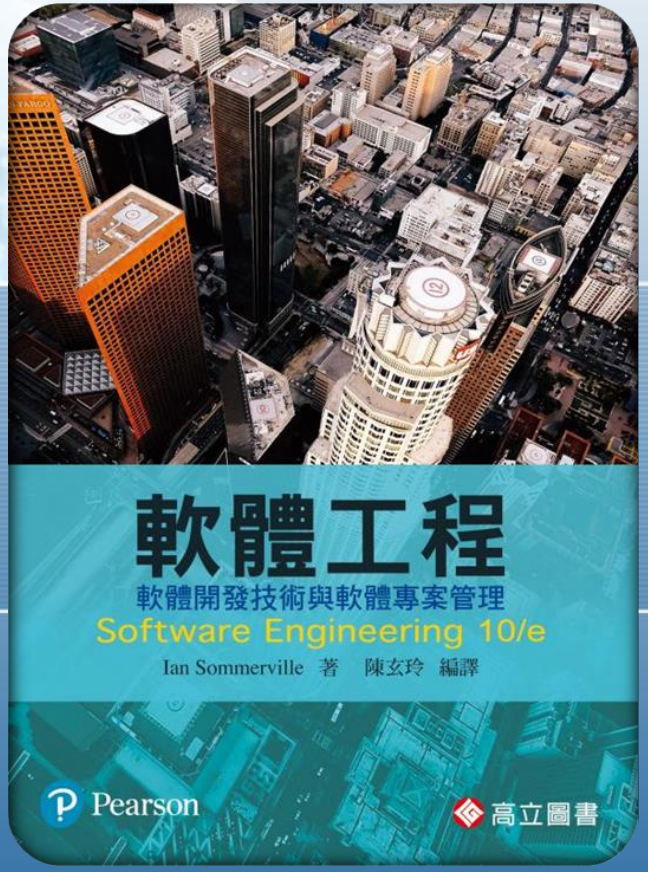
Chapter 6

架構設計



本章內容

- 架構設計 (architectural design) 是有關了解軟體系統應該如何組織，以及設計系統的整體架構。
- 在第2章介紹的軟體開發程序模型中，架構設計是軟體設計程序的第一個階段，它是介於設計與需求工程之間的重要橋樑，因為它負責找出系統的主要結構元件，以及它們之間的關係。
- 架構設計程序的輸出是架構模型 (architectural model)，用來描述系統是如何組織成一組相互溝通的元件。
- 應該在敏捷式開發程序的早期階段就設計出整體系統架構 (system architecture)，這個想法一般都能被接受。
- 為了應付變更而重構 (refactor) 個別元件通常比較簡單，但重構系統架構成本就很高了。

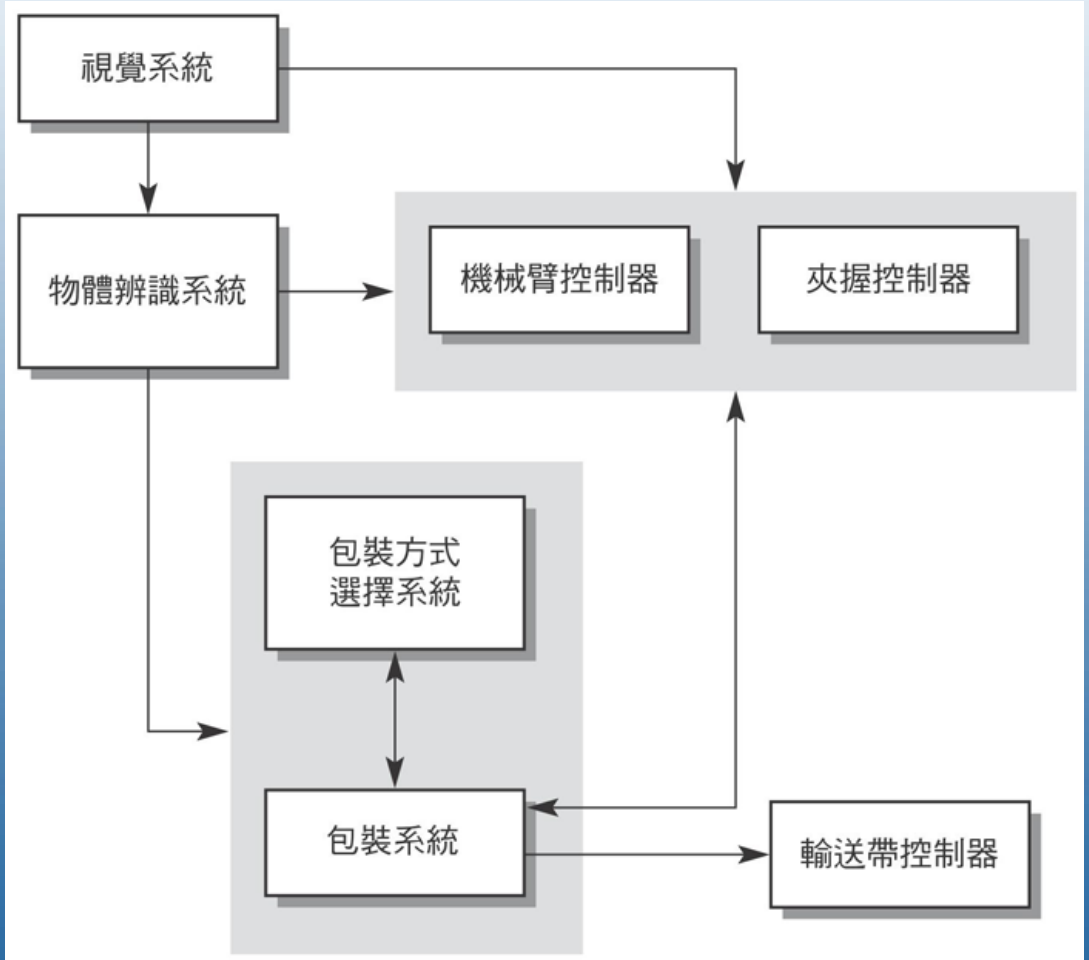


圖 6.1　包裝機器人控制系統的控制器架構

- 軟體架構的設計可分成大小兩種抽象層次：
  1. 小型的架構主要是著眼於個別程式的架構，也就是個別程式是如何分解成元件。
  2. 大型的架構則主要是針對複雜的企業系統的架構，裡面包含其他的系統、程式和程式元件。這些企業系統分散在不同的電腦上，甚至可能是不同的公司所擁有與管理。

- 明確的設計與記錄軟體架構有3個優點：
  1. **與關係人進行溝通**：適合成為不同領域的關係人共同討論的焦點。
  2. **系統分析**：對於系統將來能否符合某些關鍵需求，如執行效能、可靠性以及易維護性等，具有很深遠的影響。
  3. **大規模再利用**：需求相似的系統，架構經常也會類似，因此能大規模的再利用已完成的軟體。甚至還可以開發類似生產線的架構，這也是一種再利用的方式。

- 一般常用簡單的方塊圖 (block diagram) 來非正式的描述系統架構，如圖6.1。
- 雖然它們被廣泛使用，不過Bass等人則認為，不應該用這類非正式的方塊圖來描述系統架構，因為它們無法表現出系統元件之間的關係類型，也無法表現元件的外在性質。

1. **用來輔助系統設計的討論：**系統的高階架構觀點 (view) 用在與系統的關係人溝通和專案規劃時很有用，因為裡面沒有摻雜細節。關係人比較能夠認同而了解系統的抽象觀點，之後討論系統時就不會被細節混淆。
2. **當作已設計架構的紀錄文件：**這裡的目標是產生完整的系統模型，裡面顯示出系統中各個元件、它們的介面，還有它們的連結。

6.1 架構設計決策

- 架構設計是一個需要發揮創造力的過程，會因為被開發系統的類型、系統架構師 (system architect) 的經驗與背景，以及系統的專門需求不同而不同。
- 在同一應用領域的系統經常會有類似的架構，這反映出該領域的基本概念。比如說，所謂的應用程式生產線 (application product line) 是以某個核心架構為中心，而根據個別客戶的需求而製作出不同的版本。
- 軟體系統的架構可能是以某個特定的架構模式 (pattern) 或樣式 (style) 為基礎 (樣式與模式在此是指同樣的東西)。架構模式是擷取在不同軟體系統中使用過的架構它們的要素。

6.1 架構設計決策

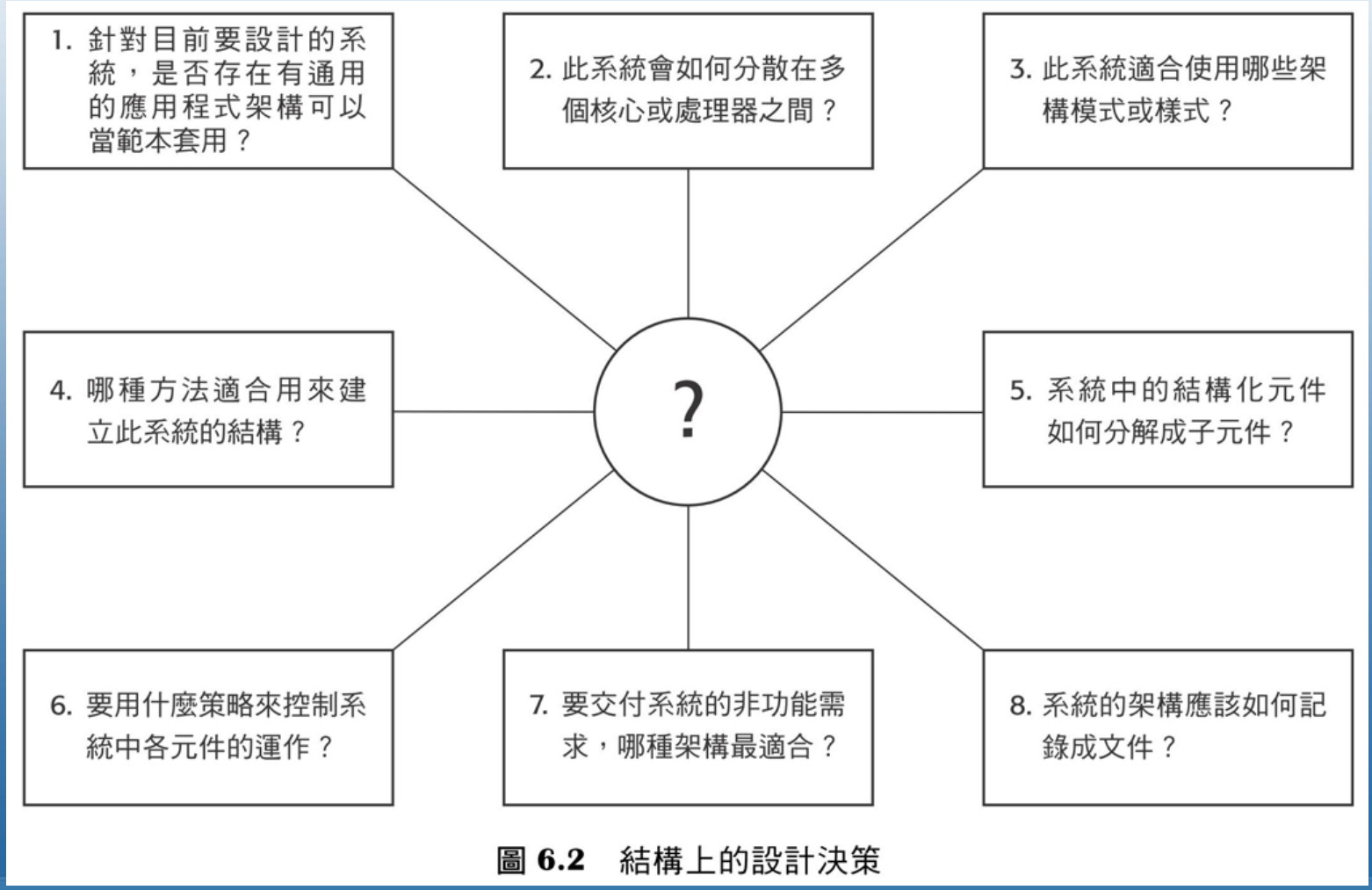


圖 6.2　結構上的設計決策

6.1 架構設計決策

- 應該根據下列幾項系統的非功能需求，來選擇系統所使用的架構模式和結構：
  1. **執行效能 (performance)**：應該將重要的運算動作集中在少數幾個元件中，使用少數幾個比較大的元件，而不要很多個功能很細的小元件。
  2. **保全性 (security)**：將最重要的資產保護在最內層。
  3. **安全性 (safety)**：與安全有關的所有運算動作集中放在單一元件或少量幾個元件中。
  4. **可用性 (availability)**：第11章會說明為可用性需求較高的系統所設計的容錯系統。
  5. **易維護性 (maintainability)**：應該使用功能較細微且能自給自足的元件。

6.2 架構的各種觀點

- 本節我將探討兩個與這兩者相關的主題：
  1. 在設計系統架構或記錄其文件時，什麼觀點 (view) 或面向會有用？
  2. 應該用什麼符號表示法來描述架構模型？
- 我們不可能在單一圖形中，表達出系統架構所有相關的資訊，因為每種圖形化的模型，都只能表達出系統的一個觀點或面向。
- 例如展示系統是如何分解成模組、執行期間的程序如何互動，或者系統元件分散在網路上的各種不同方式。因此對於設計和記錄文件而言，通常需要表達軟體架構的多種面向。

6.2 架構的各種觀點

- Krutchen (Krutchen 1995) 在其著名的軟體架構4 + 1觀點模型中，曾經建議有4種基本的架構觀點是應該要有的，它們可透過共同的使用案例或情境連結起來。

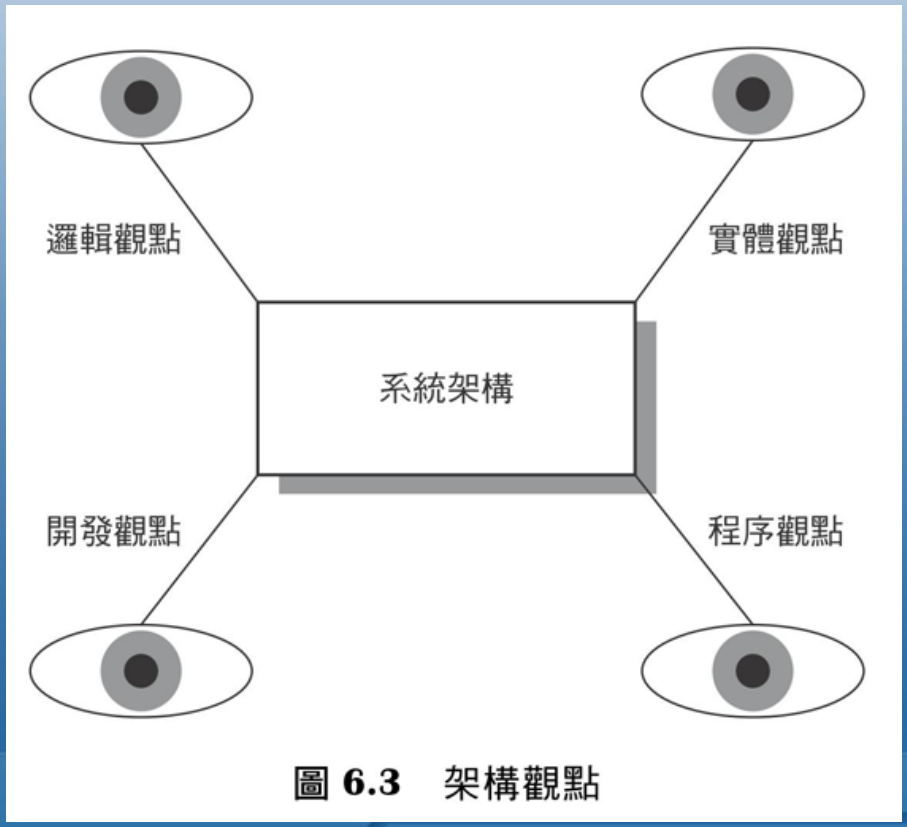


圖 6.3 架構觀點

6.2 架構的各種觀點

1. **邏輯觀點 (logical view)**：用來將系統中的關鍵抽象資訊表達成物件或物件類別。
2. **程序觀點 (process view)**：用來展示在執行期間，系統是如何由一些相互溝通的程序所組成。
3. **開發觀點 (development view)**：用來展示為了進行開發要如何分解軟體。
4. **實體觀點 (physical view)**：表達系統硬體和軟體元件如何分散在系統的各個處理器上。

6.2 架構的各種觀點

- 對於軟體架構師是否應該使用UML來描述和記錄系統架構，目前的看法不一。一份2006年的研究顯示，使用UML的時候大多是以鬆散而非正式的方式，論文作者認為這是件壞事。
- 不過我不同意這個看法，UML是設計來描述物件導向系統，而且在架構設計階段通常會想要以更高階的抽象層次來描述系統。
- 有些專家建議使用更專門的架構描述語言 (ADL) 來描述系統架構。不過也因為ADL是專業用的語言，其他領域和應用程式的專家們，發現ADL不容易瞭解和使用。
- 我不認為它們能成為軟體工程的主流做法。

6.3 架構模式

- 模式 (pattern) 是一種表達、分享和再利用關於軟體系統各種知識的方式。
- 你可以把架構模式想像成，將已經在各種不同的系統和環境中試驗過和測試過的良好實務經驗，加以樣式化和抽象化的描述。其中應該包含何時適合和不適合使用此模式，以及此模式的長處和弱點。
- 經樣式化之後的模式描述中，包含模式名稱、簡短說明、圖形模型，以及一個應用此模式的系統類型範例。

6.3 架構模式

名稱	MVC (Model-View-Controller)
說明	把展示和互動功能與系統資料分開。系統的結構分成 3 個能彼此相互溝通的邏輯元件。Model 元件負責管理系統資料和該資料的相關運算動作。View 元件負責定義和管理如何將資料展示在使用者眼前。Controller 元件負責管理使用者互動（如按鍵盤和滑鼠點選等），以及將這些互動動作傳遞給 View 和 Model 元件。參見圖 6.5
範例	圖 6.6 展示使用 MVC 模式所組織成的網站應用程式系統的架構
使用時機	當資料有多種檢視方式和互動方式時使用。另外當資料未來可能有未知的互動需求和展示需求時使用
優點	讓資料與它的展示方式分開而能夠獨立改變，反之亦然。支援相同資料可以用不同方式來展示的功能，而且當資料在某一種展示方式中被修改時，從所有其他展示方式也會看到同樣結果
缺點	當資料模型和互動情形很簡單時，這樣做可能會增加額外的程式碼和程式碼複雜度

圖 6.4　MVC (Model-View-Controller) 模式

6.3 架構模式

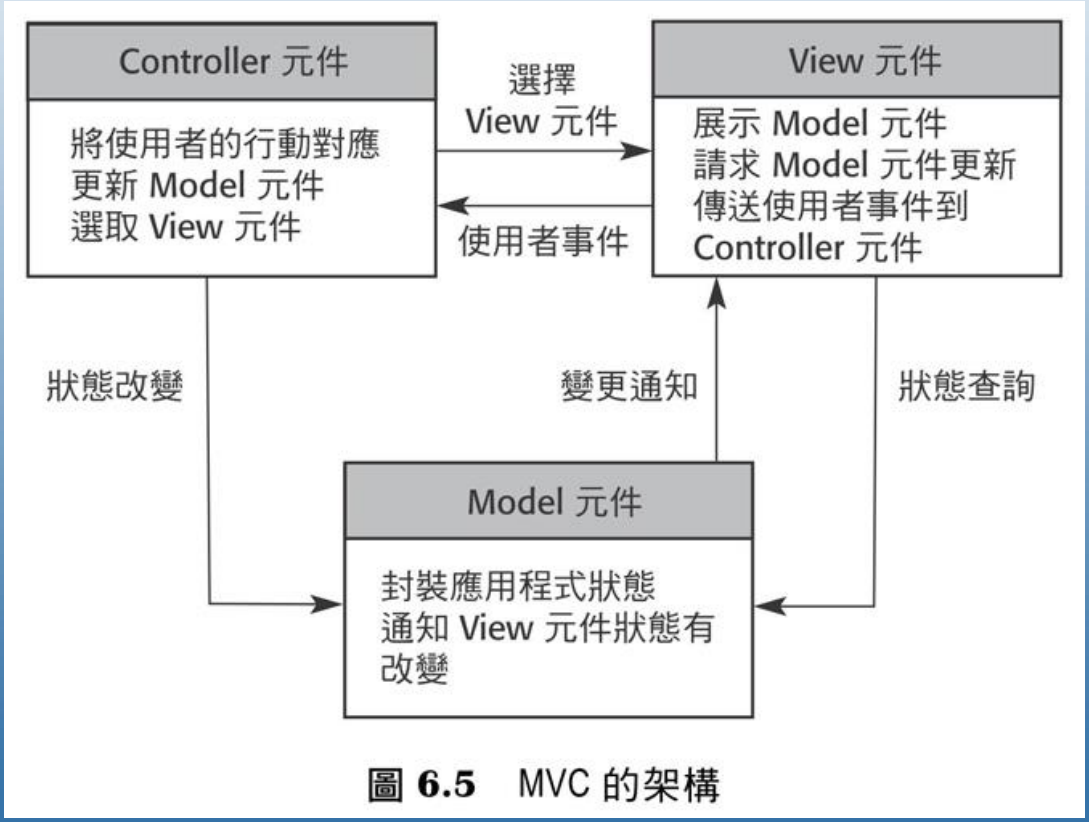


圖 6.5 MVC 的架構

6.3 架構模式

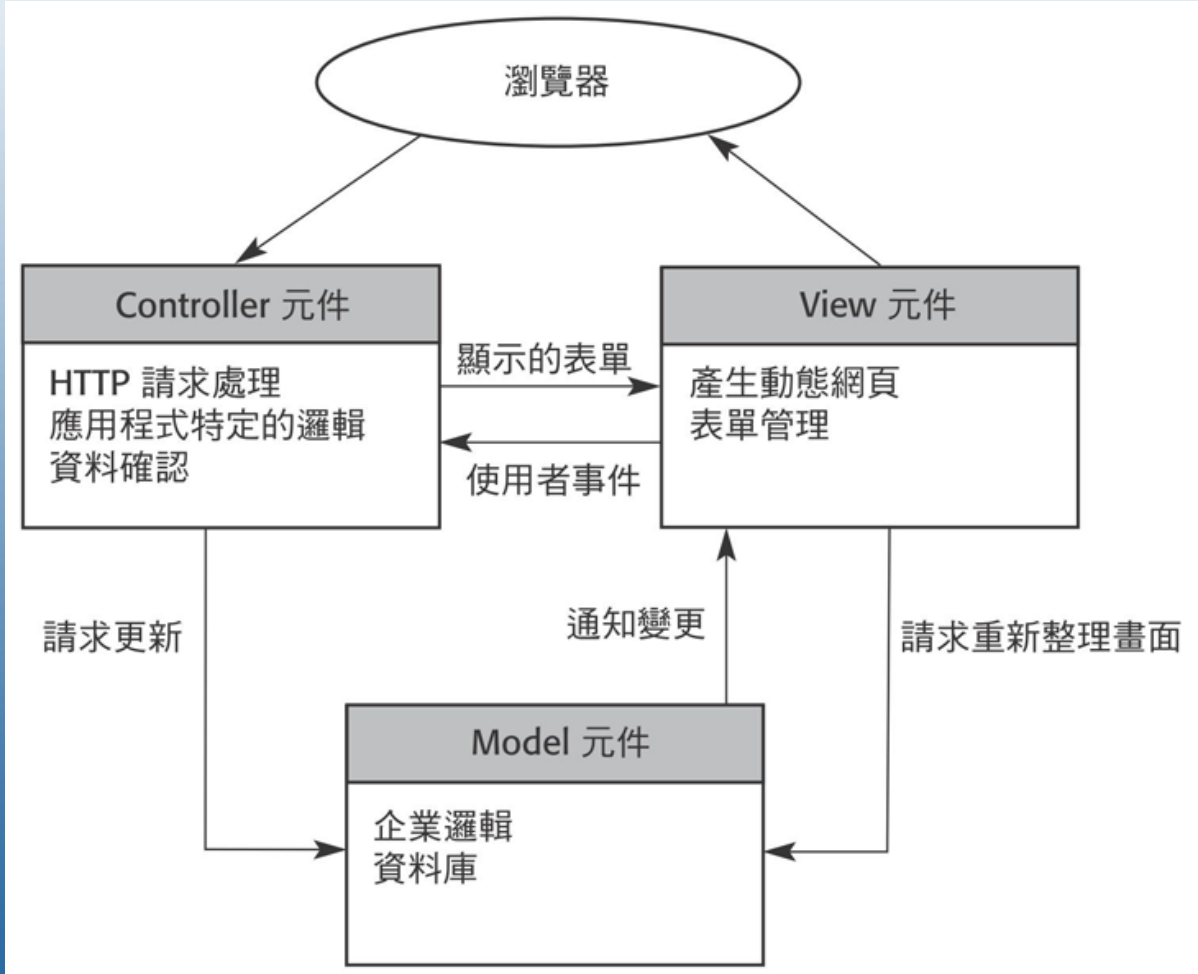


圖 6.6　使用 MVC 模式的網站應用程式架構

6.3.1 分層式架構

- 分層式架構 (layered architecture) 模式是另一種達到分隔和獨立原則的方式。
- 在這裡系統功能被組織成幾個分開的層級，每一層 (layer) 只依賴緊鄰的下一層所提供的功能和服務。
- 分層法支援增量式的系統開發方式。只要它的介面不變，就可以用已擴充新功能的新的一層來取代舊有的這一層，而不會動到系統的其他部分。
- 假如分層的介面改變或添加了新功能，也只會影響到相鄰的分層。

6.3.1 分層式架構

名稱	分層式架構 (Layered architecture)
說明	將系統架構成多個層級，把相關的功能集中在同一層。每層提供服務給上面的層級，換言之最底下的層級是提供核心服務，也就是整個系統都可能使用到的服務。參見圖 6.8
範例	提供校內不同群組學習支援的數位學習系統，在此它採用分層式模型（圖 6.9）
使用時機	當想要在現有系統的上層建立新功能時；當開發工作是分配給數個團隊，而每個團隊負責一層功能時；當有多層次保全的需求時
優點	只要介面還在，就可以換掉整層。也可以在每一層提供某些重複的功能（如驗證）以提高系統的可信賴度
缺點	在實務上，要將每個層級分隔清楚是很困難的，而且高階層級也可能需要直接與較低層進行溝通，而不透過相鄰的下一層。執行效能也是一個問題，因為每個服務請求可能必須經過許多層處理

圖 6.7　分層式架構模式

6.3.1 分層式架構

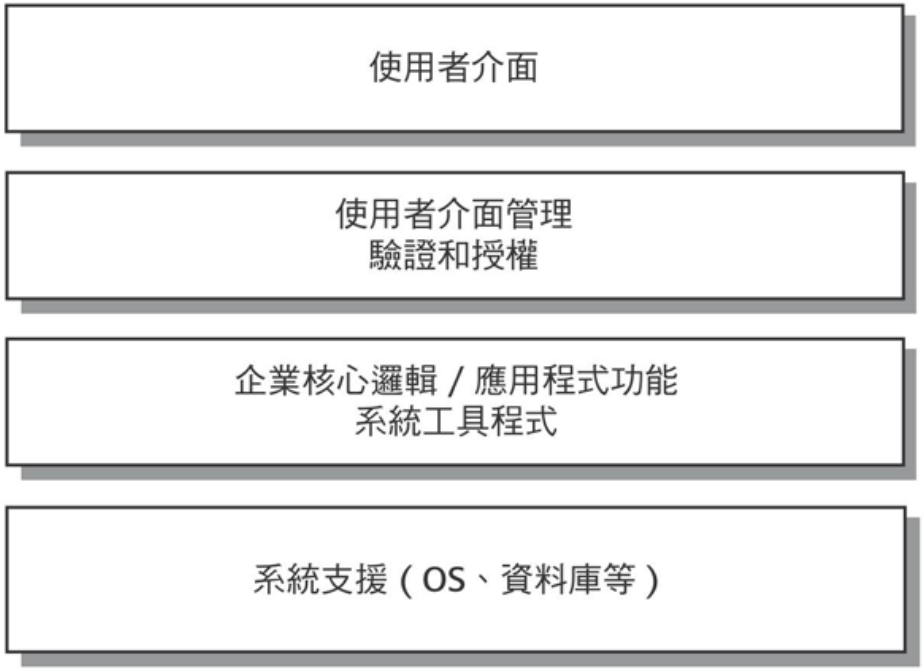


圖 6.8　一種通用的分層式架構

6.3.1 分層式架構

圖 6.9 iLearn 系統的架構

6.3.2 儲藏庫架構

名稱	儲藏庫 (Repository)
說明	將所有共用資料保存在一個集中式儲藏庫，讓所有系統元件都可以存取。元件彼此不直接溝通，只能透過儲藏庫來溝通
範例	圖 6.11 是個 IDE 範例，裡面的元件使用一個系統設計資訊的儲藏庫。每個軟體工具所產生的資訊，之後可以給其他工具來使用
使用時機	當系統會產生大量的資訊而且必須儲存以備日後使用時，應該使用這個模式。你也許可以在資料驅動 (data-driven) 系統中使用它，此時在儲藏庫裡的資料可以觸發某個動作或工具
優點	元件可以是獨立的，它們不需要知道其他元件的存在。一個元件所做的變更可以傳播到所有元件。所有資料的管理都是一致的 (例如在同一時間進行備份)，因為它們全儲存在同一個地方
缺點	儲藏庫是本系統單點故障 (single point of failure) 的弱點，因此如果儲藏庫出問題會影響整個系統。所有的溝通都要透過儲藏庫可能會導致效率不彰。要將儲藏庫分散到數台電腦上可能很困難

圖 6.10　儲藏庫模式

- 大多數會使用大量資料的系統，都會建構成共用資料庫或儲藏庫的架構。

6.3.2 儲藏庫架構

- 圖6.11是儲藏庫其中一種可能的用法。

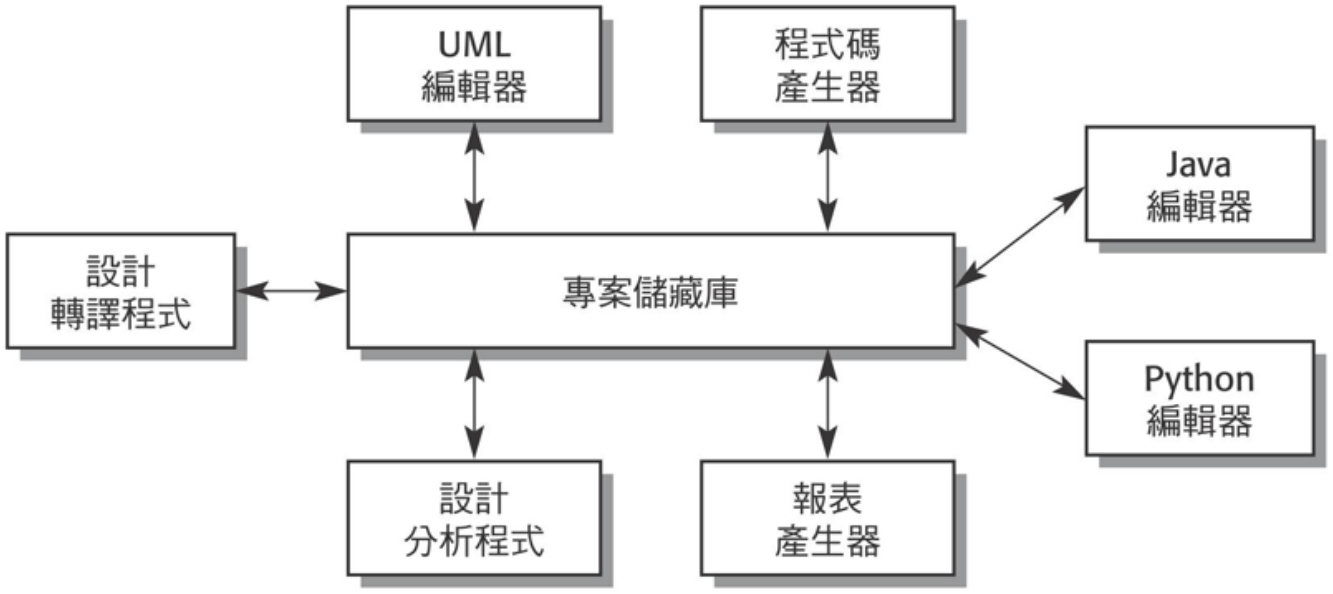


圖 6.11　IDE 的儲藏庫架構

6.3.3 主從式架構

- 主從式 (Client-Server) 模式。使用主從式模式的系統是架構成一組服務和相關聯的伺服器，以及存取和使用這些服務的用戶端。

  1. 伺服器：一組提供服務給其他元件的伺服器。

  2. 用戶端：一組向伺服器要求提供服務的用戶端。

  3. 網路：一個能讓用戶端存取這些服務的網路。

6.3.3 主從式架構

名稱	主從式 (Client-server)
說明	在主從式架構中，系統的功能是組織成一個個服務，每個服務由不同的伺服器提供。用戶端是這些服務的使用者，藉由存取伺服器來利用它們
範例	圖 6.13 是架構成主從式系統的影片與相片收藏系統
使用時機	當資料是保存在共用資料庫，而必須讓來自多個地點的存取動作使用時。因為伺服器可以被複製，所以當系統上的負載會變動時，也可以考慮使用
優點	這個模型主要的優點是伺服器可以分散在網路上。通用的功能（如列印服務）能提供給所有用戶端都使用，不需要所有的服務都實作這類功能
缺點	每個服務都是可能發生單點故障 (single point of failure) 的弱點，容易受阻斷服務攻擊或伺服器故障的影響。效能可能不易掌握，因為要看網路和系統的情況而定。假如各伺服器是隸屬於不同的機構，可能會有管理上的問題

圖 6.12　主從式模式

6.3.3 主從式架構

- 主從式架構通常被視為分散式系統架構，但邏輯模型上在不同伺服器執行的獨立服務，其實可實作在同一台電腦上。
- 用戶端是使用某個請求／回應 (request-reply) 協定 (如http)，透過遠端程序呼叫 (remote procedure call) 來存取伺服器所提供的服務。

6.3.3 主從式架構

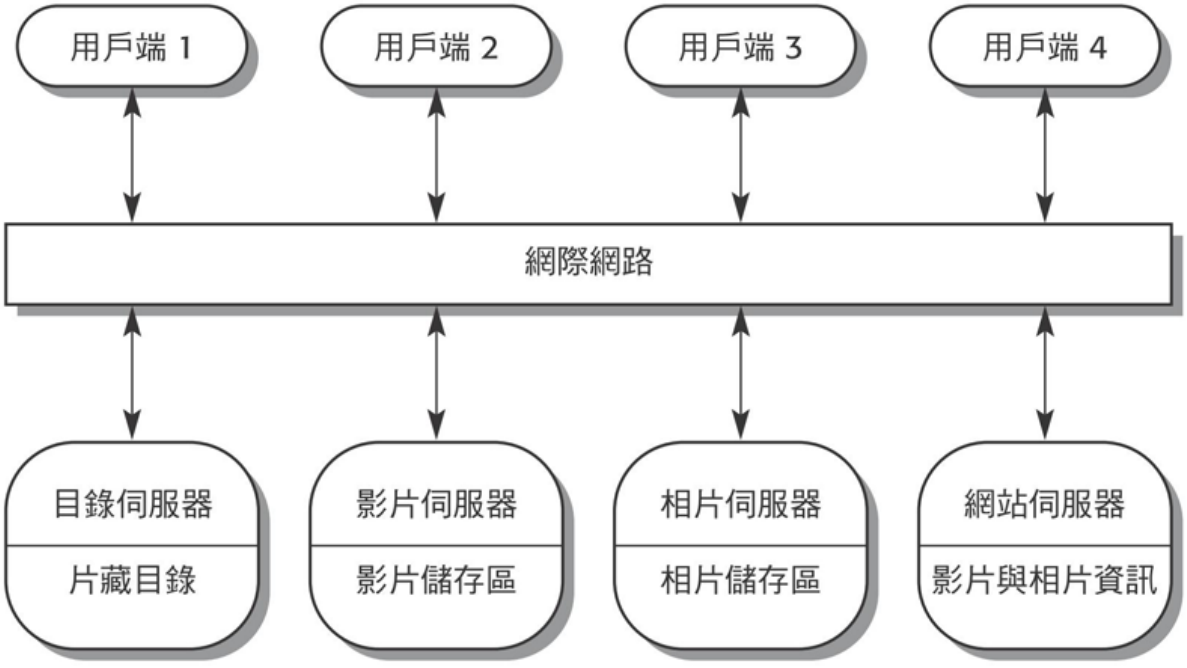


圖 6.13 影片與相片收藏系統的主從式架構

6.3.4 管道與篩選架構

- 管道與篩選 (Pipe and Filter) 模式，透過功能轉換處理輸入值並產生輸出結果。
- 這個名稱是出自Unix系統的術語。
- 從電腦一開始用在自動化資料處理時，就一直在使用這種模式的各種變化形式。
- 當它的轉換動作是以批次方式循序處理資料時，這種管道與篩選架構模型就變成一種批次循序模型 (batch sequential model)。

6.3.4 管道與篩選架構

名稱	管道與篩選 (Pipe and Filter)
說明	系統中資料處理動作的組織方式是每個處理元件（篩選）是分離的，而且各自負責一種資料轉換。而資料是從一個元件流向（像在管道中）另一個元件來處理
範例	圖 6.15 是應用在帳務處理的管道與篩選系統範例
使用時機	常用在資料處理應用上（批次或異動為主兩者皆是），其輸入會經過多個分開的階段處理後產生輸出
優點	容易了解，並且支援轉換功能再利用。這樣的工作流程風格符合很多商業流程的結構。軟體演進時只要加上轉換功能，所以很容易。可實作成循序或平行系統
缺點	在溝通轉換時必須使用雙方同意的資料傳輸格式，每個轉換動作都必須剖析它的輸入，然後再把輸出還原成公認格式，這會增加系統的額外負擔；而且意味著那些使用不相容資料結構的轉換功能，可能就無法再利用

圖 6.14　管道與篩選模式

6.3.4 管道與篩選架構

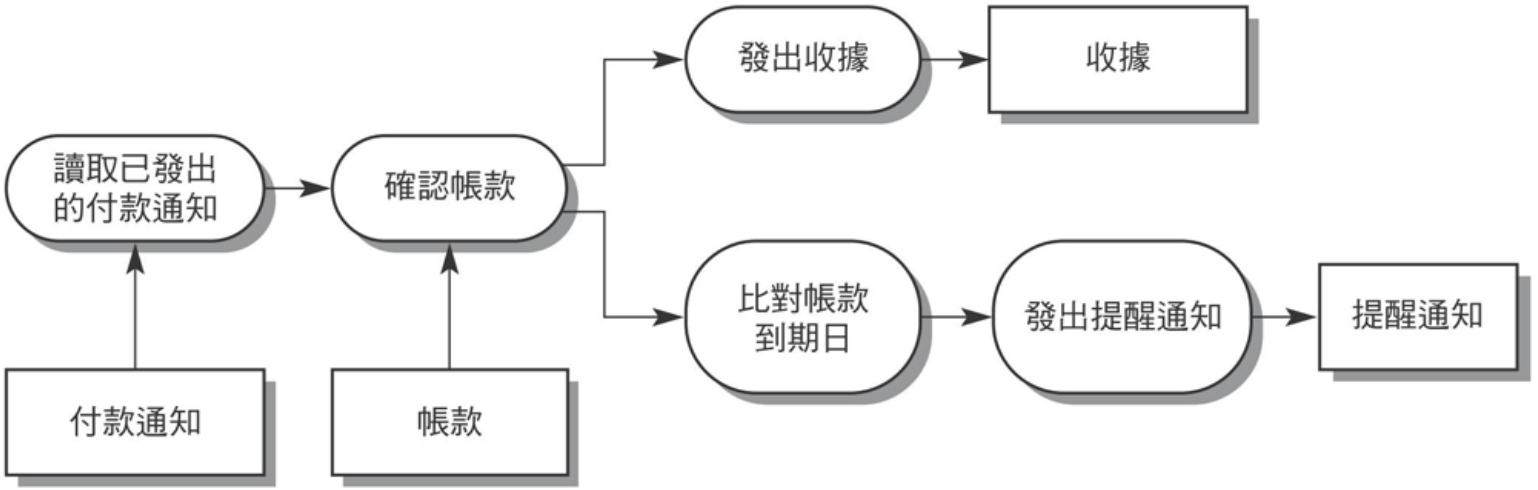


圖 6.15　管道與篩選架構的例子

- 管道與篩選系統最適合批次處理系統和嵌入式系統，因為它們只提供有限的使用者互動功能。互動式系統如果用管道與篩選模型會很難寫。

6.4 應用程式架構

- 應用程式系統是為了符合企業或組織的某些需求而建立的。所有的企業有很多地方都是共同的，它們需要僱用員工、開發票、保存帳號等。因此這些企業使用的應用程式系統也會有很多共同處。
- 很多企業系統是採用將通用的應用程式系統，經過組態設定而建立出一個新的應用程式，等於是以隱含的方式再利用應用程式架構。

6.4 應用程式架構

- 應用程式架構模型：
  1. 當作架構設計程序的起點
  2. 當作設計的檢核清單
  3. 當作開發團隊工作的組織方式
  4. 當作評估元件能否再利用的一種方式
  5. 當作討論應用程式的詞彙

6.4 應用程式架構

- 目前有非常多種應用程式系統，而且看起來似乎都大不相同。本章將描述其中兩種應用程式的架構：

1. **異動處理應用程式**：異動處理 (transaction processing) 應用程式是以資料庫為中心的應用程式，它們負責處理使用者的資訊請求，並更新資料庫裡的資訊。這類系統包括互動式銀行系統、電子商務系統、資訊系統和訂位系統等。
2. **語言處理系統**：語言處理 (language processing) 系統是將使用者的想法以正規化語言 (如程式語言) 來表達的系統。最有名的語言處理系統是編譯器 (compiler)。

6.4.1 異動處理系統

- 異動處理系統是設計來處理那些想要讀取或更新資料庫中資訊的使用者請求。
- 以使用者的角度而言，一個異動是由一連串為了達到目標所做的運算動作組合而成，例如「尋找從倫敦到巴黎的航班時間表」。

6.4.1 異動處理系統

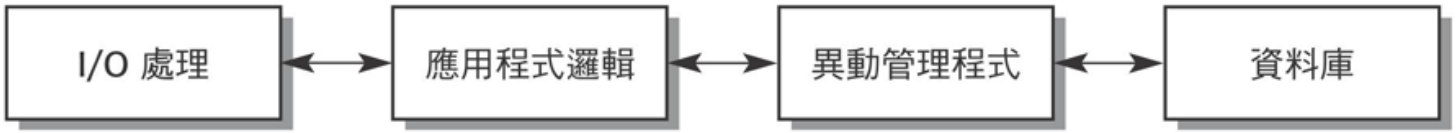


圖 6.16　異動處理應用程式的結構

- 首先使用者透過I/O處理元件向系統發出請求，這個請求會由某些應用程式邏輯進行處理而建立一筆異動。然後將該筆異動傳遞給異動管理程式 (transaction manager)，它通常是內建在資料庫管理系統中。等異動管理程式確認異動已經正確完成，它會通知應用程式處理已完成。

6.4.1 異動處理系統

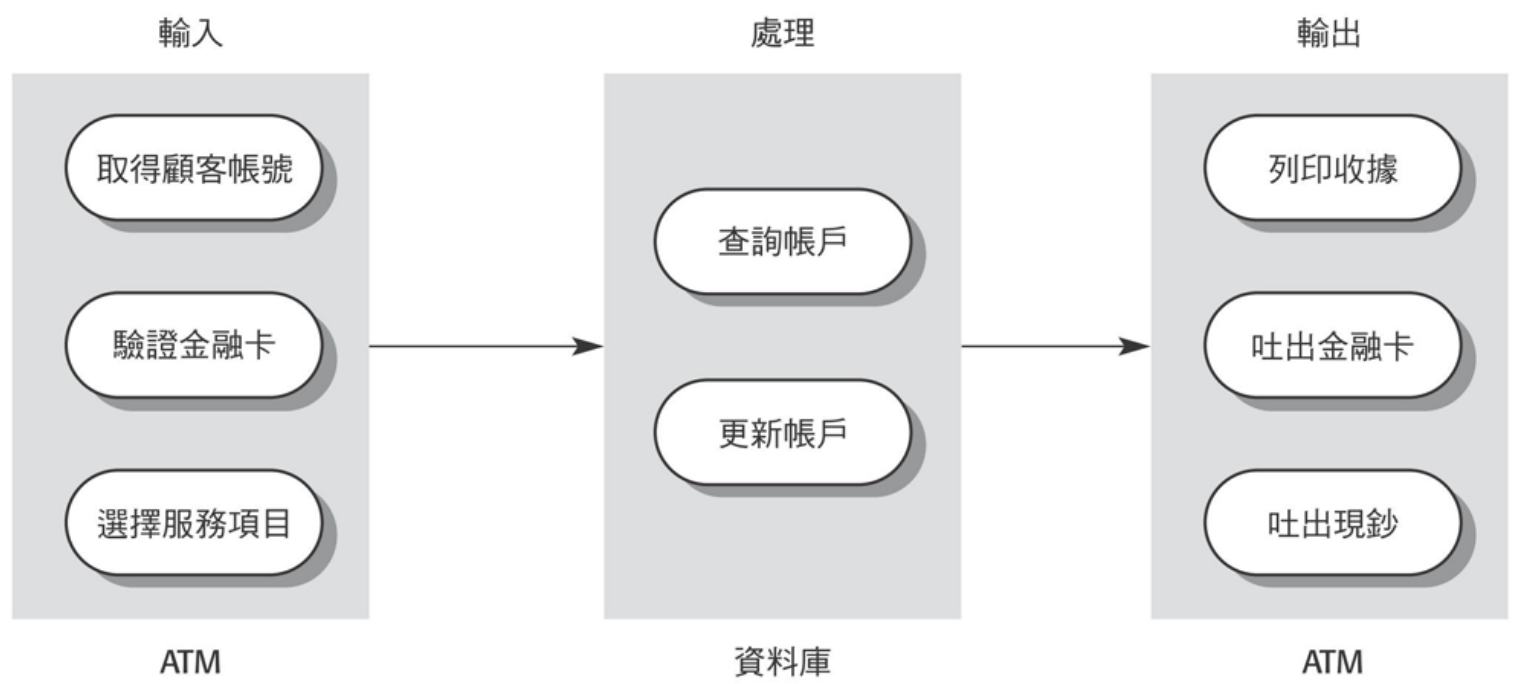


圖 6.17　ATM 系統的軟體架構

6.4.2 資訊系統

- 圖6.18是個很常見的資訊系統模型。它將系統塑模為分層式(參見6.3.1節)，其中最上層是支援使用者介面，而最底層是系統資料庫。使用者通訊層負責處理所有進出使用者介面的輸出入，而資訊擷取層則包含負責存取和更新資料庫的應用程式邏輯。

圖 6.18　資訊系統的分層式架構模型

6.4.2 資訊系統

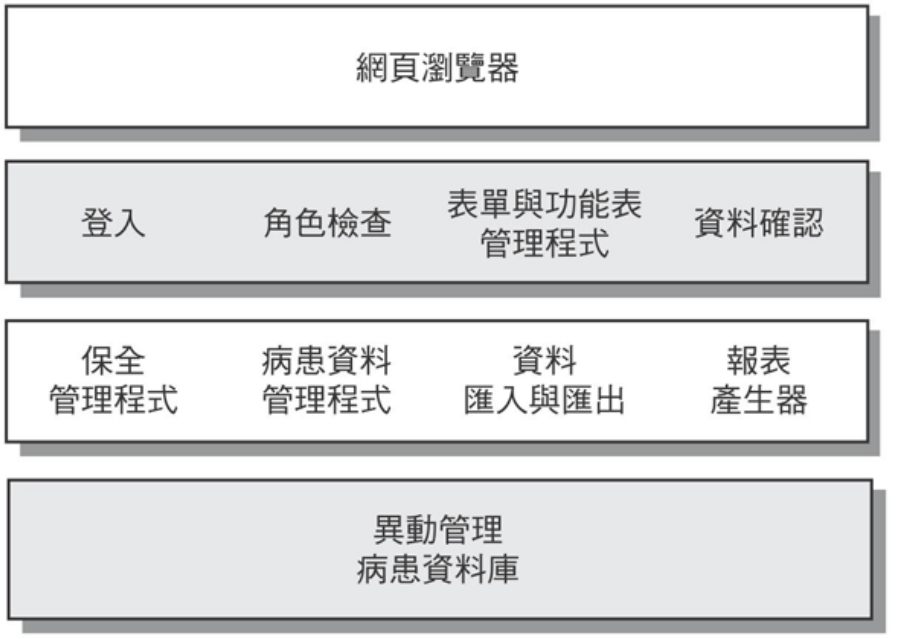


圖 6.19 Mentcare 系統的架構

6.4.2 資訊系統

- 資訊與資源管理系統有時也會是一種異動處理系統。例如電子商務系統是以Internet為基礎的資源管理系統，它接受商品或服務的電子訂單，然後安排遞送這些商品或服務給顧客。在電子商務系統中，應用程式層將額外加上「購物車」功能，讓使用者可分多次異動將購物品項放入，最後在同一筆異動(交易)中一起結帳。

6.4.2 資訊系統

- 這種系統通常是實作成具備多層式 (multitier) 主從架構的分散式系統 (參見第17章)：
  1. 網站伺服器負責所有的使用者通訊工作，使用者介面是用網頁瀏覽器來實作。
  2. 應用程式伺服器負責實作應用程式邏輯，以及資訊儲存和擷取請求。
  3. 資料庫伺服器則是負責搬移資訊進出資料庫和異動管理。

6.4.3 語言處理系統

- 語言處理系統 (language-processing system) 能將某種語言翻譯成該語言的另一種表示法。常聽到的編譯器 (compiler) 是將程式語言翻譯成機器碼。

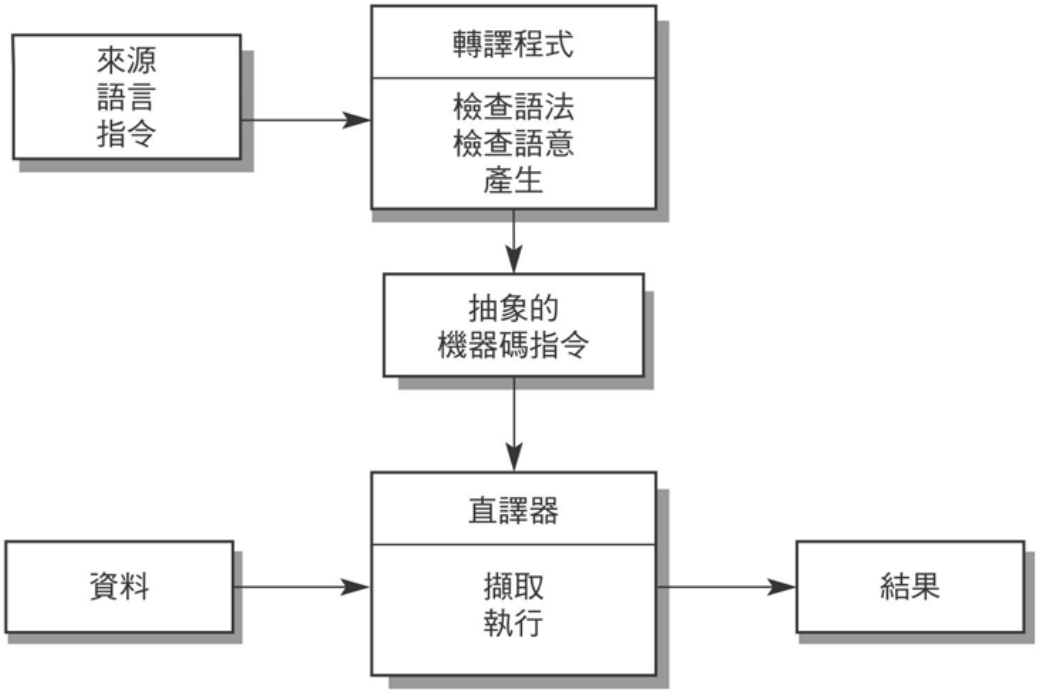


圖 6.20　語言處理系統的架構

6.4.3 語言處理系統

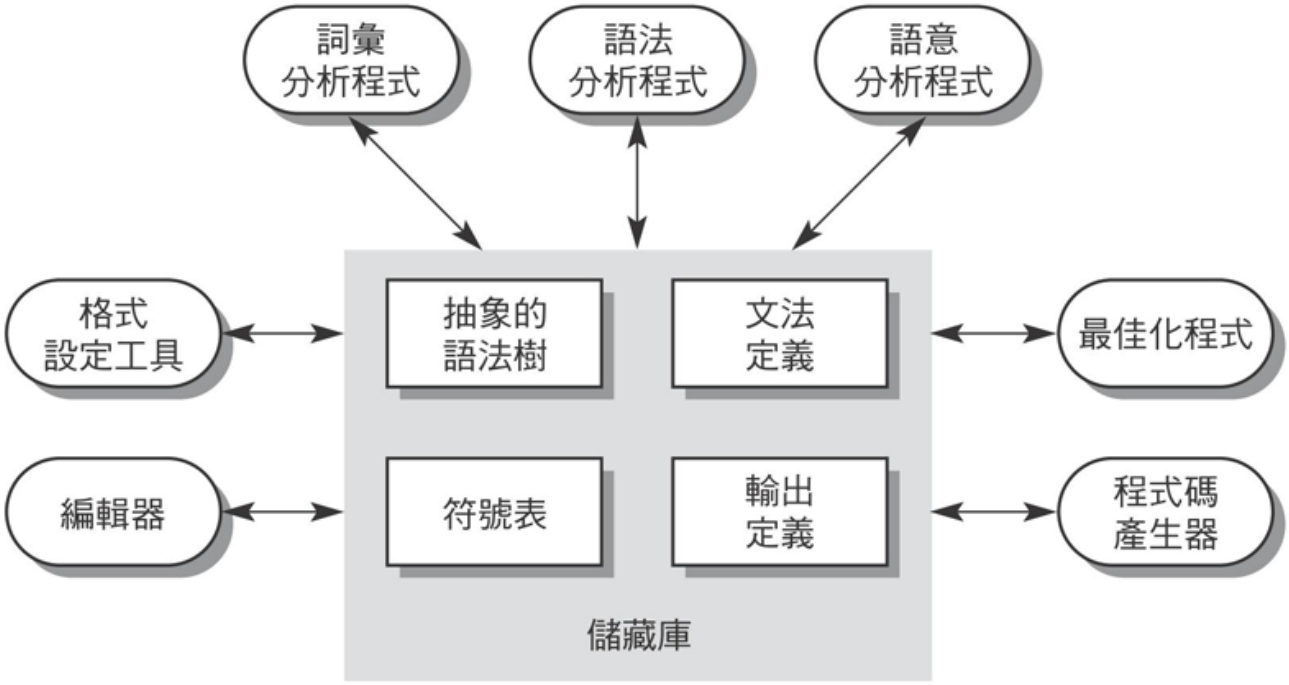


圖 6.21　語言處理系統的儲藏庫架構

6.4.3 語言處理系統

1. **詞彙分析程式 (lexical analyser)**：負責取得輸入語言token並轉換成內部格式。
2. **符號表 (symbol table)**：它儲存被翻譯文字中有關實體名稱的資訊(變數、類別名稱、物件名稱等)。
3. **語法分析程式 (syntax analyser)**：負責檢查被翻譯語言的語法。
4. **語法樹 (syntax tree)**：用來表達被編譯器的內部結構。
5. **語意分析程式 (semantic analyser)**：使用語法樹和符號表的資訊，檢查輸入語言文字的語意正確性。
6. **程式碼產生器 (code generator)**：它「走」過一遍語法樹，產生抽象機器碼。

6.4.3 語言處理系統

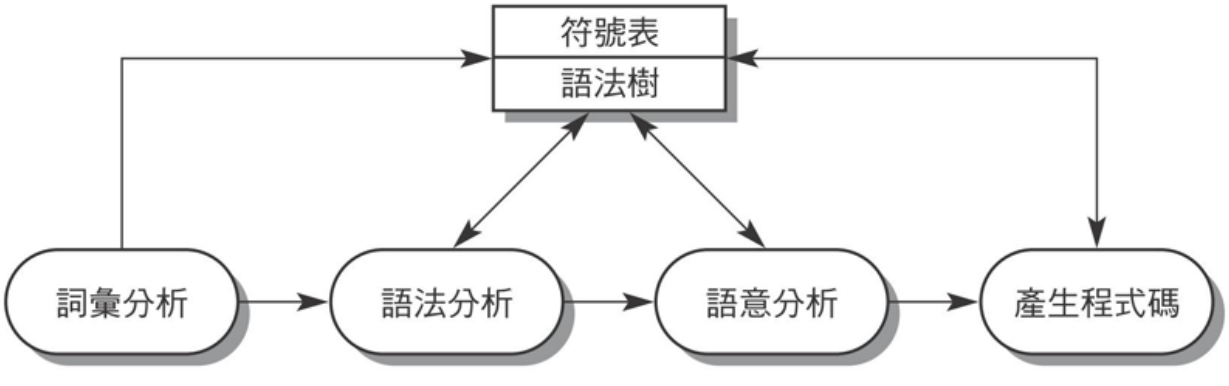


圖 6.22　管道與篩選模型的編譯器架構