自己沒有預料到這個問題！

讓我們檢視此擷取程序匯入命令後的第一行程式並找出如何處理可能拋出的例外：

```
html = urlopen('http://www.pythonscraping.com/pages/page1.html')
```

這一行可能發生兩種主要的問題：

- 在伺服器上找不到這個網頁（或讀取有錯誤）
- 找不到伺服器

第一種狀況會回傳 HTTP 錯誤，可能是“404 PageNot Found”或“500 Internal Server Error”等。在這些情況下，urlopen 函式會拋出 HTTPError 例外。你可以如下處理這種例外：

```
from urllib.request import urlopen
from urllib.error import HTTPError

try:
    html = urlopen('http://www.pythonscraping.com/pages/page1.html')
except HTTPError as e:
    print(e)
    # 回傳 null、中斷、或執行“B 計劃”
else:
    # 程式繼續。注意：若返回或因例外
    # 中斷則無需使用“else”陳述
```

若回傳 HTTP 錯誤碼，程式會輸出錯誤且不會執行 else 陳述下面的其餘程式。

若找不到伺服器（例如 *http://www.pythonscraping.com* 關機或 URL 輸入錯誤），urlopen 會拋出 URLError。這表示找不到伺服器，且由於是遠端伺服器負責回傳 HTTP 狀態碼，HTTPError 不會被拋出，所以會捕捉到更嚴重的 URLError。你可以檢查是否為這種情況：

```
from urllib.request import urlopen
from urllib.error import HTTPError
from urllib.error import URLError

try:
    html = urlopen('https://pythonscrapingthisurldoesnotexist.com')
except HTTPError as e:
    print(e)
except URLError as e:
    print('The server could not be found!')
else:
    print('It Worked!')
```

當然，若成功從伺服器取得網頁，內容還是有可能不是你所預期的。每次成功存取 BeautifulSoup 物件中的一個標籤，最好要檢查以確定標籤確實存在。若嘗試存取不存在的標籤，BeautifulSoup 會回傳 None 物件。問題是，嘗試存取 None 物件上的標籤會拋出 AttributeError。

下面這一行（nonExistentTag 是個標籤而非 BeautifulSoup 的函式）

```
print(bs.nonExistentTag)
```

回傳 None 物件。此物件可處理與檢查。若不檢查而如下嘗試對此 None 物件呼叫其他函式：

```
print(bs.nonExistentTag.someTag)
```

這會回傳一個例外：

```
AttributeError: 'NoneType' object has no attribute 'someTag'
```

所以要如何防止這兩種狀況？最簡單的方法是明確的檢查：

```
try:
    badContent = bs.nonExistingTag.anotherTag
except AttributeError as e:
    print('Tag was not found')
else:
    if badContent == None:
        print ('Tag was not found')
    else:
        print(badContent)
```

檢查並處理每個錯誤乍看之下很麻煩，但這個程式很容易重新安排成比較好寫的方式（更重要的是比較好讀）。舉例來說，下面的程式以稍微不同的方式撰寫：

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

def getTitle(url):
    try:
        html = urlopen(url)
    except HTTPError as e:
        return None
    try:
        bs = BeautifulSoup(html.read(), 'html.parser')
        title = bs.body.h1
    except AttributeError as e:
        return None
    return title

title = getTitle('http://www.pythonscraping.com/pages/page1.html')
if title == None:
    print('Title could not be found')
else:
    print(title)
```

此例建構一個 getTitle 函式，回傳網頁標題或讀取有問題時回傳 None 物件。在 getTitle 中，你如同前面的例子一樣檢查 HTTPError 並將兩行 BeautifulSoup 包圍在一個 try 陳述中。兩者其中之一可能會拋出 AttributeError（伺服器不存在、html 是個 None 物件、html.read() 都可能拋出 AttributeError）。事實上，你可以在 try 陳述中包圍可能會拋出 AttributeError 的多行程式或呼叫其他函式。

撰寫擷取程序時，同時思考處理例外與可讀性的整體模式很重要。你也可能想要撰寫可重複使用的程式。類似 getSiteHTML 與 getTitle 的通用函式（具有例外處理）使網站擷取能夠很快——以及可靠——的完成。