深入淺出軟體架構的作者們

Raju Gandhi

Raju 是建築師、顧問、作者和教師，經常受邀在世界各地的研討會上發表演說。他堅信事物應保持簡單，他的學習方法一向是理解並解釋「為什麼」，而不是「怎麼做」。他和他的賢妻 Michelle 住在俄亥俄州的哥倫布市，同住的還有他們的兒子 Mason 和 Micah，女兒 Delphine，以及三隻毛小子 Buddy、Skye 和 Princess Zara。你可以在 *RajuGandhi.com* 找到他的聯絡資訊。

Mark Richards

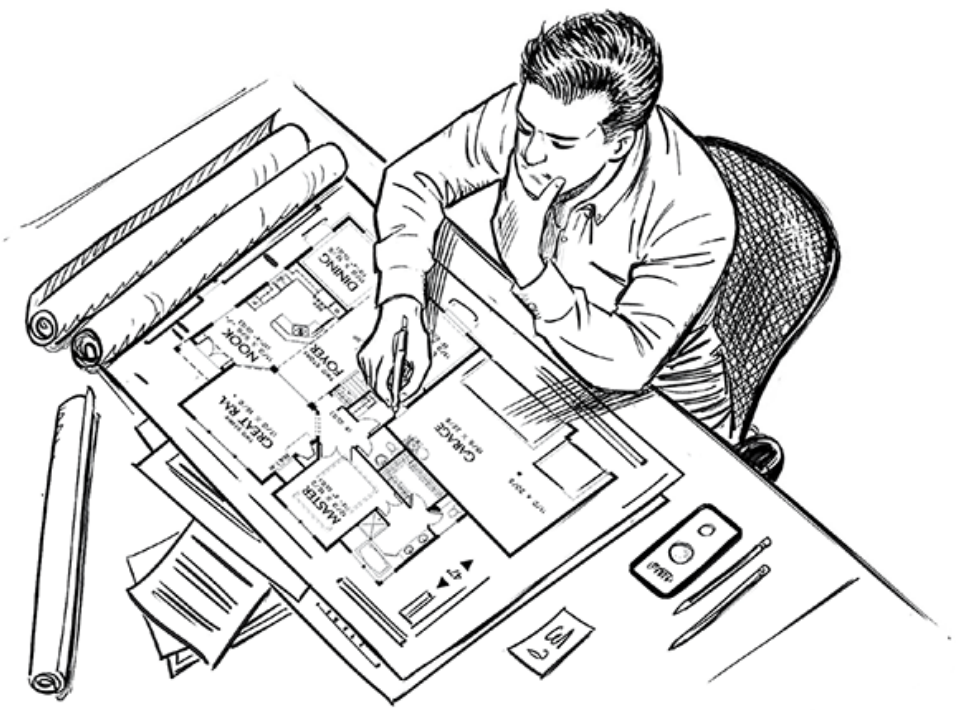
Mark 是經驗豐富的實作型軟體架構師，也是 DeveloperToArchitect.com 的創辦人。DeveloperToArchitect.com 是免費的網站，致力於協助開發者成為軟體架構師。他自 1983 年起便投入軟體產業，在應用程式、整合和企業架構等領域具備豐富的專業經驗。Mark 出版了多本技術書籍和影片，包括《*Fundamentals of Software Architecture*》，以及與 Neal Ford 合著的《*Software Architecture: The Hard Parts*》（O'Reilly）。Mark 也從事教育訓練，並在世界各地的數百場研討會和使用者群組中發表演說。

Neal Ford

Neal Ford 是 ThoughtWorks 的總監、軟體架構師和謎因管理員。ThoughtWorks 是一家專門指導端到端軟體開發和交付的全球 IT 諮詢公司。他也設計和開發了許多應用程式、文章和影片，並且著作和編輯越來越多書籍，涵蓋各種主題和技術。他的專業主要是設計和建立大型企業應用程式。他也是國際知名的演講者，在全球超過 300 場開發者會議上發表演講，並進行了超過 2,000 場演講。

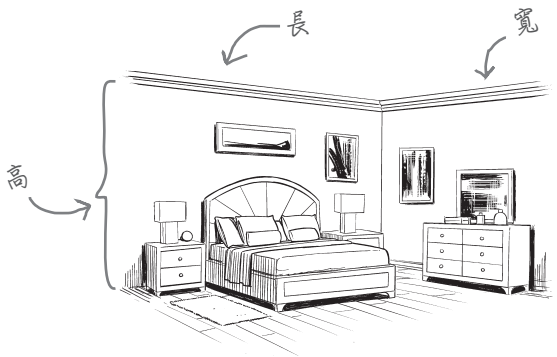
1 揭開軟體架構的神祕面紗

我們出發囉！



軟體架構是成功系統的基石。這一章要為你揭開軟體架構的神祕面紗。你將瞭解架構的各個維度，以及架構和設計之間的差異。為什麼這件事很重要？因為瞭解架構實踐法並加以應用，可協助你建構更高效且正確的軟體系統，這種系統不但表現得更好，也符合商業需求和關注點，而且能夠隨著商業和技術環境不斷變遷而持續運作。事不宜遲，我們開始吧。

軟體架構的維度



在我們四周的東西幾乎都是多維的。例如，你可能會說，你家有一間房間是長 5 米、寬 4 米，天花板高度 2.5 米。注意，為了正確地描述房間，你要說出全部的三個維度，包括高度、長度和寬度。

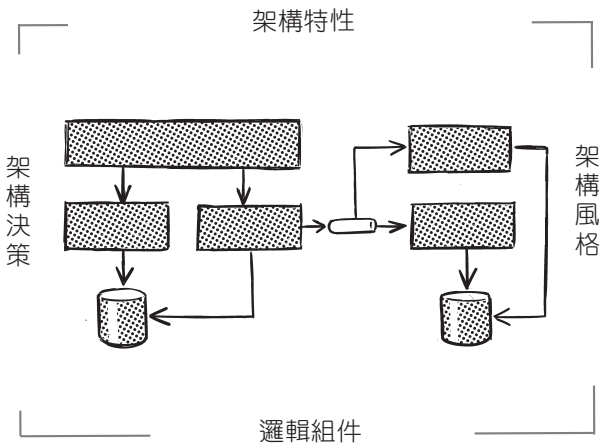
你也可以用維度來描述軟體架構，不同的是，軟體架構有**四個維度**。

1 架構特性

這個維度定義了架構必須支援系統的哪一些層面，例如可以隨需擴展、容易測試、妥善性…等。

2 架構決策

這個維度包含對系統而言，長期或重要的決策，例如，它使用哪一種資料庫、它有多少服務，這些服務之間如何溝通。



3 邏輯組件

這個維度定義了系統功能的基本要素，以及它們如何互動。例如，電子商務系統可能有庫存管理、支付處理…等組件。

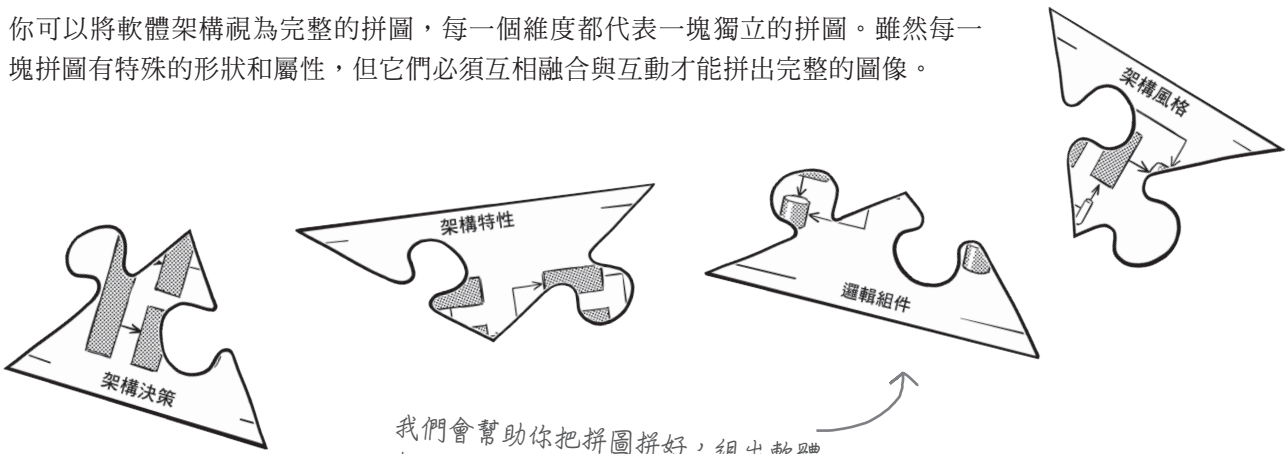
4 架構風格

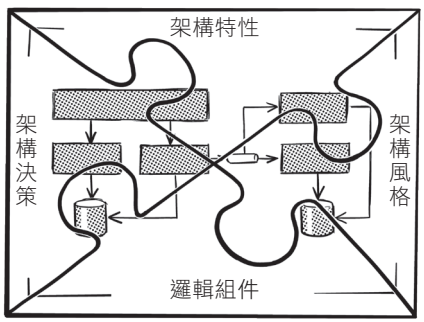
這個維度定義了軟體系統的整體實體外形和結構，就像建築平面圖定義了你家的整體外形和結構一樣。

本書會教你最常見的五種架構風格。

拚湊維度

你可以將軟體架構視為完整的拼圖，每一個維度都代表一塊獨立的拼圖。雖然每一塊拼圖有特殊的形狀和屬性，但它們必須互相融合與互動才能拼出完整的圖像。





一切都是互相關聯的。

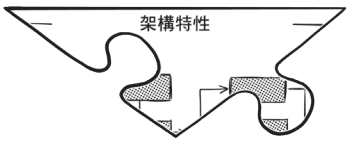
你是否注意到，在這個拼圖的中央，各個部分是怎麼合併起來的？這正是軟體架構的工作原理：每一個維度都必須協調一致。

架構風格和你選擇的架構特性以及你所做的架構決策必須一致。同理，你定義的邏輯組件必須與特性、架構風格以及你的決策一致。

問：在建立架構時，是否需要全部的四個維度？還是說，如果時間不夠的話，可以跳過其中的幾個？

答：很遺憾，這些維度都不能跳過，它們都是建立和定義架構時的必要維度。軟體架構師常犯的錯誤之一就是只用其中的一兩個維度來定義架構。「我們的架構是微服務」這句話只定義了一個維度，即架構風格，卻留下太多未解的問題。例如，哪些架構特性對系統的成功來說極其關鍵？它的邏輯組件（功能基本要素）是什麼？你為架構的實作方式做出哪些重大的決策？

第一維：架構特性



架構特性是軟體系統架構的基礎。沒有它們就無法做出架構決策，或權衡重要的取捨。

想像你在兩棟房子之間做出選擇。其中一棟房子很寬敞，但旁邊有一條繁忙、嘈雜的高速公路。另一棟房子位於漂亮、安靜的社區，但面積小得多。**對你來說，哪一個特性比較重要？**是房子的大小，還是噪音的大小及交通的便利程度？如果你不知道這件事，你就無法做出正確的選擇。

軟體架構也是如此。假設你要決定新系統該使用哪一種資料庫，究竟要用關聯資料庫、簡單的鍵值資料庫，還是複雜的圖資料庫（graph database）？答案取決於哪些架構特性對你來說非常重要。例如，如果你需要快速搜尋的能力（我們稱之為**效能**），也許你會選擇圖資料庫。如果你需要記錄資料關係（我們稱之為**資料完整性**），傳統的關聯資料庫可能比較好。

效能
系統處理業務請求所需的時間

可善性
系統正常運行的時間，通常以「幾個九」來衡量（所以 99.9% 是三個「九」）

隨需擴展性
當使用者或請求的數量增加時，系統能否維持一致的回應時間和錯誤率

這些是比較常見的架構特性。第 2 章會介紹它們的完整知識。

習題

選出你認為屬於架構特性的項目，也就是軟體系統的結構所支援的東西。

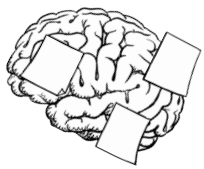
- [ ] 改變使用者介面的字體大小
- [ ] 讓變更可以快速進行
- [ ] 處理數千位同時使用的使用者
- [ ] 將資料庫內的使用者密碼加密
- [ ] 與多個外部系統互動，以完成業務請求

解答在第 30 頁。

也許**架構特性**這個術語對你而言很陌生，但這不代表你沒有聽過它們。諸如效能、隨需擴展性、可靠性，和妥善性這類的性質統稱為「非功能需求」、「系統品質屬性」，或直接稱為「-ility（…性）」，因為它們幾乎都以 *-ility*（…性）結尾。我們喜歡使用架構特性（*architectural characteristics*）這個術語，因為它們能夠幫助你定義架構的特性，以及架構需要支援的事情。

架構特性是對系統的成功而言，極其關鍵或重要的能力。

牢牢記住



為了設計軟體架構，你要先處理：

對新 app 的成功而言關鍵的能力

連連看？

給你一個機會，看看你對常見的架構特性瞭解多少。能不能將每一個架構特性（左邊）與它的定義（右邊）連起來？如你所見，定義比特性還要多，所以小心，並非每一個定義都有對應的特性。

功能延伸性 ← 我們幫你完成了這一題。

敏捷性

互操作性

容錯性

可行性

在選擇架構時，考慮時間範圍、預算和開發者技能

當系統發生致命錯誤時，其他部分維持運行的能力

改善系統以支援額外功能和特性的難易程度

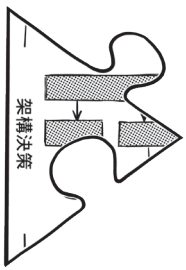
使用者獲得回應所需的時間

系統快速回應變更的能力（易維護性、易測試性和易部署性的函數）

系統與其他系統對接並互動，以完成業務請求的能力

解答在第 30 頁。

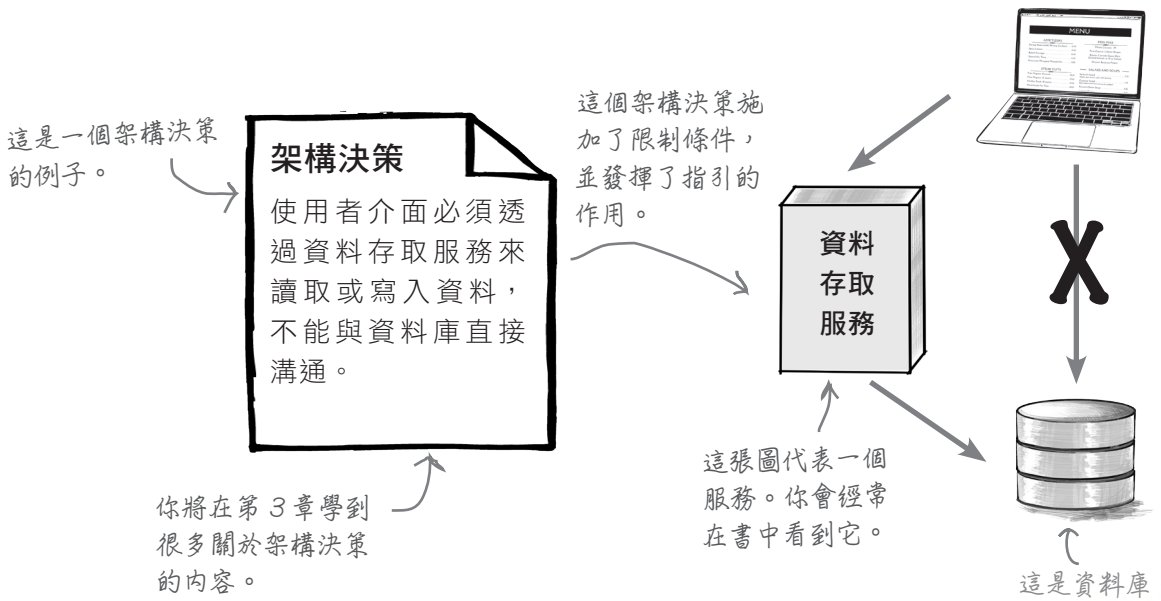
第二維：架構決策



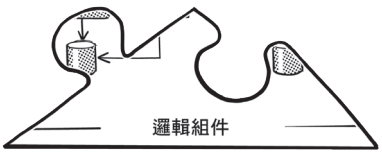
架構決策就是你為系統的結構層面做出來的選擇，那些選擇具有長期或重大的影響力。它們是限制條件，將引導你的開發團隊在規劃和建構系統時做出決策。

你的新房子要蓋成一樓還是兩樓？屋頂是平的還是尖的？你要蓋一座巨大的、廣闊的牧場式房屋嗎？以上這些都是架構決策案例，因為它們涉及房子的結構層面。

你可能決定，系統的使用者介面不應該與資料庫直接溝通，而是必須透過底層服務來讀取和更新資料。這個架構決策對使用者介面的開發施加了特定的限制條件，並且指引開發團隊如何讓其他組件存取和更新資料庫中的資料。

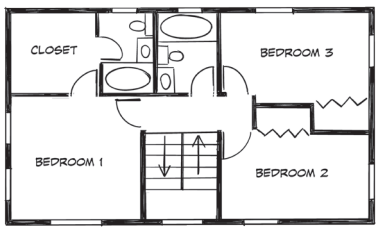


第三維：邏輯組件

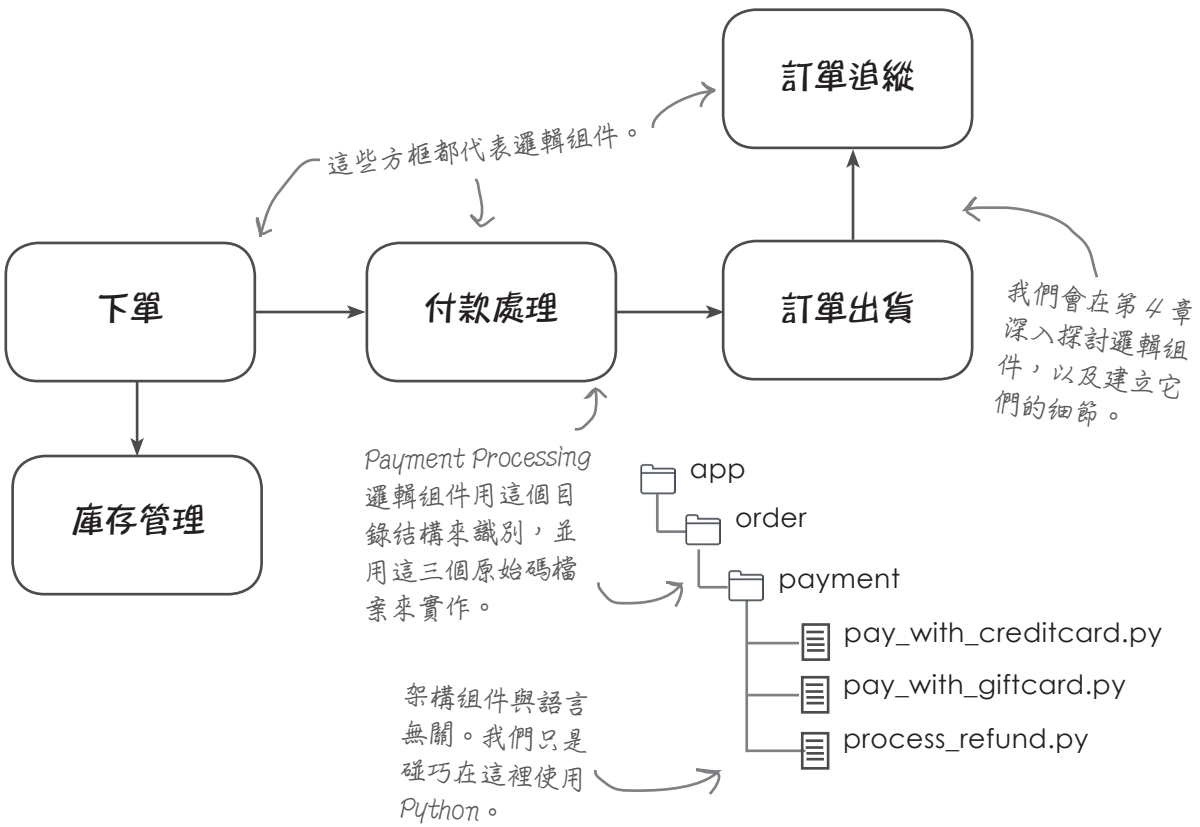


邏輯組件是系統的基本要素，就像房間是你家的基本要素一樣。邏輯組件可執行某種功能，例如處理訂單付款、管理物品庫存，或追縱訂單。

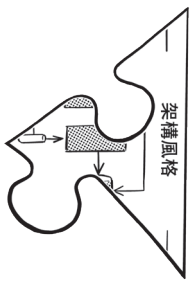
這些房間是你家的基本要素。



系統的邏輯組件通常用目錄或名稱空間來表示。例如，目錄 `app/order/payment` 及其相應的名稱空間 `app.order.payment`，代表一個名為「Payment Processing」的邏輯組件。讓使用者付款的原始碼會被存放在這個目錄中，並使用這個名稱空間。



第四維：架構風格



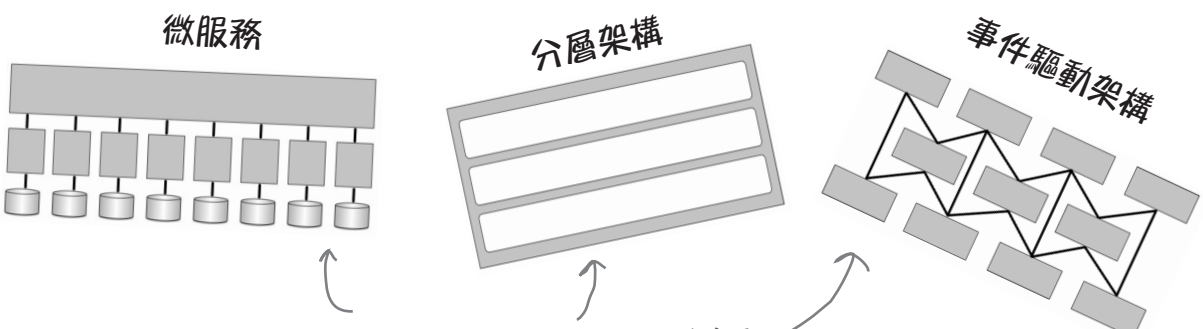
房子有各種形狀、大小和風格。雖然有些房子看起來很奇特，但大多數都屬於特定風格，例如維多利亞風格、牧場風格，或都鐸風格。房子的風格對整體結構有很大的影響。例如，牧場風格的房子通常只有一層，殖民地風格和都鐸風格的房子通常有煙囪，當代風格的房子屋頂通常是平的。

世界各地的房子有各自的風格，你可以到 https://en.wikipedia.org/wiki/List_of_house_styles 欣賞它們。

你的住家屬於哪一種風格？

架構風格定義了軟體系統的整體外形和結構，每一種風格都有獨特的特性。例如，**微服務**架構風格非常容易擴展，且極其敏捷（這是快速應對變化的能力），**分層**架構風格則成本較低且較為簡單。**事件驅動**架構風格很容易擴展，非常快速，且反應迅速。

別擔心——後面會教你這些架構風格。每一種風格都有專屬的章節。



架構風格五花八門，幸好不像房屋的風格那麼多。

並非如此！架構與設計是兩碼事。

你看，架構與外觀比較沒有關係，與結構比較有關。反之，設計與結構比較無關，與外觀比較有關。

房間牆壁的顏色、家具的位置、地板的類型（地毯或木地板）都是設計層面，而房間的物理大小、門窗的位置則是架構的一部分，換句話說，它們是房間的結構。

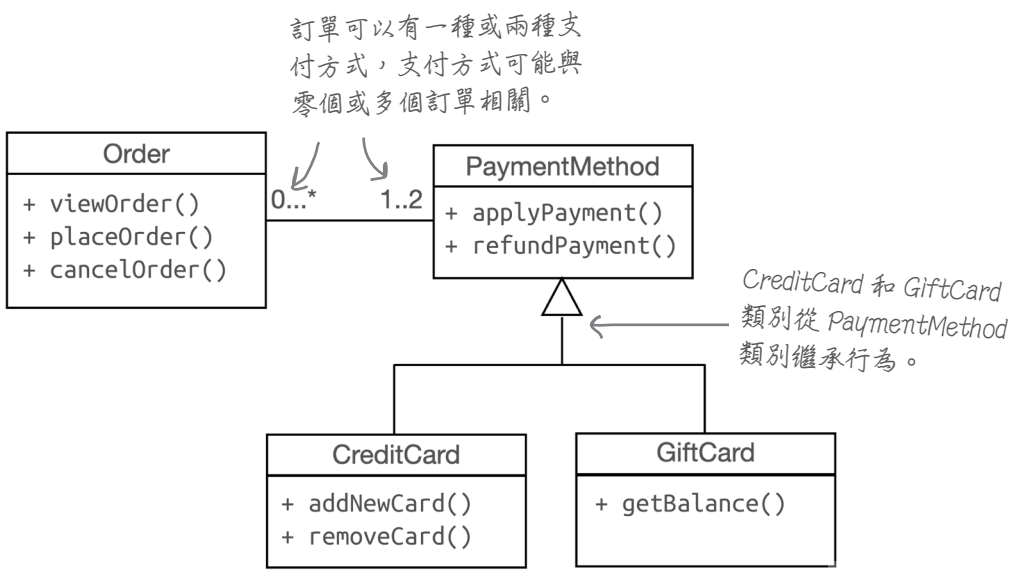
想一想一個典型的商業應用程式。架構（或結構）是關於網頁如何與後端服務和資料庫溝通以提取和保存資料，而設計是關於每一頁的外觀：顏色、欄位的位置、你使用的設計模式…等。這同樣是結構 vs. 外觀的問題。

你的問題很棒，因為很多人搞不清楚什麼是架構、什麼是設計。我們來研究一下這些差異。

設計角度

假設你的公司想要把過時的訂單處理系統換成更符合特定需求的自製新系統，讓顧客可以下單，並在下單後查看或取消訂單。他們可以使用信用卡、禮品卡，或兼用兩者來支付訂單。

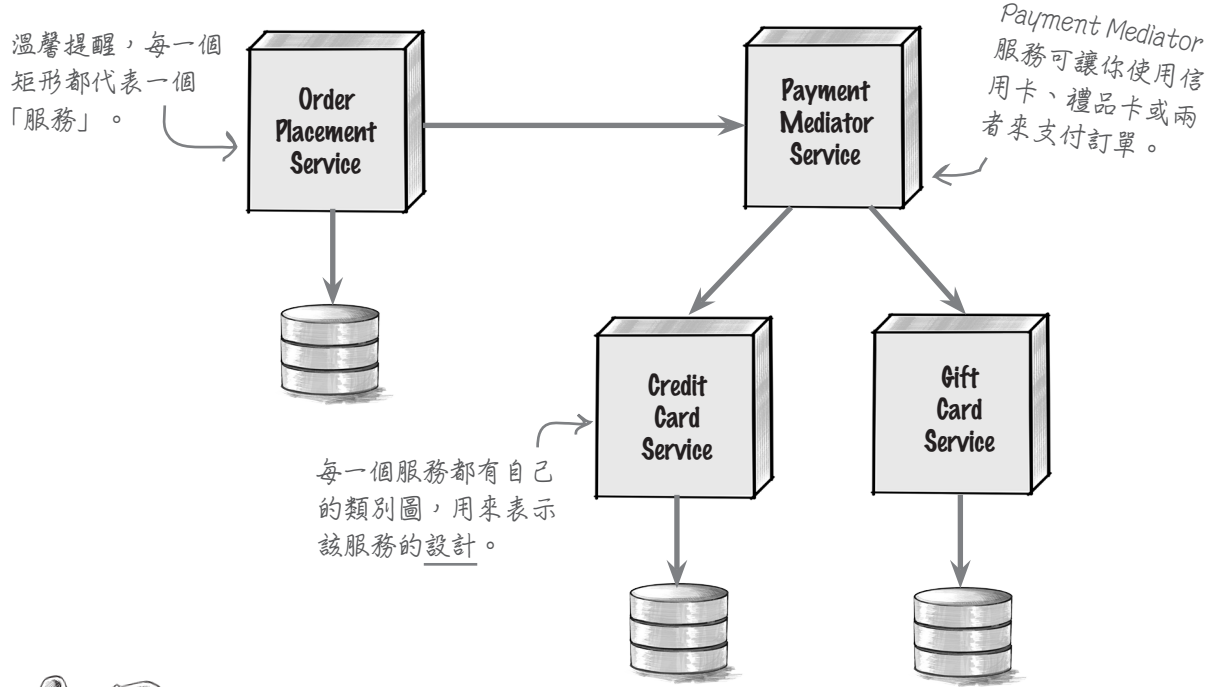
從設計的角度來看，你可能會畫一張 Unified Modeling Language（UML）類別圖（如下圖所示），以展示類別如何互動，並實現支付功能。雖然你可以編寫原始碼來實作這些類別檔案，但這個設計並未說明原始碼的實體結構，也就是這些類別檔案該如何組織和部署。



架構角度

架構與設計不同，它關注的是系統的結構——例如服務、資料庫，以及服務之間和使用者介面之間如何溝通。

我們再來考慮之前的新訂單處理系統。系統將會長怎樣？從架構的角度來看，你可能會幫訂單支付程序中的每一種支付方式建立一個獨立的服務，並安排一個指揮（orchestrator）服務來管理系統的支付處理部分，如下圖所示。



習題

選出應該加入**架構角度**圖表的東西。

- [ ] 服務之間如何溝通
- [ ] 用來實現服務的平台和語言
- [ ] 哪些服務可以訪問哪些資料庫
- [ ] 會有多少服務和資料庫

解答在第 34 頁。

分析取捨

魚與熊掌不可兼得。這個世界充斥著妥協——我們經常犧牲一件事來優化另一件事。想要用手機來拍攝和儲存大量照片嗎？要嘛，你要取得更多儲存空間，這需要付出更高的成本，要嘛，你要壓縮照片，這會降低圖像的品質。

軟體架構也不例外。你做出來的每一個選擇都涉及重大的妥協，或者用我們喜歡的說法：**取捨**。那麼，對你來說，這意味著什麼？

如果你覺得這聽起來很熟悉，這並不意外！這就是我們在第 1 章中討論重大 vs. 較不重要的取捨時的部分內容。

如果你知道哪些架構特性對你的專案而言最重要，你可以開始設計解決方案來將這些屬性之中的一些最大化。但是如果解決方案可讓你將一個（或多個）特性最大化，那麼作為代價，它將犧牲其他特性。例如，實現高隨需擴展性的解決方案可能會讓系統更不容易部署，或更不可靠。

無論你想出什麼解決方案，它都會伴隨著取捨，也就是優點和缺點。

你的工作有兩個方面：瞭解你所提出的每一個解決方案的取捨，然後選出最能夠提供最重要的架構特性的解決方案。

Clojure 程式語言的作者 Rich Hickey 說過：「程式設計師知道每一件事情的好處，卻不瞭解任何事情的權衡取捨。」我們想補充：「架構師需要瞭解兩者。」

你無法一網打盡。你必須決定哪些架構特性最重要，並選擇最能夠實現那些特性的解決方案。

做出架構決策

雖然在白板前和團隊辯論優缺點是有趣的事情，但你終究必須在某個時間點做出**架構決策**。

我們曾經在第 1 章提到架構決策，接下來要更深入討論。當你建構和設計系統時，你會做出很多決策，從系統的整體結構，到該使用哪些工具和技術。那麼，什麼樣的決策是架構決策？

在多數情況下，影響系統結構的任何選擇都是架構決策。以下有幾個決策的例子：

幫你回憶一下，決定要蓋一層樓或兩層樓的房屋是一種架構決策。

我們將使用快取來減少資料庫的負擔並提高效能。

注意，這個決策額外加入一項基礎設施。這也是實作團隊在讀取或寫入資料時必須時刻記在心裡的事情。

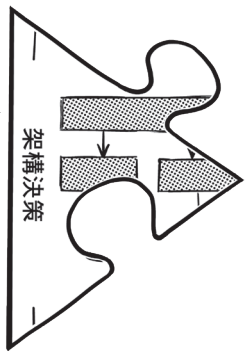
我們要把報告服務做成模組化單體架構。

這個決策很明顯，它在字面上指出服務的結構。

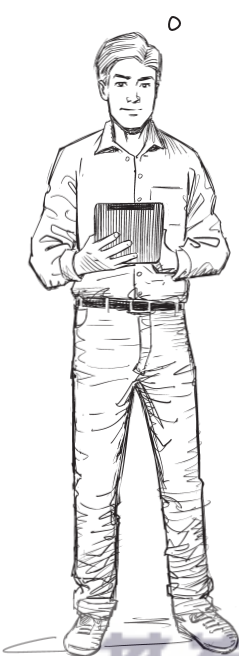
正如我們在第 1 章說過的：「架構決策是協助開發團隊瞭解架構的限制和條件的指南」。

注意這些決策都是指南而不是規則。它們幫助團隊做出選擇，但不會太具體。你做出來的大多數（但不是全部）架構決策都將圍繞著系統的結構展開。

你猜對了！我們正在討論軟體架構的第二維。



我想，我們已經有一個決策了：我們要把訂單運送服務與訂單追縱服務分開。



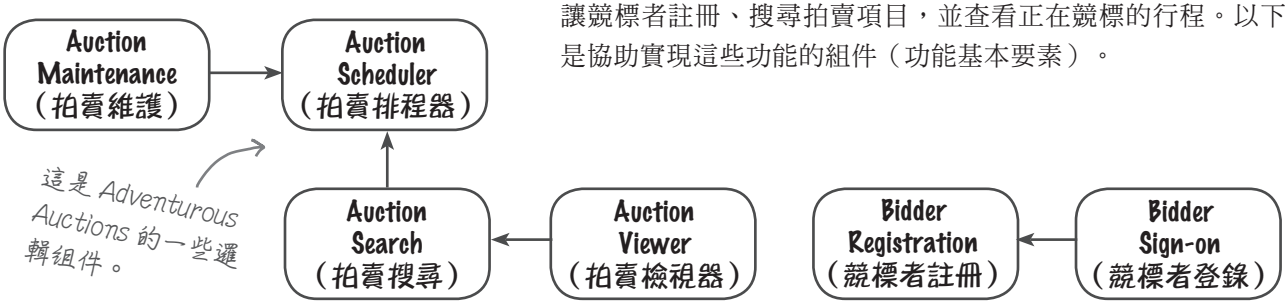
邏輯架構 vs. 實體架構

邏輯架構呈現的是系統的所有邏輯基本要素以及它們之間如何互動（稱為**耦合**）。**實體架構**則展示架構風格、服務、協定、資料庫、使用者介面、API 閘道…等事物。

本章稍後會討論關於組件耦合的許多內容。

系統的邏輯架構與實體架構無關——這意味著邏輯架構不關心資料庫、服務、協定…等。我們來看一個邏輯架構的例子。

如你所知，Adventurous Auctions 需要建立與安排線上拍賣，讓競標者註冊、搜尋拍賣項目，並查看正在競標的行程。以下是協助實現這些功能的組件（功能基本要素）。



有沒有看到邏輯架構不包含上述的實體架構中的各種組件？它是系統的不同視角。若要瞭解我的意思，你可以比較上圖與下面的實體架構圖。注意實體架構如何展示服務與邏輯架構中的組件之間的關係，並顯示系統的服務和資料庫。