Python設計模式深入解析

Mastering Python Design Patterns

藉由16種設計模式，成為能夠解決各式問題的Python大師

Kasampalis 著　江良志 譯

Design Patterns

博碩文化

Python設計模式深入解析

Mastering Python Design Patterns

藉由16種設計模式，成為能夠解決各式問題的Python大師

Sakis Kasampalis 著　江良志 譯

Design Patterns

博碩文化

Python 設計模式深入解析

作　　者：Sakis Kasampalis
譯　　者：江良志
責任編輯：沈睿哞
行銷企劃：黃譯儀
發 行 人：詹亢戎
董 事 長：蔡金崑
顧　　問：鍾英明
總 經 理：古成泉
出　　版：博碩文化股份有限公司
地　　址：221 新北市汐止區新台五路一段 112 號 10 樓 A 棟
　　電話 (02)2696-2869 傳真 (02)2696-2867
郵撥帳號：17484299　戶名：博碩文化股份有限公司
博碩網站：http://www.drmaster.com.tw
讀者服務信箱：DrService@drmaster.com.tw
讀者服務專線：(02)2696-2869 分機 216、238
（周一至周五 09:30 ～ 12:00；13:30 ～ 17:00）
版　　次：2015 年 8 月初版
建議零售價：新台幣 360 元
I S B N：978-986-434-041-5
律師顧問：永衡法律事務所 吳佳憲律師

本書如有破損或裝訂錯誤，請寄回本公司更換

國家圖書館出版品預行編目資料

Python 設計模式深入解析 / Sakis Kasampalis 著；江良志譯 . -- 初版 . -- 新北市：博碩文化，2015.08
面；　公分
譯自：Mastering python design patterns

ISBN 978-986-434-041-5(平裝)

1.Python(電腦程式語言)

312.32P97　　104015101

Printed in Taiwan

商標聲明

本書中所引用之商標、產品名稱分屬各公司所有，本書引用純屬介紹之用，並無任何侵害之意。

有限擔保責任聲明

雖然作者與出版社已全力編輯與製作本書，唯不擔保本書及其所附媒體無任何瑕疵；亦不為使用本書而引起之衍生利益損失或意外損毀之損失擔保責任。即使本公司先前已被告知前述損毀之發生。本公司依本書所負之責任，僅限於台端對本書所付之實際價款。

著作權聲明

目 錄 Contents

Python設計模式深入解析

Mastering Python Design Patterns

藉由16種設計模式，成為能夠解決各式問題的Python大師

Sakis Kasampalis 著　江良志 譯

Design Patterns

博碩文化

Python 設計模式深入解析

作　　者：Sakis Kasampalis
譯　　者：江良志
責任編輯：沈睿晔
行銷企劃：黃譯儀
發 行 人：詹亢戎
董 事 長：蔡金崑
顧　　問：鍾英明
總 經 理：古成泉
出　　版：博碩文化股份有限公司
地　　址：221 新北市汐止區新台五路一段 112 號 10 樓 A 棟
　　電話 (02)2696-2869 傳真 (02)2696-2867
郵撥帳號：17484299　戶名：博碩文化股份有限公司
博碩網站：http://www.drmaster.com.tw
讀者服務信箱：DrService@drmaster.com.tw
讀者服務專線：(02)2696-2869 分機 216、238
（周一至周五 09:30 ～ 12:00；13:30 ～ 17:00）
版　　次：2015 年 8 月初版
建議零售價：新台幣 360 元
I S B N：978-986-434-041-5
律師顧問：永衡法律事務所 吳佳憓律師

本書如有破損或裝訂錯誤，請寄回本公司更換

國家圖書館出版品預行編目資料

Python 設計模式深入解析 / Sakis Kasampalis 著；江良志譯 . -- 初版 . -- 新北市：博碩文化，2015.08
面；　公分
譯自：Mastering python design patterns

ISBN 978-986-434-041-5(平裝)

1.Python(電腦程式語言)

312.32P97　　104015101

Printed in Taiwan

商標聲明

本書中所引用之商標、產品名稱分屬各公司所有，本書引用純屬介紹之用，並無任何侵害之意。

有限擔保責任聲明

雖然作者與出版社已全力編輯與製作本書，唯不擔保本書及其所附媒體無任何瑕疵；亦不為使用本書而引起之衍生利益損失或意外損毀之損失擔保責任。即使本公司先前已被告知前述損毀之發生。本公司依本書所負之責任，僅限於台端對本書所付之實際價款。

著作權聲明

關於作者

Sakis Kasampalis（@SKasampalis）是位軟體工程師，住在荷蘭。他不會盲目獨斷地推崇某程式語言或開發工具，其理念是根據手上任務選用最適切的工具；最喜愛的工具之一是Python語言，因為生產力非常高。

Sakis也曾是《Mastering Object-oriented Python and Learning Python Design Patterns》一書的技術審閱者，Packt出版。

我想要感謝我的愛人Georgia，支持我幫助我。感謝Owen Roberts鼓勵我撰寫這本著作。還要感謝策劃編輯Sumeet Sawant，擁有一副好心腸，協助甚大。最後但絕非最不重要，我想要感謝本書的審閱者們，提供寶貴的回饋意見。

關於審閱者

Evan Dempsey，來自愛爾蘭瓦特福城的軟體開發人員，當他若不是為了樂趣與利益鑽研Python，便是在研究啤酒的滋味、Common Lisp以及追上機器學習領域的最新研究成果，也參與許多開放原始碼專案，貢獻良多。

Amitabh Sharma是位專業軟體工程師，工作內容涵蓋電子通訊與商業分析產業的企業級應用軟體，專精服務導向架構、資料倉儲與各種程式語言，諸如Java、Python與其他。

> 我想要感謝祖父與父親，讓我能盡其所能地學習，還想要感謝我的太太Komal，感謝她的支持與鼓勵。

Yogendra Sharma的出生與成長都在Pratapgarh，位於印度拉賈斯坦邦的小鎮，雖小但充滿文化氣息，在家鄉接受基本教育，之後在齋浦爾城的大學取得電腦科學的技術學士學位；內心是個工程師，天性熱愛技術。

在各個領域擁有廣泛的經驗，包括Python、Django、網頁應用程式安全性、網路、Web 2.0與C++。

除了CCNA，他獲頒許多值得令人尊重的認證資格，活躍於許多組織與社群，包括國際工程師協會、Ubuntu、印度與印度電腦協會。

最近他參與獵蟲競賽，贏得許多獎金，包括Yahoo、Ebay、PayPal舉辦的活動；被任命為安全研究人員，為許多公司組織研究相關問題，諸如Adobe、EBay、Avira、Moodle、Cisco、Atlassian、Basecamp、CodeClimate、Abacus、Rediff、Assembla、RecruiterBox、Tumbler、Wrike、Indeed、HybridSaaS、Sengrid與SnapEngag。

曾替許多知名出版社審閱各種書籍；可在LinkedIn找到他：http://in.linkedin.com/in/yogendra0sharma。

我想要感謝好友們，總是鼓勵我嘗試新東西並且相信我。

Patrycja Szabłowska是位Python開發人員，具有Java背景，主要經驗在於後端程式開發，畢業於波蘭托倫的尼古拉·哥白尼大學。

目前在波蘭華沙工作，不停地探索新穎技術，保持開放心靈，熱切學習下一套Python程式庫或開發框架，她最喜愛的程式設計座右銘是：程式碼被閱讀的機會比被修改的機會多得多。

我想要感謝我的丈夫Wacław，鼓勵我探究新的未知事物，也要感謝我的父母，教導我哪些是最重要的事物。

序

設計模式

在軟體工程領域中，設計模式指的是針對某軟體問題建議採用的解決辦法。設計模式描述如何依照一定的結構來撰寫程式，依據最佳實務作法來解決常見設計問題。有一點很重要還請記住，設計模式是高層級的解決方案，並不著重於實作細節，諸如演算法與資料結構[GOF95，第13頁]、[j.mp/srcmdp]，那些地方會交由身為軟體工程師的我們，根據手上問題，找出最適當的演算法與資料結構。

注意

如果你正感到疑惑，不清楚[]裡頭包含的文字有何意義，請暫且跳到這篇序的「書寫慣例」一節，了解本書以何種格式標示參考資料。

設計模式最重要的部份大概是它的名字吧，為所有模式賦予名稱之後，因為大家都擁有共通的詞彙，方可討論溝通[GOF95，第13頁]，因此，當你傳送程式碼給別人檢查審校，對方可能的回饋意見可能會提到「我認為你可以在這個地方使用策略模式，而不是…」，即便你當下並不知道何謂策略模式，但馬上就可以查詢。

當程式語言逐漸演進，有些設計模式，譬如單例(Singleton)模式變得陳舊無用、甚至成為反面模式[j.mp/jalfdp]，有些模式則內建在程式語言之內(迭代器)，另外也會產生新的模式(Borg/Monostate [j.mp/amdpp]、[j.mp/wikidpc])。

關於設計模式的常見誤解

關於設計模式，有幾個普遍存在的誤解，其中之一是：必須從撰寫程式的一開始便運用設計模式。有時我們常看見開發人員掙扎著思索該使用哪一個設計模式，即便他們尚未先試著以自己的方式解決手上的問題[j.mp/prsedp]、[j.mp/stedp]。

上述情況不僅錯誤，也違反設計模式的本質，應該是先有一份解決方案，然後從中發現（而非發明）可作為較佳解法的設計模式；在還沒有解答的情況下追尋更好的解答，毫無意義。請儘管動手，運用你已有的知識與技巧，盡力想出最好的辦法來解決手上的問題，如果程式審閱者沒有異議，而且經過一段時間之後，你發現你的解答不僅聰明而且也很有彈性，那麼就代表了毋須浪費時間掙扎想著該運用哪一個設計模式；你甚至可能已經發現更好的設計模式，比原先已知的都還要好，誰知道呢？重點在於，可不要強迫自己運用已有的設計模式，那只會囚禁你的創造力。

第二個誤解是：任何地方都該運用設計模式。結果只會導致過於複雜的解決方案，充斥著無謂的介面與階層架構，明明只需使用較為簡單直覺的辦法即可。請不要把設計模式當做萬靈丹，因為真的不是，只有當你能證明現存程式碼「發出臭味」，難以擴充與維護之時，才是設計模式登場的時刻。請試著考慮這兩句標語：你不會需要它（you aren't gonna need it，YAGNI [j.mp/c2yagni]）、保持簡單和愚蠢（Keep it simple stupid，KISS [j.mp/wikikis]）。到處使用設計模式跟過早最佳化一樣邪惡[j.mp/c2pro]。

設計模式與Python

本書的重心是以Python語言運用設計模式，Python不同於大部分設計模式暢銷著作所採用的程式語言（通常是Java [FFBS04]或C++ [GOF95]），Python支援鴨子型別、函式是頭等物件、以及已內建某些模式（譬如，迭代器與裝飾器）。本書企圖闡明大部分基礎且重要的設計模式，而非所有已知的設計模式[j.mp/wikidpc]。範例程式碼著重於符合Python風格與慣常用法[j.mp/idiompyt]，若你不熟悉Python之禪（Zen of Python）的話，最好現在開啟Python REPL，執行`import this`，Python之禪不僅發人省思，也相當有趣。

本書涵蓋內容

第1部分：創建型設計模式，與物件建立相關的設計模式。

第1章「工廠模式」：學習如何運用工廠設計模式（工廠方法與抽象工廠模式）來初始化物件，內容涵蓋運用此模式而非直接實體化物件所帶來的優點。

第2章「建造者模式」：學習如何簡化複雜物件的建立程序，典型的複雜物件由多個相關物件組成。

第3章「原型模式」：學習如何為既存物件建立完整複製的新物件。

第2部分：結構型設計模式，處理系統中個體（類別、物件等等）之間關係的設計模式。

第4章「轉接器模式」：學習如何以最小幅度的修改，讓原有的程式碼相容於外部介面（譬如，第三方程式庫）。

第5章「裝飾器模式」：學習如何擴充物件的功能，不使用繼承。

第6章「外觀模式」：學習如何為系統建立單一進入點，隱藏掉繁複的內部細節。

第7章「輕量模式」:學習如何從物件池重複使用物件，降低應用程式的記憶體用量，也可能增進效能表現。

第8章「模型-檢視-控制器模式」：學習如何降低應用軟體的維護難度，作法是分離商務功能邏輯與使用者介面。

第9章「代理模式」：學習提昇應用程式的安全性，作法是加入額外的保護層。

第3部分：行為型設計模式，處理系統個體之間如何進行溝通的設計模式。

第10章「責任鏈模式」：學習如何發送請求給多個接收方。

第11章「命令模式」：學習如何讓應用軟體能夠回復已經執行的操作動作。

第12章「直譯器模式」：學習如何在Python之上建立簡單的語言，供領域專家使用，而不是強迫他們學習如何寫Python程式。

第13章「觀察者模式」：學習如何在物件狀態改變時，發送通知給想要知道且已註冊的其他物件。

第14章「狀態模式」：學習如何建立狀態機，作為程式的概略結構，討論這種技巧可帶來的好處。

第15章「策略模式」：學習如何根據某些輸入條件（譬如，元素大小），在眾多可用演算法中挑出（於執行期間）適當的那一個。

第16章「模板模式」：學習如何清楚地分離演算法的共通部分與相異部分，避免重複程式碼。

閱讀本書需要哪些東西

本書範例程式碼皆以Python 3撰寫。Python 3在許多方面都不相容於Python 2.x [j.mp/p2orp3]，雖然以3.4.0為主，不過使用3.3.0應無問題，因為這兩個版本之間並無語法差異[j.mp/py3dot4]。總而言之，只要您從www.python.org下載最新的Python 3版本並安裝，便可毫無問題地執行本書的範例。範例程式使用的模組／程式庫，大都屬於Python 3標準程式庫的一部分，如果需要安裝額外的模組，在列出相關程式碼之前，文中將會說明安裝的指令。

適合本書的讀者

本書瞄準的讀者是已具備基本程度的Python程式設計師，而且想要以符合Python精神的程式寫法來實作設計模式。其他程式語言的程式設計師，若對Python有興趣，也能從本書內容得益，然而仍建議先閱讀其他解釋如何撰寫Python風格程式碼的文章與著作[j.mp/idiompyt]、[j.mp/dspython]。

書寫慣例

在本書裡，帶有不同種類資訊的文字，將以不同的樣式呈現，底下列出各樣式的範例，並解釋其意義。

內文中的程式碼、資料庫表格名稱、資料夾名稱、檔案名稱與副檔名、路徑名、URL、使用者的輸入以及Twitter標記，以如下方式呈現：「我們將使用兩套