私 立 東 海 大 學

資訊工程與科學所

碩 士 論 文

指導教授：朱正忠 博士

重整舊有之 Web-based Java 系統至以 J2EE 架構為基礎的可延展性系統

Re-engineering Legacy Web-based Java Systems to J2EE-based Scalable Systems

研 究 生：潘義清

中華民國九十一年六月

摘　要

隨著 Internet 及 WWW 的流行，各行各業的電腦化及自動化亦是蓬勃發展，Web-based 應用程式更是企業快速、即時提供客戶服務的絕佳選擇。然而市場競爭激烈，行銷策略異動頻繁，復因上網人口激增，企業為了維持快速回應客戶連線需求的情況之下，必須對原系統進行整合、升級。然而，在該軟體系統缺乏良好的架構規劃之下，往往因此付出了相當的成本，甚至原本耗時費資所開發出的系統因而被迫提早除役。

軟體重整的技術，由舊有系統中截取有用資訊進行軟體再造，不僅降低系統開發成本，又能因應企業市場變動需求。因此如何將舊有 Legacy System 重整(Re-engineering)至一個具有高可靠性、易維護性，又能方便、彈性地依企業策略而調整服務內容的可延展性系統，是一項迫切的需求。

由於採用『以元件為基礎的多層式軟體架構』開發軟體，已是現今大型軟體開發的主流模式，再加上 Java 在分散式環境及跨平台上的優點。因此，本論文採用 J2EE(Java2 Enterprise Edition) 中的伺服器端元件化標準模型，並使用其多層式架構作為重整後的目標架構。

論文中以一個『線上圖書訂購管理系統』為例，利用我們所提出的標準程序，由不具結構化與擴充性的 JSP 與 Java Beans 程式中，萃取出有用的資訊，再加以轉換處理成 J2EE 多層式架構中相對應的 J2EE 元件。由於這些元件是遵循 J2EE 標準而進行開發，因此，可依其功能與角色而分布於 J2EE 的 N-tier architecture 中，並與 J2EE 之 Container 進行互動。本論文採用 Orion 公司所研發出的 Orion Application Server[36] 為系統平台，透由其 J2EE Container 管理應用程式在安全性、同時共用、永續儲存、以及交易等系統層級的問題，並研究 J2EE 在分散式計算的技術，以協助建置一個具有元件式負載平衡機制，以及高可靠性、擴充性與彈性的可延展性系統。

關鍵詞：可延展性系統 、重整工程、逆向工程、順向工程、設計樣版、多層式軟體架構

Abstract

With the fashion of Internet and WWW, computerizing and automation in all kinds of industry are prospering. ” Web-base Application “ is a wonderful choice for enterprise to provide service quickly and instantly. However, because of the keen competition among market, marketing strategy is changeable. Also with the increasing population of using Internet so rapidly, enterprise must to integrate and upgrade the original system for maintaining quickly response to customers.

However, in the situation of lack well-designed software architecture, it constantly costs much, even results of the original system that developed cost much time and expense be compelled to eliminate early.

The technology of software re-engineering extracts the useful information from Legacy System to re-manufacture the software. It not only reduces the system development cost but also can satisfy the requirement for according the variable market. Consequently, how to re-engineer the Legacy System to a high reliability, maintainability Scalable System that can adjust the service content conveniently and flexibly is an urgent requirement.

Owing to adapt the “Component-based and N-tier Software Architecture” to develop the software has become the main stream in large-scale software development. It also because of the advantages of Java in distribute environment and across-platform, the thesis adapt the J2EE’s (Java2 Enterprise Edition) “Sever-side Component Standard model ” and utilize its N-tier architecture to be the target architecture for re-engineering.

The thesis takes ” On-line book order management system “ for example, utilizing the standard procedure addressed by us to extract the useful information from the un-organized and inexpansible program of JSP and Java Beans, then transform to the corresponding J2EE component in J2EE N-tier Architecture. Due to these components are developed following the J2EE standard, so they can distributed in the

N-tier architecture of J2EE according to their functions and roles, and interact with J2EE container. And utilize the J2EE container to manage the system-level problems of application in security, concurrence access, persistence store, and data transaction. Besides, we study the technology of J2EE in distributed computing , to help build a scalable system that has component load balance mechanism and high reliability、expansibility and flexibly.

Keywords: *Scalable system, J2EE, Re-engineering, Reverse Engineering, Forward Engineering, Scalable system, Design Pattern, N-tier software architecture*

章　節　目　錄

圖 表 目 錄

第一章 緒論

1.1 前言

程式設計技術與軟體工程是開發出一個高品質應用程式所不可或缺的兩大必備知識。傳統的程式設計本只是軟體發展過程中之設計與實作階段的技術，然而，隨著軟體技術的不斷演進，隨著 Rumbaugh 等人提出物件導向模型技術(Object-Oriented Modeling Technique)之後[15]，而有 UML、設計樣板(Design Pattern) 、框架(Framework)等技術的出現 。物件導向技術不但提昇軟體開發的速度、降低成本，更能得到較高的軟體品質[22][23]。而軟體工程是以達到符合成本效益的目標為原則來開發軟體系統[24]，可見物件導向技術在軟體工程中地位之重要。也因為程式設計在開發、維護、測試、擴充等問題，也使得諸如需求工程(Requirement Engineering)、重整工程[24]等方面的發展日益受到重視 。

傳統程序導向的設計方式以功能為主，程式設計師所關心的問題是"程式中會有哪些功能，要如何設計出模組化的程式"，然而，這樣的設計方式卻導致相依性過高，模組新增或更新不易的缺點。然而，物件導向程式設計(OOP)中，是以物件為主，程式設計師所關心的是 "一個系統需要哪些物件來組成 "，強調以資料為主的設計方式，透過 OOP 的資料封裝、繼承重用的技巧，寫程式就像是組裝 IC 電路一樣，只要抽換不同的元件就能產生新的系統，這使得系統的維護更具有彈性和擴充能力[18]。

物件導向設計技術的興起之後，不僅改進了許多傳統程序式導向設計的缺點，伴隨著而來的更是以"物件導向技術"為中心而衍生出的一系列理論與應用，如 Component-based software development[24] , CORBA[28][29] 等分散式物件相關的技術。

而在所有物件導向程式語言之中，Java 更是異軍突起，以其在語言特性上的優點，發展出一系列的技術，並以 API 套件的方式提供程式開發人員相當多的便利，不僅在圖形化介面、2D 處理、分散式運用上頗受好評，在網站應用程

式的開發上所提供的元件技術更廣受業界喜愛[19]。時至今日，以 Java 發展應用程式更是日益普及。

然而，隨著軟體使用率普遍增加，需求變動日益頻繁，人們發現舊有系統在開發時期的疏失，將導致日後擴充、整合不易的問題[3]。即使是同樣以 Java 語言開發的系統，由於規模大小、設計考量、技術演進、需求擴大等因素，也面臨升級擴充等問題。至此，軟體的維護性才被視為軟體的生命週期中不可或缺的一環。而軟體重整工程之技術，在可以降低升級成本、因應需求之情況下，成為一個極佳的替代方案[24]。

1.2 研究動機

Java 流行以後，企業間採用 Java 開發應用程式的情形日益普遍，或許因為應用程式伺服器的價格考量以及 J2EE 技術的門檻，使得許多企業網站仍只採用網站伺服器(Web Server) 來建構其應用程式。然而，電子商務日益蓬勃，網站內容與服務也日益多元，在網站擴充不易，J2EE 技術日益成熟的情況之下，以應用程式伺服器來作為企業網站平台，已成企業提供一個高效率又不易中斷服務的絕佳方案。

因此本論文的研究動機 ，在了解 Java 在網站伺服器開發上的一系列技術，並透由軟體工程的技術重整系統而得到一個高維護性的元件導向軟體系統，以提昇效能、降低升級成本。

1.3 研究目的

有鑑於上述的動機，本論之研究目的主要有二：

1. 提出一套標準程序並建立對應方法以重整、轉換舊有 Java 網站應用程式

系統，至以元件為基礎的 J2EE 多層式架構 (N-tier Component-based Architecture) 系統，以提昇系統功能、彈性，減低維護成本。

2. 研究 J2EE 元件的相關技術，以建立一個易於維護、擴充、再使用的可延展性系統。

1.4 章節安排

本論文之架構如下：第一章為序論，第二章介紹背景知識與相關研究，第三章介紹重整的流程與方法，第四章為結論及未來工作。

第二章 背景知識與相關研究

2.1 軟體重整工程(Software Re-engineering)

如 Sommerville 在其"Software Engineering"一書中所述[24]，許多公司或組織之中，仍存在使用多年、功能及效率上已經不敷需求的舊有軟體系統(Legacy System) 。這些系統常沒有完整的規格(specification)，缺乏系統文件，甚至是利用過時的語言所開發完成。雖然有這麼多的問題，但因為重新開發的昂貴成本以及不能確保新系統能夠完全取代舊系統完整的功能，因此，被迫持續沿用 Legacy System 來提供重要的商業服務，這也導致所花費的時間與成本較原先開發時高出許多[4]。倚賴 Legacy System 的公司，必須在有限的預算之下選擇最符合商業利益的策略來處理這些 Legacy System[24]。而在 Warren 的文章[31]中，認為軟體變更(Software change)有幾種不同的策略: 軟體維護(Software maintenance)、架構轉換(Architecture transformation)、軟體重整(Software re-engineering)等。其中，採用軟體重整之方法，對系統發展(system evolution)而言，具有底下兩項主要好處[24]:

- 降低風險 (Reduced risk)　重新開發系統具有高風險，可能會在系統發展階段產生錯誤，而導致新系統無法順利取代舊系統的功能而運作。
- 減少成本 (Reduced cost)　進行重整所耗費的成本遠比重新開發新系統來的低。Ulrich 指出一個商業系統的例子，說明重整系統比重新改寫系統減少了約四倍的成本[32]。

本論文採用軟體重整之策略，重整 Legacy System 以提昇軟體系統的可讀性及可維護性。依照處理的方式與流程，可將重整工程分為兩階段:

- 逆向工程(Reverse Engineering)階段

逆向工程階段的目的是由舊系統中之原始程式碼，利用一連串的分析與處理而取得有用的資訊。

- **順向工程**(Forward Engineering)**階段**

  順向工程階段對逆向工程中所得到的資訊進行解析處理以產生新的程式碼。

2.2 可延展性系統(Scalable System)

Scalable System **具備下列幾項特性：**1.**開放性**—**能與既有之軟體平台和應用程式整合。**2.**擴張性**—**可延伸使用者介面和應用程式特徵。**3.**維護性**—**容易升級及發展。**4.**模組化**—**允許元件可嵌入及不同元件可互相組合、模組可以隨時加入運作或移除和模組容易擴充** 5.**分散式。

在本論文中的可延展性系統之定義為: **一個支援多層式架構的系統，在網站連線需求擴大時，能夠不需修改程式碼即可將層級中之元件分散到不同伺服器上運作的系統，即是能提供元件化負載平衡的系統。**

負載平衡(Load Balancing)**可以分為網路式負載平衡**(Network load balance)**和元件式負載平衡**(Component load balance)**兩種方式。網路式負載平衡主要是利用硬體設備與網路環境的搭配來作負載平衡。目前常用的轉發機制（**Traffic Forward Mechanism**）有** Virtual Server via NAT (Network Address Translation)、Virtual Server via IP Tunneling、Virtual Server via Direct Routing **等三種。另外也有四種演算法來作負載分配**: **輪流排程** Round-Robin Scheduling (RRS)、**加權輪流排程** Weighted Round-Robin Scheduling (WRRS) 、**最小連結數排程** Least-Connection Scheduling (LCS) 、**加權最小連接數排程** Weighted Least-Connection Scheduling (WLCS)**等。本論文主要是採用** J2EE **應用程式伺服器所提供之元件式負載平衡技術，介紹如後。**

- **元件式負載平衡(Component Load balance)**

現有許多應用程式伺服器提供元件負載平衡功能如: BEA **的** WebLogic**，** Orion **的** OrionServer , MicroSoft **的** Application Center 2000 **等。**

- **Cluster EJBs**

Cluster **技術是大規模的應用程式系統為了確保連線服務所普遍採用的方式， 以一種鬆散藕合的方式來結合許多伺服器共同提供服務，對客戶端而言，就如同是由一個伺服器提供服務一般 。架構** J2EE **伺服器的** cluster **系統可以有兩種選擇方式**[37]:

(1) 三層式架構 Web server component (servlets **及** JSPs)**和** application server component (EJBs) **在同一個** process **中執行。**

(2) 四層式架構 Web server component **和** Application server component **各在兩個不同的** process **中執行。**

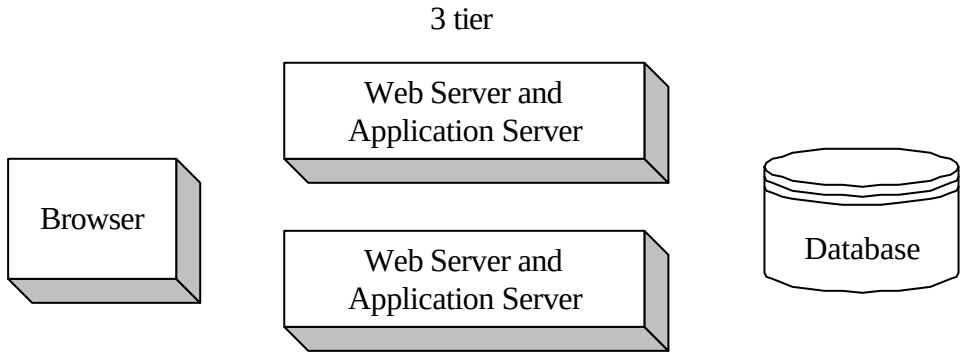


圖 2.1 **三層式** Cluster **架構**

4 tier

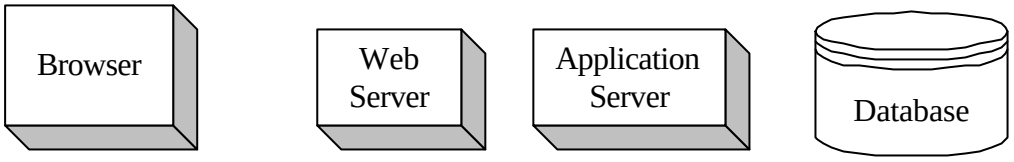


圖 2.2 四層式 Cluster 架構

- **使用 EJB 進行 Cluster**

  J2EE 伺服器廠商可以在幾個地方提供 cluster 機制的支援：

  - **JNDI driver**：可以透過 JNDI driver 找尋 home object 的機制來進行負載平衡或錯誤回復機制。
  - **Container**：可以直接透由 container 使用 interserver communication protocol 的方式來執行 cluster 的運作。
  - **Home Stub**：這是第一個被遠端客戶端所存取的第一個物件，在客戶端的虛擬機器中執行，由於它是由廠商的伺服器軟體自動產生的，因此廠商可以直接透過此 stub 來提供 load balancing 和 fail-over 的機制。
  - **Remote Stub**：廠商可以如利用 home stub 般的方式進行 load banlancing 和 fail-over 的機制，但是必須要小心避免處理 request 時影響到系統。

2.3 軟體架構(Software Architecture)

小至程式間的函式架構，大如系統之軟體架構，甚至跨系統整合之軟體架構，都需要講究軟體架構之設計。在[27]中，認為物件導向系統中的軟體是由許多子系統(subsystem)所組合成的，而軟體架構(Software architecture)則定義了那些透由介面進行互動的子系統間之靜態組織關係。軟體架構對於系統的發展和維護是很

重要的，採用了適當的軟體架構，可以使軟體工程師的設計與實作更有效率也更能預見成效;一個好的軟體架構，可以使得系統中的元件和整個應用程式系統隨著時間經過而能優雅不紊亂的發展著[27]。另外，kruchten 也提出了五種主要的系統觀點來以提供發展軟體架構時作為參考[11]。

- **軟體架構(Software Architecture)之探討**

  - **以元件為基礎的軟體架構**(Component based software architecture)

在 Sommervill **的書中指出**[24]，**以元件為基礎發展軟體**(Component-based software development)**的理論，於** 1990 **年代末開端，強調以重複使用的方式來發展軟體系統，其動機乃是為了加強物件導向發展的『重複使用性』。在**[9]**中指出元件是組成軟體系統的基本單位，並透由其介面而定義元件間交互運作的函式。因此，介面好比是元件對外溝通的窗口，提供管道以供其他元件使用該元件的功能。**

元件比物件類別更為抽象獨立，Meyer **更針對元件定義了五種層次的抽象**[25]: **Functional abstraction – 這類的元件實作了一個單一的功能，如進行數學運算。Casual groupings – 這類元件可能是一群關係鬆散的資料宣告及函式。Data abstractions – 這類元件代表一種物件導向語言中資料的抽象表示或是代表一種類別。Cluster abstractions – 這種元件是一群相關類別的集合，有時也被稱為框架**(Framework)**。System abstraction –這類元件是一個完全自足的系統，透過** API(Application Programming Interface)**作為存取系統的介面。因此，以元件為基礎之軟體架構，具有下列之優點**:

(1) **模組化 ：可以把企業系統想成由許多伺服器元件構成的集合，不同的元件代表了諸如顧客、商品等概念的模組，可以使系統的開發較具彈性。**

(2) **結構化 ：由於以元件為基礎，因此元件間的組合與互動具有一定的規則，因此架構分明容易維護。**

➢ **多層式軟體架構(N-tier Software Architecture)**

由於企業應用程式系統往往具有更高的複雜度與處理流程，因此，Steiert **在**[13]**指出，必須使用多層式架構來彌補** two-tier **或** three-tier **架構之不足。而** Aizman **也提出使用** N-tier **來建構網路管理系統的架構，可以使系統的整體彈性大幅提昇**[14]**。多層式軟體架構，在** Jacobson[27]**等人的書中稱之為** Layered architecture**，並定義其為一個以層級來組織軟體的軟體架構，越上的層級越不具有共通性，亦即越能表現出所屬應用程式之特性。其所提出的** Layered architecture **層級如下圖，這些層級的數量、名稱和內容並不是固定的，是依據軟體工程師的需求而自行規劃設計，簡介各層級如下**:

- **應用程式系統層(application system layer)　此層中的應用程式系統可以直接透過介面進行內部溝通，或是也能間接透過較低層的某些物件或服務來進行，如** ORB(Object Request Broker)**，作業系統**(operation system)**。**
- **特定商業應用層(business-specific layer)　包含了在許多不同的商業應用領域中之所使用的元件系統**(component system)**，此層的元件建構在中介層之上，並提供應用程式層發展的基礎。**
- **中介層(middleware layer)　中介層提供了公用類別和平台獨立的服務，如異質環境的分散式計算。**ORBs **及** OLE **元件即屬於此層。有了此層所提供的元件，軟體工程師可以集中心力於應用程式的開發。**
- **系統軟體層(system software layer)　此層包含了進行運算以及網路等基礎架構**(infrastructure) **相關的軟體，如作業系統或是與硬體溝通的軟體。**

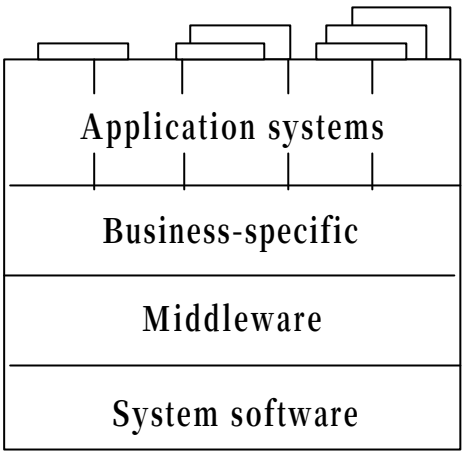


圖 2.3 **典型的** Layered architecture

將軟體架構分層可以降低軟體的相依性(dependency) **，各層級間可以透過介面來降低溝通成本**[27]**，並具有下列之優點**:

(1) **降低元件模組間之偶合力(Coupling) ： 元件模組間之偶合力越小，越容易抽換元件，因此系統越有彈性。**

(2) **提昇元件模組內部之內聚力(Cohesion) :元件模組內之內聚力越強，越容易進行，元件的粒子性越明顯也越易分工。**

- **分散式系統架構(Distributed System Architecture)**

在[24]**中，將『系統』分為三類:個人式系統(Personal systems) - 被設計為只能在單一個人電腦或工作站上執行的系統。嵌入式系統(Embedded systems) –在一顆或經整合的一群處理器上執行的系統。分散式系統(distributed system) –是指一個將資訊處理工作分散於幾部透過網路而鬆散整合在一起的電腦之上執行的系統。**Coulouris **等**[26]**指出分散式系統具有六大重要特徵**:

(1) **資源共享(Resource sharing) 分散式系統允許共享網路中的軟硬體資源，並透過一部電腦進行中央控管以提供多使用者的存取服務。**

(2) **開放性(Openness) 系統的開放性是指系統可以加入新的資源以延伸系**

統功能，分散式系統即是一個包含了許多來自不同廠商的軟硬體的開放性系統。

(3) **一致性(Concurrency)　在分散式系統中，幾個流程**(process)**可能同時在網路上的不同電腦上運作。**

(4) **可延展性(Scalability)　分散式系統中可以加入新的資源來處理系統的新需求。**

(5) **容錯(Fault tolerance)　透過分散環境中幾部電腦的合作來處理錯誤狀況以避免導致整個系統因而停擺。**

(6) **通透性(Transparency)　分散式系統的位置通透性，使用戶端不需要知道提供服務之物件的實際位置即能存取資源。**

為了要達到上述特性，必須要選擇適合的系統架構。主從式架構(Client-server architecture)**，在系統中區分為提供服務的** server **與要求服務的** client **兩角色，**client **必須知道** server **所提供的服務並知道如何跟** server **溝通。最簡單的** client-server **架構為** two-tier client-server **架構，依** client **及** server **的工作量分配而可分為** Thin-client model **及** Fat-client model [24]**。而在分散式物件架構** (Distributed object architecture)**中，透由元件的介面來提供服務，並沒有** client **和** server **的邏輯性區分，分散式物件也可以透由** middleware **來進行溝通。採用分散式物件架構允許系統設計人員延緩決定提供服務的主機與方式，而且系統更開放、更有彈性，也可以透由物件的移動**(migrate)**而能動態地重組**(reconfigure)**系統**[24]**。**

2.4　J2EE 與其伺服器端元件模型

- **J2EE 簡介**

J2EE (Java 2 Platform, Enterprise Edition) **是** Sun **公司於** 1999 **年六月所公佈的一項新架構**[21]**。**J2EE **的來源是整合分散於不同套件**(package)**中的** API**，在**

J2EE Blueprints(J2EE 藍圖)中有詳細的定義[19]。在 J2EE 中包含了下列 API :

- Java Server Pages (JSP)
- Java Servlet
- Enterprise JavaBeans (EJB)
- Java Database Connection (JDBC)
- Java Transaction API (JTA) 與 Java Transaction Service (JTS)
- Java Naming and Directory Interface (JNDI)
- Java Message Service (JMS)
- Java IDL 與 Remote Method Invocation (RMI)
- Java XML

其中 JSP、Servlet、EJB 皆代表一種元件技術(Component Technology)，J2EE 的文件中說明了所有的文件都是由容器(Container)來控管並提供元件的執行期環境(Run Time Environment)。

- **J2EE Container**

元件模型是一組(封裝在 Java package 中)擁有特定功能的類別及物件，用來區分其他具備不同功能的類別及介面[19]。一旦完成元件的定義，元件便成為一個獨立的軟體，可以在不同的應用中使用。J2EE 的元件模型中包括了 Bean Class 、遠端介面、主介面、EJB object、EJB home 以及 EJB 伺服器彼此間的關係，適用於 CTM 的架構之下，分散式伺服器端元件的設計[19]。

J2EE 平台是透由 container 來進行元件管理，Container 可以透由呼叫

Callback method 而控管 Enterprise JavaBeans 元件的生命週期，以因應用戶端的請求及操作行為。J2EE container 在系統層面的提供了如:交易管理、資料庫連結等機制，當應用程式連線負荷過重時，也能透由 Web container 來自動執行負載平衡(load balancing) 。

- **Enterprise JavaBeans 的伺服器端元件模型**

  EJB 整合了伺服器端元件以及分散式物件的技術，以簡化企業應用系統的開發。EJB 伺服器將元件以分散式物件的形式來呈現，並提供企業應用系統在安全性(Security)、同時共用(Concurrency)、永續儲存(Persistence)、以及完整交易(Transaction integrity)上的管理機制[19]。而 EJB 元件(Enterprise JavaBeans)也可以在遵循 J2EE 規格的不同元件交易監控伺服器(Component Transaction Monitor, CTM)上開發、使用，因此元件的開發、部署(deploy)都能擁有更大的彈性與選擇。使用 Enterprise JavaBeans 的平台可以簡化系統開發的工作，使得程式設計師可以專注在企業應用目的 (business purpose) 上，而提昇設計程式的效率[21]。

- **MVC 架構**

MVC 架構最早在 Smalltalk-80 語言中被使用來處理使用者介面[16] ，之後 MVC 架構成為一個被廣泛運用來處理使用者介面的框架(framework)[17]。在[27]中也提到 Java Swing 的使用者介面即是以 MVC 架構為基礎的。使用 MVC 的做法是將軟體元件分為 Model (模型)、View(顯像) 、Controller(控制者)等三個獨立單元[20]。而伺服器的應用程式，可以被歸納為幾個部分:商業邏輯(Business Logic)、外觀顯示(Presentation)與請求處理(request process)三部份，『商業邏輯』是指對應用程式資料(如客戶資料、訂單)的操作與應用，『外觀顯示』是指將應用程式以圖形化介面顯示在使用者面前，而『請求處理』則是在外觀顯示與商業邏輯間扮演聯繫的角色。在[20]中也提到一個 EJB-based 的應用程式，也必須要遵守 MVC 的原則，才能讓應用程式易於擴充和維護。

2.5 設計樣版(Design Patterns)

- **設計樣版(Design Pattern)**

樣板(Patterns)是一種描述事情的共通方法，透由對標準模式的認同，而得以傳授經驗，增進彼此的標準[6]。而設計樣板則是在程式設計時可以透由導入設計樣板而得以提昇軟體的設計品質，禰補思慮不周的疏失。由於設計樣板收錄專家們在許多領域的軟體開發過程中，針對重複出現的特定問題所使用的優良設計或解決經驗，並以易於接受、溝通的方式呈現出來。透過樣板的應用可以使程式的撰寫依據優良的設計或開發模式，熟悉 pattern 的軟體設計師們可以很快的得之應用程式的基本架構及目的，因此，Pattern 成了可以加速開發提昇軟體品質的基本建構單位(building block)[12] ，也使得程式架構在可讀性或維護性都能有更好的表現。熟悉越多的 pattern 越能夠快速地使用不同領域的專家知識解決問題並設計高品質的軟體。自從 The Gang of Four [6]，收錄了 23 個 Design Pattern，並將設計樣板分為三大類: **Creational Patterns**、**Structure Patterns**、**Behavior Patterns** 之後，設計樣板的應用與相關研究便成為炙手可熱的焦點，更有導入 pattern 於重整工程中[5][7]之研究。

2.6 UML (Unified Modeling Language)

由於過去開發軟體缺乏標準化且易懂的溝通模式，因此往往很難建立在軟體開發流程中執掌不同職務的人員間建立共識，也因此許多軟體在時間壓力下，往往缺乏良好的系統架構設計，描述系統的相關文件更是不足。也因此導致後續的維護與擴充產生許多後遺症[1][2]。

UML 的產生解決了上述問題，透由一套標準化的語言提供軟體生命週期中不同階段之發展人員以圖形化的方式建立架構和文件以提供一個溝通標準。UML 於 1997 年被提出，主要是整合了 Rumbaugh 、Jacobson、和 Booch 等三位大師所提之方法及其他方法論而得，使用九種圖以視覺化的方式來描述系統，

並使用五種構面(view)來描述軟體架構[2]。UML 中的這些圖形可以依據發展人員的需求來使用。如 UML 圖形中最常用到的類別圖(Class Diagram)，可用來描述一組類別、介面間的合作關係。 而藉由使用者案例圖(Use Case Diagram)藉由一組 use case 和 actors 來組織化(organizing)、圖形化(modeling)系統的行為。當要描述物件間的互動關係時，則可以使用循序圖(Sequence Diagram)或合作圖(Collaboration diagarm)。

目前支援 UML 的廠商與團體紛紛推出許多的 tool，除了最著名的 Rational 的 ROSE (Rational Object Software Engineering)[30]之外， TogetherSoft[33] 與 VisualUML[34]也推出了 UML 的 CASE tool，Microsoft Visio 2000[35]也支援 UML 的 diagram 繪製模組。

第三章 重整流程

為了解決舊有系統維護不易，擴充困難等問題，本論文從技術層次及系統架構的兩方面來進行改善。敘述如下:

- 引用新技術　原始系統使用 JSP 與 Java Bean 元件的技術來開發網站，並不能滿足企業策略快速變動以及網站持續擴充的需求。因此 ，我們採用 J2EE 的技術來建構大型網站，使元件的開發依循 J2EE 中定義的標準規格，而能利用廠商所開發的 J2EE 應用程式伺服器，滿足企業網站多元化服務且高可靠度的需求。

- 改善系統架構 在此我們採用 J2EE 多層式軟體架構來改善原始系統的架構 ，使依循元件依其功能而分散於不同層級間分工合作。另外，透由對新舊系統之 MVC 架構的探討，而能得知新舊系統在架構上的優劣。如圖 3.7、圖 3.8。

3.1 相關元件介紹

在本章中將先針對重整流程中的三大類主要元件: JSP 元件、Java Bean 元件以及 EJB 元件進行介紹，以由其基本規則中分析出可用以協助重整工程的特性。也透由了解 JSP、JavaBean、Enterprise JavaBeans 程式之運作機制，以利於逆向工程中資訊之截取，及順向工程中元件的生成與佈署。

3.1.1 JSP 元件

在 J2EE 的伺服器端元件標準模型中，Web container 中的 Web component 可分為 JSP 元件及 Servlet 元件，由於本論文所提的轉換方法並沒有使用 Servlet 元件，故僅作簡介如下。

- **Servlet**

在[18]中提到 Servlet 元件是一個執行於伺服器端的 Java 類別檔，Servlet Engine 接收用戶端的請求之後便會將其載入，並為 Servlet 建立一個執行緒以服務請求。之後，Servlet 便能夠隨時處於提供服務的狀態，一旦又有來自用戶端的請求，不必再經過載入即能執行。Servlet 的命名有(Server Side Applet)之意，可以如同 Applet 一樣在 Java 的 Security Manager 安全機制之下，執行在『Secure Sandbox』的保護模型之中，以避免隨意對系統資源進行存取。執行效率及安全性防護即是 Servlet 程式強於傳統 CGI 程式之所在。

- **Servlet 與 JSP**

JSP 為 Java Server Page 的簡稱，不但具有 Servlet 的所有特性及優點，更增加了程式開發上的彈性，可以跨平台，更擁有直接將程式碼嵌入於 HTML 網頁的功能，這使得網站應用程式(Web-based Application)的開發更為簡便[20]。JSP 程式完全是架構於 Servlet 程式之上，先經 JSP Engine 將其轉換為對應的 Servlet 程式碼，在將其編譯成類別檔後載入執行。之後便與 Servlet 的運作模式一樣，也就是說，只有在用戶端是第一次請求 JSP 網頁時，才會轉換、載入、執行 JSP 程式[18]。

- **JSP 組件**

JSP 組件包含了 JSP 規格中所定義的指令(directive)、程式(scripting)、動作(action)等三大組件，以及由程式設計師依需求所自訂的動作(custom actions)組件[20]。組件中包含了標籤與屬性，經由 JSP 的標籤與屬性可以設定動態網頁與伺服器間互動的資訊。可於 JSP 網頁中利用 JSP 組件而存取伺服器上的 Java Bean 元件。

3.1.2 Java Bean 元件

Java Bean (簡稱 Bean)是 Java 的一種元件結構，同時也是一種遵循特定規則的 Java 類別。一個標準 Java Bean 具有下列特性:

(1) 必須是宣告為公開的類別。

(2) 其建構式不可傳入參數。

(3) 定義了一組"get"類型的公開函式(getter)以提供外界擷取內部屬性值。

(4) 定義了一組"set"類型的公開函式(setter)以提供外界改變內部屬性值。

對 JSP 而言，Bean 是很好的資訊載體不僅封裝了許多資訊，還將一些資料處理的程序隱藏在 Bean 的內部。只要透過 setter 或 getter 便能協助 JSP 程式處理商業行為。因此，Java Bean 並不屬於分散式元件，在 J2EE 的多層式架構中 Java Bean 元件和 JSP 同屬於顯示層元件[20]。

- **Java Bean 與 JSP**

JSP 網頁可以透由<JSP:useBean>標籤來使用 Java Beans，而要設定或取得 Java Bean 的屬性則可分別透由 <JSP:setProperty>及<JSP:getProperty>標籤來完成。在這些標籤的包裝之下，許多複雜的流程可以轉交給 JSP 引擎去處理 ，因此可以將一些特定功能的程式邏輯用 Java Bean 的方式包裝起來，不僅可以縮減 JSP 程式碼提昇可讀性，更能提昇程式碼的再使用性。

- Java Bean 的生命週期

可以透由<jsp:useBean>標籤中的 scope 屬性來設定 Java Bean 的有效區域及生命週期，從作用上來區分， Bean 的 Scope 可以分為 Page、Request、Session、Application 等四種類型[18] 。

3.1.3 EJB 元件

Enterprise JavaBeans(簡稱 EJB) 是一個用 Java 語言寫成、封裝了應用程式商業邏輯的分散式伺服器端元件，利用商業函式(Business method)來實作那些商

業邏輯[19]。遠端的使用者可以透由呼叫這些 method 而存取應用程式所提供的服務。

- **採用 Enterprise Beans 的好處[21]**

在發展大型、分散式的應用程式時，採用 Enterprise Bean 的有許多好處:

(1) Bean 的開發人員可以集中心力在解決商業問題

因為 EJB container 提供了許多管理機制，來處理分散式物件與應用程式伺服器及系統等溝通的問題。另外，也提供了如安全授權、資料庫存取等系統層面(system-level)的服務。

(2)可使 Client 端的應用程式瘦身，有助於 client 在小型裝置上執行

因為 beans 包含了應用程式的商業邏輯，所以 Client 的發展者不需要去撰寫實作商業規則或是存取資料庫的程式。這些 Client 可以是輕薄短小的，也可以是各式各樣的應用程式、甚至能夠許多 Client 一起進行存取。

(3) Enterprise bean 的可攜式特性，方便應用程式的發展

應用程式的組合程式 (application assembler) 可以從存在的 bean 建立出新的應用程式，而這些應用程式可以在相容的 J2EE server 上執行。

(4)滿足應用程式的可擴充性(Scability)需求

當使用者的數量持續成長，必須要分散應用程式的元件到別的機器上時，就可以利用 Enterprise Bean 來滿足應用程式的擴充性(scalability)需求，以確保應用程式能繼續提供服務。

- **Enterprise Bean 的種類**

Enterprise Bean 可以依其功能與目的分為三類[21]：**Session Bean、Entity Bean 、Message Driven Bean**，如表格 3.1 所示。分別介紹於下。

Enterprise Bean 種類	目的	說明
Session Bean	代替 Client 於伺服器上執行商業行為。	依是否需維持對話狀而分為兩種類型: stateful 和 stateless 。
Entity Bean	代表一個存在一永續儲存體上的商業物件。	依永續狀態而區分為兩種類型: bean-managed 和 container-managed。
Message-Driven Bean	扮演一個 JMS 訊息聆聽者的角色。	可以非同步的方式處理訊息。

表 3.1 Enterprise JavaBeans 分類表

- **Entity Bean**

一個 entity bean 代表一個在一永續儲存體(persistent storage)上的商業物件(business object) 。如顧客、訂單、產品都算是一個商業物件。典型的 entity bean 的 instance，是對應到關聯式資料庫中某一個資料庫的欄位，因此增加或刪除 entity bean 會影響到資料庫中記錄(record)的增減。其特性簡述如下:

- **交易管理** 由於許多不同的 Client 可以共用同一個 Entity bean 實體，也因此每一個 Client 都有可能會去改變同樣的資料，因此，entity bean 必須使用交易管理(transaction management) 來處理共享存取(Shared Access)問題。一般來說，EJB container 都會提供交易管理，只要在 bean 的部署描述子中指定交易的相關屬性即可。

- **Primary Key** 每一個 entity bean 都有一個唯一的物件辨識子(object identifier)，通常叫做 primary key，可以透由它使 Client 找到特定的 entity bean。在 entity bean 的佈署時會由資料表格中的欄位指定 Primary Key。

- **Container Managed Persistence(CMP)** EJB container 會自動產生必須的資料庫存取呼叫，在執行時期自動的將 bean 的狀態與資料庫做同步，而且

在 bean 的程式碼中沒有資料庫存取的呼叫(SQL calls)。也因這樣的彈性，即使在使用不同資料庫的不同 J2EE 伺服器上重新佈署同樣的 entity bean，也不需要重新編譯或編輯 bean 的程式碼，簡而言之，就是 entity bean 因此更具有可攜性了。

- **Bean Managed Persistence (BMP)** 在 Bean managed persistence 的情況下，entity bean 中的程式碼必須要包含進行存取資料庫呼叫的程式碼，但卻也因此能使用 entity bean 對資料庫作更多樣的控制。
- **Session Bean**

被布署在伺服器上的一個 session bean 代表一個單一的 Client 在 J2ee server 中存取應用程式，用戶端呼叫 session bean 的 method，即能使 session bean 為它而執行工作，免去了解伺服器端複雜的處理流程。session bean，不能共享也不是一個永續的元件，當 client 結束時，session bean 隨即跟著結束不再與 client 有關聯。另外，簡介 Stateful Session Beans 及 Stateless Session Beans 如下:

- **Stateful Session Beans**

物件的狀態由它的實例變數(instance variable)的值所組成，在 stateful session bean 中，實例變數代表 Client 和 Bean 間一個唯一的 session。因為 client 和 bean 互動的互動模式，因此這狀態常被叫為(Conversational State)。這狀態會在 client-bean session 期間被保留，如果 client 移除掉 Bean 或是結束對應用程式的存取，這個 session 也就跟著結束，而且狀態消失。

- **Stateless Session Beans**

stateless session bean 不需要為特定的 client 維持對話狀態，因此比 stateful beans 有較好的執行效率。當一個 Client 呼叫一個 stateless bean 的函式， bean 的實例變數也有可能會包含一個狀態，但卻只限於呼叫期間，當該函式執行結束了， 狀態也就不再保留了。除了呼叫期間以外，所有 stateless bean 的實體(instances) 都是等效的 ，允許 EJB container 指定一個 instance 給任何一個 client。

- **Message-Driven Bean**

Message-driven bean 是一個允許 J2EE 應用程式能以非同步方式處理訊息的元件。它扮演一個 JMS (Java Message Service)訊息聆聽者的角色，這些訊息可以被任何 J2EE 元件如: 用戶端應用程式、網頁元件(Web component)或是另一個 enterprise bean 元件來傳送，也可以使用 JMS 應用程式或系統來傳送。由於目前進行重整的舊有系統並不具訊息服務的需求，因此，在本論文中並沒有使用到 Message-Driven Bean。

- **Enterprise JavaBeans 的組成**

Enterprise JavaBeans 不是一個單獨的物件，而是一由 Bean 類別與遠端介面及主介面的組合，不論是用戶端的程式或是在同一 Container 中的其他 EJB 元件也都必須透過 EJB 元件中的介面進行存取。Enterprise Beans 的組成包括了 Bean Class、Remote Interface 、 Home Interface 以及只有 entity bean 才有的 Primary Key Class ，簡介如下。

- **主介面（Home Interface)** 主介面中定義了所有和 bean 的生命週期相關的函式: 包括建立、移除及尋找 bean 等函式。
- **遠端介面(Remote Interface)** 遠端介面中定義了 bean 的商業函式(business method)，以提供用戶端進行呼叫。
- **Enterprise Bean 類別(Enterprise Bean Class)** Enterprise Bean 類別中實作了定義於遠端介面中的商業函式(business method)，也實作了與主介面中的函式相對應的函式。所以，當用戶端對遠端介面中的商業函式進行呼叫，實際上是被對應到 Bean 類別中的函式作處理的。
- **Primary Key 類別**(Primary Key Class)

Primary Key 類別的用途是給定每一個 entity bean 一個獨一無二的識別碼，可用以尋找 entity bean 並在與資料庫對應過程中發揮作用，因此，此類別被要求必須實作可序列化(Serializable)介面。

- **EJB 元件介面與 EJB Bean Class**

如下圖 3.1 所示，EJB 元件中 Bean Class 與遠端介面及主介面間具有對應關係。在 Bean Class 中具有和遠端介面及主介面中同名同參數的對應函式，而對應的動作是由 EJB Container 透由 EJB Object 及 EJB Home 而完成。

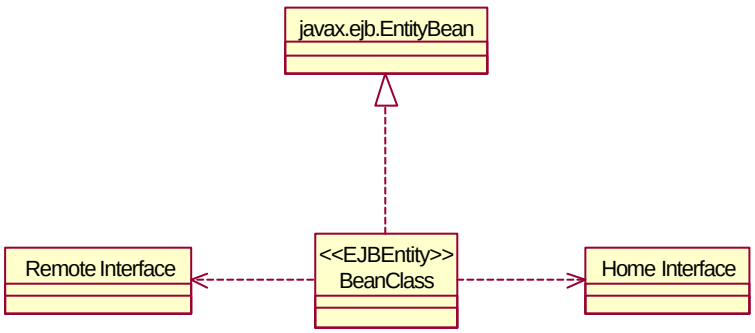


圖 3.1 EJB 元件介面與 EJB Bean Class 關係

- **EJB 元件介面與 Container**

遠端介面與主介面分別繼承自 java.ejb.EJBObject 及 java.ejb.EJBHome，而這兩介面都繼承自 java.rmi.Remote。而透由 EJB Container 業者所提供的工具產生的 EJB Object 以及 EJB Home，則將分別實作遠端介面以及主介面。要開發 EJB 元件時，程式設計師只要開發遠端介面、主介面、Bean Class，至於 EJB 元件與 Container、環境、或是其他

EJB 元件的互動都透由 EJB Object 與 EJB Home 協助完成。圖 3.2 為 EJB 元件介面與 Container 之關係。

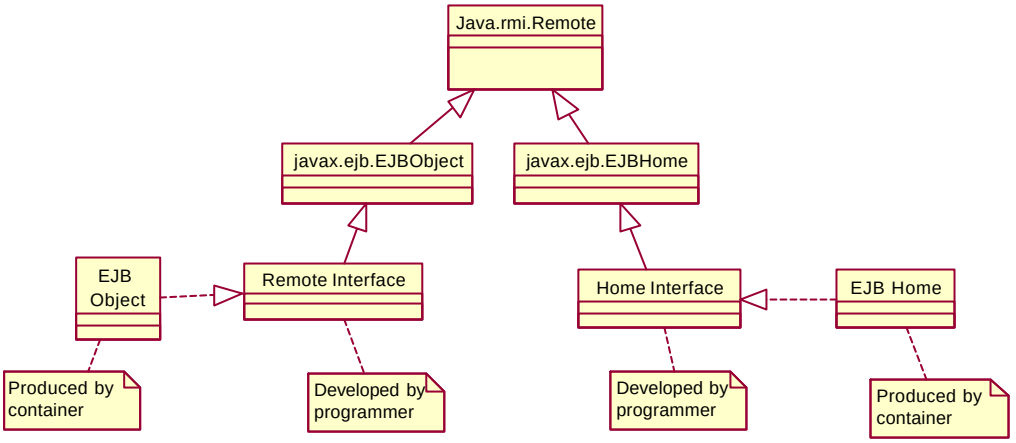


圖 3.2 EJB 元件介面與 Container 關係

- **EJBContext 介面與 EJB**

javax.ejb.EJBContext 介面由 container 負責實作，提供 EJB 與週遭環境有關的資訊 。當用戶端呼叫商業函式(business method)或是 container 呼叫 callback method 時，Session Bean 或 Entity Bean 會分別參照到繼承自 EJBContext 的介面，以供存取資訊。EJBContext 介面與 EJB 之關係如下圖 3.3 所示。

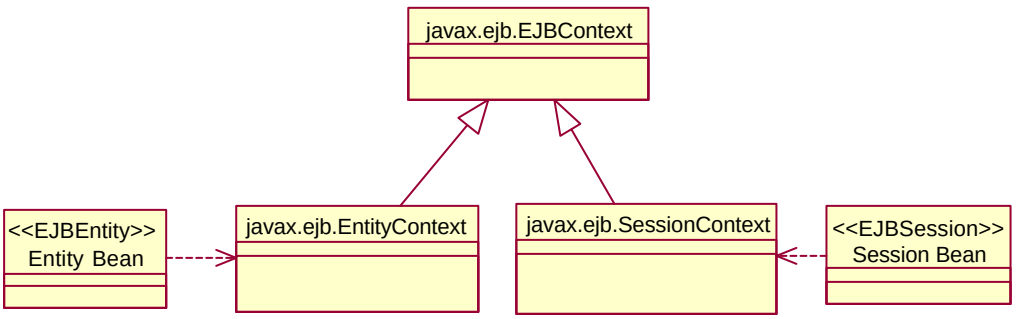


圖 3.3 EJBContext 介面與 EJB 關係

3.2 系統架構(System Architecture)

接下來會針對要進行重整前後的系統及架構進行描述 ， 並說明原始系統的缺點以及轉換成目標系統的好處。

3.2.1 原始系統架構

於本論文中，鎖定一個以 Java 語言為發展工具的線上圖書管理系統[18]作為實驗範例進行重整，該系統的原始程式檔案，只有 HTML、JSP、Java Bean 三種類型。

針對原始系統之程式碼進行分析後所得之系統架構如圖 3.5 所示。該系統的前端使用者是以瀏覽器的方式透過 HTTP 協定而連線進入系統，系統依據使用者的連線請求而以 JSP 網頁來服務使用者，並以 JSP 網頁回應伺服器的處理結果給使用者。在使用者的連線過程中，可以進行如查詢、訂購圖書等商業行為(Business behavior)，這些行為具有商業邏輯(Business Logic)，可能只是一連串的運算處理，也可能包含了多次資料庫存取動作。在此系統中透由 JSP 網頁與 Java Bean(此指 value bean 性質的 Java Bean)互動，而利用根據商業行為所定義的商業函式(business method)來完成使用者的請求。而對於底層與資料庫運作的細節，包括指定資料庫的 JDBC 驅動程式類型，則透過另外兩個公用的 Java Beans (此指 process bean 性質的 Java Bean) : SQLBridge 及 conPool 來負責。每次只要有資料庫存取之行為，都一定會經過這兩個 Java Beans 進行處理。其中，SQLBridge 進行四項主要任務: 連結資料庫、傳送 SQL 指令、接收及處理執行結果、關閉資料庫連結等[18]。此外，由於建立資料庫連結的過程中，資料庫系統都必須先為新建的連結配置記憶體資源，然後才能進行資料庫存取，因此，當系統建立資料庫連結的需求大時， 便會使得整體效能因而降低。因此，該系統使用 conPool Bean 來建立連結物件儲存池(Connection Pool)。

連結物件儲存池(Connection Pool)的做法是預先產生一群物件並將它們先存放在一個池(Pool)中。在此系統中，是利用 JSP 程式的 init()函式在對 JSP 程式初始化時去呼叫 connPool Java Bean 而預先建立起資料庫的連結，因此，當 Client 端有連線的需求時, JSP 程式的執行緒就能透過系統分配而得到預先產生的連結

物件，進而存取資料庫，也能因此縮短回應時間節省系統資源。

- **原始系統的缺失**

雖然原始系統使用了 Java Bean 來處理商業邏輯，但 JSP 網頁中仍嵌有許多具有商業邏輯處理性質的 Java 程式碼並不適合以 Java bean 來處理，而且當系統不斷的擴充，連線需求不斷增加時，以 JSP 及 Java Bean 技術開發的原始系統，便因為軟體架構與元件技術上的瓶頸而面臨了擴充上的困難，所引發的問題敘述如下：

(1) JSP 網頁中顯示與處理用途的 Scriptlet 程式碼、甚至 SQL 語法交互參雜、不但導致程式過長更影響可讀性。

(2) 因為 Scriptlet 程式碼寫死於 JSP 網頁中，彈性不足、擴充不易，欲增加新的功能只得因循舊法寫作， 經過幾次擴充下來，JSP 程式更為冗長、複雜。

(3) 當網站規模擴大，程式模組增多，系統更需重視專業分工，負責介面美觀處理的美工人員，往往為錯雜於 JSP 網頁中的程式碼所苦；程式設計師在處理商業行為之時，必須往返於 JSP 網頁與 Java Bean 程式之間，無法開發出較具結構化的程式。

(4) 系統於 JSP 程式產生時預先產生連結物件，會導致極少被執行的 JSP 程式佔有一個資料庫連結，但使用頻繁的 JSP 程式卻可能出現多個執行緒共用同一個連結物件的不正常情況。

(5) 當連線人數暴增時，系統不具負載平衡功能，因而影響伺服器執行效率，嚴重時更導致當機(crash)。

這些問題，都起因於缺乏元件化、結構化的設計缺失。而且單用 JSP 搭配 Java Bean 元件的做法雖然能夠達到網站建構的需求，但是對於那些架構較複雜要處理繁瑣的商業行為與控制流程的大型網站而言，顯然功能不足且不夠彈性，無法滿足需求[24]。

3.2.2 目標系統架構

由前 3.2.1 節可以得知，沒有完善的軟體架構將會導致許多後遺症。而且為了軟體的擴充性需求，需要更有效的方式與工具來處理請求與商業邏輯的運算。而 Sun 的 J2EE 平台是伺服器端應用程式依功能而劃分層級，並界定出層級中元件的權責。

為了改善舊有的系統，我們參考 J2EE 多層式架構(如圖 3.4 所示)，並依重整需求而得到如圖 3.6 的目標架構。

- **J2EE 多層式架構 (J2EE N-tier Architecture)**

在 J2EE 的多層式架構[8]中，依據 J2EE 元件的角色與功能將應用程式系統邏輯性地劃分為五個層級，目的在使具有類似功能的元件集中在同一層級中，透過 container 的協助而完成層級間的互動。

圖 3.4 J2EE N-tier 系統架構圖

- **用戶端層(Client tier)**

用戶端層代表了可以用來存取系統的其他應用程式或裝置，可以是透由 HTTP 協定與 Web Container 通訊的瀏覽器，或是其他使用 HTTP 外尚可利用 RMI 或 IIOP 協定來進行溝通的 GUI 應用程式。在無線網路興起之後， 也可以透由 WAP 手機或 PDA 等裝置透由無線存取協定(Wireless Access Protocol ，WAP)來存取系統。

- **顯示層(Presentation tier)**

顯示層中透過如 JSP 或 Servlet 等 Web 元件將所有用戶端存取所需的顯示邏輯封裝起來，用來當作存取 EJB 元件服務的顯示介面。在這一層中用戶端的請求會被攔截下來，並透由 Session management 機制來控管其存取商業服務(business service)的有效時期。經過處理所得的回應資訊再透由JSP 或 Servlet 元件顯現到用戶端。

- **商務層(Business tier)**

商務層提供用戶端所需的商業服務，所有商業邏輯的處理即是在此層完成，因此也較其他層複雜而重要。商務層的主要元件為 EJB 元件，是由 EJB container 進行管理。任何獨立的應用程式，都可以直接與 EJB 元件溝通。透過此層中 EJB 元件的適度分工與交互作用，宛如黑箱機制般地處理掉複雜的商業邏輯，而滿足用戶端的請求。

- **整合層(Integration tier)**

整合層負責應用程式與其他外部系統或資源的溝通，可以協助商務層對資源層的資源進行存取。在這層中的元件可以使用 JDBC、JMS 或 J2EE connector 的技術來與資源層互動，抑或透過其它中介軟體(middleware)來存取資源。

- **資源層(Resource tier)**

資源層中包含了商業資料(business data)**和其他外部資源, 通常是一個或多個資料庫管理系統**(DBMS)**，或是提供應用服務的** legacy system **和** business-to-business (B2B) system**。**

- **使用 J2EE 多層式架構的理由**

(1) **建立良好軟體結構，降低元件模組間之偶合力**(Coupling)**，提昇元件模組內部之內聚力**(Cohesion) **。**

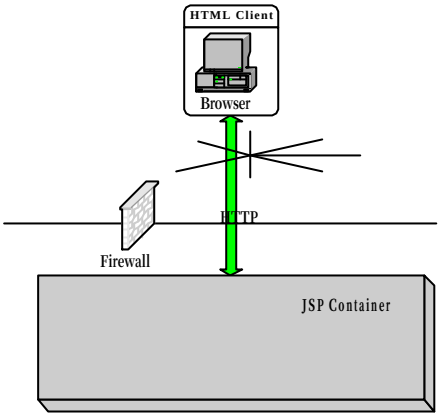
(2) **利用** EJB **元件的設計，精簡用戶端程式碼， 提昇應用程式的品質與服務效能。**

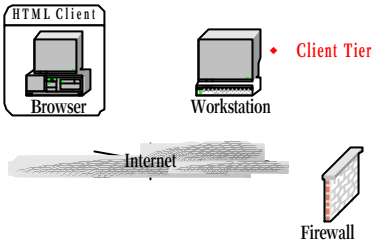
(3) **使用** EJB Container **的功能，並使應用程式具備負載平衡能力，程式設計師能專心於商業邏輯的處理。**

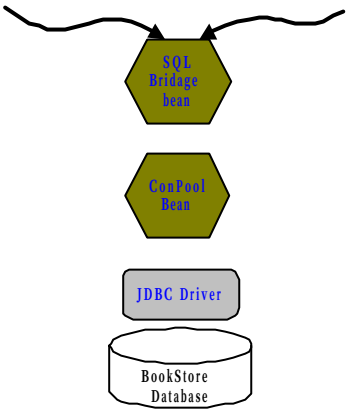
(4) **解決了** JSP **中使用** Connection Pool **的缺點，**EJB Container **提供了建立** connection pool **的服務，節省系統資源。**

(5) **容易擴充系統功能，模組化的設計，增進設計與維護上的彈性，提昇軟體再使用率。**

圖 3.5 **及 圖** 3.6 **為原始系統與新系統之架構圖，由二圖對照比較可以發現到轉換前後的元件對應關係以及運作機制。新系統中使用了多層式架構以及** J2EE **伺服器來改善舊系統的架構並提昇系統的功能 。重整後的新系統不僅元件分層分工清楚，而且維護容易，其他大型應用程式更能透過** HTTP **協定穿透防火強並存取** J2EE **伺服器上所提供的服務 。**







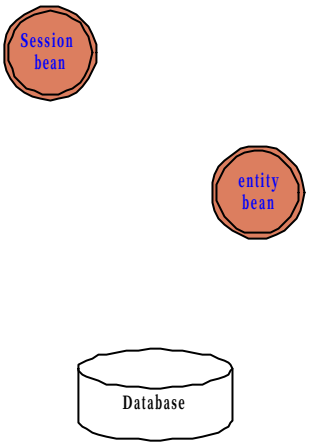


圖 3.5 原始系統架構

圖 3.6 新系統架構

- MVC 架構圖

以 Model-View-Controller 的角色來分類原始系統的元件，所得到的分布圖如圖 3.7。其中，View 中為負責顯示回應給 Client 的角色，由 HTML 網頁及 JSP 網頁所扮演; Controller 負責進行商業邏輯處理，由 Processs Bean 及 JSP 網頁扮演; Model 則是代表由資料庫中取得的資訊，由 Value Bean 扮演。由此可發現，JSP 元件同時扮演了 View 與 Controller 的角色，而且這樣的 JSP 元件不僅包含了顯示資訊(Presentation Information)，更包含了顯示邏輯(Presentation Logic)與商業邏輯(Business Logic)。這使得 JSP 網頁充斥過多程式碼，不僅缺乏可讀性更是增加後續擴充維護的困難。而且，使用 Java Bean 來扮演 View 和 Controller 的角色在搭配 JSP 元件，這樣的組合方式雖然能夠建構出一個商業應用網站，但是要提供顧客更完善更可靠的服務，就得遷就舊系統小規模的開發方式來加入新功能，並且必須花費許多的成本來撰寫許多關於系統層次複雜的程式。

而採用新系統架構，則利用 J2EE 伺服器提供系統層次的服務，使得程式設計人員只要專心於 Model-View-Controller 三大類元件的程式開發。其中，View 中就純粹只有代表顯示資訊的 HTML 及經過萃取、重整的 Pure Jsp；Controller 也改由 Custom Bean 與 Session Bean 的搭配來處理商業邏輯與顯示邏輯，大大降低 Jsp 程式碼的複雜度; 而 Model 則改由代表資料庫記錄的 Entity Bean 元件來扮演，透由 J2EE 伺服器而與資料庫進行同步更新，作為 View 之資料顯示來源。由此可見，使用 J2EE 分層式架構可以使元件依據 Model-View-Controller 的功能劃分清楚 以建構出低藕合高聚合的元件導向系統。新系統之 MVC 架構如下圖 3.8 所示。

由新舊系統各元件之 MVC 架構可見系統設計的差異，值得注意的是，在圖 3.7 中 Value Bean 是扮演 Model 角色，在圖 3.8 中扮演 Model 角色為 Entity Bean，但 Entity Bean 元件的來源卻是來自 Jsp 中的 SQL 語法及資料庫表格。因此，新舊系統之 MVC 架構，並不具直接對應關係，而是要依據各元件之特性以進行轉換，詳細的轉換對應關係請見圖 3.9。

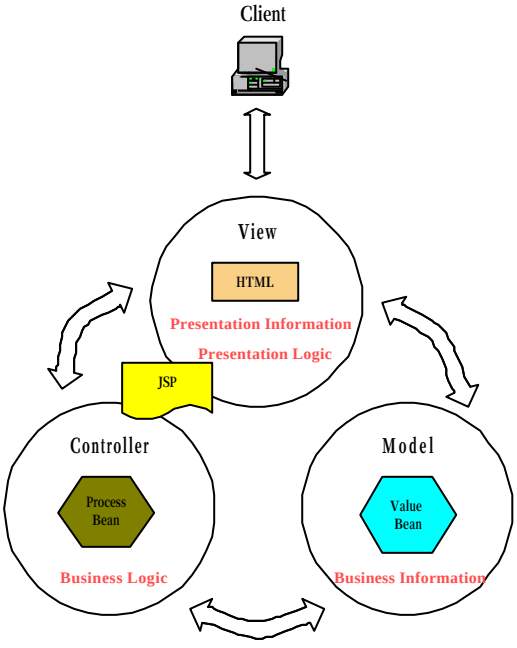


圖 3.7　原始系統之 MVC 架構圖

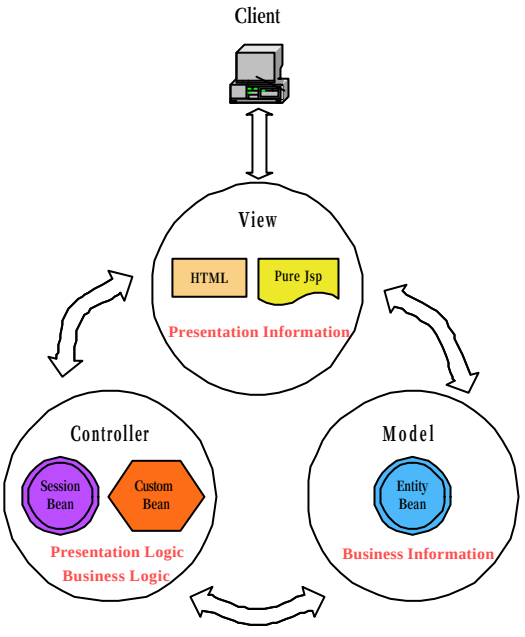


圖 3.8　新系統之 MVC 架構圖

3.3 重整方法

為了使原始系統的功能能夠得以保留下來，並提昇系統的性能，本論文採用重整的方式來擷取原始系統中的資訊，並經一系列轉換處理而得到符合 J2EE 多層式架構的新系統。針對原始系統中各元件的重整方法說明如下。

- **JSP 網頁的重整**

  - **問題描述**

    在 JSP 網頁之中的資訊可分為底下四類型:

(1) 用以顯示靜態網頁資訊的 HTML 碼。

(2) 用於協助欄位確認、檢查的 Script 碼，如: JavaScript。

(3) 協助了解 JSP 網頁的 JSP 註解 。

(4) 使 JSP 網頁呈現互動效果的 JSP 組件。

其中之 HTML 碼用來顯示網頁的靜態資訊，Script 碼進行運算或檢查以協助網頁顯示互動之結果，而 JSP 註解則只是說明程式碼的作用，因此這些非 JSP 組件的程式碼在重整前後都扮演同樣的角色，因此在重整的過程中可以原封不動的保留著。如表 3.1 所示。然而 JSP 中的程式組件(Scripting Element)，往往會因為程式開發人員的寫作風格不一而使得 JSP 網頁充斥太多複雜程式碼，一方面導致後續維護擴充的困難，另一方面更影響美工人員進行網頁編輯作業。由於程式組件必須透過與其它組件搭配才能完成網頁的互動效果，因此 JSP 組件便成為 JSP 網頁重整工程中的主要焦點。

類型	例子說明	作法
Html Tag	<html> ..</html>	**保留不擷取**
Script 語法	<script src="bookStore.js"></script>	**保留不擷取**
JSP 註解	<%--**註解區**%>、<%//**註解區**%>、<%/*註解區*/%>	**保留不擷取**

表 3.2 **非** JSP **組件的重整處理**

- **解決方法**

本論文採用的解決方法如下:

(1) 減少使用程式組件(scripting elements)的時機

JSP **網頁顯示動態資訊的效果，是經過與伺服器端元件互動而得的處理果。因此，只要能夠使處理結果透由** JSP **程式組件顯示於網頁上，便能夠避免** JSP **網頁程式碼過長，而使得** JSP **網頁的可讀性增加。也就是說，讓程式組件只負責協助顯示動態網頁的工作。**

(2) 使用自訂組件(Custom Action)及 EJB 元件的優點

處理複雜的商業邏輯的部份就交由自訂組件與 EJB **元件來進行。自定組件本身也是一個** Java **類別，可以透過此類別定義出** JSP Container **能辨識的自訂標籤，而將處理的細節於自訂組件的類別中進行，再將處理結果透由自訂的標籤而顯示於** JSP **網頁上。由於** J2EE **中的** Enterprise JavaBeans **元件具有** 3.1.3 **節中所述之優點，因此當有需要時，可以** Session Bean**元件來處理商業邏輯時，或是透由** Entity Bean **來與資料庫進行存取對應時，也可以在自訂組件類別中使用** JNDI (Java Naming and Directive Interface)**的機制來呼叫這些** EJB **元件。**

由上述可知，商業邏輯的處理方式是原始系統與新系統的差異所在，因此重整工作的進行主要是將原始系統中的商業邏輯處理工作轉移給新系統中功能更完整的元件來進行。方法如下:

(1) 擷取 JSP 組件中與商業邏輯處理有關的屬性或資訊　再經轉換處理以得到只具純顯示邏輯的 JSP 元件，以及封裝了商業邏輯的自訂組件。

(2) 將 Java Beans 元件中的資訊一一對應轉換至 Session Bean 元件中　再經過加入標準 Session Bean 元件中與 J2EE container 互動的必備函式，以及處理 EJB 元件繼承與例外情況的程式碼而得 Session Bean 元件的 Bean Class。再由 Bean Class 中的函式定義而得到產生遠端介面與主介面的資訊。

(3) 由原始系統資料庫表格中之欄位擷取資訊　依據資料庫欄位的訊息而設計出 Entity Bean 元件的商業函式。再透由參照標準 Entity Bean 元件的規格而陸續生成 Entity Bean 元件的 Bean Class 、遠端介面與主介面。

- **Java Bean 的重整方法**

依 Java Bean 在伺服器應用程式中的功能來看，可以歸類出兩類：一類是在 bean 中只有 setter 和 getter 等函式的 Java Bean，可以直接對應到資料庫中，我們稱之為 value bean。而另一類則是除了 setter 及 getter 以外，尚包含了其他商業函式，我們稱之為 process bean。這些商業函式之中常會因為複雜的處理流程，而牽扯到其他函式的呼叫或資料庫存取。因此，在新的系統架構中，為了擴充性考量，以 Session Bean 來接手上述兩種 Java Bean 的工作，只有在『自訂動作組件』的部分是採用 Java Bean 元件。

在原系統架構中，Java Bean 的用途為協助 JSP 程式處理商業邏輯，以 MVC 的架構來看即是扮演了 Model 的角色。而新系統中則以 entity bean 來扮演 Model 的角色，以 Session bean 來代替 Java Bean 中商業行為的處理，使得系統的藕合度因而降低 。

- **資料庫表格的重整方法**

資料庫表格在重整的過程並不會改變資訊，但要生 Entity Bean 元件時，則必須要依據表格中欄位的資訊，來定義出Entity Bean 元件中所當具備之call back method。

3. 4 重整流程概觀

要進行重整的原始系統中只有 HTML 、JSP 、 Java Bean 以及資料庫表格等四類型的檔案，其中 HTML 檔案並不需要重整，而資料庫表格的資料也將保留，且其中的資訊可提供順向工程階段生成 Entity Bean 元件。而 JSP 和 Java Bean 則分別經歷了三個主要處理階段：擷取(Retrive)、生成(Gererate)、佈署(Deploy)。在擷取階段中進行原始程式碼分析的工作，並截取出有用的資訊，此屬於逆向工程的範疇；而生成階段則對截取出的資訊依據目標元件的種類與需求進行轉換處理以得到新元件，此則屬於順向工程。而佈署階段的主要目的是將生成階段所得到的所有元件，透過伺服器廠商所提供的佈署配置工具佈署至伺服器上。由於新元件都是依據 J2EE 多層式架構中元件的標準規則而產生的，因此，只要元件被佈署到 J2EE 伺服器上，即能顯現各元件在 J2EE 多層式架構中所扮演的角色，並發揮其功能以與其他元件合作互動，至此重整流程即告完成。完整的重整流程概觀圖如圖 3.7 所示。

圖中的 Process Bean 及 Value Bean 皆屬於 Java Bean，各元件在不同階段所進行的轉換與處理將陸續介紹如下。

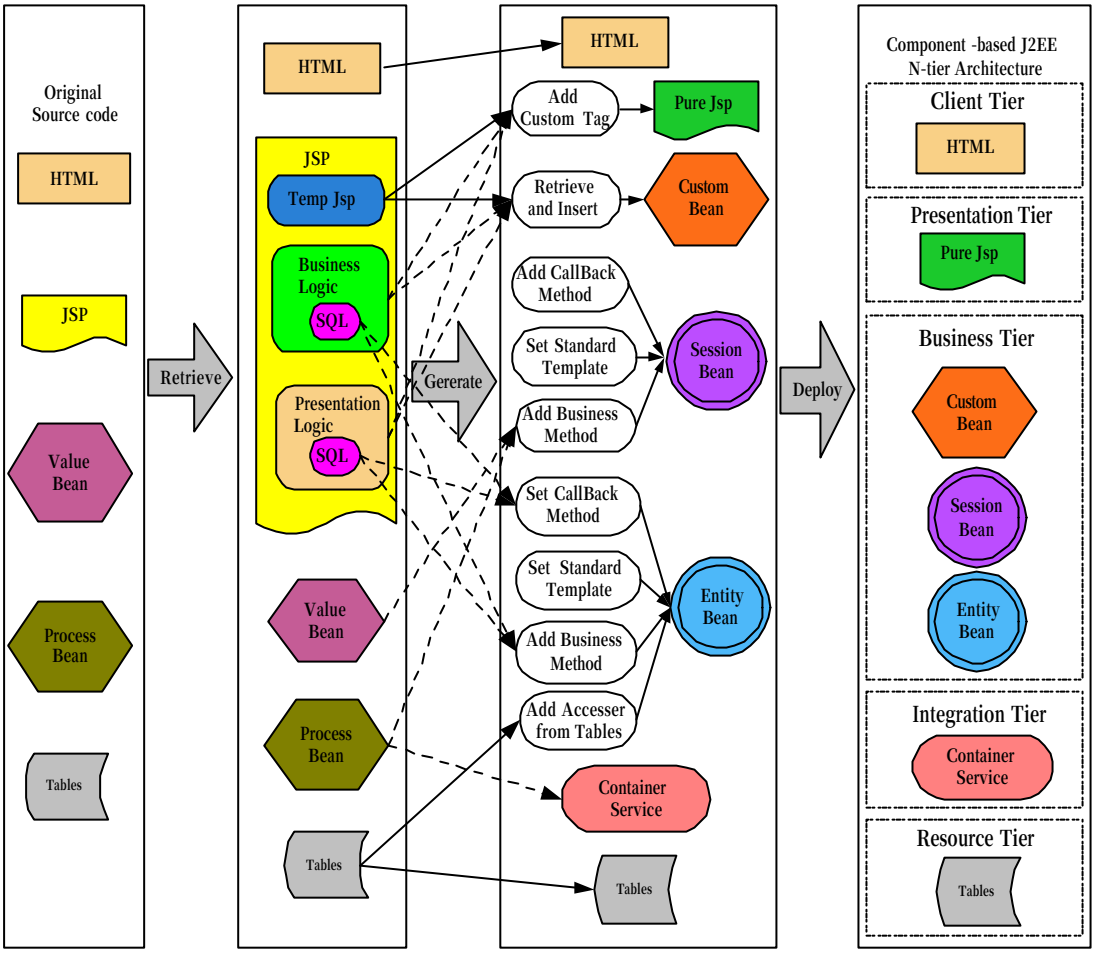


圖 3.9 **重整流程概觀**

3.4.1 資訊擷取(Retrive)階段

- **擷取 JSP 組件的資訊**

原始系統中的 JSP **程式不但於** Scriptlet **中撰寫了冗長的程式碼，甚至還嵌入了** SQL **語法，因而大大降低可讀性，不僅開發時期除錯不易，更影響了後續的維護與擴充。一個較好的** JSP **程式中的** JSP **組件，應該是除了基本的** JSP **標籤與屬性以外，只存在包含顯示邏輯**(presentation logic)**的程式碼，具有較複雜處理細節與流程的商業邏輯程式碼應該被抽離出來獨立處理。然而，嵌入於** JSP **網頁中的程式碼，引用到的變數型態及該變數的生命週期與** JSP **標籤中的屬性值設定**

息息相關 。因此，要對 JSP 網頁進行重整必須分析 JSP 組件，以得知該 JSP 網頁中針對動態資訊的相關設定，並定義新舊系統間的對應方法。Jsp 之三大標準組件之分析與對應處理描述如下。

- **指令(Directive Elements)組件**

  - **描述　指定關於 JSP 網頁本身的資訊，尤其是不因請求而異的共同資訊，會影響 JSP 引擎對 JSP 網頁的處理，指令組件在編譯時期被編譯一次後便能夠服務之後每次客戶端的請求。**

  - **種類　共有三種指令**:

(1) **include 指令: 用來引入靜態檔案, 將該檔案內容與 JSP 網頁內容合併在一起。**

語法: <%@ include **屬性值設定**%>

分析: 該指令之中並沒有和 scriptlet 相關的屬性, 因此與商業邏輯的處理無關。

例子:

```
<%@ include file="checksum.html" %>
```

對應處理: 保留該指令於 JSP 網頁中，不作變動。

(2) **page 編譯指令: 用來定義該網頁中專有的屬性。**

語法: <%@ page **屬性值設定**%>

分析: 該指令之中的 import 屬性需被 scriptlet 中的程式碼所參考。

例子:

```
<%@ page language =”java” contentType=”text/html;charset=Shift_JIS” %>

<%@ page import =”java.io.*, java.util.*”    %>
```

對應處理: 保留該指令並擷取 import **屬性，轉為自訂組件類別中的** import **敘述。**

(3) taglib 指令: 當網頁中要引用到自訂的動作組件(custom actions) **，也就是非標準** JSP **組件時，必須用** taglib **指令來宣告含有這類型動作組件的標籤庫**(tag library ,**常被簡稱為** CTL) **。**

語法: <%@ taglib **屬性值設定**%>

分析: 該指令與商業邏輯的處理無關。

例子:

```
<%@ taglib uri=”/rattaglib” prefix=”rat” %>
```

對應處理: 保留該指令於 JSP **網頁中，不作變動。**

- **程式組件(Scripting Elements)**
  - **描述 可以利用程式組件而將程式碼加入** JSP **網頁中，用以進行顯示邏輯或商業邏輯的處理。**
  - **種類 共有三種應用情形**:

(1) 宣告(Declaration)：定義和宣告一些 class level **的變數或函數其生命週期與** JSP **程式同步。**

語法: <%!**完整、合法的宣告句**%>

分析: 在宣告區中進行變數與函式的宣告，可能隱含商業邏輯的處理或進行資料庫存取等動作。而且宣告的變數或函式會影響到之後 Scriptlet **區程式碼的變數參照。**

對應處理: 針對宣告區的處理可分為三個階段，說明如下 :

Pass one: 裁取程式，加入 Custom Tag**。將宣告區中的程式碼全部取出，並在取出處放入一個作為標註之用的自訂標籤。**

Pass two: 進行下列處理:

1.**將取出的程式碼放入自訂組件的類別之中。**

2.**將類別中有關** Process Bean **的函式刪除(因為使用** CMP**，故可由** Container **代為執行)。**

3. **設計、定義包含相對應於** Value Bean **那些函式的** Session Bean **元件及其函式。**

4.**針對與資料庫存取相關的函式與** SQL **語法設計、定義相對應的** Entity Bean **元件及其函式。**

Pass three: 進行置換工作:

1.**將自訂組件類別中有關** Value Bean **的函式以** Session Bean **元件之對應函式進行置換。**

2.**將自訂組件類別中有關資料庫存取之函式與** SQL **語法以** Entity Bean**進行置換，其中的函式有部分因為使用** CMP **而不再需要，有些以** Entity Bean **的** Business method **來對應其處理工作。而** SQL **語法的部分則依**

其語法以 Entity Bean **之** Callback Method **或** Business method **進行置換。**

例子:

```
<%!
   void deleteOrder(){//(略)}
   void  updateOrder(){//(略)}
%>
```

(2) 運算式(Expression)：**可以進行運算處理，並將結果顯示出來，處理結果必須是字串或可被轉成字串型態。**

語法: <%= **完整、合法的宣告句**%>

分析: 此區的程式碼是用來顯示動態網頁的處理結果。

對應處理: 依據該運算是使用的時機而有不同的對應處理方式，說明如下:

情況一: 運算式出現於顯示邏輯之 Scriptlet **程式碼中。**

Pass one: **定義** custom bean**。**

在 Jsp **網頁中原本是利用運算式語法並透過** Java Bean **而取得運算結果的，就改以自訂標籤來設定、取得運算結果。自訂標籤封裝了運算處理的過程，真正處理的細節撰寫於對應的自訂動作組件類別(即** Custom Bean) **中。而此類型之運算式，會先經過迴圈或條件敘述而決定運算式中的顯示結果，因此，必須先撰寫迭代式動作組件的本體及其相對應的處理函式。**

Pass two: **置換為自訂標籤。**

將 JSP **網頁中含於顯示邏輯** scriptlet **中的運算式置換成迭代式動作組件，以代替原本利用迴圈或條件式敘述並透由** Java Bean **而得的運算結果。**

情況二: 運算式出現於顯示邏輯之外:

Pass one: **定義** custom bean**。**

此類型之運算式透過 Java Bean **而取得運算結果，因此，改以自訂標籤來設定、取得運算結果。自訂動作組件類別中定義許多函式，可以在自訂標籤中設定相對應的屬性而得到處理結果。因此，必須先撰寫選擇式動作組件的本體及其相對應的處理函式。**

Pass two: **置換為自訂標籤。**

將該類型運算式置換成選擇式動作組件，以利用 Custom Bean **代替** Java Bean **而得運算結果。**

(3) Scriptlet : 在此區中可以撰寫以 page **指令指定之程式碼，沒有經過良好設計的** JSP **程式，即是在此區中包含了冗長的程式碼。**

語法: <% **合法的程式碼片段** %>

分析: 原始系統在此區中可能撰寫了執行 SQL **語法、以及身分驗證、判斷按鈕事件等商業邏輯處理之程式碼 。**

對應處理: 針對上述之各種情況，以自訂動作組件及 EJB **元件來進行轉換處理。其步驟與宣告區同。說明如下**:

Pass one: **取出** scriptlet **區程式碼，並放入自訂標籤。**

Pass two: **將** Pass one **取出的程式碼放入** custom bean **中，並移除該** bean **類別中有關** process bean **的函式。之後再定義所需之** Session Bean **及** Entity Bean **元件。**

Pass three :進行置換工作:

Scriptlet 區的置換動作與轉換原理與宣告區同，目的在以Value Bean 的函式以 Session Bean 元件之對應函式進行置換。而將自訂組件類別中有關資料庫存取之函式與 SQL 語法以 Entity Bean 及其對應函式進行置換。

- **動作組件(Action Elements)**

  - **描述　動作組件的設計是為了要為每一次進行 JSP 網頁連線請求的用戶端進行客製化的服務，因此，動作組件是在用戶端請求時期才被執行的。和其他 JSP 組件不同的是，動作組件是以 XML 的語法安插於網頁中，因此，除了開頭與結尾標籤外，中間還包括了一個主體，而主體之中還可以包括其它組件，因此其語法也較多樣。**

  - **種類 可為 9 種 JSP 標準動作組件以及由使用者自訂的動作組件。不過** <jsp: fallback>、<jsp: forward>、<jsp: include/>、<jsp: param>、<jsp: params>、<jsp: plugin>**等動作組件與重整過程無關，因此不作對應處理。剩下三種組件，描述如下**:

    (1) <jsp:getProperty/>　用來取得 bean **的內涵值，以供在** JSP **網頁中之** Scriptlet **使用。**

    語法:　<jsp:getProperty **屬性設定值**/>。

    對應處理:　**因** Bean **將被取代，故將該組件移除。**

    (2) <jsp: setProperty > 用來設定 bean **的一項或多項內涵。**

    語法: <jsp:getProperty **屬性設定值**/>

    對應處理: **因** Bean **將被取代，故將該組件移除。**

    (3) <jsp: useBean> JSP **用此動作組件來建立** Java Bean **檔案與該** JSP **網頁的關連, 並透由該組件中屬性的設定決定** Java Bean **的使用方式。**

語法: <jsp:useBean **屬性設定值**/>

對應處理: **因** Bean **將被取代，故將該組件移除。**

各組件之對應處理會整如下表 3.3**。**

組件名稱	JSP 標籤語法		Pass1 對應處理	Pass 2 對應處理	Pass 3 對應處理
指令組件	<%@page 屬性設定%>		擷取 import 屬性值	放入 custom bean	
動作組件	<jsp:useBean/>		取得 id、class 屬性	移除該組件	
	<jsp: setProperty>		置換為對應之 C.T.	移除該組件	
程式組件	<%!宣告句%>		裁取程式，加入 C.T.	放入 custom bean	
				移除 Process Bean 之函式	
				定義 S.B.元件	引用 S.B.置換 Java Beans
				定義 E.B.元件	引用 E.B.置換 SQL 語法
	<%=運算式%>	包含於 P.L.C.中	定義 custom bean	置換為迭代式 custom tag	
		包含於 B.L.C.中	定義 custom bean	置換為選擇式 custom tag	
	<%程式碼 %>	含 SQL 語法 身分確認函式 按鈕事件判斷 其它	裁取程式碼，加入 C.T.	放入 custom bean	
				移除 Process Bean 之函式	
				定義 S.B.元件	引用 S.B.置換 Java Beans
				定義 E.B.元件	引用 E.B.置換 SQL 語法

縮寫: P.L.C : Presentation Logic Code E.B. : Entity Bean F.M.: Finder Method
B.L.C : Business Logic Code S.B. :Session Bean C.T.: Custom Tag

表 3.3 JSP **組件的對應處理表**

- **擷取 Java Beans 的資訊**

在原始系統中使用 Java Beans **搭配** JSP **網頁來處理使用者的請求，而新系統之中，則改以自訂動作組件來呼叫** Session Bean **中之商業函式來處理。因此，在擷取階段，取出** Java Bean **中之程式碼並放入** Session Bean **的** Bean Class **中，以完成** Session Bean **元件中商業函式**(business method)**及資料**(data)**的設定。**

依 Java Bean 在伺服器應用程式中的功能，將其區分為底下兩類，並說明其截取方式如下。

- **Value Bean**

為 bean 中只有 setter 和 getter 等函式的 Java Bean，可以直接對應到資料庫中。在順向工程中，將被轉成 Session Bean，因此 bean 中的所有程式碼都需截取出來，以待下階段對應至 Session Bean 中。

- **Process Bean**

除了 setter 及 getter 以外，尚包含了其他商業函式。這些商業函式之中常會因為複雜的處理流程，而牽扯到其他函式的呼叫或資料庫存取。而 process bean 的截取處理則要依據該 bean 的性質決定。

- **資料庫表格資訊的截取**

資料庫表格在逆向工程階段並不需要進行轉換處理，只需取得資料庫中欄位的資訊以供順向工程階段進行對應。

3.4.2 元件生成(Generate)階段

在此階段之中主要是由資訊擷取階段中獲得資訊以生成新系統中之元件，各元件之生成流程詳述如後:

- **JSP 元件的生成**

原始系統之 JSP 網頁分析後可分離出四大類元素，Temp Jsp、Business Logic、Presentation Logic 以及 SQL 敘述等。

- **Temp JSP 元素**

為抽取 JSP 程式組件後所遺留下來的 JSP 網頁。因此，Temp JSP 只具顯示處理結果的性質而無任何商業邏輯。Temp JSP 中的資訊包含了 JSP 中的指令組件與動作組件以及所有非 JSP 組件。由於不使用 Java Beans，因此，Temp JSP

之動作組件中已無<jsp:useBean>、<jsp:setProperty>、<jsp:getProperty>等標籤。

- **Business Logic 元素**

為 JSP 網頁中程式組件之程式碼，即是往往導致 JSP 網頁程式碼過於冗長之問題所在。Business Logic 元素被取出後，將在順向工程被放入 Custom Bean 中。

- **Presentation Logic 元素**

為 JSP 網頁中用以動態協助顯示處理結果的程式碼。Presentation Logic 的元素的種類可分為底下幾類，各有不同之截取方式:

1.以 If 敘述句來顯示結果

截取 Presentation Logic 的資訊以放入 Custom Bean 中。

2.以 for 迴圈或 while 迴圈來顯示結果

截取 Presentation Logic 的資訊以定義迭代式動作組件。其中，迭代式動作組件是為了取代 for 迴圈或 while 迴圈而自訂的一種動作組件。

3.先以 If 敘述句再配合 for 迴圈或 while 迴圈來顯示結果

截取 Presentation Logic 的資訊放入 Custom Bean 中，並定義迭代式動作組件。

- **SQL 敘述元素**

是指針對資料庫進行存取的 SQL 語句。在逆向工程階段中，將這些 SQL 語法取出以供順向工程階段開發 Entity Bean 元件之用。

如圖 3.8 所示，新系統最後生成之 Pure Jsp 元件，是由 Temp Jsp 及 Custom Tag 所組成的，圖中的實線代表 Temp Jsp 中的程式碼將被移入 Pure Jsp 元件中，而虛線表示商業邏輯與顯示邏輯的程式碼將被移出 Jsp 程式中，而以自訂標籤來顯示伺服器端的處理結果。

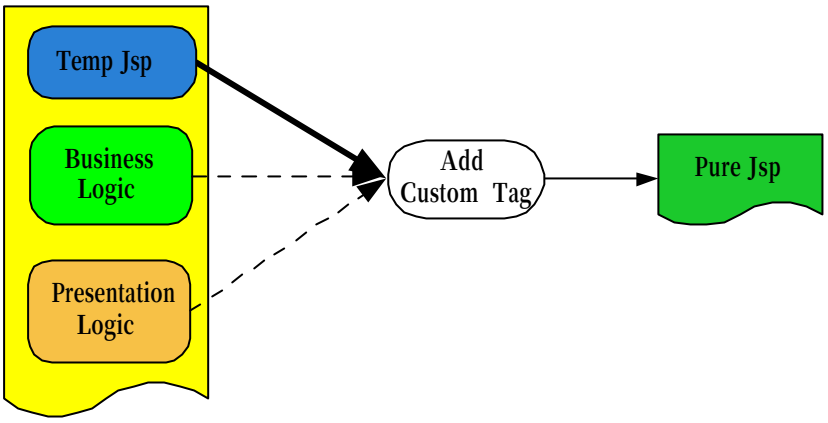


圖 3.10 JSP 元件生成流程

- **Custom Beans 的生成**

Custom Beans 的來源擷取自 Jsp 的三大標準組件。Temp Jsp 中與重整有關之程式碼，經過擷取後將連同 Business Logic 及 Presentation Logic 之程式碼安插入 Custom Bean 中。

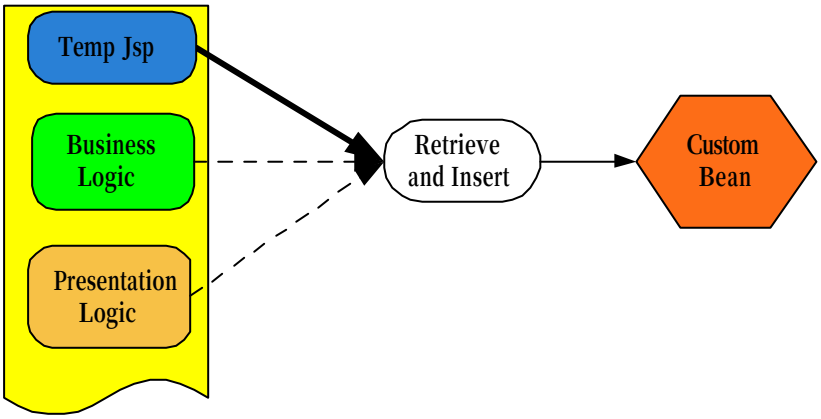


圖 3.11 **Custom Beans 生成流程**

- **範例說明**

底下範例為從原始系統中 userOrderMaintain.jsp 檔案重整而得的 Custom

Bean **之類別檔，依據其重整來源之不同說明各片段程式碼如下**:

範例說明: **從** userOrderMaintain.jsp **重整而得的** handle class

```
import    javax.servlet.jsp.*;                                    A
import    javax.servlet.jsp.tagext.*;
import    java.io.*;
import    java.util.*;
```

A **來自** Jsp **網頁之** import **屬性**

..(**略**)…

```
public class HandleTag extends TagSupport{                        B
```

B **中之** TagSupport **為** Jsp **所提供之標準類別，自訂標籤時必繼承之。**

```
        private int check;                                        C
        private String situation;                                 D
```

C、D **是為此自訂標籤所自訂之屬性。**

```
        public void setCheck(String check){                       E
                this.check=Integer.valueOf(check);
        }
        public ovid setSituation(String situ){                    F
                this.situation=situ;
        }
```

E、F **中進行自訂屬性值之設**

//**來自** jsp **的宣告區**<%! .... %>**程式**

```
        private void deleteOrder(){//(略)}                        G
        private void updateOrder(){//(略)}
```

G **中之程式碼來自於** Jsp **網頁宣告區的程式碼。**

```
public int doEndTag() throws JspException{                    H

    switch(check){

        case  0:
              //<jsp:useBean ... /> or <jsp:setProperty ... />    I
              break;

        case  1:
              //處理  <% .. if(!user.isAdminUser())               J
                          throws Exception; %>    的問題
              break;

        case  2:
             //處理<% action=request.getParameter("Submit");
                   if(action.equals("刪除"))                      K
                      deleteUser();
                   else if(...)
                   ... %>    的問題
              break;

        case  3:
              //  處理不含在顯示邏輯中之<%=...%>的內容 L
              break;

        case 4:
              //處理含在顯示邏輯中之<%=...%>的內容     M
              break;
        default:
              break;
    }//end of switch

    return EVAL_PAGE;

}//end of doEndTag
```

H 中之 doEndTag 函式中，將放入各自訂標籤之處理函式。

I 中之使用時機為移除 <jsp:useBean> 或 <jsp:setProperty> 時。

J 之使用時機為取代 Jsp 網頁中用以判斷使用者身分之函式，以自訂標籤將處理程式碼藏於 Custom Bean 中。

K 中之使用時機為取代 Jsp 網頁中用以判斷使用者所按下的按鈕事件並改寫入 Custom Bean 中。

L 中之使用時機為使用選擇式自訂動作組件取代 Jsp 中之 Expression 式時。

M 中之使用時機為使用迭代式自訂動作組件取代 Jsp 中之 Expression 式時。

- **EJB 元件的生成**

EJB 元件的生命週期與其生成息息相關，因為 EJB 元件的生命週期是透 EJB container 利用 callback method 而進行控制。因此，每一個 EJB 元件因其種類 (session，entity 或 message-driven)之差異而有不同的 callback method，要產生元件，必須先了解其生命週期以產生每種元件的 callback method。

另外，要開發 EJB 元件的過程，只要得到 Bean Class，就能夠依據 EJB 元件的規則而產生對應 Bean Class 的遠端介面和主介面。

- **Session Bean 元件的生成**

Value Bean 及 Process Bean 均可生成 Session Bean。Bean 中的程式碼將被移入 Session Bean 中，並加入 CallBack Method 及代表 Session Bean 元件標準模板之程式碼，如下圖 3.10 及圖 3.11 所示。

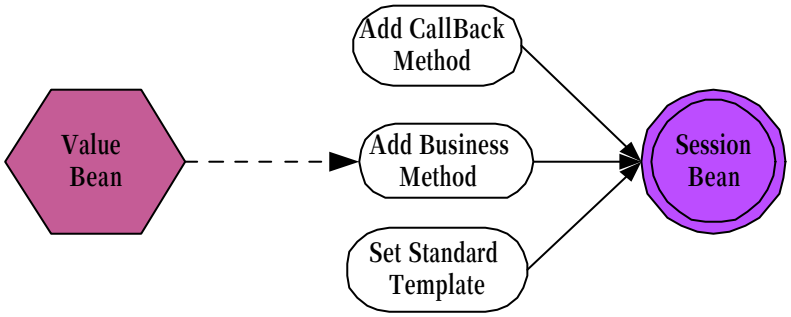


圖 3.12 由 Value Bean 生成 Session Bean

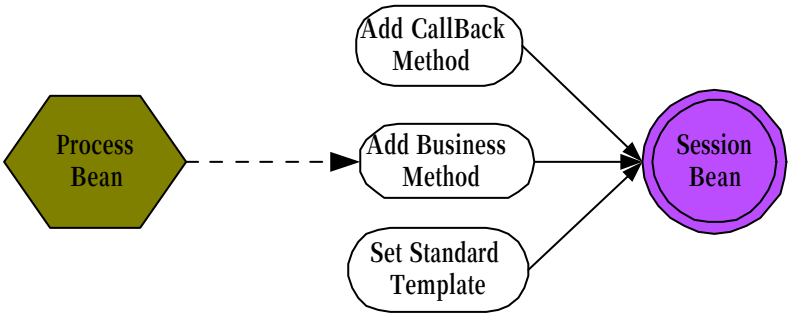


圖 3.13 由 Process Bean 生成 Session Bean

其中的標準模板(Standard Template)包含了 Session Bean 元件所當繼承之介面、所當丟出或處理之例外情形、以及對應而生的遠端介面(Remote Interface)及主介面(Home Interface)等。

- **Entity Bean 元件的生成**

由 Jsp 網頁中所抽離出的 SQL 語法，經過轉換對應而得到 Entity Bean 元件的 CallBack Method 以及 Business Method 。而 Entity Bean 元件中的 Accesser(getter 及 setter)則可由資料庫表格的欄位而設定，如資料表格中有一名為"age"的資料欄位，則可在 Entity Bean 的 Bean Class 類別中撰寫一組名為"setAge" 及"getAge"的函式。另外，在加入 Entity Bean 元件的共通標準模板之程式碼，即完成 Entity Bean 元件的開發。

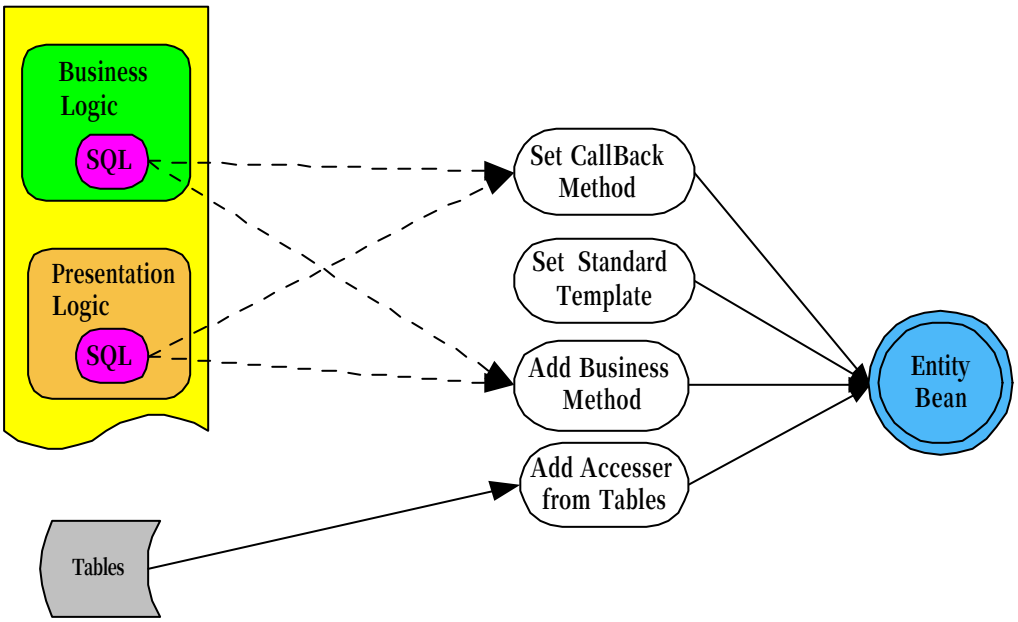


圖 3.14 Entity Eean 元件的生成流程

3.4.3 元件佈署(Deploy) 階段

以往發展軟體，常會遵照系統分析、系統設計、系統實作、系統測試、系統維護的軟體發展流程進行開發[10] [24]，然而對於為了因應社會進步，提昇企業競爭力的大型應用程式，在系統的可維護性(Maintainability)與可擴充性(Scalability)對企業電子化的建置成本考量之下，選用應用程式伺服器逐漸企業成為企業提供高效能網站服務的解決方案。也因此使軟體的產生過程又多了一個系統佈署的階段。

在本論文中佈署Enterprise Java Beans 是透由Orion Application server的部署工具來協助佈署，佈署完成後，將會產生佈署描述子。佈署描述子是一份指定 bean 資訊如永續類型和交易屬性的 XML 檔案 ，每一個佈署於 J2EE 伺服器之上執行的應用程式，皆會有一個包裝的其程式碼與XML 文件的 EAR(Enterprise application archive file)檔，之中描述了被一起佈署進去的 JAR 檔(Java archive file)、WAR 檔(Web application archive file)以及其他 XML 檔。其中的 JAR 檔包裝了 EJB 元件，WAR 檔包裝了 Web Component，XML 檔則是用以描述被包裝的檔案資訊，如圖 3.13 之元件佈署說明圖。

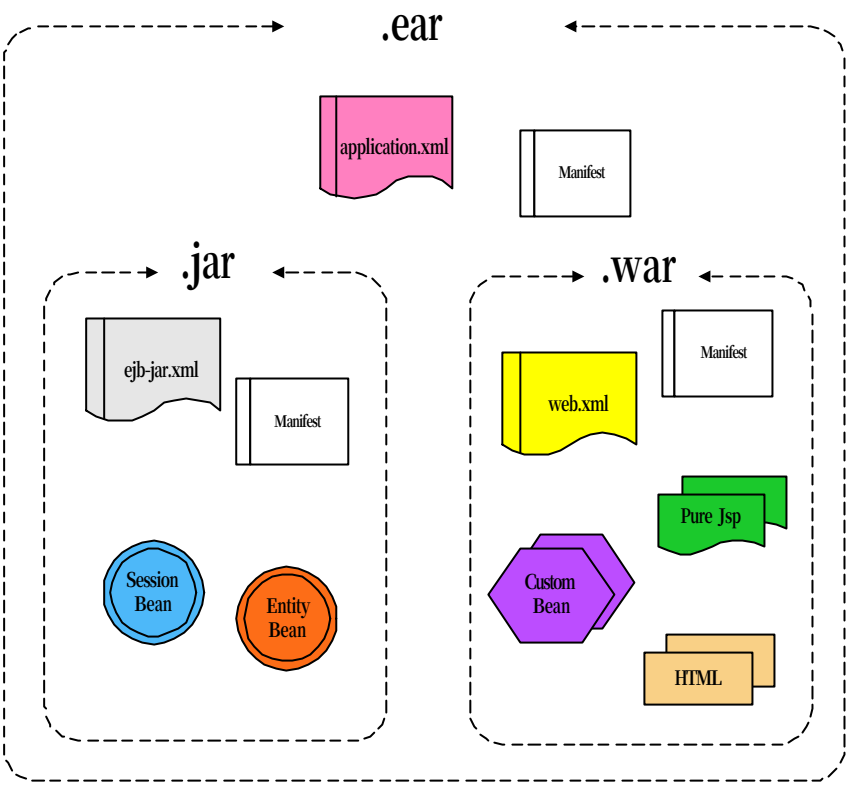


圖 3.15 元件佈署說明圖

由於各元件的生成是依循 J2EE 的規格開發而成。因此，元件生成之後將會自動的分布於 J2EE 的 N-tier Architecture 之中。每一層的元件透過介面與其鄰近層或同層之元件互動溝通，如此一來，不但提高了同層級元件間的內聚力，更能有效率地使不同層級間的元件溝通合作，降低整個系統的藕合度。也因為各元件的權責分明，因此可以彈性地調整系統架構，提昇元件的再使用率。

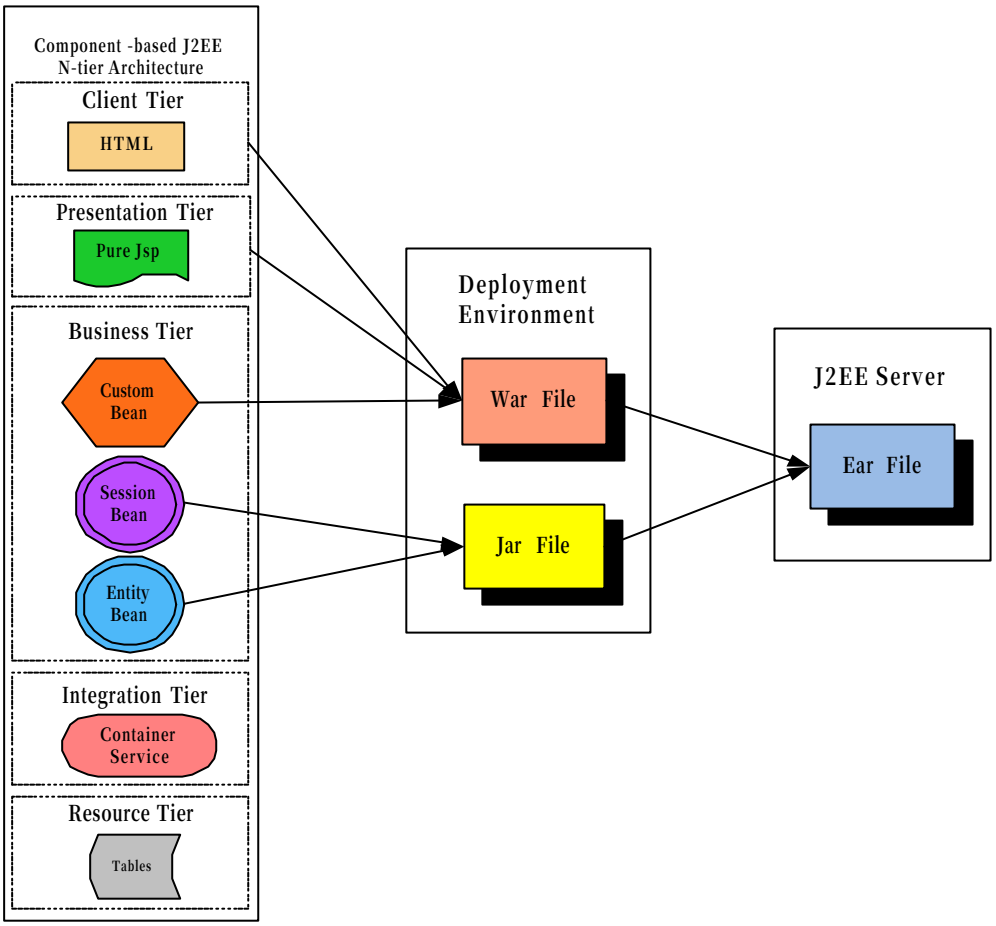


圖 3.16 元件佈署流程

元件在各層級之分布屬於邏輯性劃分，而真正要執行系統之時，必須先將這些元件經過包裝佈署於 J2EE 伺服器之上。如圖 3.14 所示，HTML、Pure Jsp、Custom Bean 等將被包裝於 War 檔中，而 EJB 元件則一起被包裝於 Jar 檔中。最後 War 檔及 Jar 檔在一起被包裝成 Ear 檔於 J2EE 伺服器上執行。

3.4.4 JSP 中之重整轉換處理規則

- **SQL 敘述置換**

Entity Bean 中的函式與 SQL 敘述的對應關係如下表 3.4。在此我們採

CMP(Container-Managed Persistence)的方式，讓 EJB Container 處理掉所有 Entity Bean 所需有關資料庫存取的部分，因此，SQL 的語法都能夠在 Entity Bean 的元件所定義的 Callback Method 中進行對應處理。比較特別的是 select 語法與 Entity Bean 的 finder 函式之對應，必須要等到佈署階段才經由佈署配置工具的設定為每一個 finder 函式下 EJBQL 的語法使其透由 container 自動對應到 SQL 之語法。這使得程式設計師可以更專心的處理應用程式的商業邏輯，也精簡了 JSP 及 EJB 的程式碼。

Method	SQL Statement
ejbCreate	insert
ejbFindByPrimaryKey	select
ejbLoad	select
ejbRemove	delete
ejbStore	update

表 3.4　SQL 指令語法與 Enterprise JavaBeans 中函式之對照表

如上表所示，Jsp 網頁中之 insert 語法將對應至 Entity Bean 中之 ejbCreate 函式，一旦執行了 ejbCreate 函式，J2EE container 將指定 bean instance 給 EJB object，此時的 Entity Bean 由輪調狀態(Pooling state)進入就緒狀態(Ready State)。接著，在系統呼叫並執行過 ejbPostCreate()函式後，主介面(Home interface)便會傳給用戶端 EJB object 的遠端參照(stub)。此時，該 Entity bean 即能夠為用戶端提供服務，同時也在資料庫中完成一筆資料(即一筆 record)的建置。

而 delete 語法則對應到 Entity Bean 中之 ejbRemove 函式，只要系統執行了 ejbRemove()函式，即會使得 bean instance 與 EJB Object 分離，此時 Entity Bean 由就緒狀態回到輪調狀態中，bean instance 等待再次被選上與 EJB Object 結合，而此時資料表中與該 Entity Bean 對應之 record 也將透由 J2EE container 而將之刪除。底下說明以 Select 語法來說明 SQL 語法之轉換處理過程。

➢ SQL 語法處理- Select

```
<%
  try   {
     //執行SQL Select 語法
    sqlBridge.execSQL( "select * from book where del_mark = 'n'
    order by id" );
     //判斷所找出的記錄
    while ( sqlBridge.nextRow() ){
     //取得id 屬性的值
    String id = sqlBridge.getFieldString( "id" )
  %>
    (略)
```

轉換為Entity Bean 之 FindAllById () 函式

於佈署時定義FindAllById () 函式之EJBQL

由Container 將EJBQL自動轉換為SQL語法

- **顯示邏輯判斷**

分析判斷是否有顯示邏輯存在。採用之判斷規則為:由不成對的 Block 語法判斷"{"，"}"。舉例如下:

(1) Scriptlet 間含有一段字串或敘述。

如:

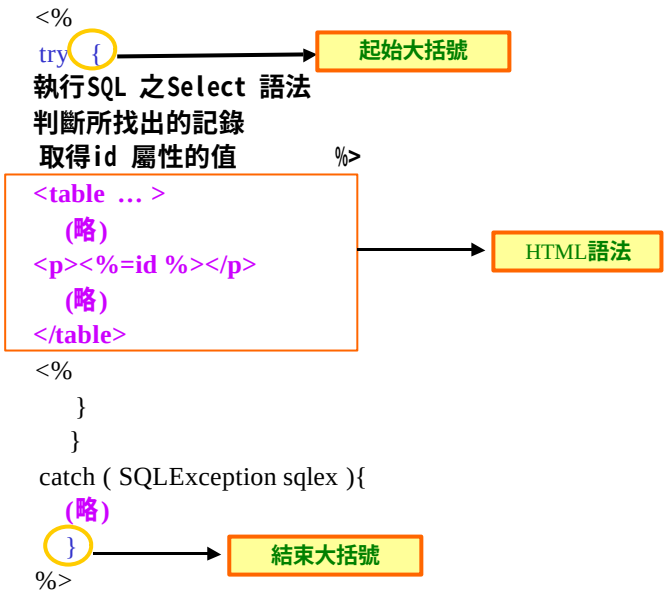
```
<% java.util Date clock= new java.util.Date( ) %>
<% if  (clock.getHours() <  12 ) { %>
  Good morning!
<% } else if  (clock.getHours( ) <17 )  %>
  Good day!
<%  } else { %>
  Good evening!
<% } %>
```

起始大括號

敘述句

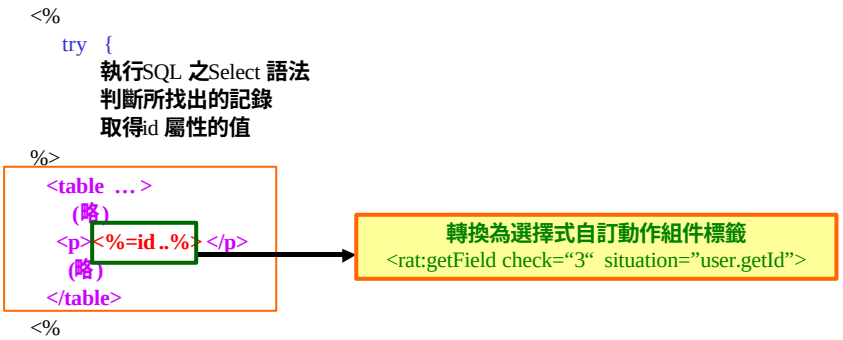
結束大括號

(2) **跨** Scriptlet **處理的情況。**




```
<%
 try  {
執行SQL 之Select 語法
判斷所找出的記錄
 取得id 屬性的值                %>
  <table  … >
     (略)
  <p><%=id %></p>
     (略)
  </table>
  <%
       }
     }
 catch ( SQLException sqlex ){
    (略)
    }
%>
```

- **顯示邏輯程式碼中 Expression 之處理**




```
<%
    try  {
         執行SQL 之Select 語法
         判斷所找出的記錄
         取得id 屬性的值
%>
   <table  … >
      (略)
    <p><%=id ..%></p>
      (略)
   </table>
<%
       }
     }
     catch( SQLException sqlex ){
        (略)
     }
%>
```

如上程式碼中的<%id= ..%> **，因其包含於顯示邏輯之中，因此將被置換成以選擇式自訂動作組件。在置換式中，**rat **為我們自訂之動態組件標籤庫**(Tag

library)，getField 為自訂標籤的名字，而 check 及 situation 為該組件之屬性值。這是 JSP 支援自訂動作組件的標準格式，透過自訂標籤及其屬性值的設定可以告知 Jsp Container 要到 Custom bean 中去取得運算處理之結果。

此範例表示 Expression 將要轉換成 rat 標籤庫中定義的 getField 標籤，而該標籤所要回傳之 id 值，將來自 Custom Bean 類別中定義的第三種運算處理方式。此外，由於在本論文中，以 Session Bean 來置換 Java Bean 之服務，因此原在原始程式中的 id 值，本是來自前段 Jsp 程式碼中名為 ”user” 之 Java Bean 的 getId 函式，在重整過後，將改由 Session Bean 中之 getId 函式取得。

第四章 結論與未來工作

本論文提供了一套重整程序與方法，使以運用 JSP 及 Java Beans 建構網站應用程式的系統，可以依據我們所自訂的規則與程序進行轉換。在軟體開發各階段中的人員，只要依循所提之對應處理的方法與步驟，就能夠截取出原始系統的可再用資訊轉換成各相對應之 J2EE 元件，並佈署於 J2EE 伺服器上執行。

另一方面，本論文採用 J2EE 多層式架構，研究並整合 Java 在 Web-based 應用程式開發上的新舊技術，以一個實例進行分析、歸納、驗證，而提出方法來使舊有系統在缺乏良好軟體架構及元件規劃下，可以不必耗費高成本進行重新開發即能得到一個維護性、再使用性更高的系統。

未來工作，在理論的延伸部分，將更深入了解 J2EE 在 Cluster 以及 load balance 的技術，以善用其在分散式環境下的元件互動模式，完成具有元件式負載平衡機制之可延展性系統的建置。另一方面，也將導入 Design Pattern、J2EE Pattern、Architecture Pattern 以建構出一個更具彈性與再使用性的多層式架構。

在實作面，將依據這一套重整程序與對應方法而設計重整輔助工具 (Reengineering Auxiary Tool，簡稱 RAT) ，以進行自動對應轉換，減少以人工判斷的繁瑣與可能缺失。並期望此套工具能藉由現階段所建立的對應規則，完成擷取舊有元件(JSP、Java Bean)資訊、生成 J2EE 元件(JSP、 Custom Bean、EJB) 與元件佈署(以 Pattern 建構元件間關係)等轉換處理工作。

參考文獻 Reference

[1] *The Unified Modeling Language User Guide,* Rational Software Corporation 1999.

[2] Object Management Group. (2001, August). *OMG unified modeling language specification*. Version 1.4, Retrieved July 16, 2001 from http://www.omg.org/technology/documents/recent/omg_modeling.htm

[3] Bennett, K. H. (1993). An overview of maintenance and reverse engineering, *The REDO Compendium*, John Wiley & Sons, Inc., Chichester.

[4] Booch, G. (1994). *Object-oriented analysis and design with applications* 2nd ed. Redwood City, Calif. : Benjamin/Cummings Pub. Co., 3-25.

[5] Chu, W.C., Lu, C.W., Chang, C.H., & Chung, Y.C. (2001). Pattern based software re-engineering. Handbook of Software Engineering and Knowledge Engineering, Vol. 1, Skokie, IL.: Knowledge Systems Institute.

[6] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, MA.: Addison-Wesley.

[7] Lano, K., & Malik, N. (1997). Reengineering legacy applications using design patterns. in the *Proceedings of the 8th International Workshop on Software Technology and Engineering Practice*, IEEE, 326-338.

[8] Sun Java Center J2EE™ Patterns,http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/

[9] Wirfs-Brock, R. J., & Johnson, R. E. (1990). Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33(9),105-124.

[10] B. Boehm, "Software Engineering," *IEEE Transactions on Computer*, Vol. 25, No. 12, 1976, pp. 1226-1241.

[11] kruchten P.(1995). The 4+1 view model of architecture. *IEEE Software*,**12**(6),42-50

[12] Frank Buschmann,"pattern-oriented software architecture", Wiley & Sons,2000

[13] H.P. Steiert, "Towards a Component-based n-Tier C/S-Architecture," *In Proceedings of the Third International Workshop on Software Architecture*, Orlando, FL USA, 1998, pp. 137-140.

[14] A. Aizman, "Application Framework for Rapid Agent Development", *IEEE Third International Workshop on Systems Management*, 1998 Newport, Rhode Island.

[15] J. Rumbaugh, "Depending on Collaborations: Dependencies as Contextual Associations," *Journal of Object-Oriented Programming*, Vol. 8, No. 7, 1998, pp. 5-9.

[16] Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller(MVC), **http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html**

[17] MVC,**http://minnow.cc.gatech.edu/squeak/1767**

[18] **楊洸,沈建男**,"Java Server Page **程式設計實務**,"**學貫**, **民國** 89 **年** 10 **月**

[19] Richard Monson-Haefel,**譯 黃奕勤,蔡寶進**,"Enterprise JAVABEANS **技術**,"**歐萊禮**, **民國** 89 **年** 10 **月**

[20] Hans Bergesten,**譯 楊子毅,林長毅**,"Java Server Pages **設計技巧**," **歐萊禮**, **民國** 90 **年** 8 **月**

[21] The J2EE Tutorial, http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html

[22]Rine, D. C. (1997). Supporting reuse with object technology. *IEEE Computer*, 30(10), 43-45.

[23]Wirfs-Brock, R. J., & Johnson, R. E. (1990). Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33(9),105-124.

[24]Sommerville, I.,*Software Engineering*, 6th edition. Reading, MA.: Addison-Wesley, 2001.

[25]Meyer, B. (1999). On to components.*IEEE Computer*, 32(1),139-40.(Ch.14)

[26]Coulouris,G., Dollimore,J. et al.(1994) Distributed Systems: Concepts and Design.Workingham: Addison-Wesley.(Ch. 11)

[27]Ivar Jacobson,Martin Griss,Patrik Jonsson,SOFTWARE REUSE, Addison-Wesley,1997

[28]http://www.gemstone.com/products/s/papers_corba.html

[29]http://www.omg.org/news/whitepapers/#CORBA/IIOP

[30] http://www.rational.com/index.jsp

[31]Warren,I.(ed.)(1998).The Renaissance of Legacy Systems.London:Springer.

(Chs 26,27)

[32]Ulrich, W. M.(1990). The evolutionary growth of software reengineering and the decade ahead. American Programmer,3(10) ,14-20.(Ch. 28)

[33] http://www.togethersoft.com/

[34] http://www.VisualUML.com

[35] http://www.microsoft.com/ms.htm

[36] http://www.orionserver.com

[37]Cluster EJBs Ed Roman ,Scott Ambler,Tyler Jewell,Mastering Enterprise JavaBeans,Second Edition, John Wiley & Sons,Inc.,2002.

誌　謝

半年多以來忙於論文寫作，讓我成長許多，走過了研究生發覺問題、廣集資料尋求解答的求學歷程，經歷了焚膏繼晷、以實驗室為家的生活，當然，也無數次品嚐了由絞盡腦汁、山窮水盡的地步而昇華到茅塞頓開、柳暗花明境界的甜美果實。這段難忘的錘鍊光陰，使我領悟到『鍥而不捨』的成功者哲學、以及『天助自助者』的自強之道，凡事盡心，雖不中亦不遠矣!

完成這份論文的此刻，內心百感交集，內心除了感謝與輕鬆以外，卻多了一份遺憾。論文完成接近尾聲之際，奶奶的驟然離世，使我痛苦不已，頓然失去動力，更憾恨自己未能早日完成論文，導致聚少離多未盡孝道之不是。

完成這份論文，第一位要誌謝的是我的指導老師　朱正忠教授，在這兩年多的時間，有形無形間給予我相當多的言教與身教，使我除了學術研究外又多學習到許多寶貴的知識。更感謝 您在我最低潮的時候，給予我正面的激勵與肯定，使我感受到莫大的溫馨，您的每一次鼓勵與肯定，是我一次次突破瓶頸的主要動力來源。

再來要感謝的是實驗室的學弟們 : 逸群、依恆、祚民、俊雄，由於你們的幫忙，張羅實驗室事務，使我們得以專心的完成論文。因為論文的壓力，使得這半年來鮮少與你們促膝清談，沒能參與你們的"下午茶"，敬請見諒! 另外，兩位專題學弟，育睿、盈達，在後半年中一路上跟著我一起鑽研技術、共同成長，使我深有『獨學而無友，則孤陋而寡聞』之慨，和你們一起努力、奮鬥的感覺真好! 還有那些曾經跟我一起討論，給予我莫大幫助的實驗室學長: 志偉及志宏學長，以及實驗室夥伴:文達、靜慧、逸群、繼賢、士嘉，和朋友: 珮姍、英彥及 AAJC 的學弟士軒、佳瑋及冠綸等，謝謝你們的協助使我釐清許多盲點。也感謝這兩年來學校與老師、同學們，給予我一個良好又充滿溫情的求學環境，使苦悶的研究生活多了幾分亮麗、愉悅的色彩。

最後要感謝的是我的家人及女友，你們不但感受了我在論文寫作過程中的壓力，更不時給予我最大的關懷與支持，在此更向你們致上最高的敬謝之意。尤其是家人寬厚的諒解與鼓勵，使我在服喪期間得以定下心來，返校完成論文修改作業。最後，謹將此篇論文，敬獻給培育我多年、教我、養我並給予我最大安定力量的父母與奶奶。期勉自己日後有成以不忝所生。