第五章

並行處理 AI 運算作業

本章目標

在本章中，你將學到：

- 多執行緒、多程序與非同步程式設計在提升 AI 應用效能與擴展能力時發揮的作用與優勢
- FastAPI 伺服器裡的執行緒池與事件迴圈的角色
- 如何在處理請求時避免伺服器被阻塞
- 使用非同步程式設計來與資料庫、AI 模型及 web 內容等外部系統互動，並建立網頁爬蟲與 RAG 模組
- 模型部署策略與 LLM 記憶體優化策略，它們將減少耗用大量記憶體的阻塞操作
- 處理長時間執行的 AI 推論任務的策略

在本章中，你將深入瞭解非同步程式設計在提升 GenAI 服務效能與擴展能力時扮演的角色與帶來的好處，並學習如何管理並行的使用者互動，以及與資料庫等外部系統對接、實作 RAG，和讀取網頁內容以豐富提示詞脈絡。你也會學習有效處理 I/O 密集型與 CPU 密集型操作的技巧，尤其是在處理外部服務或長時間執行的推論任務時。

我們也會深入探討如何有效處理長時間執行的生成式 AI 推論任務，包括使用 FastAPI 的事件迴圈來執行背景任務的策略。

以下是圖 5-2 所示的三種 Python 執行模式：

無並行（同步）

單一程序（在單核心上）依序執行任務。

無平行並行

單一程序之中（在單核心上）有多個執行緒並行處理任務，但由於 Python 的 GIL 機制，它們無法平行執行。

平行化並行

多個程序在多核心上平行執行任務，充分利用多核心處理器以達到最高效率。

在多程序運作中，每一個程序都能存取自己的記憶體空間與資源，並與其他程序隔離，獨自完成任務。這種隔離機制能提升程序的穩定性，因為一個程序的崩潰不會影響其他程序，但也會讓程序之間的通訊比共用記憶體空間的執行緒更複雜，如圖 5-3 所示。

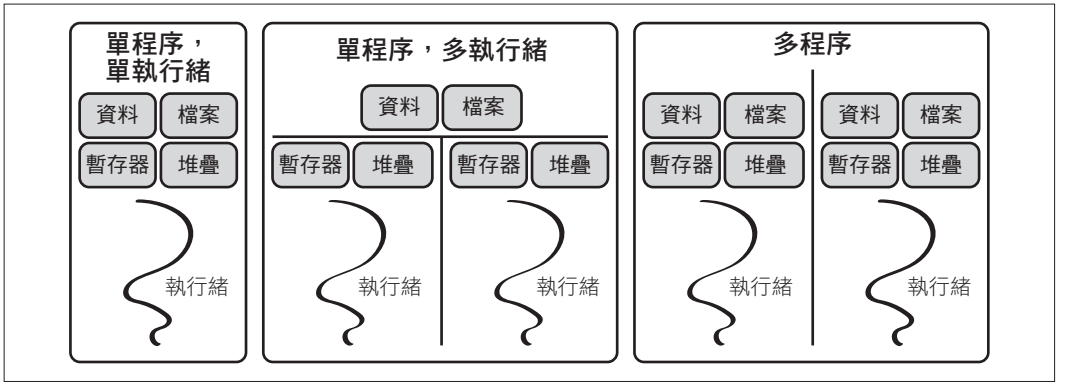

圖 5-3　多執行緒與多程序的資源共享情形

當系統將運算作業分散開來時，通常會用一個管理程序來協調各程序的執行與合作，以避免資料毀損與重複工作。多程序的典型例子是使用負載平衡器來分配流量給多個容器，讓每一個容器皆執行你的應用程式實例。

圖 5-8 是完整的 pipeline。

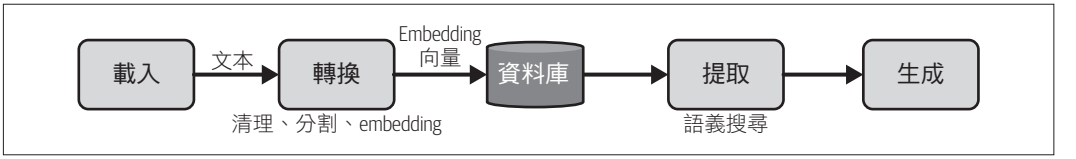

圖 5-8　RAG pipeline

你可以將圖 5-8 所示的 pipeline 整合到現有服務中。圖 5-9 是使用 RAG 的「與文件對話」服務系統架構。

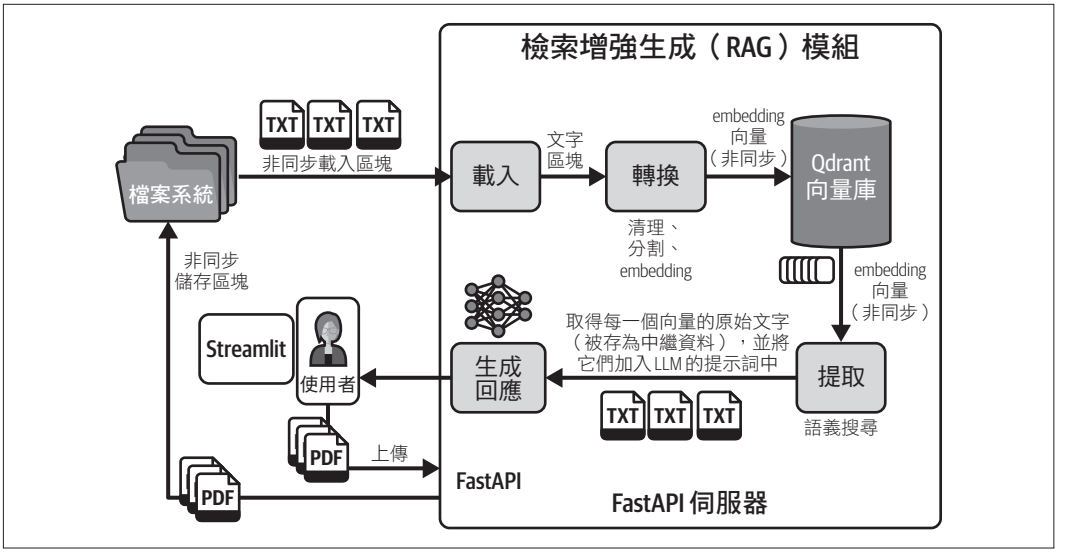

圖 5-9　「與文件對話」系統的架構

在圖 5-9 所示的系統中，我們儲存使用者透過 Streamlit 介面上傳的文件，接著將它們取出、處理、存入資料庫，然後在需要增強 LLM 提示詞時，提取並使用它們。

在實作圖 5-9 所示的 RAG 系統之前，我們的第一步是為 Streamlit 用戶端與後端 API 加入檔案上傳功能。

你可以使用 FastAPI 的 `UploadFile` 類別來分塊接收使用者上傳的文件，並將它們存入檔案系統或其他儲存方案（例如 blob storage）。注意，這個 I/O 操作是非阻塞的，因為 FastAPI 的 `UploadFile` 類別支援非同步程式設計。

將資料庫整合至 AI 服務

本章目標

在本章中，你將學到：

- 何時需要使用資料庫？如何為你的專案選擇合適的資料庫類型？
- 關聯式資料庫的運作機制，以及非關聯式資料庫的使用情境
- 使用關聯式資料庫的開發流程、工具，與最佳策略
- 在使用資料庫時，提升查詢效能與效率的技巧
- 如何管理持續變動的資料庫 schema
- 在與團隊協作時，管理 codebase、資料庫 schema、資料漂移的策略

在本章中，我們要把資料庫整合到現有的 API 服務中，用來它來儲存與提取使用者互動資料。

本章假設你已經具備基本的資料庫與 Structured Query Language（SQL）使用經驗，因此不會探討 SQL 程式設計與資料庫工作流程的細節。你將學習更高階的資料庫概念、開發流程，以及如何將資料庫整合至 FastAPI 應用程式，並讓資料庫與 GenAI 模型（例如 LLM）互動。

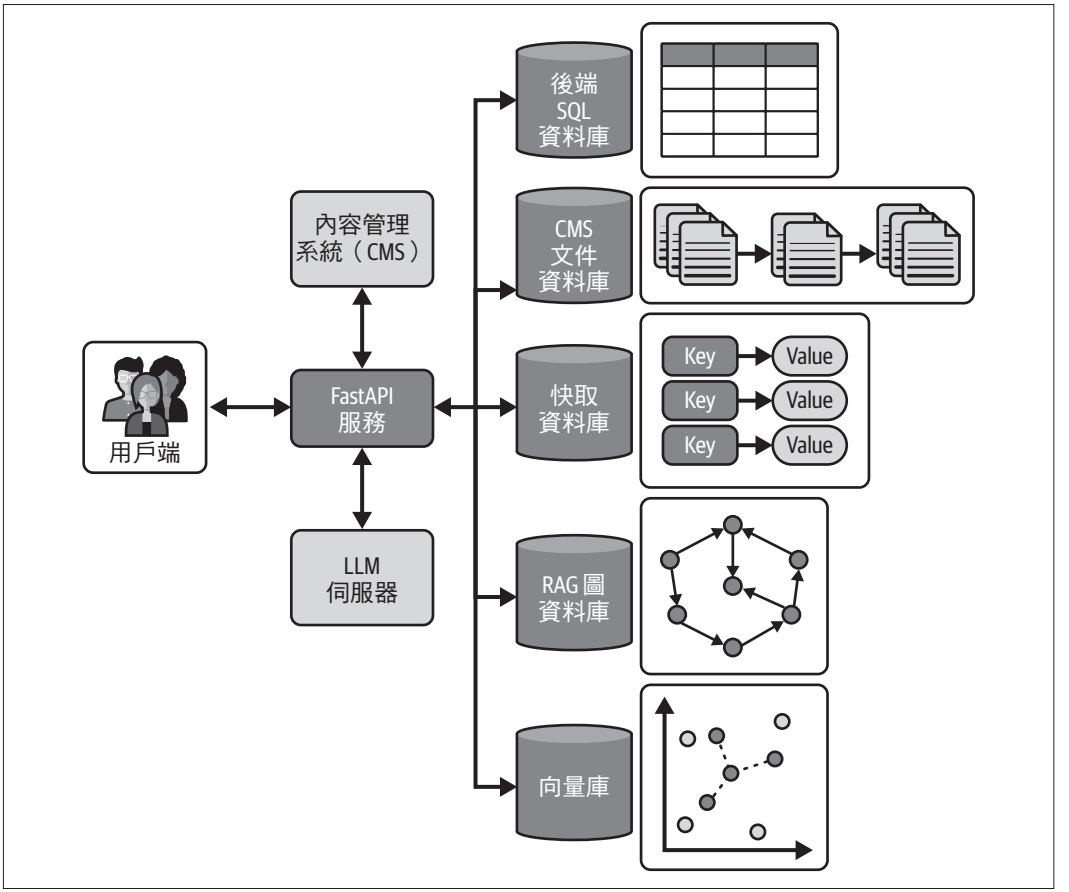

圖 7-3　一起使用各種類型的資料庫

瞭解 GenAI 服務如何與多種資料庫整合之後，接下來將專門討論如何為服務加入關聯式資料庫。

專案：在關聯式資料庫中儲存使用者與 LLM 的對話

在上一節，我們介紹了與應用程式資料持久化有關的核心資料庫概念。

在這一節，我們要將關聯式資料庫整合到 GenAI 服務中，以便將使用者與 LLM 的對話紀錄存入資料庫。作為這項專案的延伸，你還會學習管理 schema 變更與資料遷移的最佳策略、工具，以及開發流程。

第九章

保護 AI 服務

本章目標

在本章中，你將學到：

- 如何防範 GenAI 服務被惡意攻擊、誤用，與濫用
- 如何自行實作倫理與安全護欄、評估流程，與內容過濾層，用來管理使用行為
- 如何實作使用管理、串流限流與各種 API 速率限制策略，以降低伺服器負載

你已經在前幾章學習如何建立可支援多種 AI 生成器的 GenAI 服務，並知道如何支援並行和即時資料串流了。此外，你還整合了外部系統（如資料庫），並實作了自訂的身分驗證與授權機制。最後，你寫了一套測試工具，用來驗證整個系統的功能與效能。

本章要教你實作「使用管理」與「濫用防護」機制，以確保 GenAI 服務的安全。

使用管理與濫用防護

在部署 GenAI 服務時，你必須考慮惡意使用者可能如何誤用或濫用你的服務。這對保護使用者安全與維護自身聲譽來說至關重要。我們無法預測使用者會如何使用系統，因此必須預設最壞情況，並設計護欄（*guardrails*）來防止任何形式的誤用或濫用。

根據近期一項關於 GenAI 惡意使用的研究（*https://oreil.ly/ihmzR*），你的服務可能被用於表 9-1 所述的惡意目的。

護欄

護欄是一種主動進行偵測的控制措施，旨在引導應用程式產生預期的結果，它們具有極高的多樣性，可依照 GenAI 系統中可能出錯的各種情況來設定。

舉例來說，*I/O* 護欄用來驗證進入 GenAI 模型的輸入資料與輸出至下游系統或使用者的內容。這類護欄可標記不當的使用者查詢句，並檢測輸出內容是否包含有害語言、幻覺，或禁止討論的主題。圖 9-1 是加入 I/O 護欄之後的 LLM 系統範例。

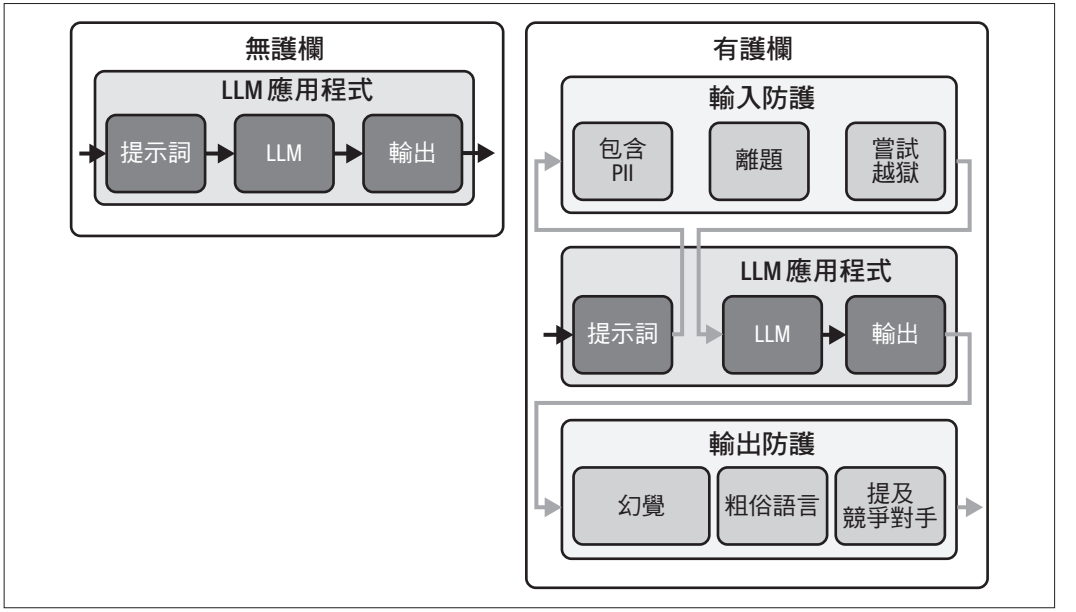

圖 9-1　比較無護欄與有護欄的 LLM 系統

目前已經有開源的護欄可以保護你的服務了，例如 NVIDIA NeMo Guardrails、LLM-Guard 與 Guardrails AI，所以你不需要從零開始實作護欄。不過，在使用這些框架之前，你可能要學習特定的框架語言，而且外部依賴套件可能造成服務速度下降與應用程式膨脹。

市面上的其他商用護欄（如 OpenAI 的 Moderation API、Microsoft Azure AI Content Safety API 與 Google 的 Guardrails API）有的不是開源的，有的缺乏測量品質所需的細節與內容。

實作速率限制的方法是在特定時間監視輸入請求，並使用佇列來平衡負載。

你有許多速率限制策略可以選擇，表 9-5 比較這些策略，圖 9-2 則展示了其運作方式。

表 9-5　速率限制策略

策略	優點	限制	使用情境
代幣桶（Token Bucket） 系統以固定速率填充代幣桶，每一個請求都會消耗一個代幣。若代幣不足，系統會拒絕新的請求。	• 能處理暫時的流量高峰與動態的流量模式 • 可以更仔細地控制請求	做法比較複雜	常見於多數 API 與服務，以及請求速率不規則的互動式或事件驅動型 GenAI 系統
漏桶（Leaky Bucket） 將輸入請求加入佇列，並以固定速率處理它們，讓流量保持穩定。若佇列滿出來，系統會拒絕新請求。	• 實作簡單 • 維持穩定的流量輸出	• 較無法靈活地處理動態流量 • 在流量暴增時，可能拒絕合法的請求	適合需要維持穩定回應時間的 AI 推論服務
固定時間窗口（Fixed Window） 限制出現在一個固定時間窗口之內的請求數量（例如每分鐘 100 次請求）。	容易實作	不擅長處理暴增的流量	• 適用於昂貴的 AI 推論與 API 呼叫，強制執行嚴格的使用政策 • 適合免費使用者，或使用模式可預測的批次處理系統 • 每一個請求都被平等對待
滑動時間窗口（Sliding Window） 在不斷滑動的一段時間內統計請求數量。	更有彈性、更仔細地管理，並且更好的突發流量平滑能力	• 程式比較複雜 • 需要使用較多記憶體來追蹤請求	• 較擅長處理突發流量 • 適合對話式 AI，或需要高頻率、彈性存取的高階用戶