序

對任何一間公司、組織、機構或政府機關來說，要製作與使用 API，就必須知道如何在持續擴展和演變的數位生態系統中正確地管理數位資源。由於過去五年來不斷發展的數位轉型，API 生態系統已經有了明顯的轉變，公司再也不自問**要不要做** API 了，而是開始瞭解**如何妥善地建構** *API*。各機構開始意識到，除了建構 API 之外，它們還要在整個 API 生命週期中處理與 API 的交付有關的所有事項。*Continuous API Management* 的 API Academy 作者群對於如何設計與實作 API、如何實現一致性、如何大規模發表，以及如何採取可重複的做法有獨到的見解，提供了一個非常特別的學習機會。

大部分的 API 從業者在製作 API 時都只考慮單一 API 組合。Medjaoui、Wilde、Mitra 與 Amundse 與眾不同，他們從 250 英尺的高空往下看，視野橫跨上千個 API、大量的產業，以及一些最大型的當代企業機構。我沒辦法用十根手指頭列舉世界上所有的 API 天才，但是 Medjaoui、Wilde、Mitra 與 Amundsen 一定是前幾位。如果你想要知道如何從 API 的草擬階段進入設計階段、從開發階段進入生產階段，以及如何從頭再來一次，這群作者可以跟你分享豐富的經驗。這群 API 專家的知識範圍與廣度是絕無僅有的，因為你會反覆地閱讀這本書，它注定會變成一本被你翻爛的 O'Reilly 書籍。

我讀過許多關於 API 建構技術、Hypermedia 與 REST，以及如何以各種程式語言和平台來進行實作的書籍，但是它們都無法像這本書一樣完整地說明 API 的交付，它不但說明技術面的細節，也談到運作 API 的關鍵商業元素，其中也包括與人有關的面向，例如 API 教育、實現以及在大型企業機構促進發展。本書有條不紊地列出 API 建構師在大規模交付可靠、安全與一致的 API 時需要考慮的基本元素，它可以協助任何 API 團隊將工作量化，更批判性地思考如何改進 API，同時建立與改善結構化的敏捷方法，以標準的方式在團隊間交付 API。

放下這本書之後，我對現代的 API 生命週期有了全新的看法——更重要的是，我充分瞭解如何量化與評量 API 的經營，也更瞭解經營管理的 API 生命週期策略。就算我對這個領域已經有一定程度的理解了，這本書也讓我不得不用全新的觀點看待生態系統。這本書補充了我已經知道的知識，但也改變一些我**認為**已經知道的事情，迫使我更改一些既有的做法。對我來說，API 旅程的意義就是：不斷接受挑戰、學習、計畫、執行、評量以及重複這個過程，直到你找到想要的結果為止。*Continuous API Management* 反映了這項“API 交付”的事實，它是可重複使用的指南，教導我們在企業裡面大規模地製作 API 所需的技術、事務與策略。

你要多看幾次這本書，並且在閱讀之後付諸執行，實現你的願景。你要演化 API 策略，並且定義一個自己的 API 週期版本，將 Medjaoui、Wilde、Mitra 與 Amundsen 教你的東西運用在實際的工作上。你每隔一段時間都要重新閱讀這本書，我保證每當你重新閱讀這本書的時候，都會有一些新的發現與啟發，有時它會幫助你更瞭解 API 園林正在發生的事情，並且協助你在不斷擴展的網路經濟中更有自信地參與（或帶領）API 業務。

— *Kin Lane，API 宣揚者*

前言

隨著社會與商業日益數位化，互連軟體的需求已呈爆炸式成長。由於應用程式開發介面（API）促進軟體的相連，它已經成為現代機構最重要的資源了。但是有效地管理這些 API 是一項新的挑戰。要從 API 獲得最大的價值，你必須知道如何管理 API 的設計、開發、部署、成長、品質與安全防護，並且處理與背景、時間及規模有關的複雜因素。

誰該閱讀這本書？

如果你正準備開始編寫 API 程式，並且想要瞭解將來的工作，或是你已經有一些 API 了，但是想要知道如何更妥善地管理它們，這本書就很適合你。

在這本書裡面，我們試著建構一個可以在各種背景之下運用的 API 管理框架。藉由本書的內容，你會知道如何管理一個可讓全世界的開發者使用的 API、如何建立專為內部開發者設計的微服務架構 API 組合，以及介於兩者之間的所有知識。

這本書也盡量保持技術面的中立。我們提供的建議與分析適合任何 API 結構，包括 HTTP CRUD、REST、GraphQL 與事件驅動型互動。本書適合所有想要改善 API 決策的人。

本書的內容

本書包含我們多年來設計、開發與改善自己和別人的 API 時累積的知識。我們把所有經驗都濃縮在這本書裡面了。我們發現有效地開發 API 有兩個核心要素：採取產品觀點及建立正確的團隊。我們也發現實現它們的三個基本要素：治理、讓產品成熟與設計生態環境。

這五種 API 管理元素是建構成功的 API 管理專案的基礎。本書將介紹這些主題，並教導你如何根據自己的組織背景來塑造它們。

大綱

本書的章節經過刻意安排，希望隨著章節的進展而增加管理的範圍。我們先介紹“決策導向治理”以及“API 即產品”的基本概念。接著介紹建構 API 產品時必須管理的所有工作。

我們將從單一 API 的觀點開始探討“變動 API”是什麼意思，以及“API 的成熟度如何影響這些變動決策”，並加入時間層面。接著討論進行變動的團隊與人員。本書的後半部討論擴展的複雜性以及管理 API 產品生態系統的挑戰。

以下是各章的摘要：

第 1 章介紹 API 管理領域，並解釋為何有效管理 API 如此困難。

第 2 章從決策的角度來討論治理，決策導向工作是 API 管理的基本概念。

第 3 章建立 API 即產品的觀點，以及為何它是任何一種 API 決策的重要元素。

第 4 章概述 API 產品領域的十大基本工作支柱。這些支柱構成你必須管理的決策工作。

第 5 章深入介紹持續變動 API 的含義。本章介紹持續變動心態的重要性，並說明你將遭遇的各種 API 變動種類（與它們的影響）。

第 6 章介紹 API 產品生命週期，這個框架可以協助你在 API 產品的整個生命週期中管理橫跨十大支柱的 API 工作。

第 7 章處理 API 管理系統的人員元素，本章將探討 API 產品生命週期中，API 團隊的典型角色、責任與設計模式。

第 8 章在 API 管理問題中加入擴展的角度，介紹在同一時間變動多個 API 時需要處理的八個 V——多樣性、詞彙、數量、速度、漏洞、能見度、版本系統與易變性。

第 9 章介紹持續性生態系統設計，可用來大規模、持續地管理 API 的變動。

第 10 章將生態系統觀點對映到“API 即產品”概念，並指出當生態系統不斷演變時，API 工作會如何變動。

第 11 章將之前提過的 API 管理故事聯結起來，建議你如何為將來做好準備，以及如何立即開始你的旅程。

本書沒有談到的內容

API 管理領域十分廣大，且環境、平台與協定都有大量的差異。限於本書的時間與空間，我們無法解決 API 工作的所有具體實作方法。本書不是設計 REST API 或挑選安全閘道產品的指南。如果你想要尋找編寫 API 程式或設計 HTTP API 的說明，這本書不適合你。

雖然我們有一些範例談到特定的實作，但這不是一本專門討論 API 實作的書（好消息是坊間有大量的書籍、部落格與影片可以滿足你的需求）。本書解決的是一個很少被處理的問題：如何在一個複雜、持續變動的組織系統中有效地管理 API 建構工作。

本書編排規則

本書使用以下的編排規則：

斜體字（*Italic*）

代表新的術語、URL、電子郵件地址、檔案名稱及副檔名，中文用楷體字。

定寬字（`Constant width`）

代表程式元素，例如變數或函式名稱、資料型態、陳述式與關鍵字。

定寬斜體字（*`Constant width italic`*）

代表應換成使用者提供的值，或依上下文而決定的值。

第一章

管理 API 時面臨的挑戰

歸根究柢，管理是結合藝術、科學與工藝的工作。

—Henry Mintzberg

Coleman Parkes 在 2017 年發表的一項調查（*http://bit.ly/CP_APIs_survey*）指出，全球的企業，九成都有某種形式的 API 專案。同一項調查表示，這些公司都在它們的 API 專案中看到各式各樣的好處，包括平均上市速度增加了 18% 左右。但是其中只有大約 50% 的公司表示它們有先進的 API 管理計畫。這指出許多企業級 API 專案的主要鴻溝——“貢獻收入的主力 API”以及“支持這種製造利潤的 API 所需的管理技能與基礎設施”之間的距離。這也是本書希望處理的鴻溝。

好消息是，目前已經有許多公司成功地管理他們的 API 專案了。但比較不好的消息是，他們不太願意分享經驗與專業知識，或你不容易知道它們。這有幾個原因。多數情況下，很會管理 API 的機構只是因為太忙了，才無法與別人分享他們的經驗。少數情況下，有一些公司很在乎他們與外界分享了多少 API 管理專業知識，根據我們的訪談，他們確信 API 技術是一種競爭優勢，所以不想要迅速地公開他們發現的技術。最後，雖然有些公司會在公開的會議上或透過文章與部落格文章分享他們的經驗，但他們分享的資訊通常只適合特定的公司，很難轉換成適合各種機構的 API 專案。

本書試著解決第二個問題——將適合特定公司的案例變成適合所有機構的共同經驗。為此，我們訪問了數十間公司，採訪了許多 API 技術員，試著在這些公司與我們分享及與公眾分享的案例中，找出共同點。本書會討論許多主題，而本章會先介紹它們。

當你聽到別人談到 API 時，必須先確定他的意思是什麼。首先，“API”這個詞指的可能只是**介面**（例如 HTTP 請求 URL 與 JSON 回應）。它也可以代表將一項服務放入產品環境所需的程式碼與部署元件（例如 `customerOnBoarding` API）。最後，我們有時會用“API”來代表一個正在運行的 API **實例**（例如，在 AWS 雲端運行的 `customerOnBoarding` API vs. 在 Azure 雲端運行的 `customerOnBoarding` API）。

當你管理 API 時，另一個重要的挑戰是區分設計、建構與發表**單個** API vs. 支援和管理**多個** *API*（我們稱之為 *API* **園林**（*landscape*）[譯註]）之間的差異。本書會用大部分的篇幅來討論這個光譜的兩端。*API* **即服務**（*API-as-a-Product*，AaaP）之類的概念與建立、維護 API 的技術（我們稱之為 *API* **支柱**（*pillar*））的目的都是為了處理單個 API 帶來的挑戰。我們也會討論 API 成熟度模型的角色，以及處理隨著時間造成的變動所需的工作，它們都是管理一個 API 時很重要的層面。

在光譜的另一端是管理 API 園林的工作。園林是來自所有商務領域、在所有平台上運行、由公司的所有 API 團隊管理的 API 組合。園林帶來的挑戰有許多層面，包括規模與範圍如何改變 API 的設計與實作，以及為何只因為擁有一個大規模的生態系統，就會增加它們的易變性與漏洞。

最後，我們要討論管理 API 生態系統的決策程序。根據我們的經驗，這是為 API 專案建立成功的**治理**計畫之關鍵。你會看到，決策程序必須隨著園林而改變，堅持採取舊的治理模式可能會限制 API 計畫的成功，甚至讓既有的 API 陷入更多風險。

在深入討論如何處理這兩項挑戰（單獨的 API 及 API 園林）之前，我們先來看一下兩個重要的問題：什麼是 API 管理，以及為何它如此困難？

什麼是 API 管理？

如前所述，API 管理並非只與設計、實作與發表 API 有關。它也包含 API 生態系統的管理、如何在機構裡面發表決策，甚至將現有的 API 遷移到正在成長的 API 園林等程序。在這一節，我們要花一點時間討論這些概念，但首先，我們要簡單地解釋這裡的“API”是什麼意思。

[譯註] landscape 本身有景觀、環境等含意，作者用它來代表 API 生態系統，本書將它譯為“園林”，你可以想像自己像園丁一樣，照顧由許多 API 組成的生態系統。API landscape 的定義見第 149 頁。

什麼是 API ？

有時“API”這個名詞代表的不僅僅是介面，也代表功能，也就是介面背後的程式。例如，有人可能會說“我們必須快點發表更新後的 Customer API，讓別的團隊可以使用新的搜尋功能。”有時人們只用它來代表介面本身的細節。例如，團隊成員可能會說“我想要幫現有的 SOAP 服務設計一個新的 JSON API，來支援顧客入門流程。”當然，這兩種意思都正確（而且它們都清楚地表達含義），只不過有時會造成混淆。

為了釐清這種區別，以更方便說明介面與功能，我們要介紹一些額外的名詞：介面、實作與實例。

介面、實作與實例

API 這個縮詞代表**應用程式開發介面**。我們用**介面**來使用在 API “背後”運行的東西。例如，你可能會用一個 API 來公開“管理使用者帳號”這項工作。開發者可以用這個介面：

- 建立新帳號。
- 編輯既有的帳號設定。
- 改變帳號狀態（暫停或啟動）。

這個介面通常以共用的協定來表示，例如 HTTP、Thrift、TCP/IP 等等，並且使用標準的格式，例如 JSON、XML 或 HTML。

但是它只是個介面，此外還需要其他的東西才可以執行你要求的工作，那些其他的東西就是所謂的**實作**。實作是提供實際功能的部分，通常實作是用 Java、C#、Ruby 或 Python 等程式語言寫成的。延續使用者帳號的例子，`UserManagement` 實作可能包含建立、加入、編輯與移除使用者等功能，這些功能使用之前提到的介面來公開。

將介面與實作解耦

請注意，前面所談到實作的功能是一組“建立、讀取、更新、刪除（CRUD）模式”動作，但是介面有三項操作（`OnboardAccount`、`EditAccount` 與 `ChangeAccountStatus`）。這種實作與介面間的“不一致”很常見，而且有時有很大的功用，它可將各項服務的實作與使用那項服務的介面解耦，讓你更容易隨時變動它，而不會造成損壞。

第三種名詞是**實例**。API 實例是介面與實作的結合，很適合用來代表已經被發表為產品並且已經在運行的 API。我們會用一些指標來管理實例，以確保它們是健康的，我們會註冊與記錄實例，以方便開發者尋找與使用 API 來解決真實世界的問題，我們也會保護實例的安全，確保只有經過授權的使用者可以執行操作，以及讀取 / 寫入必要的資料來完成那些操作。

圖 1-1 是這三種元素之間的關係。在本書中，當我們談到“API”時，通常是指 API 的實例：完全可操作的介面與實例組合。如果我們的意思**只**是介面或**只**是實作，就會在內容那樣稱呼它們。

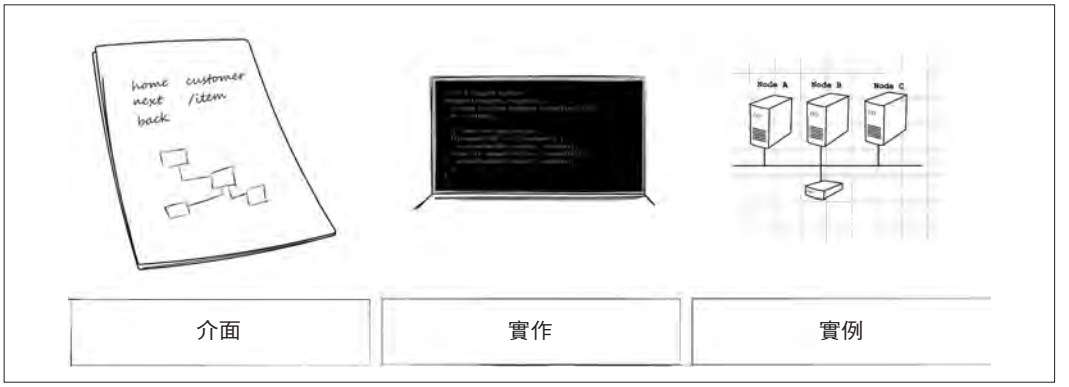

圖 1-1　三種 API 元素

不僅僅是 API

API 本身（介面與實作的技術細節）也只是整個故事的一部分而已。設計 / 建構 / 部署等傳統元素在 API 的生命週期之中當然也很重要。但是**管理** API 通常也代表測試它們、記錄它們，以及在網站入口發表它們，讓正確的用戶（內部開發者、夥伴、不知名

的第三方 app 開發者等等）可以找到並且學習正確地使用它們。你也要保護 API、在執行期監控它們，以及在它們的生命週期中維護它們（包括處理變動）。這些額外的 API 元素就是我們所謂的 *API 支柱*：它們是所有 API 都需要，且所有 API 專案經理都必須處理的元素。我們會在第 4 章深入討論支柱，屆時我們將探討建立與維護健康的 API 的十大關鍵方法。

好消息是這些實務領域不限於任何單一的 API。例如，API 團隊可將妥善記載 API 的技術交給下一個團隊，測試技巧、安全防護等層面也一樣。這也代表就算你有許多不同的團隊負責各個 API 領域（銷售團隊、產品團隊、後台團隊等等），不同團隊的成員人間也會有“交叉”利益的存在[1]，所以管理 API 的另一個重點是：支援與規劃 API 建構團隊。我們會在第 7 章進一步討論不同的機構如何運作它。

API 成熟階段

知道與瞭解 API 支柱還不代表知道全部的事情，在專案之中的每一個 API 都會經歷它自己的“生命週期”——一系列可預測且實用（useful）的階段。知道目前處於 API 旅程的哪個階段可協助你決定目前應該投入多少時間與資源在 API 之中。瞭解 API 有多麼成熟可讓你辨認出許多 API 的同一個階段，並協助你為各個階段不同的時間與精力需求做好準備與做出回應。

表面上，在設計、建構與發表 API 時，合理的做法是考慮需要處理的所有 API 支柱，但是現實的情況並非如此。例如，在 API 的早期階段，最重要的事情通常是專注在設計與建構層面，減少文件工作。在其他的階段（例如，當原型已經在 beta 測試者手中時），比較重要的事情是花更多時間監控 API 的使用，以及防止它被濫用。瞭解成熟階段可以協助你安排有限的資源來產生最大的效果。我們會在第 6 章帶你經歷這個過程。

多個 API

讀者可能知道，在管理許多 API 時，我們將面對不同的事情。我們的顧客會有成千上萬個必須建構、監控與管理的 API，此時，你比較不會在意實作單一 API 的細節，而是這些 API 如何在一個不斷成長、動態的生態系統中共存。如前所述，我們將這種生態系統稱為 API 園林，本書的後半部分會用好幾章專門討論這個概念。

1 音樂串流服務 Spotify 將這種交叉群體稱為“guilds”。要進一步瞭解這個主題，見第 137 頁的“擴展你的團隊”。

此時大多數的挑戰都是確保一定程度的一致性，同時避免因為集中管理以及查看所有的 API 細節而造成瓶頸和速度的下降，做法通常是將這些細節的責任分配給各個 API 團隊，並且把中央單位的管理 / 治理重心放在規範 API 彼此的互動，建立一套核心的共享服務或基礎架構（安全防護、監控等等），並且讓所有的 API 團隊使用，通常我們也會幫自立的團隊提供方針與指導。也就是說，我們通常要脫離一般的集權式命令與控制模式。

當你在機構中更深入地授與決策權與自主權時，有一個挑戰在於機構的高層很容易忽視底層團隊的重要活動。在過去，團隊可能要請求許可才可以採取行動，公司將額外的自主權授與各個團隊就是鼓勵他們不需要上層的審查與許可就可以採取行動。

管理 API 園林的挑戰大都與**規模**與**範圍**有關。事實上，隨著 API 專案的成長，它不但會變大，也會變**形**。本章將進一步討論這個主題（見第 7 頁的“為什麼管理 API 很困難？”）。

API 的商務

除了在園林中建立 API 與管理它們的細節之外，你一定要記得這項工作的目的是支援商業目標和目的。API 不僅僅是 JSON 或 XML、同步或非同步等技術細節，它們是將商業元素連接起來，以便公開重要的功能和知識，讓公司更有效率的方法。API 通常是為機構釋出既有價值的手段，例如透過建立新的 app、開發新的收入來源，以及啟動新的業務。

這一種思維方式比較專注於 API 用戶的需求，而不是生產與發表 API 人員的需求。這種用戶導向的做法通常稱為“Jobs to Be Done”或 JTBD，它是哈佛商學院的 Clayton Christensen 提出的做法，他在 *The Innovator's Dilemma* 與 *The Innovator's Solution*（Harvard Business Review Press）中深入介紹這種做法的威力。如果你要啟動與管理成功的 API 專案，它可以清楚地提醒你：API 的目的是為了解決商業問題。根據我們的經驗，擅長用 API 解決商業問題的公司都把它們的 API 當成用來“完成工作”的**產品**，這種看法與 Christensen 解決顧客問題的 JTBD 框架不謀而合。

用 API 專案來輔助商業活動的做法之一就是研發一套靈活的“工具”（API）來建構新的解決方案，以免產生高昂的成本。例如，如果你有一個 `OnlineSales` API 可讓重要的夥伴管理與追蹤他們的銷售活動，以及一個 `MarketingPromotions` API 可讓行銷團隊設計與追蹤產品促銷活動，或許你可以建立一個新的夥伴解決方案：`SalesAndPromotions` app。

API 協助商業的另一種方式就是讓人們更容易取得重要的顧客或市場資料，它們或許與新客戶群的新趨勢或獨特行為有關。藉著讓這些資料更安全與更容易取得（經過妥善地匿名與篩選），API 或許可讓你的公司發現新的商機、建立新的產品 / 服務，甚至以更低的成本與更快的時間來推出新的活動。

我們會在第 3 章討論 AaaP 這一個重要面向。

為什麼管理 API 很困難？

在本章的開頭談過，雖然大部分的公司都已經啟動了 API 專案，但是只有 50% 認為他們把 API **管理**得很好。怎麼了？這個領域的挑戰是什麼？你該如何協助公司克服它們？

當我們拜訪世界各地的公司，討論如何管理 API 生命週期時，看過一些基本的主題：

範圍

隨著時間的推移，中央軟體架構團隊在管理 API 時應該注意什麼事項？

規模

一般來說，研發 API 時，少數幾個小團隊在初期有效實施的方法，無法隨著專案擴展成全球的計畫而擴展。

標準

我們發現，隨著專案的成熟，管理與治理工作必須從“詳細地建議如何設計與實作 API”變成較通用的 API 園林標準化，讓團隊可以自行做出更多更詳細的決策。

持續平衡這三個要素（範圍、規模與標準）有助於在本質上創造一個健康、持續成長的 API 管理專案。因此，這些要素值得我們更深入地探討。

範圍

執行健康的 API 管理專案有一個巨大的挑戰，在於中央單位對於控制程度的拿捏。事情不僅於此，更有挑戰性的是，適當的控制程度會隨著專案的成熟而改變。

在專案的早期，你應該直接把重心放在設計 API 的細節。當 API 剛起步時，這些設計細節可能直接來自創造 API 的團隊——他們會查看“坊間”既有的專案，使用適合 API 的工具與程式庫，然後動手製作那個 API。

在這個 API 專案的“早期階段”，所有事項都是新的，所有問題都是第一次遇到（與解決）。這種最初的經驗通常會被視為公司的“API 最佳做法”或公司方針等等。在一開始，讓小團隊製作少量的 API 是合理的做法。但是這些最初的方針可能是不完整的。

隨著公司中負責 API 的團隊數量的成長，各種風格、經驗與觀點也會隨之增加。你將越來越難以維持團隊之間的一致性，原因不僅僅是有些團隊不遵守已經發表的公司方針，也有可能是新團隊使用了一組不同的產品，讓他們難以遵守最初的方針。或許是因為他們無法在事件串流（event-streaming）環境中工作，因為他們支援 XML 呼叫 / 回應風格的 API。當然，他們都需要指導，但這些指導必須符合**他們的**領域、**他們的**顧客需求。

你當然會規定一些讓所有團隊遵守的共同方針，但那些方針必須符合他們的問題領域以及 API 顧客的需求。隨著你的社群越來越大，你的多樣性也會增加，此時非常重要的一件事就是你不能犯下試圖消除這種多樣性的錯誤。這就是你要將控制槓桿從**發號施令**（例如“所有的 API **都必須**使用以下的 URL 格式…”）切換到提供**指引**的時機（例如“在 HTTP 上面運作的 API 都**應該**使用以下的 URL 模板之一…”）。

換句話說，隨著專案範圍的擴展，你的方針也要相應地擴展。這一點對全球企業而言特別重要，因為各個地區的文化、語言與歷史會影響各個團隊思考、創造與解決問題的方式。

這一點引出下一個重要元素：專案規模。

專案規模

為了建立與維護健康的 API 管理專案，你的另一項挑戰是處理專案的規模隨著時間的推移產生的變化。上一節談過，增加團隊數量與這些團隊製作的 API 數量可能是一項挑戰。在執行期監控與管理 API 所需的程序也會隨著系統的成熟而改變。追蹤同一個團隊在同一個地理位置建構的少量 API 所需的工具，與追蹤分布多個時區與國家的成千上百個 API 入口點所需的工具有很大的不同。

在本書中，我們將這種 API 管理的層面稱為“園林”。隨著專案的擴展，你必須設法關注各地團隊的許多流程。你更需要監控執行期的行為，以隨時瞭解系統的健康程度。本書的第二個部分（從第 8 章開始）將探討 API 園林管理概念如何協助你找出哪些元素值得關注，以及哪些工具與程序可以協助你持續處理不斷成長的 API 平台。

API 園林會帶來一系列的新挑戰。設計、實作與維護單一 API 的程序與擴展生態系統的程序不一定一樣。基本上，這與數量有關：你的系統有越多 API，它們就越有機會彼此互動，因此就更有機會出現意外的行為（或“錯誤”）。大型系統本身就有更多的互動與意外，僅僅試著消除這些意外並不會讓你全身而退，因為你不可能移除所有的 bug。

所以，正在成長的 API 專案都會遇到第三項挑戰：如何在 API 專案中採取適當的標準來減少意外的變動？

標準

當你開始管理園林層面而不是 API 層面時，有一個重大的轉變就是標準（standard）指引團隊以一致的做法設計、實作與部署 API 的能力將有所不同。

群體規模變大 —— 包括負責 API 的團隊 —— 協調成本就會增加（見第 16 頁的“決策”）。擴展規模需要改變範圍。處理這項挑戰的關鍵做法是更依靠一般性的標準，而不是特定的設計限制。

例如，全球資訊網從 1990 出現以來能夠持續良好地運作的原因之一，在於它的設計師早就決定採取適合任何軟體平台與語言的通用標準，而不是專門針對單一語言或框架建立範圍狹隘的實作方針。如此一來，創新團隊可以發明新的語言、結構模式，甚至執行期框架，而不會破壞既有的實作。

在這個協助 web 持續成功的長期標準中，有一個共同的主軸就是將元件與系統之間的*互動*標準化。web 標準的目的是讓各方用它來瞭解彼此，而不是將內部元件的做法標準化（例如你要使用這個程式庫、這個資料模型等等）。類似的情況，隨著 API 專案更成熟，提供給 API 社群的指南就必須更著重通用的互動標準，而不是特定的實作細節。

這種轉變可能是艱辛的過程，但它對過渡到健康的 API 園林至關重要，因為團隊可以用它來建構可和現有與未來的 API 輕鬆互動的 API。

管理 API 園林

本章開頭說過，API 管理領域有兩項重大的挑戰：管理單一 API 的生命週期，以及管理所有 API 組成的園林。我們在拜訪許多公司並且研究如何管理 API 時，發現許多的“管理單一 API”的方法。外界有許多可讓你在設計、建構與部署 API 時，用來辨認與減輕困難的“生命週期”與“成熟度模型”。但是我們找不到太多關於 API 生態系統（我們稱為園林）的建議。

園林有它自己的挑戰、它自己的行為與傾向。設計單一 API 時應考慮的事項與支援數十、上百或上千個 API 時應考慮的事項不一樣。在生態系統中，你會遇到關於**規模**的新挑戰——事情不僅僅發生在單一 API 實例或實作中。本書稍後會更深入討論 API 園林，但是我們想要在本書的開頭指出 API 園林為 API 管理帶來的三種特殊挑戰：

- 技術的擴展
- 團隊的擴展
- 治理的擴展

讓我們花點時間回顧一下這些 API 園林管理層面。

技術

當你第一次開始進行 API 專案時，必須做一些將會影響所有 API 的技術決策。此時，你“所有”的 API 只有兩、三個並不是重點，重要的是，你要在建立第一個 API 專案時，就有一組一致的工具與技術可以依靠。你會在討論 API 生命週期（第 6 章）與 API 成熟度時看到，API 專案並不便宜，你必須仔細地監控你在這類的活動中投入的時間與精力：也就是在不過早投入大量資金的情況下，會對 API 的成功產生重大影響的活動。這通常意味著你要選擇與提供一組小工具，以及非常明確、通常也很詳細的指導文件，協助 API 團隊設計與建構既能解決商業問題，又能良好地合作的 API。換句話說，你可以藉由限制技術的範圍，在早期就取得成功。

這種做法在一開始很管用。但是隨著專案數量的增加（見第 166 頁的“數量”）以及範圍的擴大（例如有更多團隊建構更多 API，在更多地點為更多業務提供服務等等），挑戰也會改變。當你發展 API 專案時，依靠有限的工具與技術可能是降低速度的關鍵因素。雖然在團隊還很小的初期，限制選擇可加快工作的進度，但是對大型的團隊施加限制則是代價高昂且風險很大的做法，當你在遙遠的地點增設團隊，並且 / 或者接收新的

業務單位或收購新公司，讓它們進入 API 園林時更是如此。此時，多樣性（見第 161 頁的“多樣性”）對生態系統而言是更重要的成功因素。

所以，當你管理讓 API 園林使用的技術時，重點是認出園林的規模何時已經大到可以開始提升技術的多樣性，而不是限制它們。其中有一些因素與既有實作的現況有關，如果你的 API 園林需要支援既有的 TCP/IP SOAP 服務，你就不能要求所有服務都使用之前為全新的 HTTP CRUD API 建立的同一份 URL 指南。為新的事件導向 Angular 作品或舊的遠端程序呼叫（RPC）作品建立服務也是如此。

更廣大的範圍代表園林有更高的技術多樣性。

團隊

隨著專案的成長，管理 API 不是只會遇到技術層面的新挑戰，團隊本身的組成也需要隨著園林的變動而調整。同樣的，在 API 專案開始的時候，你可以和少數幾位忠誠的夥伴一起（在大部分的情況下）做任何事情。這就是“full-stack developer”或“MEAN”MongoDB、Express.js、Angular.js、Node.js 開發者或其他稱謂的由來，它們代表有位開發者擁有 API 專案所有層面的技術。你可能也聽過“創業團隊”或“獨立團隊”，它們都代表一個擁有所有技術的團隊。

當你的 API 很少，而且它們全部都是用同一組工具來設計與實作的時候，你可以採取這種做法（見第 10 頁的“技術”），但是隨著 API 的規模與範圍變大，建構與維護 API 的技術量也會增加，你再也不能期望各個 API 團隊都是由一組擅長設計、資料庫、後端、前端、測試與部署的成員組成的，你可能會指派一個團隊負責設計與建構資料導向的儀表板介面，來讓其他團隊使用。舉例來說，他們可能要瞭解所有資料格式以及收集那些資料的工具。或者，你可能會讓一個團隊負責藉由單項技術（例如 GraphQL 或某些其他的查詢程式庫）來建構行動 app，隨著技術種類的成長，你的團隊可能要更專業化。我們會在第 7 章更詳細討論這個主題。

團隊需要隨著 API 領域的成長而改變的另一種層面是他們參與日常決策程序的方式。當團隊很小且成員的經驗還不太豐富時，由一個指導小組專門負責決策是合理的做法。在大型的組織中，這種團隊通常稱為 Enterprise Architecture 團隊或其他類似的名稱。這種團隊在較小的規模與範圍是有效的，但是當生態系統的同質性變低且範圍變廣時，它就會變成一個大問題。隨著技術越來越多，單一團隊很有可能跟不上各種工具與框架的細節。而且隨著團隊數量的增加，你可能也要將決策權分配出去，因為中央單位不太可能瞭解一個全球性企業的日常運作情況。

解決這種問題的做法是將決策程序拆成所謂的決策元素（見第 26 頁的“決策元素”），並將這些元素分配給公司內適當的階級。持續成長的生態系統意味著各個團隊必須在技術層面上更專業化，並且在決策層面上肩負更多責任。

治理

關於 API 園林的挑戰，最後一個環節是**治理** API 專案的一般性做法。與之前的其他案例一樣，我們發現，隨著生態系統的成長，治理的角色與手段也會改變。你會遇到新的挑戰，且舊方法也不像過往那麼有效。事實上，堅持採取舊的治理模式可能會減緩甚至阻礙 API 的成功，對企業而言尤其如此。

如同任何一個領導領域，當規模與範圍有限時，最有效的做法是提供直接的指導。對小型的團隊如此，對**新**團隊來說也是如此。如果團隊沒有太多經驗，快速成功的方法就是用詳細的指南與 / 或流程文件來傳授經驗。例如，我們發現在早期治理 API 專案的方式通常是用多頁的流程文件來解釋特定的工作：如何為 API 設計 URL，或 URL 可以使用哪些名稱，或版本號碼應該放在 HTTP 標頭的哪裡。提供明確的、幾乎沒有什麼選項的方針可讓開發者確實執行已被核准的 API 實作方式。

但是同樣的，隨著專案的成長，當你加入更多團隊與支援更多商業領域時，社群的規模與範圍開始難以讓所有團隊使用單一指導文件。雖然你可以把編寫與維護流程文件這項工作“外包”出去，但是這種做法通常不好——第 10 頁的“技術”談過，技術的多樣性在大型的生態系統之中是一種優勢，試著在企業治理層面上控制它可能會減緩專案的進度。

這就是當 API 領域擴展時，治理文件也要從提供直接處理方針轉變成提供一般原則的原因。例如，與其提供文件來說明哪些東西可以構成有效的 URL，比較好的做法是引導開發者查看 Internet Engineering Task Force 關於 URI design and ownership 的指南（RFC 7320），並提供如何在機構內執行這項公共標準的一般性指導。你也可以在大部分的 UI/UX 指南中找到這種**原則性指南**的例子，例如 Nielsen Norman Group 的“10 Usability Heuristics for User Interface Design”（*https://www.nngroup.com/articles/ten-usability-heuristics/*）。這類文件提供的是許多選項，以及使用某種 UI 模式而非另一種的理由。它們讓開發者與設計師知道為何 / 何時應使用某個東西，而不是直接規定他們必須遵循哪些規則。

最後，對非常大型的機構而言（尤其是在多個地點與時區營運的公司），治理的方式必須從公布原則轉向收集建議。實質上，這種做法翻轉了典型的中央治理模式。中央治理單位的主要工作是收集現場的經驗資訊、找到關聯性，並提供“最佳做法”指南給廣泛的內部單位，而不是告訴各個團隊該做什麼。

所以，隨著 API 園林的成長，API 治理模式也要從提供直接的建議變成提出一般性的原則，收集有豐富經驗團隊的做法分享給其他團隊。第 2 章會介紹一些建立適合你公司的治理模式的原則與做法。

總結

在開頭的這一章，我們談到一些即將在本書中出現的 API 管理面向。我們承認，雖然 API 是一種驅動力，但是在接受調查的公司中，只有 50% 的公司對於自己是否正確地管理 API 充滿信心。我們也釐清了“API”這個名詞的各種意思，以及這些意思為何讓你很難提供一致的專案治理模式。

而且，最重要的是，我們介紹了管理“一個 API”與管理“API 園林”有很大的不同。在第一種情況下，你可以依靠 API 即產品、API 生命週期與 API 成熟度模型。當你管理 API 的變動時，也會採取這種“單一 API”的思維。但是故事不只於此。

接著我們討論了 API 園林的管理，園林就是機構的整個 API 生態系統。管理持續成長的 API 園林需要使用各種技術與指標，你要用這些技術來處理多樣性、數量、易變性、漏洞與其他的層面。事實上，這些園林層面都會影響 API 生命週期，本書稍後會再討論它們。

最後，我們指出做出 API 專案決策的方式也要隨著時間而改變。隨著系統的成長，你也要像分配 IT 元素（例如資料存放區、計算功能、安全防護與公司基礎架構的其他部分）一樣分配決策權。

以本章的介紹為背景，接下來我們要先討論**治理**的概念，以及如何將決策與分配決策當成整體 API 管理方法的主要元素。