第二章

實施 SLO

Steven Thurgood、David Ferguson 與
Alex Hidalgo 及 Betsy Beyer 合撰

服務水準目標（SLO）明確定義了您服務可靠性的目標水準，因為 SLO 是做攸關可靠性決策（而決策最好由資料所驅動）的關鍵，所以也是 SRE 實踐的核心，從很多方面來說，這是本書最重要的一章。

一旦具備了些許指導方針，設定初始 SLO 以及完善之的流程就很簡單了，我們第一本 SRE 書的第 4 章（*http://bit.ly/2szBKsK*）就介紹了 SLO 及 SLI（服務水準指標），並且給了如何使用它們的建議。

在討論了 SLO 及犯錯預算背後的動機之後，本章提供了循序漸進的步驟，讓您開始思考 SLO，以及繼此之後迭代前行的建議。接著我們會涉及如何利用 SLO 做有效的業務決策，然後探討一些進階的主題。最後，我們會提供關於不同種類服務的 SLO 範例，以及在特定情況下如何創建更精密複雜的 SLO[1]。

為何 SRE 工程師需要 SLO

即使在大公司中，工程師都是稀有資源，所以做工程的時間應該投注在最重要服務中的關鍵屬性上。這之間的權衡取捨是很困難的：該投資在功能開發上吸引新客戶或留住現有客戶；抑或投資在可靠性與可擴充性上以取悅現有客戶。在 Google，我們學到了

1 術語上的注意事項：本章自始至終，我們用**可靠性**一詞來討論，以服務全部的 SLI 而言該服務績效如何，而這也指涉了其他相關術語：如可用性及延遲。

教訓：深思熟慮過後採用的 SLO，對於我們做關於可靠性工作的機會成本之資料考量（data-informed）決策、以及如何妥善排定工作的優先順序皆至為關鍵。

SRE 工程師的核心職責不僅僅在自動化「所有事情」，然後抓住傳呼器不放。他們每天的任務以及專案都是被 SLO 驅動的：短期目標為確實守衛 SLO，而中長程目標為維護這些 SLO，甚至可以斷言：沒有 SLO，就不需要 SRE 工程師。

SLO 是幫助決定何種工程工作之優先順序需要被排定的工具。例如，來衡量看看以下兩個可靠性專案工程上的取捨：自動化回滾機制，以及移至某複寫資料存放區。藉由計算對犯錯預算的預估衝擊，我們可以決定哪個專案對於我們的使用者最有利，請參見第 35 頁的「運用 SLO 及犯錯預算做決策」一節以了解更多相關細節，還有《網站可靠性工程》書中的〈管理風險〉章節（*http://bit.ly/2xzGm83*）。

捲起袖子

建立基礎 SLO 伊始，讓我們假設您的服務是某種程式，已被編譯過，被發佈過，並在網絡基礎設施上運行，俾使用者上網存取使用，您系統的成熟度層級可能是下列之一：

- 綠地開發：目前沒有任何部署。
- 生產環境上的系統，有些監控機制可以告知意外出錯，但沒有制定正式目標，沒有犯錯預算的概念，還有心照不宣的目標：系統應永遠正常運行。
- 有著低於 100% SLO 的運行中部署，對 SLO 的重要性沒有共識，也沒有共識如何利用之以資持續改進，換句話說就是無牙[*]的 SLO。

為了採行以犯錯預算為基礎的網站可靠性工程方法，您需要達到誠如下述的狀態：

- 組織中所有的利害關係人，都認可是符合產品方向的 SLO 存在著。
- 負責確保服務滿足 SLO 的同仁們，同意這個 SLO 在正常狀況下是可以達成的。
- 組織已經承諾以犯錯預算作為決策及排定優先順序的基礎，且這承諾是以犯錯預算政策形式正式頒行。
- 完善組織 SLO 的流程已然就緒。

否則，您將無法採行以犯錯預算為基礎的可靠性工程方法，SLO 遵從度不過是另一個 KPI（關鍵績效指標）或是流於形式彙報指標，而非決策工具。

[*] 無約束力、無強制性或罰則。

可靠性目標與犯錯預算

制定適當 SLO 的第一步是討論 SLO 應該為何以及其涵蓋的範圍。

SLO 為客戶制定服務應有的可靠性目標層級，在此門檻以上，幾乎所有的客戶都應該滿意您的服務（即假設他們滿意這服務之實用性）[2]。低於這個門檻，使用者有可能開始抱怨或停用這個服務。終究最重要的是使用者滿意——滿意的使用者使用服務，為您的組織產生營收，對客戶服務團隊的要求也相對低，並且會向他們的朋友們推薦這個服務。我們履行我們的服務可靠性，以使我們的客戶滿意。

客戶滿意度是個相當模糊的概念；我們無法準確地測量之，通常我們根本對其缺乏能見度，所以我們要如何開始呢？我們要以何作為我們的第一個 SLO ？

我們的經驗在在顯明，百分百可靠度是錯誤的目標：

- 如果您的 SLO 是對準客戶滿意度，百分百不是合理的目標，即使有了備援組件、有自動化的系統運行狀況檢測、快速的失效備援（failover），仍然有那麼一丁點的機率，一個或多個組件可能同時失靈，導致低於百分百的可用性。
- 即使您的系統能達成百分百的可靠性，您的客戶也無法體驗百分百的可靠性，在您與客戶間，有一長串複雜的系統鏈，而任一個組件都有可能失靈[3]。這也意味著當您從 99% 可靠性進境到 99.9%，甚至到 99.99%，只要每多個 9，就得付出多更多成本，但是對您客戶的邊際效益卻穩定地趨近於零。
- 如果您奮力想營造有百分百可靠度的使用者體驗，並想要維持那種水準的可靠性，您將永遠無法更新或改進您的服務，系統故障停機（outage）的頭號戰犯就是變更：推出新功能、套用安全性補釘、部署新硬體以及擴充容量以滿足客戶的需求等等，在在都會衝擊這個百分百可靠度的目標。遲早您的服務會停滯不前，您的客戶也會琵琶別抱，這會挑戰任何人的底限。
- 100% 的 SLO 意味著您只有時間被動回應，您名副其實地束手無策，除了疲於奔命回應小於 100% 的可用性[*]——這是保證會發生的常態。100% 可靠性不是符合工程文化的 SLO，它其實是運維團隊的 SLO。

---

2 這與**服務層級協議**（SLA）有顯著的不同，SLA 是業務上的合約，有法律效力，當您的使用者非常不滿意的時候，您必須以某種形式補償他們。

3 更多關於把系統依存性因素考慮進您服務的可靠度，請參見 Ben Treynor, Mike Dahlin, Vivek Rau, and Betsy Beyer, "The Calculus of Service Availability," *ACM Queue* 15, no. 2 (2017), *https://queue.acm.org/detail.cfm?id=3096459*。

[*] 一般系統絕大部分時間都不是 100% 可用。

一旦您有低於 100% 的 SLO 目標，那麼組織裡須有人被授權擔負起這個 SLO，在開發速率及可靠性之間權衡取捨。在小一點的組織裡，這樣的角色可能是 CTO 在扮演；而在大一點的組織裡，通常是由產品負責人（或產品經理）擔綱。

測量什麼：使用 SLI

一旦您同意 100% 是個錯誤的度量（number），那您如何決定正確的度量呢？不管怎麼說，您究竟在測量什麼呢？在此服務水準指標就派上用場了：SLI 是一種**指標**，鑑別您所提供之服務水準。

雖然很多度量都可以當作 SLI，我們一般建議把這兩個度量的比例當作 SLI：良好事件的數目除以事件總數，比如：

- 成功的 HTTP 請求數 / 全部的 HTTP 請求數（成功率）
- Google 遠端程式呼叫框架（gRPC）在 100 毫秒內成功完成的呼叫數 / 全部 gPRC 請求數
- 使用了整個語料庫（corpus）的搜尋結果數 / 包含那些被優雅降級（degraded gracefully）過了的總搜尋結果數
- 產品搜尋中運用了晚近十分鐘內的新庫存資料的「庫存檢查計數」請求數 / 總庫存檢查請求數
- 根據某指標的延伸標準清單而定義的「良好使用者體驗分鐘數」/ 總使用者體驗分鐘數

這種形式的 SLI 有很多特別有用的特性，SLI 的範圍從 0% 到 100%，0% 意指沒東西能運作，而 100% 則意謂沒東西壞掉。我們發現這樣的尺度很直覺，其風格也容易適用在犯錯預算上：SLO 是目標百分比，而犯錯預算則是 100% 減去 SLO。舉例來說，如果您的成功率 SLO 是 99.9%，那麼一個在四週期間內接受三百萬個請求的服務則有 3,000 個（0.1%）的預算空間可以出錯，如果一次故障造成了 1,500 次錯誤，那這些錯誤就花費了 50% 的犯錯預算 [4]。

此外，讓您所有的 SLI 遵循一致的風格也允許您更妥善利用工具應用：您可以寫就告警邏輯、SLO 分析工具、犯錯預算計量以及報表等等，以期待同樣的輸入資料：分子、分母及門檻，簡單化是這種風格的加分項目。

4 如果在某日曆期間測量 SLO，比如說一季，而且是基於不可預測的指標如流量的話，那麼可能要到季末才會知道這預算有多大。參見第 27 頁〈選擇適當的時窗〉一節可窺更多有關匯報期間的討論。

- 定期從關聯式資料庫讀取資料，並寫入分散式雜湊（hash）資料表以優化伺服的系統
- 一個轉換影片格式的影片處理服務
- 一個從多種來源讀取記錄檔以產生報表的系統
- 一個從遠端伺服器擷取指標並產生時間序列及警報的監測系統

存貯（storage）

一個接受資料（例如位元組、紀錄、檔案、影片）並將其儲存俾以後能擷取之的系統。

一個有效的範例

試想一個手機遊戲的簡化架構，如圖 2-1 所示。

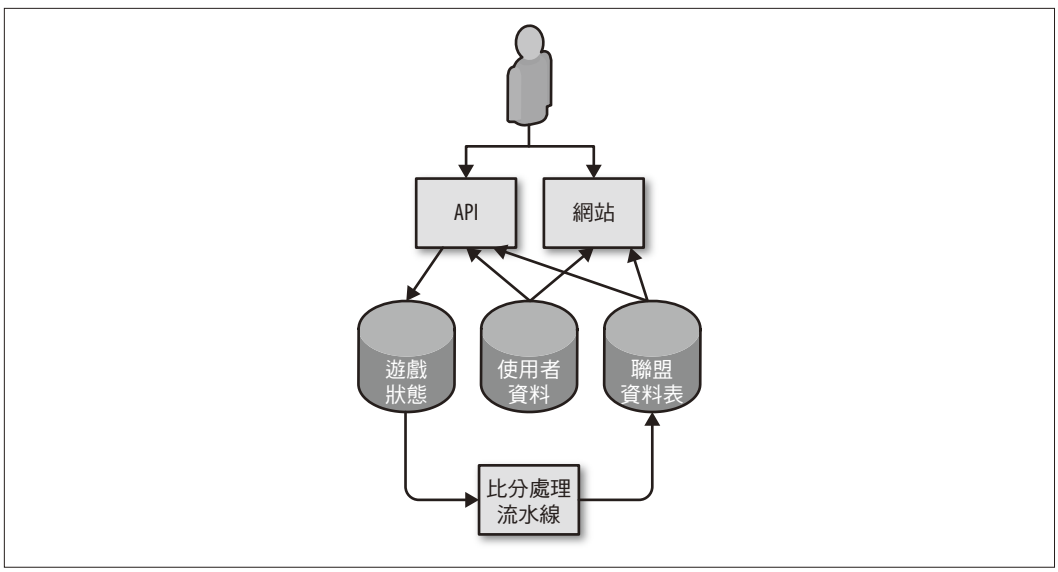

圖 2-1　範例手機遊戲的架構

在使用者手機上運行的這個遊戲應用程式，跟一個在雲端上運行的 HTTP API 互動，這個 API 把遊戲狀態的變化寫到一個永久的存貯系統。一個流水線定期處理這些資料，產生聯盟資料表，該表可以提供本日高分、本週、或是歷史最高分。這些資料會被記錄到不同的聯盟資料表資料存儲，結果將會在行動應用程式上（遊戲中的分數紀錄）以及網站上顯示。使用者可以上傳自製頭像（avatar）到使用者資料表，可透過 API 供遊戲中以及高分排行榜網頁上使用。

有了這樣的設置，我們可以開始思考使用者如何與系統互動，以及哪種 SLI 可以測量使用者經驗的多種面向。

有些 SLI 可能會重疊：請求驅動的服務可能會有個正確性 SLI，處理流水線有個可用性 SLI，而耐久性 SLI 可以被視為正確性 SLI 的變體。我們建議選用少數（五種或更少）有代表性的 SLI 來體現您系統提供給客戶最關鍵的功能。

為了捕捉典型使用者體驗以及長尾效應，我們也建議為某些型態的 SLI 採用多種等級的 SLO。例如，若 90% 的使用者請求在 100 毫秒內返回[*]，但剩下的 10% 卻要 10 秒才返回，那很多使用者都會不開心。延遲性 SLO 可以不同的門檻取得使用者群資訊：如 90% 的請求快過 100 毫秒，或 99% 的請求快過 400 毫秒。如此原則皆適用於有測量使用者不滿意度之參數的 SLI。

表 2-1 提供了一些不同型態服務所常見的 SLI。

表 2-1　不同型態組件的潛在 SLI

服務型態	SLI 型態	描述
請求驅動	可用性	有成功回應之請求所佔的比例。
請求驅動	延遲性	回應速度高於某個門檻之請求所佔的比例。
請求驅動	品質	當服務過載或後端無法使用而導致服務優雅降級，您需要測量非優雅降級所伺服之回應所佔的比例。例如，如果使用者資料存儲無法使用，使用者仍然可以玩遊戲，只是用通用圖像顯示。
處理流水線	新鮮度	在某時間門檻之內資料有晚近被更新過的比例，理想中這種指標數算使用者存取該資料的次數，所以可以更精確反映使用者經驗。
處理流水線	正確性	匯入處理流水線的記錄中，能產出正確輸出者所佔比例。
處理流水線	覆蓋	當批次處理時，處理超過某標的量資料的任務所佔比例。當串流處理時，在某限定時窗（time window）內成功處理完的匯入記錄所佔比例。

[*] return，有回應。

圖 3-1　VALET 儀表板

SLO 湧現

一旦 SLO 深植組織集體意識，有效自動化及報表服務就緒後，新的 SLO 就會迅速激增。在年初追蹤約 50 個服務的 SLO 後，到年底前我們就追蹤多至 800 個服務的 SLO，差不多每個月約有 50 個新服務註冊到 VALET。

因為 VALET 讓我們在家得寶上下擴大 SLO 採用，需要來開發自動化的時間與努力花的非常值得。然而，其他公司不應該因為他們不能開發類似複雜的自動化，而懼於採用以 SLO 為基礎的方法。雖然自動化為家得寶帶來額外的好處，在改革伊始就算只把 SLO 寫下來，也很有益處。

在批次應用程式上運用 VALET

在圍繞著 SLO 開發耐用的報表系統的同時，我們發現一些 VALET 的其他用途，只需要些微調整，批次應用程式也可如下適用此框架：

第六章

消滅苦工

David Challoner、*Joanna Wijntjes*、*David Huska*、
Matthew Sartwell、*Chris Coykendall*、*Chris Schrier*、
John Looney、*Vivek Rau* 與 *Betsy Beyer*、
Max Luebbe、*Alex Perry* 與 *Murali Suriar* 合撰

Google 的 SRE 工程師花很多時間在優化上——透過專案工作加上和開發人員協作，盡力把系統每一點一滴的效能榨出來，但優化的範圍不侷限在運算資源上：SRE 工程師優化他們自己如何好好利用時間也同樣重要。基本上，我們想要避免執行可歸類為**苦工**的任務，更詳盡的討論，參見《**網站可靠性工程**》一書的第 5 章（*http://bit.ly/2Lg1TEN*）。就本章而言，我們將苦工定義為與維護服務有關的那些重複、可預測和源源不絕之任務。

對任何管理生產環境服務的團隊來說，苦工看似無法避免。系統維護不可避免地需要若干上線、升級、重啟、警報鑑別等等工作。若任其蔓延不查察也不交代清楚的話，這些活動會很快把團隊精力消耗殆盡。Google 限制 SRE 團隊只能花 50% 的時間（脈絡請詳我們第一本書的第 5 章（*http://bit.ly/2Lg1TEN*））在運維工作上（包括苦工與非苦工繁重的工作）。這個目標或許在您的組織並不那麼恰當，但設置一個苦工上限還是有好處，因為能夠識別和量化苦工，是優化團隊時間運用的第一步。

什麼是苦工？

苦工不一而足，可以下列特徵估量，這些特徵在我們的第一本書已經描述過了。這裡針對每個苦工特徵提供具體範例：

- 工程專案工作逐漸增長，有些工作會更進一步減少苦工
- 團隊士氣增進，團隊離職率和過勞程度降低
- 由於中斷 * 導致的上下文切換（context swtiching）更少了，提高團隊生產力
- 流程更清晰易懂與標準化之促進
- 團隊成員技術能力和職涯成長之提升
- 訓練時間減少
- 歸因於人為失誤的故障停機事件更少
- 資安改善
- 使用者請求的回應時間更短

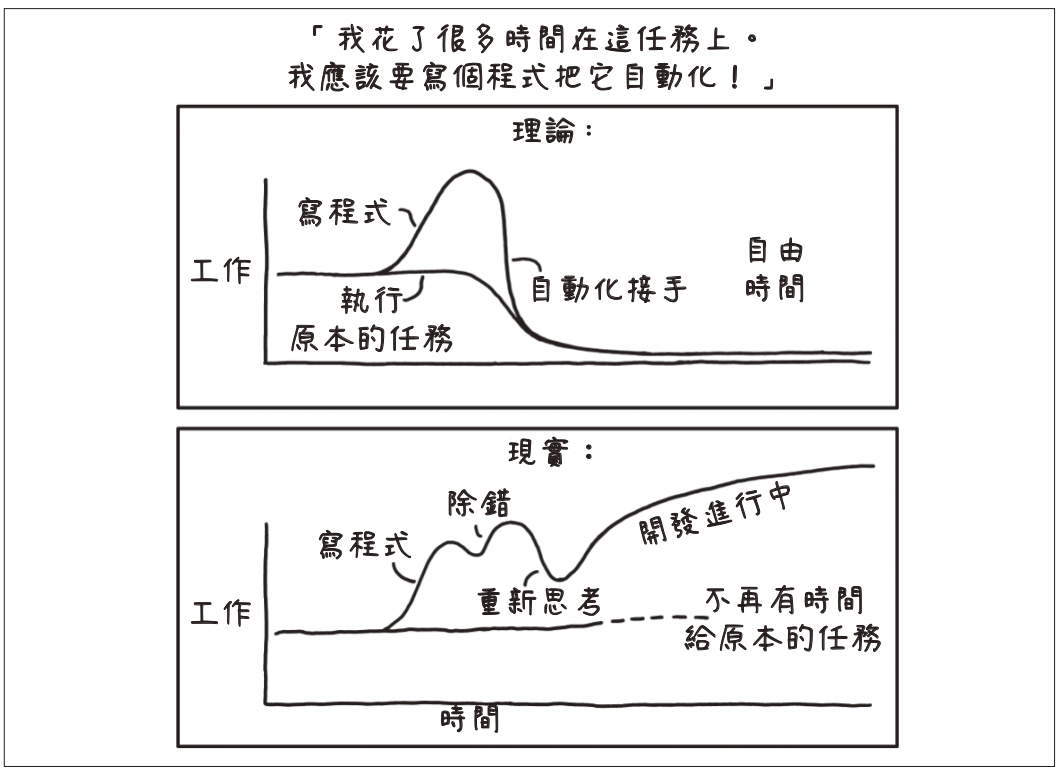

圖 6-1　估算您在減少苦工的工事上會花多少時間，並確保可以回本（來源：*http://xkcd.com/1319/*）

* 比如被其他高優先的任務打擾導致目前手上工作被迫中斷，如高優先請求或故障停機事故。

案例研究 3：展示廣告蜘蛛網的簡化

背景

Google 的展示廣告業務有很多相關產品，包含一些收購來的（DoubleClick、AdMob、Invite Media 等等）。這些產品須經相當改動，方能與 Google 的基礎設施及既有產品協同運作。舉例來說，我們想讓某使用 DFP（DoubleClick for Publishers）廣告管理系統的網站能展示 Google AdSense 所選定的廣告；同樣地，我們想要讓使用 DBM（DoubleClick Bid Manager）廣告需求端服務平台的競標者能參與在 Google 的 Ad Exchange 廣告系統上進行的即時拍賣。

這些各自獨立開發的產品，構築成彼此相連的後端系統，錯綜複雜故難以理解。觀察當流量通過這些元件時發生什麼事是困難的，並且為每個元件供應正確的容量是不方便也不精確的。我們一度添加測試，以確保我們移除了查詢流程中所有無限迴圈。

我們決定做什麼

廣告伺服 SRE 工程師是推動標準化的當然人選：雖說每個元件都有特定的開發團隊，但 SRE 工程師為整個技術堆棧 on-call（待命值班）。我們的首要任務之一就是草擬一致標準，並與開發團隊合作以逐步採用這些標準。這些標準是：

- 建立複製大型資料集的唯一方式
- 建立執行外部資料查找的唯一方式
- 為監控、服務開通及組態提供常用範本

在此項倡議之前，每個產品的前端及拍賣功能是由各自的程式提供。如圖 7-1 所示，當一個廣告請求可能打到兩個目標系統時（AdMob 及 AdSense），我們重寫該請求以迎合第二個系統之預期。這需要額外程式碼及處理，而且也為不理想迴圈開了方便之門。

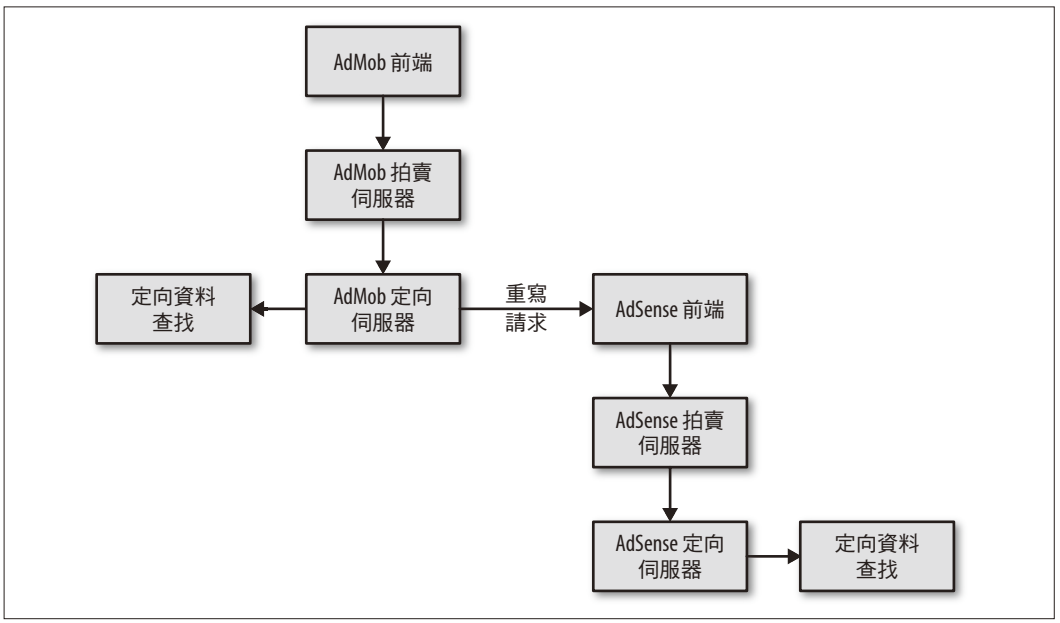

圖 7-1　先前，一個廣告請求可能打到兩個系統：AdMob 及 AdSense

為了簡化系統，我們於常用程式中添加一些邏輯以滿足所有的使用案例，也添加一些旗標以保護這些程式。我們逐漸將這些旗標移除，並將功能整併進較少的伺服器後端。

伺服器一旦整合，拍賣伺服器就可以與兩個定向伺服器直接溝通。如圖 7-2 所示，當多個定向伺服器需要查找資料的時候，只需要在整合的拍賣伺服器查找一次即可。

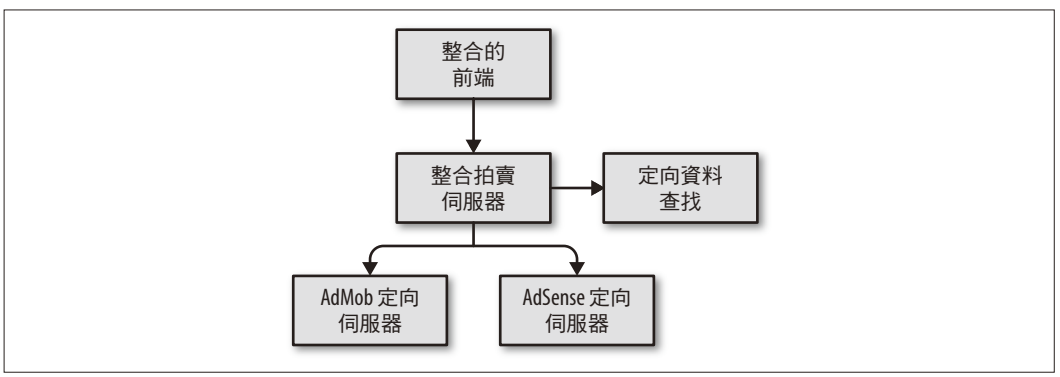

圖 7-2　整合的拍賣伺服器現在只需要執行一次資料查找

習得教訓

若某系統已經運行多時，最好將之漸進式整合進您自己的基礎設施。

正如同單一程式中，某些非常類似函式的存在代表某種「程式碼異味」，顯示更深層的設計問題；單一請求中重複的查找也代表某種「系統異味」。

當您建立定義明確的標準，並為 SRE 及開發人員所贊同，您就可以提供清晰的藍圖以去除複雜度，而主管階層也比較傾向支持並獎勵這樣的藍圖。

案例研究 4：在共享平台上運行數以百計的微服務

Mike Curtis 撰

背景

過去 15 年來，Google 開發出多個成功的產品（搜尋、廣告及 Gmail，僅舉幾例），並源源不絕產出嶄新並重構好的系統。許多如此的系統都有其專職的 SRE 團隊及對應領域特定的生產堆棧，包含量身訂做的開發工作流程、持續整合與持續交付（CI/CD）的軟體週期及監控。而這些獨一無二的生產堆棧，會在幾個方面引致可觀的間接成本：維護、開發成本以及各自獨立的 SRE 積極參與。以上這些也會讓團隊間服務（或人員！）調度或增加新服務難上加難。

我們決定做什麼

一群社群網路產品區塊的 SRE 團隊，努力想把他們服務的生產堆棧匯合成單一管控微服務平台，由區區一組 SRE 工程師管理。這個共享的平台遵循最佳實踐，並搭配且自動設定許多先前未充分利用的功能，可以改善可靠性並讓除錯更便捷。無論他們 SRE 積極參與度如何，該 SRE 團隊守備範圍內的新服務，都需要使用這個共用的平台，並且舊有服務也要遷移到新平台，不然就要被逐步淘汰。

在社群網路產品區塊獲致成功後，這個共享平台也漸漸獲得 Google 上下其他 SRE 或非 SRE 團隊的採用。