# 前言

## 人生苦短

這是一本關於秘訣的書，內容包括：一般問題的解決方案、程式碼片段的重利用、解釋、範例，和簡短的教學指南。

本書的意圖在於為讀者節省時間。人們總是說，時間就是金錢，而時間也是個人生命的構成元素。我們的生命最好花費在創造新事物上，而不是用在對抗自己所犯的錯誤，或者解決那些已經被解決過的問題。我們對本書的期許是：讓所有讀者節省的時間，遠超過我們撰寫它所花費的時間。

- Ruby 程式語言本身就是一種非常節省時間的工具，與其他程式語言相比，它能讓你具有更高的生產力，因為，你會花費較多時間讓電腦做你想要做的事，而與程式語言本身相周旋的時間則較少。但是，對 Ruby 程式員來說，有許多事情需要花費時間，但實際上卻沒有完成什麼成果，下面是我們遇過的這類事情：
- 將時間花在撰寫常見演算法的 Ruby 實作上。
- 將時間花在偵錯常見演算法的 Ruby 實作上。
- 將時間花在發現和變通 Ruby 特有的缺陷上。
- 將時間花在應當自動執行的重複性工作上（包括重複性的編程工作!）。
- 將時間花在重複其他人已經公開的實作成果上。
- 將時間花在尋找執行某項工作的程式庫上。
- 將時間花在評估和抉擇執行某項工作的諸多程式庫上。
- 由於文件說明的匱乏或過時，必需花時間學習如何使用某個程式庫。

- 不敢接觸看似嚇人、實則有用的技術，因而將時間浪費在其他不良的解法上。

我們，以及本書的諸位貢獻者，對於自己過去所浪費的時間，依然感到歷歷在目。我們將自身的經驗粹取到本書中，冀望讀者不再重蹈覆策，浪費生命 — 或者，至少讓讀者愉快地將時間花費在其他更有趣的問題上。

我們的另一個目標，是增加讀者的興趣。如果你在閱讀本書之後，希望能夠使用 Ruby，透過演算法，產生動人的音樂，沒問題，秘訣 12.14 所提供的錦囊妙技將節省你的時間，讓你不必從頭開始，無中生有。本書中每個秘訣的形成和編寫，在理念上都具有下面兩個目標：節省你的時間，以及讓你的頭腦對新觀點保有活力。

# 讀者

本書所設定的讀者群，是那些至少懂一點 Ruby，或者對一般性編程具有一定程度瞭解的人。這不是一本 Ruby 的教學指南（欲查看真正的教學指南，請參見後面的〈其他資源〉一節），但是，如果你已經熟悉某些其他的程式語言，那麼，透過閱讀本書前 10 章，並且在過程中實作書中所列的程式碼，應該就能夠掌握 Ruby，無師自通。

本書所包含的秘訣適用於各種層次的讀者，從剛開始使用 Ruby 的初學者，到那些偶爾需要參考資料的專家。我們主要致力於一般性的編程技術，但也涵蓋特定的應用程式框架（像是 Ruby on Rails 和 GUI 程式庫）和最佳實務（像是單元測試）。

即使你只是準備將本書作為參考用書，仍然建議你將全書略讀一遍，以便對我們所解決的問題有個大致上的瞭解。這本書很厚，但並沒有解決所有的問題。如果在閱讀後，仍然找不到關於你自己問題的解決方案，或者對問題可能有幫助的訊息，那麼很抱歉，浪費你寶貴的時間。

如果你事先略讀本書一次，對本書所探討的問題自然會有清晰的概念，因而獲得更好的使用效率，知道這本書何時有幫助，何時又應當參考其他書籍、進行 Web 搜尋、詢問朋友、或者從其他途徑取得幫助。

# 本書組織架構

本書共分 23 章，每一章皆著力於一種類型的編程或者特定的資料型別。這裡的章節概述應當能夠讓你瞭解我們是如何劃分各類型的秘訣。每一章裡也包含對該章本身的詳實介紹，對當中各節作更詳細的描述。強烈建議讀者瀏覽下面的章節介紹和內容表列。

我們用前 6 章介紹 Ruby 的內建資料結構。

- 第 1 章：字串，包含建構、處理、和操作文字字串的秘訣。我們特別介紹若干有關正規表達式的秘訣（1.17 節至 1.19 節），但是，我們所關注的重點是 Ruby 特定的問題，而正規表達式則是一種非常通用的工具。如果讀者尚未使用過，或者對它感到恐懼，那麼，建議你閱讀線上教學指南，或者由 Jeffrey Friedl 所撰寫的《Mastering Regular Expressions》(O'Reilly)。
- 第 2 章：數字，介紹不同類型數字的表示，包括：實數、複數、任意精度的小數等等。還包括常見的數學和統計演算法的 Ruby 實作，並解釋在建立自己的數字類型時，會遇到的一些 Ruby 獨有的特性（2.13 節和 2.14 節）。
- 第 3 章：日期與時間，涵蓋 Ruby 在時間處理上的兩個介面：一個是奠基於 C 的時間程式庫，這在其他程式語言中比較常見；另一個則是純 Ruby 的實作，這比較合乎語言的習慣。
- 第 4 章：陣列，介紹 Ruby 最簡單的複合資料型別（compound data type）：陣列。陣列的許多方法實際上來自 Enumerable mixin【譯註 1】；這意味著你可以將這些秘訣應用在雜湊（hash）或其他的資料型別上【譯註 2】。Enumerable 的一些特性被涵蓋在本章之中（4.4 節和 4.6 節），還有一些特性則在第 7 章介紹。
- 第 5 章：雜湊，介紹 Ruby 的另一種基本複合資料型別：雜湊。雜湊讓物件與其名稱相關聯，以便爾後查詢（雜湊有時被稱為 " 查詢表 " 或 " 字典 "）。結合雜湊與陣列的使用，可以輕鬆地建構深化且複雜的資料結構。
- 第 6 章：檔案與目錄，包含讀取、寫入、和操作檔案的技術。Ruby 的檔案存取介面奠基於 C 的標準檔案程式庫，因此，你可能已經很熟悉。本章還涉及 Ruby 用於搜索和操作檔案系統的標準程式庫，許多秘訣將在第 23 章中再次說明。

譯註 1　mixin：混成（名詞）或混進（動詞）。在某些物件導向語言中，mixin 是一種收集功能性，供類別重利用的機制，但是，mixin 本身不是類別（class），也無法獨立存在。mixin 的優點在於促成程式碼的重利用，並且避開多重繼承的缺點。Ruby 利用模組（module）的概念完成 mixin 機制。大體上來說，模組是單一名稱空間下一組方法與常數（甚至實例變數、類別、或其他模組）的集合體，但不能用來直接建立實例（instance）。在類別定義中使用 "include"，可將模組裡所定義的內容「混進」類別定義裡。當多個類別 include 同一個模組時，它們會參照到同一份實例方法（而不是分別複製一份），但就實例變數而言，則是每個物件會有自己的一份實例變數。Java 與 Ruby 都不支援多重繼承，但在 Java 裡，可利用類別搭配介面（interface）達成多重繼承之效；而在 Ruby 裡，則是利用類別搭配 mixin，達到相同的效果。

譯註 2　這些是指有 "include" Enumerable 模組的資料型別。

前 6 章處理特定的演算法問題，接下來的 4 章比較抽象：牽涉到 Ruby 的語言特性和基本原理。如果你無法讓 Ruby 語言做你想要做的事，或者對以 Ruby「應有」的方式來撰寫程式感到捉襟見肘，那麼，下面這 4 章的秘訣將有所幫助。

- 第 7 章：*程式碼區塊與迭代*，包含探討 Ruby 程式碼區塊（也稱為閉包，closure）之各種可能性的秘訣。
- 第 8 章：*物件與類別*，介紹 Ruby 對物件導向編程的處理。包括撰寫不同類型的類別與方法的秘訣，以及幾個說明所有 Ruby 物件能力的秘訣（例如 freezing 和 cloning）。
- 第 9 章：*模組與名稱空間*，介紹 Ruby 的模組（module）。這種構造（construct）用於將新行為「混進」（mixin）既有類別中，以及將功能性分隔到不同的名稱空間中。
- 第 10 章：*反射機制與中介編程*，涵蓋以程式化手段探索和修改 Ruby 類別定義的技術。

第 6 章介紹基本的檔案存取，但對於特定檔案格式的著墨並不多。我們將利用下面 3 章的內容，介紹廣受歡迎的資料儲存方式。

- 第 11 章：**XML 與 HTML**，說明如何處理最受歡迎的資料交換格式。本章主要涉及剖析他人的 XML 文件和 Web 網頁（參見秘訣 11.9）。
- 第 12 章：*圖形與其他檔案格式*，介紹 XML 和 HTML 以外的其他資料交換格式，特別關注於圖形的生成與操作。
- 第 13 章：*資料庫和永續儲存*，涵蓋 Ruby 對資料儲存格式最好的介面，不管是將 Ruby 物件序列化到磁碟上，或是在資料庫中儲存結構化資料。從序列化資料和對文字內容做索引，到 Ruby 針對常見 SQL 資料庫所提供的客戶端程式庫，再到像 ActiveRecord 這類發展成熟的抽象層（讓你完全免於撰寫 SQL），本章針對其不同作法提出說明。

當前，Ruby 最普遍的應用在於網路應用程式（多數是透過 Ruby on Rails）。我們利用下面 3 章的內容，介紹不同類型的應用：

- 第 14 章：*網際網路服務*，透過說明以 Ruby 程式庫撰寫的各種客戶端和伺服端，開始介紹我們的網際網路服務。
- 第 15 章：**Web 開發：Ruby on Rails**，介紹近來讓 Ruby 形成風起雲湧之熱潮的 Web 應用程式框架。

- 第 16 章：**Web Service** 及分散式編程，介紹在 Ruby 程式裡於不同電腦之間共享資訊的兩種技術。為了使用 Web Service，你必須對其他電腦上的程式（通常是你沒有控制權的電腦）發出 HTTP 請求。Ruby 的 DRb 程式庫可讓你在執行於一組電腦上的不同程式之間，共享 Ruby 的資料結構。

接下來，我們利用下面 3 章的內容，介紹專案主要編程作業的輔助性工作。

- 第 17 章：測試、偵錯、最佳化、及文件化，重點放在處理例外狀況以及為程式碼建立單元測試，另外，還有一些關於偵錯和最佳化程序的秘訣。
- 第 18 章：打包與部署軟體，主要是處理 Ruby 的 Gem 打包系統，以及保存許多 gem 檔的 RubyForge 伺服器。其他章節中的許多秘訣要求你安裝特定的 gem，因此，如果你對 gem 不熟悉，建議你特別閱讀 18.2 節。本章也說明如何為自己的專案建立和部署 gem。
- 第 19 章：用 **Rake** 自動執行任務，介紹最流行的 Ruby 建置工具（build tool）。藉由 Rake，你可以讓像是執行單元測試，或者將程式碼打包成 gem 的常見任務自動化。儘管通常被用在 Ruby 專案中，但它是一種通用的建置語言，你可以將它用在任何可能使用 Make 的地方。

我們利用最後 4 章，介紹其他主題。

- 第 20 章：多工與多執行緒，說明如何利用多執行緒同時進行更多工作，以及如何使用 Unix 子程序執行外部命令。
- 第 21 章：使用者介面，介紹使用者介面（除了第 15 章介紹過的 Web 介面）。我們討論了命令列介面、搭配 Curses 和 HighLine 程式庫並且以字元為基礎的 GUI、為各種平台準備的 GUI 工具組、以及更多類型的特殊使用者介面（秘訣 21.11）。
- 第 22 章：以其他語言擴展 **Ruby**，主要的焦點在於：為效能考量或為了存取更多程式庫，將 Ruby 和其他語言連繫起來。本章大部分聚焦在存取 C 的程式庫，但也有一個關於 JRuby 的秘訣，JRuby 是執行在 Java 虛擬機器（JVM）上的 Ruby 實作（秘訣 22.5）。
- 第 23 章：系統管理，介紹眾多完成管理任務（通常使用其他章節的技術）的自我完備（self-contained）程式。這些秘訣側重於 Unix 上的管理任務（但是對 Windows 使用者也有一些資源可供參考（秘訣 23.2）），以及一些跨平台的指令稿（script）。

# 本書程式碼的運作方式

從「錦囊妙技」系列學習，意味著要實際執行書中的各個秘訣。某一些秘訣定義了大量的 Ruby 程式碼，你只需要將程式碼放進自己的程式中便可使用，甚至無需真正理解它們（秘訣 19.8 就是一個好例子）。然而，多數秘訣對於它所探討的技術都做了詳實的說明，而學習一項技術的最佳方式就是實際練習看看。

我們本著這樣的精神撰寫秘訣及其程式碼，大部分程式碼的行為以單元測試的方式表現，驗證該秘訣所描述的內容：反覆測試物件並秀出結果。

目前的 Ruby 安裝隨附一個稱為 irb 的互動式直譯器。在一個 irb 期程（session）中，你可以鍵入 Ruby 程式碼並且立刻看到輸出，而不必建立 Ruby 程式檔，再透過直譯器來執行。

大部分的秘訣以你能夠直接在 irb 期程中鍵入或複製 / 貼上的形式存在。要想深入研究某個秘訣，建議你從 irb 期程開始，並且在閱讀過程中，執行書中所列的程式碼，這樣做可以讓你對概念的理解比單純只是閱讀要來得更深刻一些。完成這一步之後，讀者可以對範例程式所定義的物件，進行更進一步的試驗。

有時候，為了將你的注意力吸引到 Ruby 表達式的預期結果，我們運用包含 ASCII 箭頭的 Ruby 註解，指向表達式的預期結果。這與 irb 用來告訴你每個輸入表達式所得結果的箭頭相同。

我們也使用文字化的註解，解釋某些程式碼片段。下面是某個秘訣中，用註解格式化過的 Ruby 程式碼片段：

```
1 + 2                          # => 3

# On a long line, the expected value goes on a new line:
Math.sqrt(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)
# => 7.41619848709566
```

為了顯示 Ruby 表達式的預期輸出，我們使用不含 ASCII 箭頭的註解，並且總是另起新行：

```
puts "This string is self-referential."
# This string is self-referential.
```

如果在 irb 中，省略註解，鍵入這兩段程式碼，你可以根據這些註解文字，驗證是否得到與我們相同的結果：

```
$ irb
irb(main):001:0> 1 + 2
=> 3
irb(main):002:0> Math.sqrt(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)
```

```
=> 7.41619848709566
irb(main):003:0> puts "This string is self-referential."
This string is self-referential.
=> nil
```

如果你閱讀的是本書的電子版，便可將程式碼片段複製並貼到 irb 中。Ruby 直譯器會忽略註解，但是，你可以用它們來確保你的答案與我們的相符，而不必回頭查看書中的文字（但是，你應該瞭解，如果由你自己鍵入程式碼，將有利於對這些程式碼的理解）。

```
$ irb
irb(main):001:0> 1 + 2      # => 3
=> 3
irb(main):002:0>
irb(main):003:0* # On a long line, the expected value goes on a new line:
irb(main):004:0* Math.sqrt(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)
=> 7.41619848709566
irb(main):005:0> # => 7.41619848709566
irb(main):006:0*
irb(main):007:0* puts "This string is self-referential."
This string is self-referential.
=> nil
irb(main):008:0> # This string is self-referential.
```

我們並未抄捷徑，大部分秘訣都是從頭到尾、完整說明的 irb 期程，並且包含我們試圖闡明重點所需的匯入或初始化工作。如果你正確地按照秘訣執行程式碼，應該能夠獲得與我們相同的結果【註】。這符合我們所謂「程式碼範例應該是所屬內容之單元測試」的理念。事實上，我們使用解析秘訣內容並執行其程式碼的 Ruby 指令稿，像進行單元測試一樣地測試我們的程式碼範例。

irb 期程技術並非總是有效。與 Rails 有關的秘訣必須執行在 Rails 中，與 Curses 有關的秘訣會掌控整個螢幕，無法與 irb 一同運作，因此，我們有時會以下列格式顯現獨立運作的檔案：

```
#!/usr/bin/ruby -w
# sample_ruby_file.rb: A sample file

1 + 2
Math.sqrt(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)
puts "This string is self-referential."
```

註　當程式行為依賴於當前時間、隨機數字產生器、或特定磁碟檔案的存在，你所得到的結果可能就不會與我們完全相同，但應該會類似。

只要有可能，我們還是會秀出執行此程式將得到的結果：可能是 GUI 程式的畫面擷取（screenshot），或者是程式在 Unix 命令列下執行的輸出紀錄：

```
$ ruby sample_ruby_file.rb
This string is self-referential.
```

注意，`sample_ruby_file.rb` 的輸出與鍵入 irb 的相同程式碼的輸出看起來不同。這裡沒有加法和平方根運算的痕跡，因為它們沒有產生輸出。

## 安裝軟體

Ruby 被隨附安裝在 Mac OS X 和大多數的 Linux 中，Windows 則否，但很容易就可以取得 One-Click Installer（「一點即通」的安裝程式）：參見 *http://rubyforge.org/projects/rubyinstaller/*。

如果使用者的 Unix/Linux 系統沒有安裝 Ruby（或者希望進行升級），那麼，其發行版本的套件系統（package system）可能有提供 Ruby 套件。在 Debian GNU/Linux 上，它是 ruby-[version] 套件：例如，ruby-1.8 或 ruby-1.9。Red Hat Linux 則稱它為 ruby；Mac OS X 上的 DarwinPorts 系統也是。

如果所有這些嘗試都失敗，你可以自行下載 Ruby 原始碼進行編譯。你可以透過 *http://www.ruby-lang.org/* 的 FTP 或 HTTP 取得 Ruby 原始碼。

本書中的許多秘訣都要求使用者以 Ruby gem 的形式安裝第三方程式庫。一般而言，我們偏愛使用最獨立的解決方案（僅使用 Ruby 標準程式庫），再來是使用 gem 的解決方案，最後才是使用其他第三方軟體的解決方案。

如果你對 gem 不熟悉，可以根據自己的需要參考第 18 章。一開始，只需要知道先從 *http://rubyforge.org/projects/rubygems/* 下載 Rubygems 程式庫（在網頁中選擇最新版本），解壓縮 tarball 或 ZIP 檔案，再到 `rubygems-[version]` 目錄，以 superuser 的權限，執行下列命令：

```
$ ruby setup.rb
```

Rubygems 程式庫包含在 Windows 版本的 One-Click Installer 裡，因此，Windows 上的使用者就不必操心這個步驟。

Rubygems 程式庫安裝完畢後，就很容易安裝其他的 Ruby 程式碼。如果某個秘訣提到像是 "Ruby on Rails 以 `rails` gem 的形式提供" 的事，你就可以在命令列鍵入下列命令（再次強調，以 superuser 的權限）：

```
$ gem install rails --include-dependencies
```

RubyGems 程式庫會下載並自動安裝 `rails` gem（以及任何其他依賴的 gem）。接著，你就能夠正確地執行該秘訣中的程式碼。

對於新的 Ruby 安裝版本來說，三個最有用的 gem 是 `rails`（如果你試圖建立 Rails 應用程式的話），以及 Ruby Facets 專案所提供的兩個 gem：`Facets_Core` 和 `Facets_More`。Facets Core 程式庫以普遍有用的方法擴展 Ruby 標準程式庫的類別。Facets More 程式庫則添加了全新的類別和模組。Ruby Facets 主頁（*http://facets.rubyforge.org/*）上有完整的參考。

有些 Ruby 程式庫（特別是一些比較老的）沒有被打包成 gem。本書所提到的大多數「非 gem」程式庫在 Ruby Application Archive（*http://raa.ruby-lang.org/*）上都有其進入點，RAA 是一個 Ruby 程式與程式庫的目錄。多數情況下，你可以從 RAA 下載 tarball 或 ZIP 檔，並且使用 18.8 節所描述的技術進行安裝。

## 平台差異、版本差異、與其他疑難雜症

除非特別註明，各秘訣所描述的觀念都是跨平台的，程式碼本身應該以相同模式在 Windows、Linux、和 Mac OS X 上執行。大多數的平台差異和平台特定的秘訣出現在最後幾章，包括：第 20 章、第 21 章、和第 23 章（參考第 6 章的介紹，關於 Windows 檔案名稱的說明）。

我們使用 Ruby 1.8.4 版和 Rails 1.1.2 版撰寫和測試秘訣，這是本書撰寫時最新的穩定版本。如果你執行的是 Ruby 1.9（本書撰寫時最新的非穩定版本）或 2.0，我們會提示你應當在程式碼裡某些地方做修改。

儘管我們盡了最大的努力，本書還是可能存在著某些特定於平台、但卻沒有標示出來的程式碼，更別提一些 bug。我們為此深感抱歉。如果你對某個秘訣有疑問，請查看本書的勘誤表（參見下面內容）。

在本書的一些秘訣中，為了添加新方法，我們修改了像是 `Array` 的標準 Ruby 類別（例如，秘訣 1.10 定義一個稱為 `String#capitalize_first_letter` 的新方法），於是，這些方法便可被應用在你的程式裡的每個類別實例，這在 Ruby 中是相當常用的技術：前面提到的 Rails 和 Facets Core 程式庫都是這樣做。然而，這仍然有些爭議，並且可能造成問題（參見秘訣 8.4 更深入的討論），因此，我們覺得應該在這裡特別提出這一點，儘管對剛接觸 Ruby 的人來講，這可能太過技術性。

如果你不想要修改標準類別，你可以將我們展示的方法放進子類別，或者將它們定義在 `Kernel` 的名稱空間裡：也就是，定義 `capitalize_first_letter_of_string`，而不是重新開啟 `String`，並在裡頭定義 `capitalize_first_letter`。

# 其他資源

如果你要學習 Ruby，那麼，標準參考資料就是 Dave Thomas、Chad Fowler、和 Andy Hunt 合著的《Programming Ruby: The Pragmatic Programmer's Guide》（Pragmatic Programmer）。該書第一版以 HTML 格式公佈在網路上（*http://www.rubycentral.com/book/*），但內容略顯陳舊。第二版好很多，並且以印刷品或 PDF（*http://www.pragmaticprogrammer.com/titles/ruby/*）的形式提供。比較好的建議是購買第二版來閱讀，而以第一版作為方便性的參考，而不是試著直接閱讀第一版。

由 "why the lucky stiff" 提供的 "Why's (Poignant) Guide to Ruby"，運用故事來講授 Ruby，像是一本英語初級讀本一樣，對具有創造力的初學者而言，這是一本絕佳的書籍（*http://poignantguide.net/ruby/*）。

就學習 Rails 而言，標準讀物是 Dave Thomas、David Hansson、Leon Breedt、和 Mike Clark 撰寫的《Agile Web Development with Rails》（Pragmatic Programmer）。與此類似的還有兩本專門針對 Rails 的書籍：Rob Orsini 撰寫的《Rails Cookbook》（O'Reilly）和 Chad Fowler 撰寫的《Rails Recipes》（Pragmatic Programmer）。

一些常見的 Ruby 陷阱在下列網站上有解釋：Ruby FAQ（*http://www.rubycentral.com/faq/*，從 Section 4 開始）以及 "Things That Newcomers to Ruby Should Know"（*http://www.glue.umd.edu/~billtj/ruby.html*）。

許多人在學習 Ruby 之前就已經通曉一或多種程式語言，你也許覺得使用一本大書（它認為必須教你編程，也必須教你 Ruby）學習 Ruby，會讓你倒盡胃口，對這類讀者，我們推薦你 Ruby 創始人 Yukihiro Matsumoto 所撰寫的 "Ruby User's Guide"（*http://www.ruby-doc.org/docs/UsersGuide/rg/*）。這是一本簡短的讀物，主要介紹 Ruby 與其他程式語言的差異，它的用語稍微有點過時，並且，其程式碼範例是藉由過時的 `eval.rb` 程式來呈現的（而不是 `irb`），但它確實是我們所知的最佳簡介。

有一些文章特別適合想要學習 Ruby 的 Java 程式員：Jim Weirich 的 "10 Things Every Java Programmer Should Know About Ruby"（*http://onestepback.org/articles/10things/*）、Francis Hwang 的部落格文章 "Coming to Ruby from Java"（*http://fbwang.net/blog/40.html*）和 Chris Williams 的超連結收集 "From Java to Ruby (With Love)"（*http://cwilliams.textdriven.com/pages/ java_to_ruby*）。不管這些文章的名稱，C++ 程式員也可從中受益。

Ruby Bookshelf（*http://books.rubyveil.com/books/Bookshelf/Introduction/Bookshelf*）以容易閱讀的 HTML 格式，提供許多關於 Ruby 的免費書籍，包括上面提過的許多本。

最後，Ruby 的內建模組、類別、和方法都隨附了極佳的文件說明（當中許多原本是為了《Programming Ruby》而寫的）。你可以在 *http://www.ruby-doc.org/core/* 和 *http://www.ruby-doc.org/stdlib/* 上閱讀這些文件說明，也可以在你自己安裝的 Ruby 上，使用 `ri` 命令查看。以類別或方法名稱為引數，`ri` 將顯示相對應的文件說明。下面有幾個範例：

```
$ ri Array                                  # A class
$ ri Array.new                              # A class method
$ ri Array#compact                          # An instance method
```

# 本書印刷體裁

本書使用下列印刷體裁：

純文字

指選單標題、選單選項、選單按鈕、以及鍵盤輔助鍵（諸如 Alt 和 Ctrl）。

*斜體字*

指新術語、URL、電子郵件、及 Unix 工具。

`定寬字`

指命令、選項、變數、屬性、鍵、函式、型別、類別、名稱空間、方法、模組、特性、參數、值、物件、事件、事件處理器、XML 標籤、HTML 標籤、巨集、程式、程式庫、檔名、路徑名、目錄、檔案內容、或命令之輸出。

**`粗體定寬字`**

表示應該由使用者逐字輸入的命令或其他文字。

*`斜體定寬字`*

表示應該由使用者提供的替換文字。

# 使用程式碼範例

這本書是為了協助你把工作做好。一般而言，你可以在你的程式和說明文件中使用本書的程式碼。除非你要重製重要的程式碼，否則無需取得我們的許可。例如，使用本書的

程式碼片段撰寫程式，不需要取得我們的許可。但是，把 O'Reilly 書籍的程式範例燒成光碟片販售或散佈，就需要取得授權。引用本書的文句和範例程式碼來回答問題，不需要取得許可。把本書大量的程式範例整合到你的產品的說明文件中，則需要取得授權。

雖然並非必要，但註明引用來源，我們會很感謝。註明來源通常包括書名、作者、出版商、以及 ISBN。例如，「*Ruby Cookbook*, by Lucas Carlson and Leonard Richardson，Copyright 2006 O'Reilly Media, Inc., 0-596-52369-6」

如果覺得你對書中程式碼範例的使用有別於上述情況，不用客氣，盡量和我們連絡：*permissions@oreilly.com*。

## 連絡我們

在本書專屬的網頁中，我們列出勘誤表、範例、及其他額外資訊。網頁如下：

*http://www.oreilly.com/catalog/rubyckbk*

關於本書的意見，或者想詢問技術性問題，請寄送電子郵件到：

*bookquestions@oreilly.com*

有關其他書籍、研討會、資源中心、以及 O'Reilly Network 的資訊，可參考我們的網站：

*http://www.oreilly.com*

## 誌謝

首先，要感謝我們的編輯 Michael Loukides，謝謝他的大力幫忙，並且默許我們在秘訣的範例程式碼中使用他的名字，即使我們將他變成一隻會說話的青蛙。製作編輯 Colleen Gorman 也提供了許多許多的幫助。

如果沒有我們的投稿作者群，這本書的撰寫時間將會更長，並且缺乏趣味，他們總共撰寫了 60% 以上的秘訣。名單包括：Steve Arniel、Ben Bleything、Antonio Cangiano、Mauro Cicio、Maurice Codik、Thomas Enebo、Pat Eyler、Bill Froelich、Rod Gaither、Ben Giddings、Michael Granger、James Edward Gray II、Stefan Lang、Kevin Marshall、Matthew Palmer、Chetan Patil、Alun ap Rhisiart、Garrett Rooney、John-Mason Shackelford、Phil Tomson、以及 John Wells。他們對各種 Ruby 主題的豐富知識，大大節省了我們的時間，並且豐富了本書的內涵。

如果沒有我們的技術審查人員，本書的品質可能會低得令人毛骨悚然，他們指出大量的 bug、特定平台的問題、以及概念上的錯誤。他們是：John N. Alegre、Dave Burt、Bill Dolinar、Simen Edvardsen、Shane Emmons、Edward Faulkner、Dan Fitzpatrick、Bill Guindon、Stephen Hildrey、Meador Inge、Eric Jacoboni、Julian I. Kamil、Randy Kramer、Alex LeDonne、Steven Lumos、Keith Rosenblatt、Gene Tani 和 R Vrajmohan。

最後要感謝 Ruby 社群的程式員和作家們，不管是如 Yukihiro Matsumoto、Dave Thomas、Chad Fowler、和 "why" 這樣的名人，還是成百上千位其工作成果被加入說明本書所用之程式庫的無名英雄們，他們的能力與耐心將持續不斷地將更多的人帶進 Ruby 社群裡。