Chapter 11

11.1 導論

11.2 前後端分離技術架構

11.3 PIM 轉 DB

11.4 PIM 轉 AP

11.5 PIM 轉 UI

11.6 應用 ChatGPT 轉換之認知與方法

11.7 結論

附錄：應用 Enterprise Architect 進行 MDA 轉換之實作

附錄 11A：網路購書系統之實體類別圖轉 MS SQL Server 2005 資料表

附錄 11B：網路購書系統之類別圖轉應用程式

本章習題

參考文獻

生成式 AI x MDA 之轉換方法論

本章學習重點

詳讀本章，你至少能瞭解：

- MDA 中 PIM 轉 DB/AP/UI 的轉換概念與方法論。
- 實體類別與關係如何轉換為關聯式資料庫。
- 循序圖如何轉換為應用程式架構。
- 狀態機圖、介面藍圖與詞彙如何轉換為使用者介面及程式。
- 生成式 AI 在上述轉換程序中可扮演的角色及其應用。

11.1 導　論

雖然現代的物件導向程式語言與程式設計技術，已提供許多功能與方法以提升系統開發效率，但如第 1 章所述，目前系統開發仍面臨一些亟待克服的問題，例如生產率、可攜性、互通性、維護與文件等。針對程式階層文件維護問題，其中一種常見作法是直接由原始程式碼自動產生文件，以確保文件內容能隨程式更新而同步維護。然而，此方法僅能改善低階程式層級的文件一致性問題，對於高階的系統分析與設計文件，仍高度仰賴人工撰寫與維護，整體效益因而有限。

模型驅動架構(Model Driven Architecture, MDA)結合生成式 AI（例如 ChatGPT、Gemini 或 Cursor 等）被認為是解決或緩解上述問題的有效途徑之一。基於此，本書已於第 2 章中說明 MDA 的發展背景、核心概念與其生命週期；而本章則進一步聚焦於 MDA 的轉換流程及其背後的轉換方法論，並探討如何在此基礎上，結合生成式 AI 工具，將平台獨立模型(Platform Independent Model, PIM)中的類別圖、循序圖、狀態機圖、介面藍圖與介面詞彙等，系統性地轉換為關聯式資料庫結構、應用程式設計，以及使用者介面程式碼。

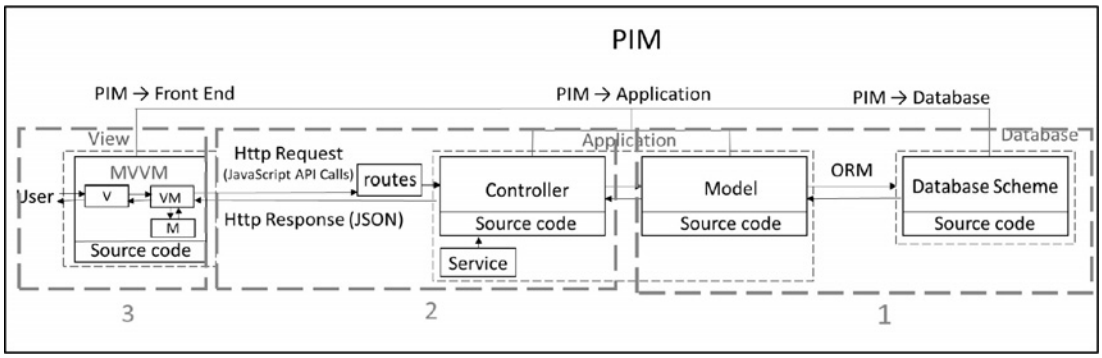
事實上，欲使 MDA 之轉換流程達到自動化或半自動化，其關鍵前提在於建立一套明確、可重複執行且具一致性的轉換方法論，作為模型轉換與程式碼生成的理論與實務依據。因此，本章將依序說明轉換流程所依據的技術基礎與框架假設，並系統性地闡述 PIM 轉換為資料庫(DB)、應用程式(AP)與使用者介面(UI)之轉換規則與設計原則；同時，透過實際工具之實作說明，進一步說明生成式 AI 在上述轉換過程中所扮演的角色及其具體應用方式。

11.2 前後端分離技術架構

PIM 雖為平台獨立模型，其設計目的在於避免過早綁定特定技術實作；然而，在實務上，PIM 轉換至 PSM 乃至最終程式碼，仍不可避免地受到所採用之平台技術、應用框架與程式語言的深刻影響。因此，在說明 PIM 轉換為最終程式碼之具體方法論與轉換規則之前，本節將先界定本書案例所假設之前後端分離技術架構，並概述其中所採用的主要技術、開發框架、程式語言與資料庫等，作為後續轉換設計與實作結果的基礎。需特別指出的是，後續章節所介紹的轉換方法與產出，皆建立在上述技術與框架之假設上；若實際應用情境採用不同的技術組合，相關轉換規則與實作細節亦須相應調整。

在本書案例中，便當王網路訂購系統採用前後端分離技術架構進行開發：前端使用者介面採用 MVVM 架構之 React(JSX、CSS)框架，後端應用程式使用 PHP Laravel 框架，資料庫則採用 MySQL。整體而言，本案例之 MDA 三個核心模型轉換流程，皆透過生成式 AI 來輔助完成 PIM 轉 Code 的工作，涵蓋資料庫(DB)、應用程式(AP)之訊息傳遞與業務邏輯，以及前端使用者介面(UI)等程式碼產出。整體轉換流程可明確劃分為三個階段：轉 DB、AP 與 UI，如圖 11-1 所示。

Figure
圖 11-1 PIM 轉換至資料庫、後端與前端之程式碼架構



11.2.1 Laravel：後端應用程式框架

Laravel 為一套以 PHP 為基礎的 Web 應用程式框架，其核心設計理念在於提供清晰的系統架構與標準化的開發流程，以降低後端系統在開發與維護上的複雜度。本書案例中，Laravel 主要用於後端系統的實作，涵蓋資料庫存取層(Database Layer)、應用程式邏輯層(Application Layer)與 API 服務層(Service Layer)。

在技術特性方面，Laravel 提供完善的資料庫 Migration 機制，使開發者得以透過程式碼管理資料庫結構（如資料表、欄位、主鍵與外鍵等），並提供可版本控制的資料庫綱要(Database Schema)。其內建之 Eloquent ORM 透過 Model 定義資料表與程式物件間的對應關係，使資料存取得以採用物件導向方式進行，有助於強化程式可讀性與維護性。此外，Laravel 採用 MVC 架構，以 Route、Controller 與 Service 的分層設計，明確區分請求處理、流程控制與業務邏輯，有助於提升系統結構的清晰度與擴充彈性。其架構亦與 RESTful API 設計規範高度契合，特別適合作為前後端分離架構中的後端服務框架。在本書所示範之案例流程中，Laravel 作為 PIM 轉換為應用程式(AP)階段的主要實作框架，承載由模型所衍生之資料結構與業務邏輯，並支援後續系統整合與介面開發。

11.2.2 React：前端使用者介面框架

React 為一套由 Meta（前 Facebook）所開發的前端使用者介面框架，採用元件化(Component-Based)的設計理念，適用於建構高互動性的 Web 使用者介面。本書案例中，React 主要用於前端使用者介面的實作，負責呈現系統功能並處理使用者互動行為。

在技術特性方面，React 以元件化開發模式為核心，將使用者介面拆解為可獨立維護與重複使用的元件，有助於提升前端程式的可讀性與維護性。其狀態(State)管理與資料綁定機制，使使用者介面能隨資料狀態的變化即時更新，進而強化整體互動體驗。此外，React 可透過 HTTP 請求與後端 API 進行整合，前端藉由呼叫後端（如 Laravel）所提供的 RESTful API，完成資料庫的存取。React 亦高度支援前後端分離的系統架構，促進前端與後端在開發、測試與部署上的模組化分工，提升系統的擴充彈性。在本案例流程中，React 作為 PIM 轉換為使用者介面(UI)階段的主要實作框架。

11.2.3 PIM 轉換目標、資料來源與產出

整體而言，本案例以 Laravel 作為後端核心，負責 MVC 架構中的 Model 與 Controller 邏輯；前端則透過 React 以狀態驅動方式實現類 MVVM 的互動模式，從而完成前後端分離的系統開發架構。

首先，在 DB 轉換階段，ChatGPT 以「類別圖（特別是實體類別及其關聯關係）」結合「介面詞彙」作為主要輸入，產出 Laravel 所需之 Migration（資料表結構定義）與 Model（ORM 映射模型）。

接著進入 AP 轉換階段，ChatGPT 同時參考「循序圖（描述控制類別與實體類別之互動流程）」以及 轉 DB 階段所產出的 Model，並輔以操作情境描述，以理解整體業務流程。此階段的目標在於產出後端 API 規格與完整的運作邏輯，包含基本的 CRUD 操作，以及較為複雜的業務邏輯。其產出成果包括 OpenAPI 規格文件、Route（路由設定）、Controller（請求接收與回應）與 Service（業務邏輯實作）等。

最後為 UI 轉換階段。生成式 AI 依據「介面藍圖」、「介面詞彙」、「狀態機圖」以及「API 規格文件」，生成完整的前端使用者介面程式。介面藍圖用以決定畫面配置，介面詞彙確保元件命名與資料欄位的一致性，狀態機圖描述畫面切換與使用者互動流程，而 API 文件則支援資料綁定與 axios 呼叫。此階段的產出

包含 React(JSX)介面程式碼、CSS 樣式設定與 API 串接實作，使前端介面能正確呈現資料，並完成跨頁導覽、狀態管理與各項互動功能，符合 MVVM 架構之前端設計理念（如表 11-1 所示）。

Table
表 11-1 PIM 轉換目標資料來源與產出

階段	轉換目標	資料來源	產出（執行層）
轉 DB	轉出資料庫綱要之可執行的遷移檔(Migration)、Model	類別圖（主要是實體類別與關係）＋介面詞彙	Migration+Model（透過 ORM 映射到實際的資料表）
轉 AP	轉出後端 API 規格、控制器中資料表基本CRUD操作與可執行的業務邏輯(Service)	循序圖（主要是控制類別與實體類別）＋ Model	OpenAPI 規格文件 Route+Controller+Service
轉 UI	轉出可執行之介面以及介面間之跳轉	介面藍圖、介面詞彙＋狀態機圖＋ OpenAPI 規格文件	UI 框架(e.g., React)+CSS+ Axios

11.3 PIM 轉 DB

在資料庫層，PIM 階段所建立的實體類別及其關聯關係，必須被具體化為關聯式資料庫可管理的資料庫綱要(Database Schema)。這些綱要包含資料表、欄位名稱與長度、主鍵與外鍵設定，以及一對一、一對多與多對多等關聯。在本書所採用的實作環境中，此一轉換階段主要對應到 Laravel 的 Migration 與 Model(Eloquent ORM)兩個程式構件。

從 MDA 的觀點，PIM 轉換為資料庫 PSM 的工作可歸納為三個面向：

1. 資料型態對應

說明 PIM 中屬性型態如何對應至目標關聯式資料庫（如 MySQL）的欄位型態與長度設定。

2. 實體類別轉換

說明哪些類別應被視為永存類別(Persistent Class)，並轉換為資料表，以及其屬性如何映射為欄位。

3. 實體類別關係轉換

說明類別間的一對一、一對多、多對多、聚合、組合與一般化關係，如何落

實為資料表之主鍵、外鍵與關聯表。

為了說明上述概念及本書所採用的實作方式，表 11-2 整理了 PIM 轉 DB 的輸入來源與對應產出的程式構件。

Table
表 *11-2* PIM 轉 DB 的輸入來源與對應產出的程式構件

轉換面向	PIM 文件來源	PIM 文件來源（分析模型）	ChatGPT 轉出程式內容
DB	類別圖	類別圖：實體類別、屬性、資料型態、長度、主鍵、外鍵、一對多／多對多關係	1. **Migration**：資料表名稱、欄位名稱與型態、長度、主鍵(PK)／外鍵(FK)等資料庫綱要(Database Schema) 2. **Model**：Eloquent ORM 關聯設定（例如 hasMany、belongsTo 等）

透過 ChatGPT 等生成式 AI 工具，開發者可將 PIM 與提示語(Prompt)作為輸入，自動生成符合開發框架規範的 Migration 與 Model 程式碼。關於提示語的設計與應用方法，將於第 12 章中詳細介紹。

11.3.1 Migration：實體類別的結構轉換

Migration 是以程式碼方式管理資料庫綱要(Database Schema)的機制，其核心目的是確保資料庫結構能隨系統版本同步演進，並提供安全、可追蹤的版本控制流程。在模型驅動架構(MDA)的轉換流程中，Migration 扮演將平台獨立模型(PIM)中的類別模型具體化為資料庫實體結構的關鍵角色。

其核心轉換規則可歸納如下：

1. 實體類別轉換爲資料表

每一個 UML 實體類別(Entity Class)對應到資料庫中的一個資料表(Table)，系統化地將類別的靜態結構映射為關聯式資料庫的儲存單元。

2. 屬性轉換爲資料表欄位

類別的屬性(Attribute)會映射為資料表中的欄位(Field)。欄位的具體屬性，如資料型態與長度，通常依據「介面詞彙」中的規範進行定義，以確保前端 UI 呈現與後端資料存取的一致性。此外，若某屬性被設計為唯一識別，則在 Migration 中應將其設為資料表的主鍵(Primary Key, PK)，並根據需求設定非空(Not Null)、自動

遞增等約束條件。

● 實體類別轉換

在 PIM 階段建立的類別模型中，並非所有類別都需要永續儲存在資料庫中。一般而言，類別可分為：

1. **永存類別(Persistent Class)**：其資料需長期保存，例如客戶資料、訂單資料等。
2. **暫存類別(Transient Class)**：僅用於系統內部處理流程，生命週期隨執行結束而消失，例如計算結果或畫面暫存資料，通常不需轉換為資料表。

因此，PIM 轉換為資料庫 PSM 的重點在於將永存類別對應為資料表，其基本轉換步驟如下：

1. 類別對應資料表

每一個永存類別轉換為一個資料表(Table)，作為相關資料的永久儲存容器。

2. 屬性對應欄位

類別的每個屬性(Attribute)對應為資料表中的欄位(Field)，並指定合適的資料型態與長度（可參照表 11-2 資料型態對應）。

3. 鍵屬性轉換爲主鍵(Primary Key, PK)

若類別中已有唯一識別屬性，則將其設為主鍵。若未明確定義，則可在資料表中新增唯一識別欄位（例如流水號或 UUID）作為主鍵。

4. 其餘屬性轉換爲一般欄位

根據需求設定是否允許空值(NULL/NOT NULL)、預設值、是否建立索引等。

5. 依據類別關聯設定外鍵(Foreign Key, FK)

參照類別圖中的一對一、一對多、多對多或其他關聯關係，在資料表中建立對應的外鍵欄位。

在本書實作環境中，以上轉換規則透過 **Laravel Migration** 具體落實為資料庫綱要(Database Schema)，並由 **Eloquent Model** 表達類別在程式中的物件存取與關聯定義。如此可確保 PIM 中的類別設計，以一致方式反映到資料庫與應用程式程式碼之中。