CONTENTS

前言

我們很榮幸和你一起學習「系統設計面試」。在所有技術性面試中，系統設計的面試題目往往最難對付。題目或許會要求受試者設計出一套軟體系統架構，完成一些像是動態訊息、Google 搜尋、聊天系統之類的功能。這種題目感覺蠻嚇人，而且往往沒有固定的模式可循。題目所涵蓋的範圍通常很廣泛，而且又很籠統。處理的方法往往很開放、不會很明確，也沒有所謂標準或正確的答案。

目前有許多公司廣泛採用這種系統設計面試的做法，因為所測試出來的溝通能力與解決問題的技能，與軟體工程師日常工作所需非常類似。只要觀察受試者如何分析這些模糊的問題、看她如何逐步解決問題，就可以對受試者做出評估。這種做法可以測試出來的能力，還包括她如何解釋其構想、如何與他人進行討論、如何對系統進行評估，以及如何進行最佳化。

在英語的文字中，使用「她」（she）總比老是用「他或她」（he or she）來得流暢些，而且也比我們在兩種說法之間變來變去好得多。為了讓各位閱讀時輕鬆一點，本書將統一使用女性的「她」。我們並不是故意不尊重男性工程師喲。

系統設計問題通常是開放式的。就像在現實世界一樣，系統經常存在許多差異與變化。我們希望得到的結果，其實就是提出一套可實現系統設計目標的架構。不同的面試官，也有可能讓討論內容偏向不同的方向。有些面試官可能會選擇比較高階的架構來涵蓋所有面向；有些人則可能選擇其中一個或多個領域來聚焦。一般來說，一開始就應該先好好理解系統的需求、限制與瓶頸，才能找出面試官與受試者共同認可的方向。

本書的目的就是提供一種可靠的策略，以解決各種系統設計問題。正確的策略與知識，對於面試的成功來說至關重要。

1

使用者人數——從零到百萬規模

設計出一個可支援好幾百萬使用者的系統，相當具有挑戰性，而且這是一個需要不斷完善、不斷改進的過程。我們打算在本章先打造出一個可支援單一使用者的系統，然後再逐步擴大規模，以服務好幾百萬個使用者。閱讀完本章之後，你就可以掌握一些技巧，協助你解決一些系統設計面試問題。

單一伺服器

千里之行始於足下，打造複雜系統也是如此。我們先從簡單的做起，一開始所有東西都只在單一伺服器上執行。圖 1-1 顯示的就是單一伺服器的配置方式，其中所有東西（包括 Web App、資料庫、快取等等）全都在同一個伺服器中執行。

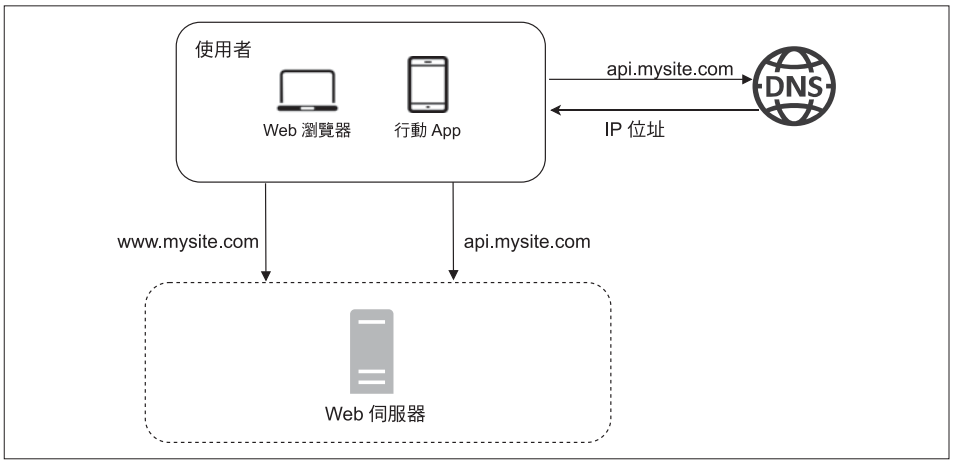

圖 1-1

如果想多瞭解這種配置方式，可以先調查一下請求的流向與流量的來源。我們先觀察一下請求的流向（圖 1-2）。

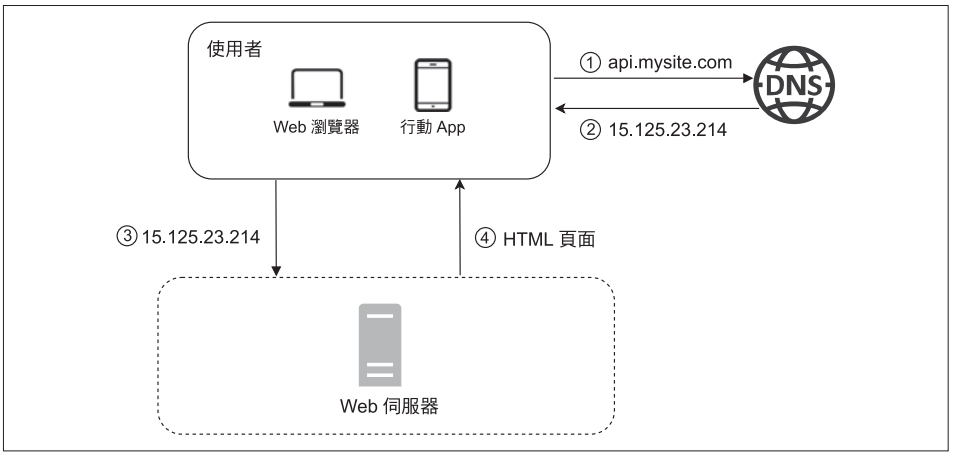

圖 1-2

1. 使用者通常會透過網域名稱（例如 api.mysite.com）來存取網站。DNS 通常是採用第三方所提供的服務，而不會放在我們的伺服器中。
2. IP 位址會被送回瀏覽器或行動 App。在這裡的範例中，送回來的 IP 位址就是 15.125.23.214。
3. 一旦取得 IP 位址，接著就會把 HTTP[1] 請求直接發送到你的 Web 伺服器。
4. Web 伺服器則會送回 HTML 頁面或 JSON 回應，以進行後續的渲染（rendering）工作。

接著我們再檢查一下流量的來源。Web 伺服器的流量主要來自兩種不同的來源：「Web 應用程式」與「行動 App」。

- **Web 應用程式**：伺服端語言（Java、Python 等）負責處理業務邏輯、資料儲存等工作，客戶端語言（HTML、JavaScript）則負責呈現結果。

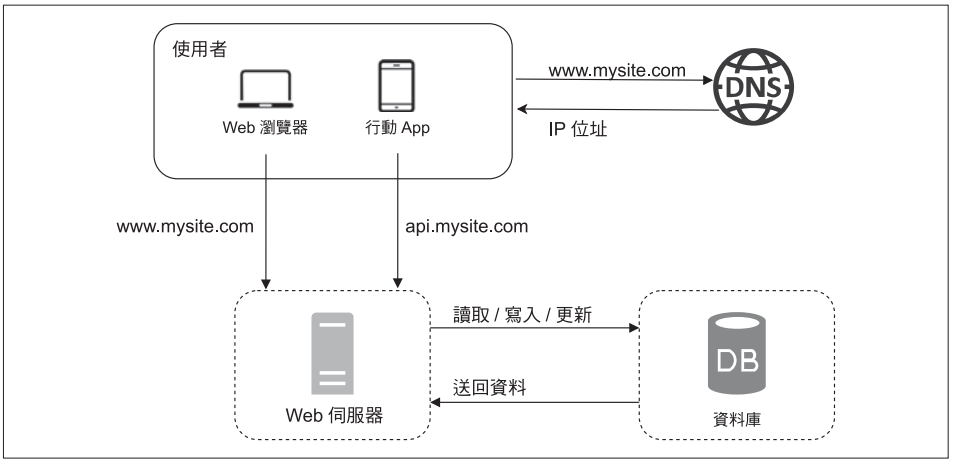

圖 1-3

該使用哪一種資料庫？

你可以在傳統的「關聯式資料庫」與「非關聯式資料庫」兩者之間進行選擇。我們就來看看兩者之間的差異。

關聯式資料庫（Relational database）也稱為「關聯式資料庫管理系統」（RDBMS；relational database management system）或 SQL 資料庫。其中最受歡迎的是 MySQL、Oracle 資料庫、PostgreSQL 等等。關聯式資料庫會把資料儲存在資料表（table）內一行一行（row）的結構之中。你可以透過 SQL 語法，在不同的資料表之間執行 join（聯結）操作。

非關聯式資料庫也稱為 NoSQL 資料庫。其中最受歡迎的就是 CouchDB、Neo4j、Cassandra、HBase、Amazon DynamoDB 等等 [2]。這種資料庫可再分為四類：鍵值（key-value）儲存系統，圖（graph）儲存系統，縱列（column）儲存系統、文件（document）儲存系統。非關聯式資料庫通常並不支援 join（聯結）操作。

對於大多數開發者而言，關聯式資料庫是最佳的選擇，因為這類資料庫已存在超過 40 年，而且從過去的歷史來看，它一直都運作得很好。不過，如果關聯式資料庫並不適合你特定的使用狀況，趁機探索一下關聯式資料庫之外的做法也不錯。對於以下幾種情況來說，非關聯式資料庫有可能是更好的選擇：

- 你的應用程式要求超低延遲（latency）。
- 你的資料是非結構化的，或是你並沒有任何關聯式的資料。
- 你只需要對資料（例如 JSON、XML、YAML 等）進行序列化（serialize）與反序列化（deserialize）的操作。
- 你需要儲存大量的資料。

垂直擴展與水平擴展

垂直擴展（vertical scaling）也稱為「往上擴展」（scale up），指的是針對伺服器添加更多資源或效能（CPU、RAM 等）的一種做法。水平擴展（horizontal scaling）也稱為「往外擴展（scale-out）」，這種做法可透過像是在「資源池」（pool of resources）添加更多伺服器的方式，來達到擴展的效果。

當流量還比較低的時候，垂直擴展是一種不錯的選擇，垂直擴展本身的簡單性就是它主要的優點。不幸的是，這種做法有嚴重的侷限性。

- 垂直擴展有硬體上的限制。我們不可能在單一伺服器中，無上限添加更多的 CPU 與記憶體。
- 垂直擴展並沒有故障轉移（failover）與提供冗餘（redundancy）的效果。如果一部伺服器出現了故障，網站 / App 就完全無法使用了。

由於垂直擴展的侷限性，因此對於大規模的應用而言，水平擴展是一種更加可取的做法。

在之前的設計中，使用者會直接連到 Web 伺服器。Web 伺服器只要一離線，使用者就無法存取該網站了。還有另一種情況是，如果有許多使用者同時存取 Web 伺服器，讓 Web 伺服器達到了負載上的限制，使用者通常就會遇到回應較慢或無法連到伺服器的問題。解決此類問題最好的做法，就是採用「負載平衡器」（load balancer）。

負載平衡器

負載平衡器會把送進來的流量，以一種很均衡的方式分散到負載平衡組合內定義的好幾部 Web 伺服器。圖 1-4 顯示的就是負載平衡器的運作方式。

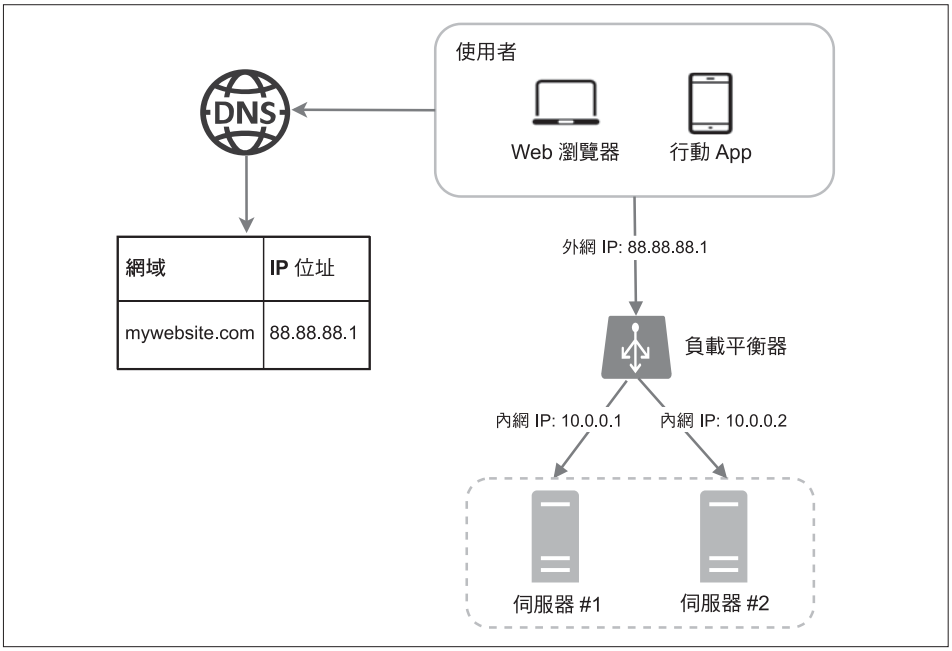

圖 1-4

如圖 1-4 所示，使用者會直接連到負載平衡器的外網 IP 。在這樣的設定下，客戶端就無法直接連到 Web 伺服器了。伺服器之間都是透過內網 IP 進行溝通，因此也可以提高安全性。內網 IP 指的是同一個內網中的 IP 位址，唯有在同一內網中的伺服器，彼此間才能進行溝通，而在 Internet 外

網的機器，則無法直接對內網進行存取。所有外部設備全都必須先連往負載平衡器，才能透過內網 IP 與 Web 伺服器進行溝通。

在圖 1-4 中，我們添加了負載平衡器與第二部 Web 伺服器之後，就可以成功解決故障轉移的問題，並提高 Web 層的可用性（availability）。詳細說明如下：

- 如果伺服器 #1 離線，所有流量就會被導向伺服器 #2。這樣就可以避免網站離線的問題。我們也可以在伺服器池（server pool）加入狀況良好的新 Web 伺服器，以平衡整體的負載。
- 如果網站的流量迅速成長，兩部伺服器不足以處理過大的流量，負載平衡器也可以很優雅地處理這個問題。你只要在 Web 伺服器池加入更多的伺服器，負載平衡器就會自動開始向這些新的伺服器發送請求。

現在從 Web 層來看狀況還不錯，那資料層呢？目前的設計只用到一個資料庫，因此並不支援故障轉移，也無法提供冗餘。資料庫複寫機制（database replication）就是解決這類問題常用的技術。我們就來看一看吧。

資料庫複寫機制

引自維基百科：「許多資料庫管理系統都可以使用資料庫複寫機制，其中原始資料庫（master）與副本資料庫（slave）之間，通常具有主 / 從（master/slave）的關係」[3]。

主資料庫（master）通常只支援寫入操作。從資料庫（slave）則會向主資料庫取得資料的副本，而且只支援讀取操作。所有資料修改的指令（例如 insert 插入、delete 刪除、update 更新）都必須發送到主資料庫。在大多數的應用中，讀取的次數通常遠大於寫入的次數；因此系統的「從資料庫」數量通常都大於「主資料庫」的數量。圖 1-5 顯示的就是一個主資料庫搭配多個從資料庫的情況。

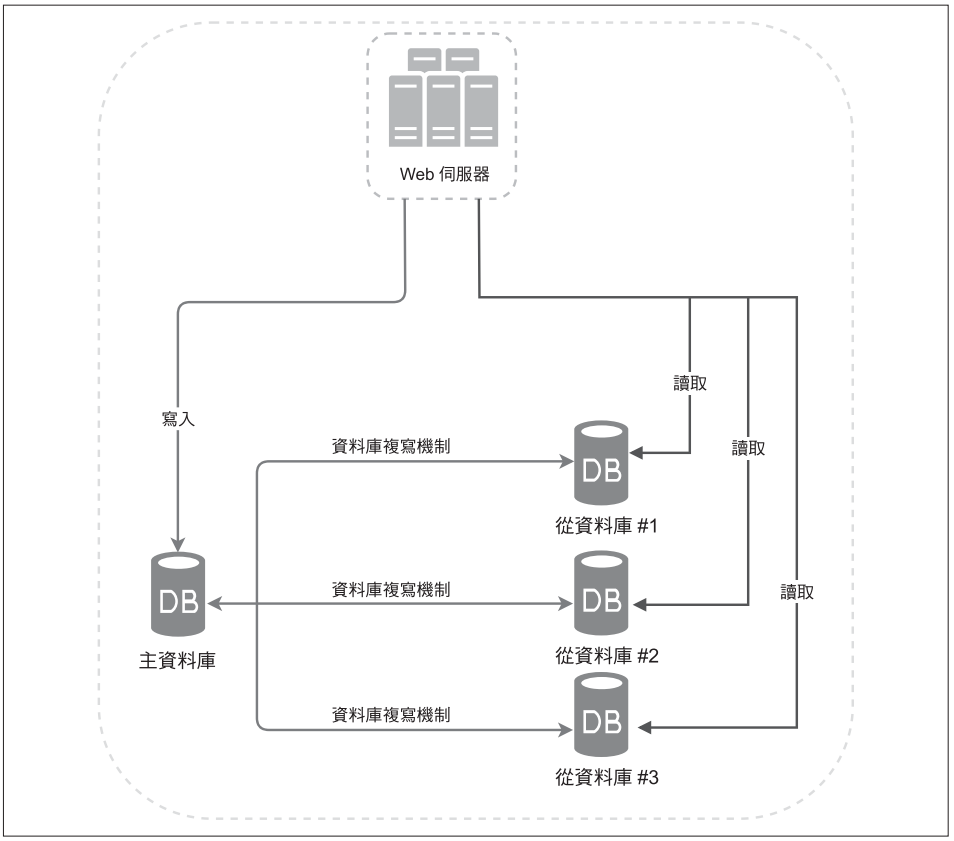

圖 1-5

資料庫複寫機制的優點：

- **更好的效能**：在主從模型中，所有寫入與更新都只會發生在主節點；讀取操作則會分散到各個從節點。這個模型可提高效能上的表現，因為這樣系統就能以平行的方式處理更多的查詢。
- **可靠性**：就算有某部資料庫伺服器因為颱風或地震等自然災害因素而受到破壞，你的資料還是會被保存下來。你不必擔心丟失資料，因為資料已事先在多個位置進行了複製。

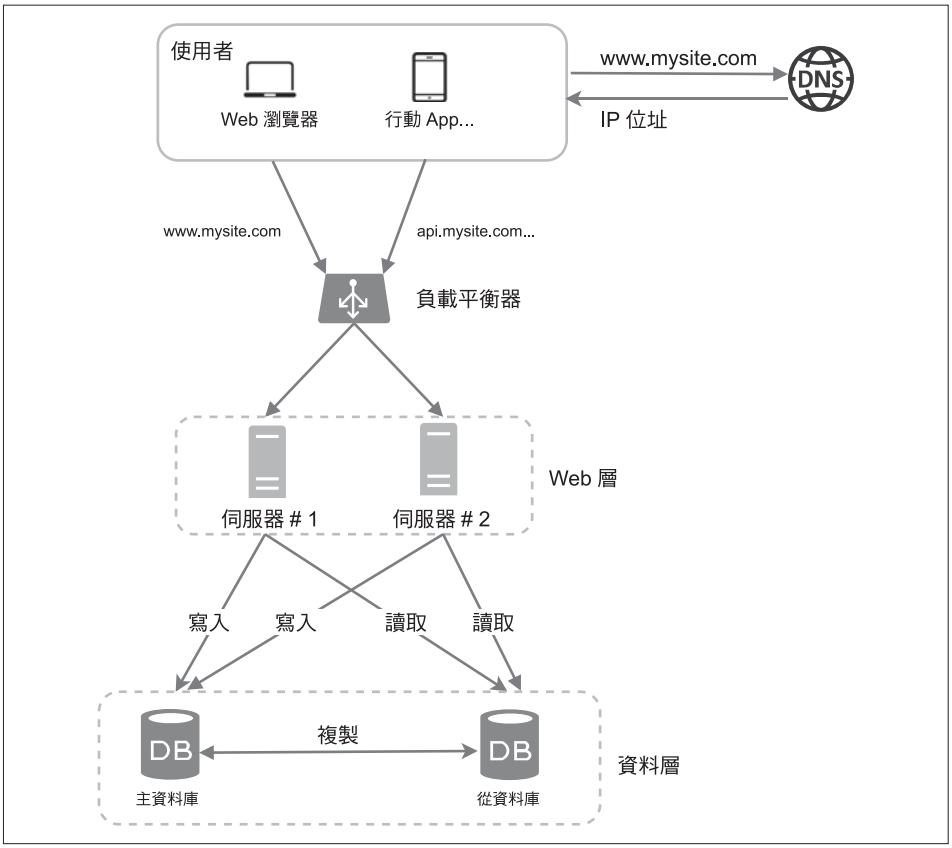

圖 1-6

我們來看看目前這個設計：

- 使用者從 DNS 取得負載平衡器的 IP 位址。
- 使用者用這個 IP 位址連往負載平衡器。
- HTTP 請求被轉送到伺服器 #1 或伺服器 #2。
- Web 伺服器讀取「從資料庫」裡的使用者資料。
- Web 伺服器會把所有資料修改相關操作轉送到「主資料庫」。包括寫入、更新、刪除，都屬於修改相關操作。

- **減輕故障的影響**：單一的快取伺服器就代表可能會有單點故障（SPOF；single point of failure）的問題，維基百科對於單點故障的定義如下：「SPOF 單點故障指的是，如果系統某部分發生故障，整個系統就會停止運作」[8]。基於這個理由，因此我們建議跨越不同資料中心，使用多個快取伺服器，以避免出現 SPOF 單點故障的問題。另一種推薦的做法則是按照一定的百分比，以超額的方式提供所需的記憶體。隨著記憶體使用量的增加，這樣將可以提供一種緩衝的效果。

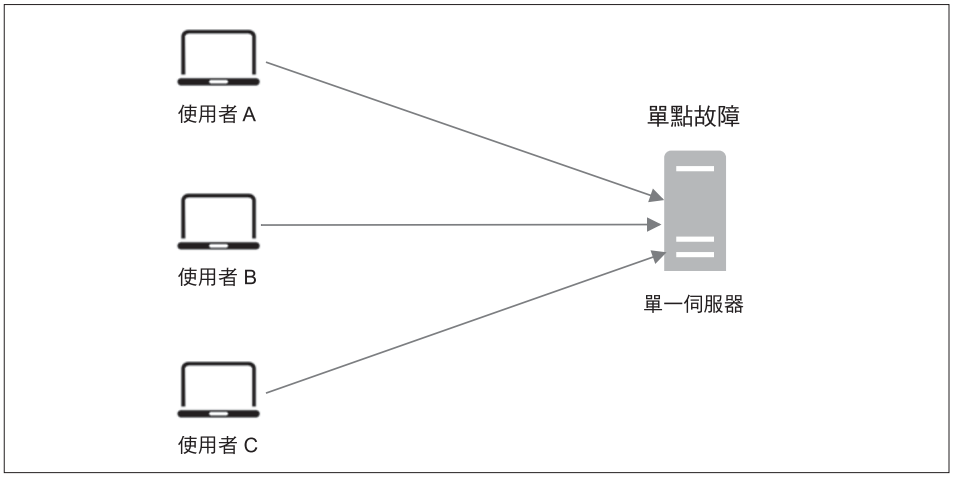

圖 1-8（圖片來源：https://bit.ly/3eGsnyH）

- **逐出策略（Eviction Policy）**：快取一旦滿了之後，如果再把新的項目加入到快取中，就有可能導致原有的項目被移除。這就是所謂的快取逐出（eviction）。其中最常見的一種快取逐出策略，就是所謂的 LRU（Least-recently-used；最近最少被使用）策略。你也可以採用其他的逐出策略，例如 LFU（最不常被使用）或 FIFO（先進先出）策略，以滿足不同的使用狀況。

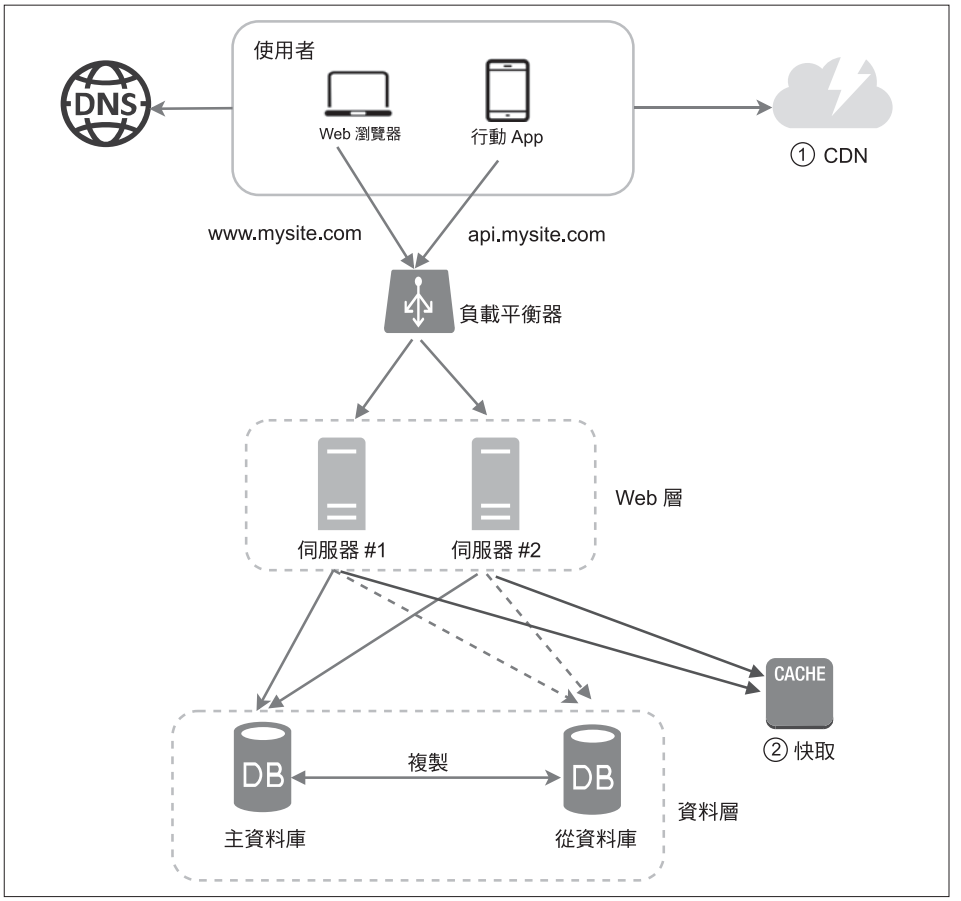

圖 1-11

無狀態網路層

現在該是時候考慮 Web 層的水平擴展了。為此，我們必須把狀態（state，例如使用者 session 資料）移出 Web 層。其中一種好做法就是把 session 資料儲存在像是關聯式資料庫或 NoSQL 之類的持久型儲存系統中。集群內的每個 Web 伺服器都可以從資料庫存取到狀態資料。這就是所謂的無狀態（stateless）Web 層。