貳、文獻探討

2.1 應用程式架構

Gartner Group 針對主從式運算提出一個「五種主從式運算」(Five Styles of Client/Server Computing)的模式(圖 3)。此模式將系統架構區分成三個分開的層次：資料管理(Data Management)、應用邏輯(Application Function)及展現(Presentation)[6]。

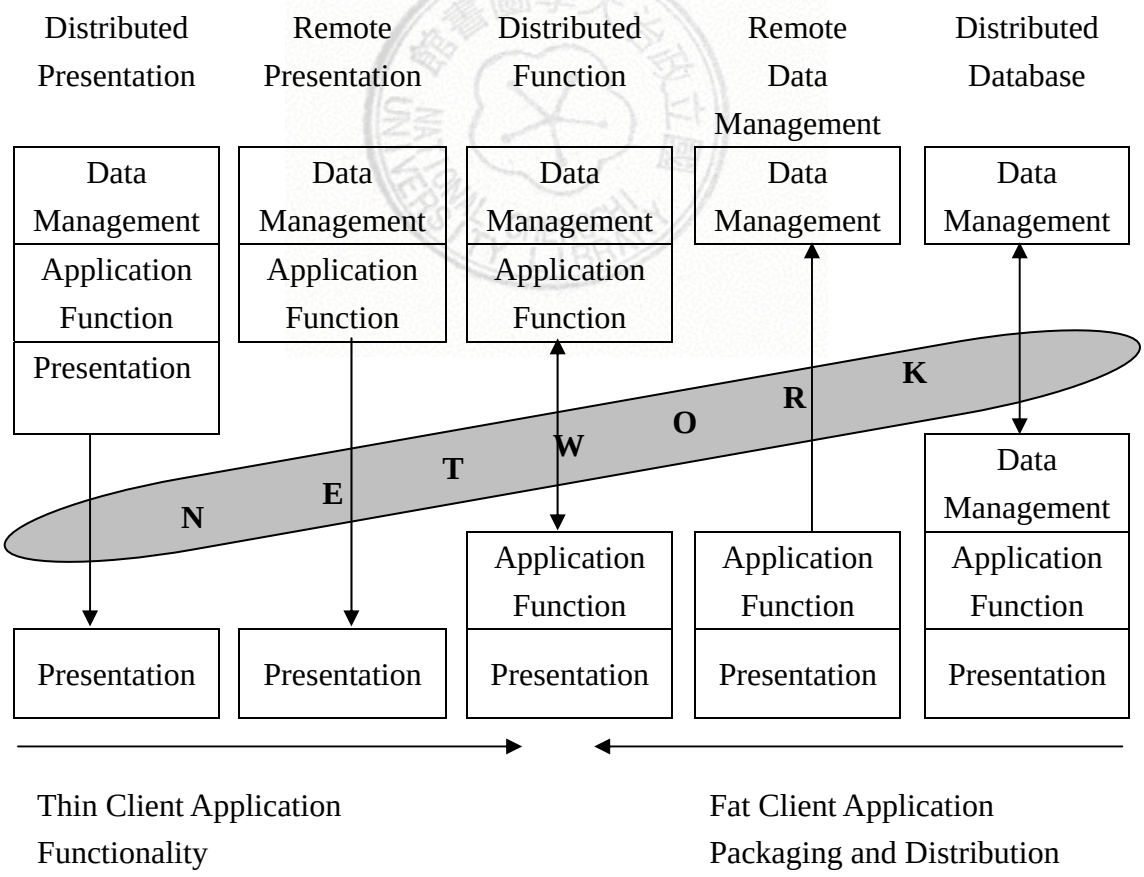

圖 3：Gartner Group's 五種主從式運算模式

第一個層次是展現層，即是面對使用者收集及展現資訊的使用者介面；第二層應用邏輯層即是依據商業規則所設計之處理邏輯的部份；第三層資料庫層則是負責資料的存取功能。設計企業應用程式之系統架構時，即可應用此種分類方法。

傳統的主從式架構多半採用 Fat Client Application 的型式，可能採用 Distributed Database、Remote Data Management 或是 Distributed Function 等方式，商業邏輯部份會被實作在使用者介面或資料管理之中。如此的切割方式最大問題在於商業邏輯與使用者介面同時存在使用者端的程式中，一旦商業邏輯變動時，時常需要重新部署使用者端的程式，重新部署成本及維護成本將提高。本研究之案例目前即是採用此種架構。

三層式架構與二層式架構最大的差異處，是將商業邏輯獨立出來(如圖 4)[23]，而形成三層邏輯分明的層次，較傾向 Thin Client Application 的架構。

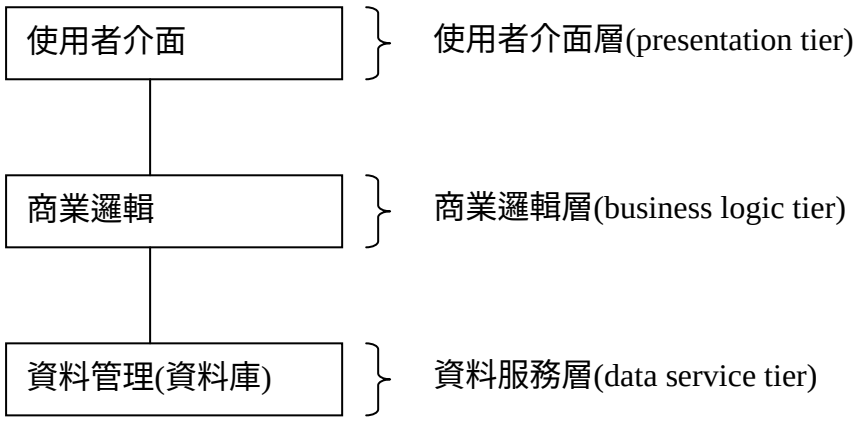

圖 4：三層式架構分層示意圖

此種架構可解決二層式架構重新部署使用者端程式的問題；加上配合物件導向技術，中間層的商業邏輯以元件實作，當商業處理邏輯變動時，僅需更動變動元件部分，程式架構比較有彈性；在網際網路的環境建置企業的應用程式，配合網頁技術，即可達到使用者端的操作簡易性及系統的延伸性。

在中間層負責處理商業邏輯的角色稱之為應用伺服器(Application Server)。應用伺服器的概念早在 1989 年，Lee[25]的全球資訊網的計畫書裡即已提及，如圖 5。

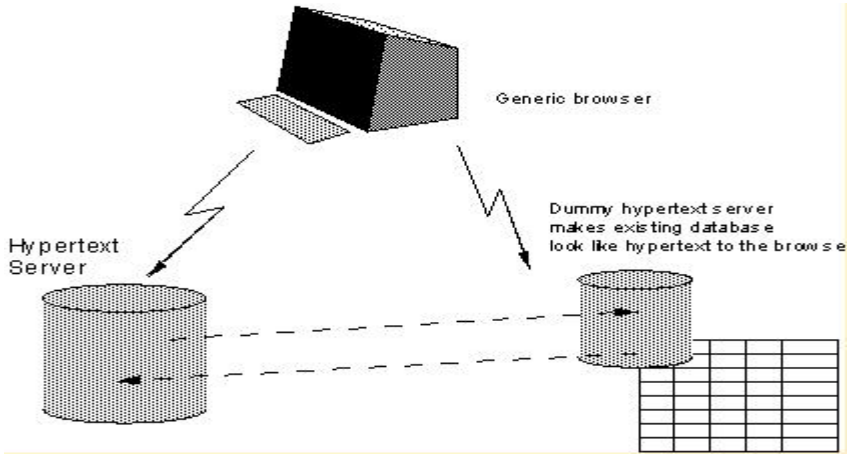

圖 5：應用伺服器定義示意圖

其中，Dummy Hypertext Server 所扮演的角色就類似今日的應用伺服器。它從資料庫中擷取資料並轉換成 hypertext 的格式並透過 Hypertext Server 傳回使用者端的網頁瀏覽器。而使用者根本不知道也不需在意有 Hypertext Server 及 Dummy Hypertext Server 的區別。因此，套用在多層式架構中所指稱之應用伺服器也就是介於網頁伺服器(Web Server)及資料庫(Database)之間的伺服器(圖 6)，主要用於處理商業邏輯元件的伺服器[6][14]。

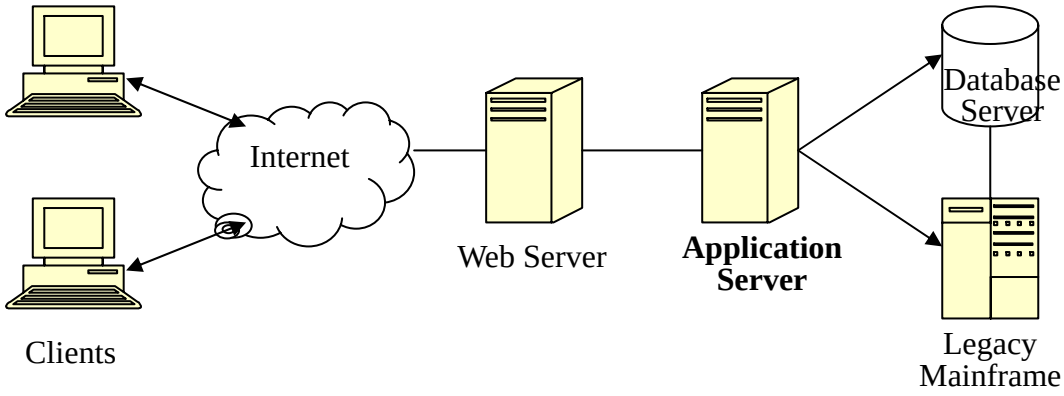

圖 6：應用伺服器部署示意圖

根據 Feiler 的分類，他把應用伺服器分為四類[5]：

1. 與作業系統(Operating Systems)綁在一起：以微軟為代表。微軟的應用伺服器與 Windows NT 一起搭配銷售已有多年；其應用伺服器包含可以建立

Web-based 應用程式(ASP, Active Server Pages)、支援交易管理(MTS, Microsoft Transaction Server)、獨家的元件標準(COM/DCOM)、資料庫連結池(Database Connection Pooling)及負載平衡(Load Balancing)等技術。而在資料庫軟體的選擇上則較為彈性。

2. 整合式(Integrated)：應用伺服器與網頁伺服器或與資料庫整合在一起。例如：FileMaker Pro 即是與網頁伺服器綁在一起，稱為 FileMaker Web Companion。如此的方式，對網頁伺服器或資料庫軟體的選擇性較無彈性。
3. plug-in 式：有點像整合式的應用伺服器，但此種方式把應用伺服器的功能包裝成元件，可以嵌入(plug into)網頁伺服器中。例如：Allaire 公司的 Cold Fusion 便屬於此類。
4. 獨立式(Standalone)：可以自由選擇網頁伺服器及資料庫軟體，彈性較大；但相對的，軟體之間的運作配合程度就不如上述三種方式。

2.2 多層式系統架構

目前最常使用之多層式架構可先概分為三層式及四層式二種架構，再依實作時使用之技術之區別，可再細分成四種，說明如下：[11][23]

1.三層式架構

實作時，使用者端之程式是由 Java 或 C++等語言所撰寫，並非網頁程式；且使用者端是透過 IIOP(Internet Inter-ROB Protocol)等通訊協定與應用伺服器溝通。就會形成如圖 7 的三層式架構。

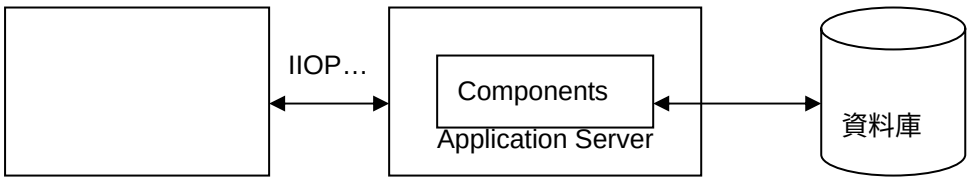

圖 7：三層式架構示意圖

若希望使用者端只需網頁瀏覽器便可讀取解譯 HTML/XML 等網頁語言，並且是透過 HTTP 等通訊協定與網頁伺服器溝通；亦即並無應用伺服器的角色，而由網頁伺服器直接與資料庫溝通，則形成如圖 8 之 Web-Based 三層式系統架構。

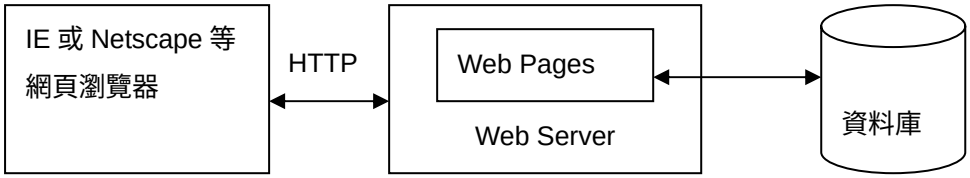

圖 8：Web-Based 架構示意圖

2.四層式架構

從原先 Web-Based 架構加入應用伺服器的角色，網頁伺服器是透過 IIOP 等通訊協定與應用伺服器溝通，則形成 Thin Client 多層式架構(圖 9)。

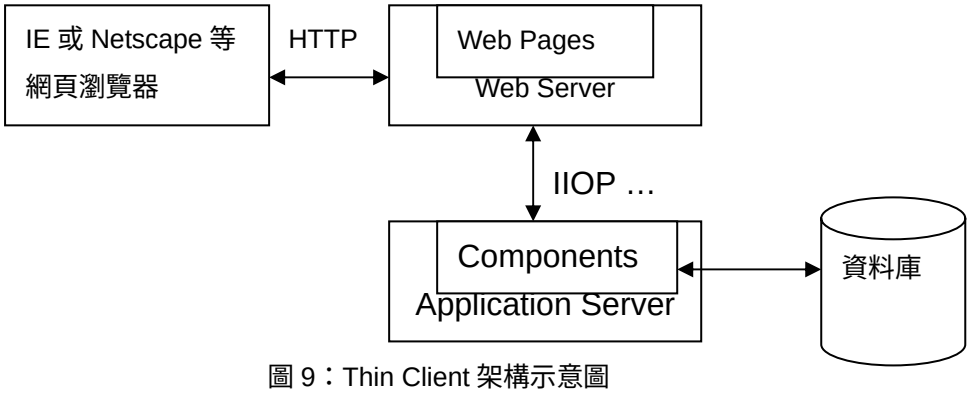

圖 9：Thin Client 架構示意圖

從 Thin Client 架構中再延伸，若使用者端是透過 HTTP 通訊協定從網頁伺服器下載 Java Applet 程式至前端，使用者端透過 Java Applet 程式以 IIOP 等通訊協定與應用伺服器溝通，如此的作法就形成 Java Applet Web-Based 架構(圖 10)。

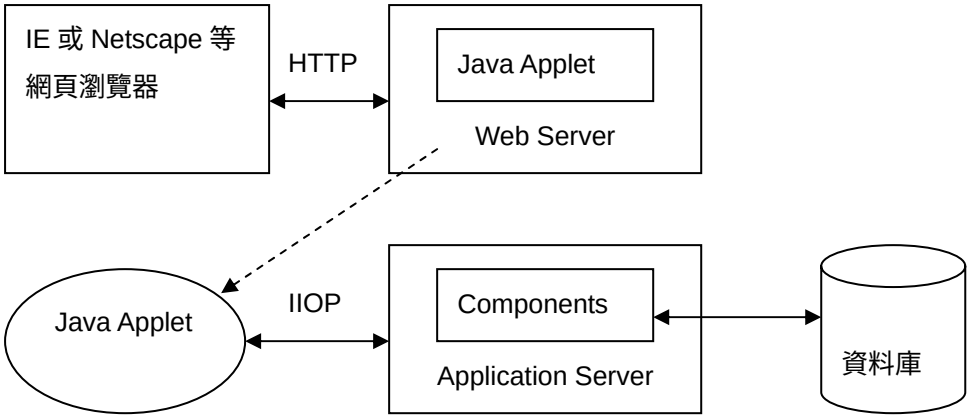

圖 10：Java Applet Web-Based 架構示意圖

2.3 多層式架構的派工法則

在多層式架構中，為了服務為數眾多的使用者，只以單獨一台伺服器恐無法負荷，通常會以多台伺服器共同服務的方式來解決；而在此架構下需要派工法則的位置有二：一是網頁伺服器、另一是應用伺服器。而多台伺服器的部署可以區分以下二種方式[11]：

1.每一台伺服器提供相同的功能，使用者端可以任選其一服務。稱之為叢集技術(Clustering)。

2.每一台伺服器提供各自不同的特有功能，使用者的需求會依照服務需求的順序，一一透過不同伺服器來完成。稱之為序列技術(Serializing)。

目前市面上之應用伺服器多數採用叢集技術，而其採用之派工法則可以概分為靜態(Static)及動態(Dynamic)兩種。以 Sybase 公司的應用伺服器產品 EAServer 所提供之派工法則有四種[16]：(前三種為靜態，第四種為動態)

1.隨機法(Random)：隨機分配工作，並未考量負載平衡的問題。

2.輪流法(Round-robin)：輪流分配工作給每一台應用伺服器，最簡單的負載平衡方法，容易實作、效能通常不佳。

3.權重法(Weighted)：依據應用伺服器的特性，事先指定其負載的權重；仍屬靜

態的負載平衡觀念，指定每台應用伺服器的權重後，並無法動態地變更。

4.調適法(Adaptive)：根據負載量度(load metrics)動態分派工作。其衡量負載量度的參數有三項：CPU 的使用率(CPU Utilization)、IIOP 連線數(IIOP Connections)及回應時間(Method Response Time)。將此三項參數透過函數計算得出負載清單(Load List)，表示每台應用伺服器的相對負載量度(load metrics)，以做為分派工作的原則。每隔一段時間(應用伺服器管理者可以自行設定)，擔任工作分派者的應用伺服器會重新計算負載清單，以做為此段時間區間的工作分派原則。

微軟在網頁伺服器叢集所採用負載平衡的派工法則稱為網路負載平衡(Network Load Balancing, NLB)。而在應用伺服器叢集所採用的是動態的負載平衡技術，稱之為元件負載平衡(Component Load Balancing, CLB)[4]；此技術的原理是所有使用者端的工作要求會導引至一台 CLB 伺服器，而 CLB 會追蹤並記錄叢集中所有應用伺服器的負載情形，當收到使用者端的建立物件的需求時，CLB 伺服器會將此需求轉送至最空閒的伺服器上。而對於計算負載的參數，則有提供 CPU Load 及 Method-timing 等函式庫供程式呼叫使用。

Teo 等人[21]在探討 1998 年世界杯足球賽所使用的 Web-based 應用程式之負載平衡策略時，討論比較三種派工法則：

1.最小負載法(Least Load)：假設可以得知每台伺服器實際且正確的負載量，並以此為分派工作的原則。但由於負載量及回應時間通常無法從伺服器的幾項參數中精確的計算出來，所以，此項方法在實務上並不可行。在研究上，通常將此種方法做為比較其他派工法則優劣的基準(Benchmark or Baseline)。

2.最少連線法(Least Connections)：計算連線數最少的伺服器，表示其負載最輕，便可將接下來的工作需求分派至該伺服器處理。實務上較最小負載法可行，效能表現也頗佳。

3.輪流法(Round-robin)：簡單容易的派工法則，效能通常不佳。當每項需求的服務時間差異不太時，此法的效能不致於太差；但若各項需求之間的服務時間差

異加大時，此法的效能就逐漸變差。