# 國立交通大學

## 資訊學院　資訊學程

## 碩 士 論 文

Object-Oriented Web Services 應用- 解決 JCOM 在大量資料交換效能不佳的問題

Object-Oriented Web Services application - Solves JCOM performance issue in great deal of information exchange

研 究 生：唐啟銘

指導教授：袁賢銘　教授

中 華 民 國　九 十 八 年 六 月

# Object-Oriented Web Services 應用
# - 解決 JCOM 在大量資料交換效能不佳的問題
# Object-Oriented Web Services application
# - Solves JCOM performance issue in great deal of information exchange


研 究 生：唐啟銘　　Student：CHI-MING Tang

指導教授：袁賢銘　　Advisor：Shyan-Ming Yuan

國 立 交 通 大 學

資訊學院　資訊學程


碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements for the Degree of

in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

# Object-Oriented Web Services 應用
## - 解決 JCOM 在大量資料交換效能不佳的問題

學生：唐啟銘　　指導教授：袁賢銘

國 立 交 通 大 學　　資 訊 學 院　　資 訊 學 程 碩 士 班

## 摘　要

根據統計資料，在全球前 1000 名的企業中，約八成以上都混合使用了 Java 和 Windows 技術，早期為了幫助這些企業解決 java 及 COM +元件之間的溝通的問題，許多中介軟體因而產生(例如 : bridge2java，J-Integra for COM，Weblogic JCOM…等)。其中，JCOM 因為整合在 Weblogic Server 中，所強調的是提供方便直覺的開發架構，更是被許多程式開發人員所接受。但近年來，隨著企業內的資料量愈來愈大，我們發現以往使用 JCOM 做為異質資料交換系統的執行效率愈來愈差。

因此我們希望能找到一種架構或方法，即能保留原來 JCOM 的直覺式開發的方便性，讓 COM+的開發人員享受 java 物件導向的優點，又能解決效能的問題。而這篇論文就是探討目前最熱門的 Object-Orient Web Service 架構與 JCOM / DCOM / COM 之間的差異性，以及嘗試用 Object-Oriented Web Service 來解決 JCOM 在大量資料交換時效能不佳的可能性。

關鍵字 : Object-Oriented Web Services, Web Services, Java to COM, DCOM, JCOM Bridge

Object-Oriented Web Services application
- Solves JCOM performance issue in great deal of information exchange

Student：Chi-Ming Tang　　Advisor：Shyan-Ming Yuan

Degree Program of Computer Science
National Chiao Tung University

# Abstract

Based on the statistical data, in the global first 1000 enterprises, approximately over 90% mixed has used Java and the Windows technology, to solve java and the COM + part's communication question, many intermediary software therefore produced (e.g.: bridge2java, J-Integra for COM, Weblogic JCOM…And so on). And, because JCOM the conformity in Weblogic Server, the convenient intuition's development construction, is accepted by many program development personnel. But in enterprise's material quantity is momentarily getting bigger and bigger, we found that the performance of JCOM is getting more and more bad. Therefore we hoped that could find one kind of construction or the skill namely could retain original JCOM the intuition type development conveniences, could also solve the potency problem. This paper is the discussions present with at present most popular Object-Oriented Web the Service construction, whether to solve JCOM in the massive material exchange effectiveness to be able not the good feasibility.

Index Terms: Object-Oriented Web Services, Java to COM, DCOM, JCOM Bridge

# 誌　　謝

首先我要誠摯地感謝我的指導教授袁賢銘博士對我的啟蒙與指導。袁教授從論文題目的確定、研究的方法到比較的方式，都非常用心指導，因此我才能順利完成我的碩士論文。在論文撰寫的這段期間，非常感謝青峰、琨耀學長及其他實驗室的學長學弟們，謝謝你們不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，提供寶貴的意見，使得本論文能夠更完整而嚴謹。

另外, 我要感謝幾位和我一起努力奮鬥的同學大衛, yummy, 富文...，回想這三年來,每當考試期間，我們總是相約在讀書館 k 書，偶而還會相約去吃大餐，有你們這群好朋友，讓三年研究所的生活增添不少歡樂回憶!

最後，我要感謝我的父母及即將畢業妹妹。你們的支持及鼓勵，是趨動我完成學業的動力之一!同時，我特別要謝謝在背後的默默支持我的老婆，每當我最無力、最需要有人依靠時，你總是在我身邊，耐心地安慰我，鼓勵我。在我論文忙得焦頭爛額之際給予我最大的包容, 我能無後顧之憂專心在論文寫作上。

如今我終於完成了這份論文了，謝謝那些曾經幫助我的師長、同學及家人們，這份榮耀應該是屬於你們的。

# 目　　錄

# 圖目錄

# 表目錄

# 第一章 緒論

## 研究背景

長久以來,Java一直許被多大型企業當作發展企業應用解決方案最佳的分散式平台，尤其是建構分散式伺服器的應用，而當中又以21世紀初提出的Enterprise JavaBeans (EJB)架構最具代表性。

什麼是EJB呢?EJB是實現3-Tier架構(將系統切分為使用者介面層(Presentation)、企業邏輯層(Business logic)及資料庫(Database))的一種方法。系統開發人員將企業邏輯程式寫成EJB元件，然後Deploy到EJB Container裡,藉由EJB的伺服器端元件架構以及分散式物件的技術(如CORBA及以Java RMI)可以減少Client端和Server邏輯之的耦合度。Client端應用程式透過RMI(Remote Method Invocation)即可呼叫遠端的EJB，而EJB則透過Connection Pool連結到資料庫。

但是JAVA最讓企業頭痛的部份,在於開發使用者介面應用程式所耗費的時間及資源，遠大於其他開發工具(Visual Basic，Delphi，C++…等)。所以在過去大多數的企業還是偏愛以其他程式語言開發使用者介面,尤其以半導體廠內的系統更是大宗(半導體廠內的系統多為封閉式的系統，無法採用Web Application架構)。因此才發展出現目前一般企業常見的架構(以VB、C++或是Delphi呼叫 EJB的分散架構)。

但Java及Windows Com+是兩種完全不同的平台，為了能讓兩者之間能進行溝通或是資料交換，許多軟體供應商開始提出一些解決方法及架構(例如 : WebLogic提供的JCOM技術，bridge2java(IBM提供的基於Java本機接口和COM技術)，J-Integra for COM…等)。

當時，Bea Weblogic提出JCOM的技術，由於提供開發人員在異質開發平台上直接使用JAVA物件結構，並且將其整合於Weblogic Server中，提供完整跨平台系統架構，獲得許多大型企業所採用，也因此廣泛被應用在企業中。

## 研究動機

近年來，許多企業不斷進行平行或垂直整合，系統與系統之間必須處理的資料量及複雜度也不斷地提高。這使得我們漸漸發現原先使用JCOM架構系統的效能

似乎愈來愈差，甚至是無法順利運作。

一般而言，大企業若遇到這類的問題時，會有兩種解決辨法 :(一) 提昇硬體規格來改善效能。(二) 重新開發新的系統來取代現在的系統。但這二種方式一則需要花費大量的金錢，二則需要耗費大量開發資源及時間。

因此，本篇就是探討是否能找到其他解決辨法，使其能在最短的時間用最少的花費來來改善JCOM效能不佳的問題。研究目的

根據Bea Weblogic官方對JCOM架構的定義，JCOM其實是一個介於Microsoft系列應用程式及JVM執行環境的中介軟體。主要是負責將來自於Microsoft COM+或JAVA的Request轉換成另一端系統所支援的傳輸協定並傳送至對方，也就是COM/DCOM及Java RMI之間的資料內容的轉換，這正是為什麼Microsoft或JAVA開發人員，能直接使對方元件/物件的主要原因。

雖然JCOM提供如此直覺化的方便性，但是JCOM的缺點在於一旦開發人員大量使用這種技術存取雙方物件/元件資料內容時，可能會因為JCOM Runtime Engine負載過重，導致執行效能不佳。

以目前的技術而言，只要談到異質資料交換機制，第一個連想到應該是Web Services。Web Services主要的優點是利用XML(text-based)作為資料交換的內容，而且它是一次性的傳輸，不需要浪費時間處理COM/DCOM與JAVA-RMI之間的轉換。所以才會有了這樣子的構想，希望用Web Services來解決JCOM效能不佳的問題。但Web Services是以Text為傳輸內容，省去了異質平台資料轉換的時間，但必須分別在Server 及Client兩端執行資料格式的轉換(Object <->XML)及透過HTTP傳送的資料封包。這兩個步驟是否會產生反效果呢?這正是本篇文章所探討的內容之一。

另外，假設以Web Server為中介軟體的架構其執行效能真的比JCOM的方法好。但我們要如何將Web Services導入即存系統中，卻又能保有JCOM提供系統開發人員直覺式使用JAVA物件的優點，則是本文中要探討的另一個重點。因此，本研究的目的有下列幾點 :

1. Web Services vs. JCOM的效能比較。
2. 如何以最少的資源，將Web Services導入即存的系統，取代JCOM。

## 論文架構

本研究論文架構共分成五個章節，分述如下 :

第一章、緒論

在緒論中先敘述論文研究的歷史背景，以及目前企業中遇到的技術瓶

頸，進而產生本篇論文的研究動機。在產生動機因素後，接著就必須找到一個解決方法，而這方法就是本篇論文研究的目的。

第二章、文獻探討

在文獻探討中主要可以分成二小節。第一小節深入討論的現行企業所使用的技術(JCOM)的幾種可能架構及其效能優劣。第二節則是介紹Web Service的基本運作法式以及OOWS的概念。

第三章、研究方法

經由第二章的詳細介紹二種架構的優勢及缺點後，在本章節會建置多組實驗性的專案，依專案性質可分為實驗組及對照組，對照組是仿照目前企業常見的幾種JCOM軟體架構，而實驗組則是本篇論文討論的以Web Services主的架構。接著會依據每組實驗的結果驗證本論文的可行性。

第四章、實際專案導入

這個章節會藉由真實案例，說明企業目前確實有面臨如此的技術瓶頸以及如何導入本論文中的架構來解決問題，用來加強佐證本篇論文的動機及價值。

第五章、結論與建議

本研究的最後章節，將對整篇文章作最後總結並且提出個具體的方法供許多企業參考使用。最後則是點出本篇論文未來可能遇到的情況與未來後續研究的建議。

# 第二章 文獻探討

## 2.1 JCOM 架構及相關技術

### 2.1.1 何謂JCOM Bridge

JCOM (Java for COM)，簡單來說就是一種中介軟體，藉由這個軟體架構，可以讓 Java 程式能使用 COM+、ActiveX 控制項、DLL 以及 EXE 等 Microsoft 元件，也可以讓 VB、VC++、ASP 程式能執行任何 Java objects 及 EJB。換句話解釋，就是無障礙的軟體開發架構。

- COM-to-Java：對於 COM 開發者而言，可以透過 JCOM 直接使用 JAVA object 的元件，享受件導向的方便性〔43〕。
- Java-to-COM：對於 Java 開發者而言，也可以透過 JCOM 直接使用 COM 元件〔43〕。

### 2.1.2 JCOM Bridge的運作架構

如下圖所示，JCOM 是一種建構在 JVM(Java Visual Machine)平台上的一個軟體。JCOM 主要功用在於兩種類型的訊息傳遞之間轉換(COM/DCOM vs. Java RMI)。

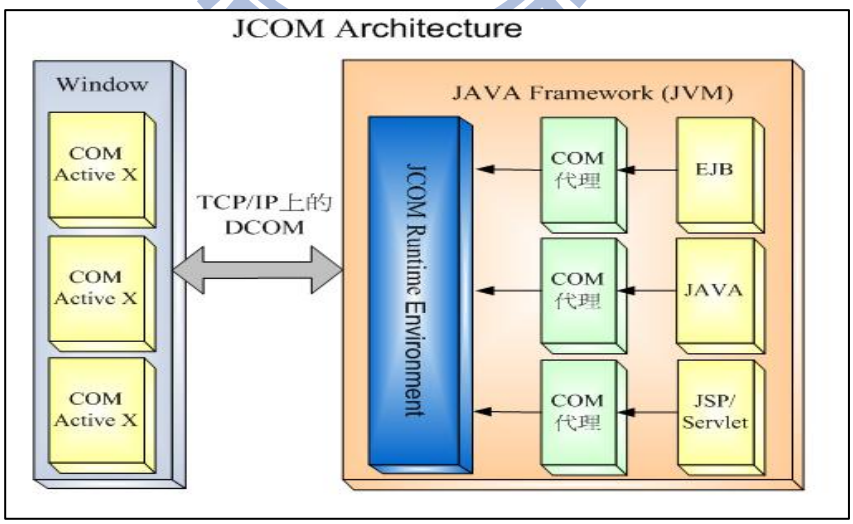



圖 1 JCOM 最基本架構圖

它會自動建立每一端通過上述協議進行通信所必需的 COM/DCOM 代理和 RMI 存取控制元件的對應關連。當 COM 元件要使用 JAVA 物件時，會透過 COM/DCOM 對 JCOM 發出 Request，JCOM 在接收到 Request 後,會將其 Request 轉換成通過 Java 遠程方法協議/Internet Inter-ORB Protocol 分

散式組件基礎結構實現遠程方法（Remote Method Invocation，簡稱 RMI）的方式，然後再將 Request 發送到 Enterprise Java Bean (EJB)，如此一來就像請求來自一般 EJB 客戶端一樣。

同樣地，當 JAVA 要使用 COM+元件時，會先向 JCOM 發出請求，JCOM 會將來 Request 轉換為符合 DCOM 的 Request，並發送給 DCOM 環境中的相應元件。如此便能達到所謂的無差異性的開發環境，無論是 Java 或 Windows 開發人員在使用上都像是在使用一般元件一樣方便。

Weblogic 這家公司為了提供企業一個完整的跨平台解決方案，除了上述的基本 JCOM 架構外，另外也有專為提昇效率的解決方案。但在正式介紹 Weblogic JCOM 前，必須了解以下幾個觀念 ：

- Late Binding vs. Early Binding: 當 JAVA 物件轉換成 COM 物件前，是否要先產生對應的 COM class。也就是說若採用 Early Binding 的技術時，必須借由 Weblogic 提供的工具，先將 JAVA 物件先轉換 IDL(Interface Description Language)後，再由 Microsoft MIDL 工具將 IDL 文件轉換成 COM 支援的 TLB 檔。如此一來就是在程式在 Compile 階段就知道程式是否正確。反之 Late Binding 則不是不需要進行任何物件 Interface 的轉換，Compile 階段不會檢查物件的正確性，而是在執行時才和 JCOM-Bridge 確認是否無誤。一般而言，Early Binding 的執行效率會比 Late Binding 來的好。
- In Process vs. Out Process: 這是用來設定 JCOM JVM 的執行是否獨立於 COM Process 中。Weblogic 為了改善透過 DCOM 執行效能的問題，提出了另外兩種方式來改善效能。以下這兩種做法有一個共同點，就是必須在客戶端執行 JCOM-Bridge。
  (1) Native (Out Process)：如下圖 2 ： 這種方式是把 JVM 移至客戶端機器上執行 JCOM Bridge，但是由於是透過 COM 來執行資料交換，所以執行速率預期會比用 DCOM 的方式快上一些。
  (2) Native (In Process)：如下圖 3 ： 這種方式一樣是把 JVM 移至客戶端機器上執行 JCOM Bridge，但 Weblogic 提供了 JVM.dll 的 library，讓 COM+程式執行時能一同將 JVM Bridge 載入。

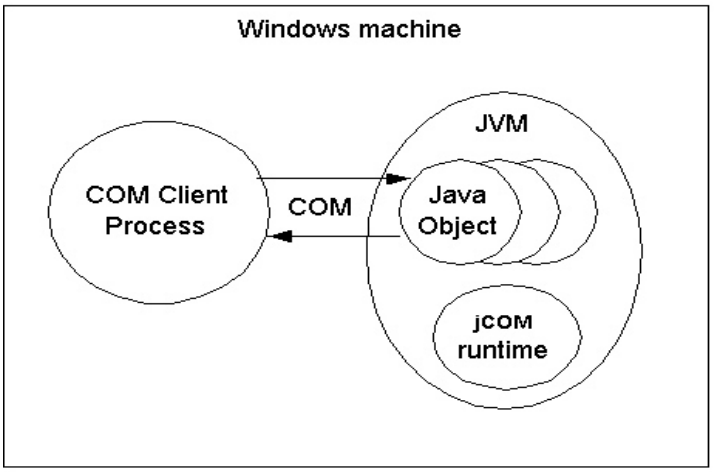



圖 2 Out Process 架構

資料來源：Web Logic 官方網站

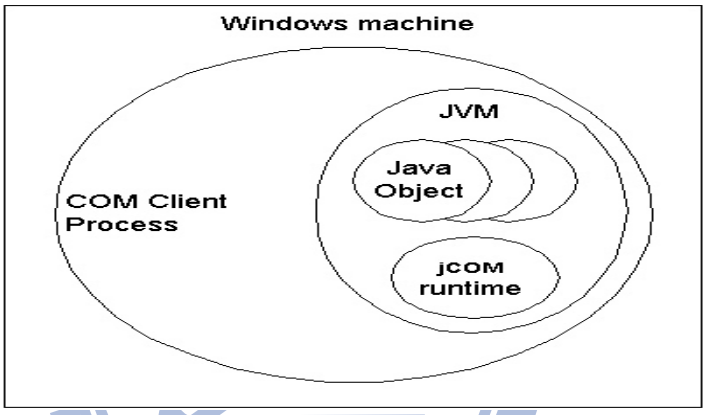



圖 3 In Process 架構

資料來源：Web Logic 官方網站

以下是我們歸納出幾種 JCOM 比較具代表性的架構 ：

1. Zero Client、DCOM 模式架構
2. Native Mode + Out Process + 後期綁定(Late Binding)
3. Native Mode + In Process + 前期綁定(Early Binding)

### 2.1.3 Zero Client、DCOM模式架構

Zero Client，就是在客戶端不需要任何額外的設定或載入 JCOM 程式就可以使用 JCOM 的架構。JCOM Zero Client 的部署很容易實現，也是企業內最普遍使用的方法之一。這種架構使用物件引用標記(objref)。Objref 字串是由 Weblogic Server 的 IP 及 Port 進行編碼產生的。Client 端程式只要取得這一字串，便可和 Server 產生連結物件(Objref)，當取得連線後，Objref 物件就如同在 COM Client 端產生對應 java 的 interface 一樣，COM+程式可以直接對 objref 執行 Java 的 Methods。

DCOM 模式和 Zero Client 模式類似，差別在於 Objref 的取得方式是由

在 Client 端必須註記 Server 的 IP 及 Port。

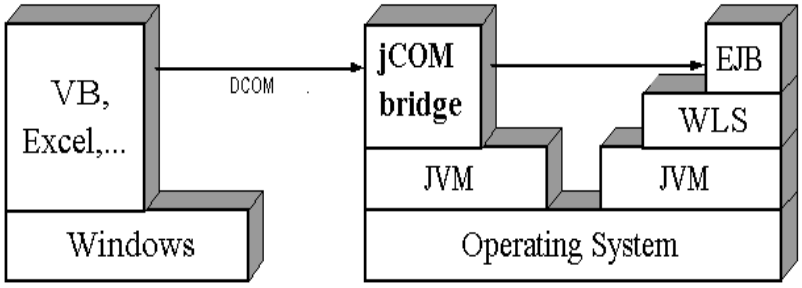



圖 4 Zero Client Mode

資料來源：Web Logic 官方網站

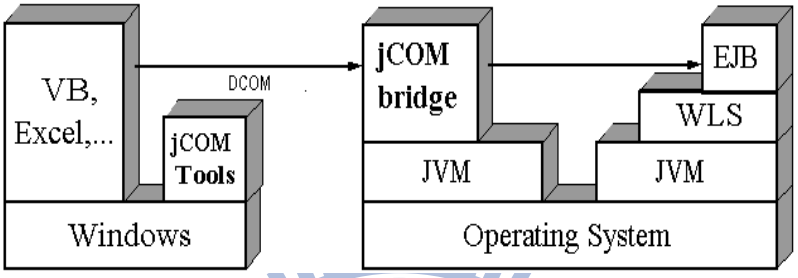



圖 5 DCOM Mode

資料來源：WebLogic 官方網站

這種架構的好處在於客戶端無需任何額外的軟體或設定，開發人員可以很直覺式的使用 Objref 就好像在使用 java object 一般。 但缺點是不適合在用大量資料交換，因為每當 com 對 Objref 讀取一次資料時，就如同透過 DCOM 連結遠端 JCOM ，再經由 JCOM 透過 RMI 的方式取得 JAVA 值後，再回傳給 COM Client 端。如下圖 6 所示：

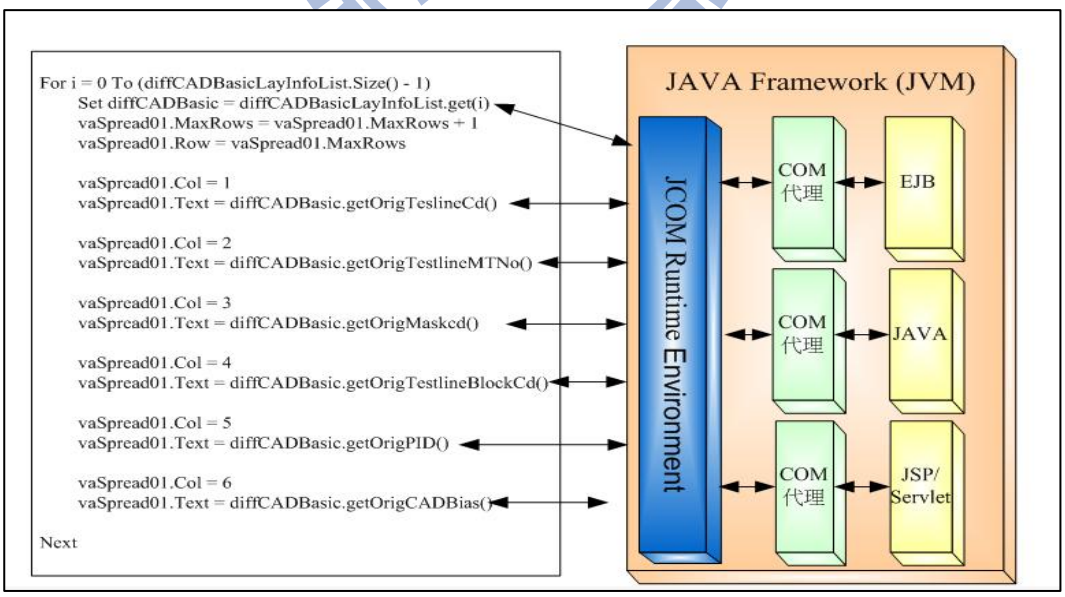



圖 6 Client 每次存取 Object 物件，都必須往返 Server

可以想像，當程式大量使用 JCOM，需花費大量時間及資源在 Java & Com+ 物件之間的轉換，導致 Server 的 Loading 急速增加，必然會造成 Client、

網路流量、Server 端相當大的 loading。

### 2.1.4 Native Mode + Out Process + Late Binding 架構

所謂的 Later Binding 指的是 COM+元件在 Compile 時不檢查有關所訪問 Object 的是否有效；而是在運行時對所訪問對象進行即時確認。這意味著，直到執行程式時才能確定所訪問的方法和屬性是否確實存在。和 Early Binding 比較起來，因為必須才能透過 JCOM 確認 Interface 是否正確，所以執行性能上不如 Early Binding 好。

所謂的 Native Mode(本地模式)指的就是 Client 模式，也就是將 Weblogic library 複製一份到 client 端，並且在 Client 端架設 JCOM Bridge。一般企業比較少使用本地模式。

由圖 7 中得知，本地模式和 DCOM 模式最大的差異性在 JCOM 所在位置由 Server 端改成本機(Client)端，而且是透過 COM 來進行資料交換。 由下圖中，我們可以很明顯看出差異性。簡單來說本地模式是為減少網路流量及 Server loading 而生的架構。

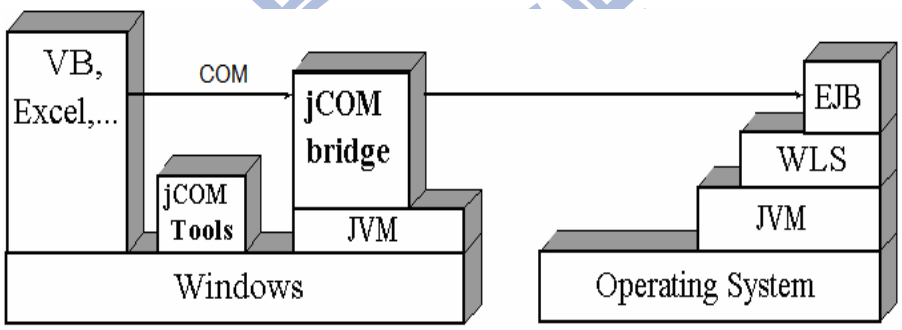



圖 7 Native Mode + Late Binding
資料來源：Web Logic 官方網站

### 2.1.5 Native Mode + In Process + Early Binding 架構

就官方的說法，這個架構是 Weblogic JCOM 中最有效率，但也是最難實做、最複雜的架構。其採用的是 Early Binding 的方式，產生本地模式使用專為本機操作系統和 CPU 編譯並優化的本機端的 DLL，配合上本地模式的應用，可提高性能。不僅如此，在本地模式下運行的 COM-to-WLS 應用程式使用 WebLogic 的 T3 / Internet InterORB (IIOP) 協議在 COM 客戶端和 WebLogic Server 之間進行通信。因此可提供以下優點

- 可減少網絡調用次數，所以與使用 DCOM 調用相比可提高性能。例如，假設您的 VB 應用程式存取 java vector 物件時，必須讀取 1000 個通過使用 WebLogic Server 返回值的次數。在 DCOM 模式下，該過程需要對服務器進行 100 次往返網路使用。在本地模式

下，則僅需要一次往返調用。〔43〕

- 採用 In Process 的方法，能直接使用 JVM 的物件，更能增加執行速度。〔43〕
- 可使用 WebLogic Server 的故障轉移和平衡 loading 的功能。

以下是 Early Binding(前期綁定)的步驟 ：

1. 先將 JAVA(EJB)的程式，透過 Weblogic 提供的 java2com 的工具，轉成 IDL 檔，
2. 再將 IDL 覆製到 Client 端，並且用 C++提供的 MIDL 工具，把 IDL 轉換成 TLB 檔，
3. 最後必須將 TLB 載入到客戶端系統上的客戶端應用程式。

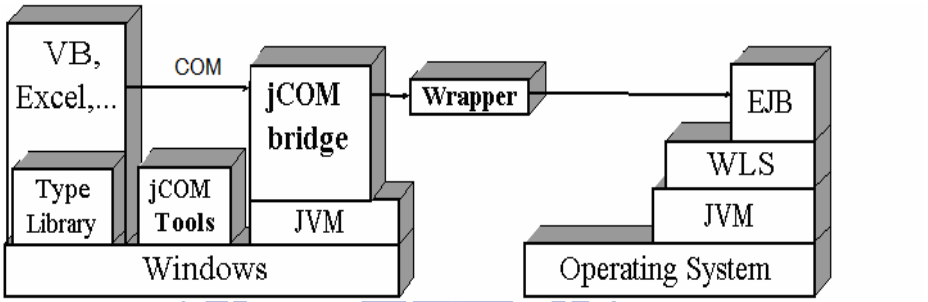



圖 8 Native Mode + Early Binding
資料來源：Web Logic 官方網站

### 2.1.6 JCOM架構模式的優劣

無論是 Zero Client/DCOM 模式或是 Native and Early Binding 模式，其最大共同的優點就是對系統開發人員而言，無論是 COM-to-WLS 或是 WLS-to-COM，都可以用直覺式的方法調用對方的元件。

但就執行的效能來說，Zero Client/DCOM 模式，但只適用在 Remote Procedure Call 的應用上，不適用在大量資料交換上的應用。而 Native and Early Binding 模式，雖然比 Zero Client/DCOM 來的有效率，但仍然有一先天上的缺點及瓶頸 ：

1. JCOM 程式必須建置在 Client 端
2. 必須先將 Java Interface 轉換成 Windows Library，而且在所有 COM 客戶端計算機上安裝 WebLogic Server Library 才能順利執行。
3. 在 Native Mode 中，雖然可以直接透過 COM/DCOM 和 JCOM 在本機上溝通，但在大量資料轉換過程中，仍浪費許多時間處理資料轉換的問題。

因此我們嘗試用其他的方法來克服以上問題。其中的方法就是用 Object Orient Web Service 來替代 JCOM。

## 2.2 Web Services 架構及相關技術

在解釋什麼是OO-WS之前，必須先了解什麼是Web Services。Web Services是一種軟體元件，它透過Web http 通訊協定及資料格式的開放式標準(如 HTTP、XML 及 SOAP等)來為其他的應用程式提供服務，達到跨平台的能力。Web Services主要可分為三大主題。 在這三大主題中，我們比較在意的是WSDL和JCOM之間的差異性，所以我們會針對WSDL部份加以詳細解說，其餘部份只會大略介紹。

- UDDI (Universal Description Discovery and Integration)：
  提供註冊與搜尋 Web Service 資訊的一個標準。主要是定義 Web services 位置的 pointer。〔46〕
- WSDL (Web Service Description Language):
  是一項以 XML 為基礎的技術，它描述一個 Web Services 的網路服務介面、資料和訊息型態、以及指示用戶端與它可能的互動方式和轉換協定。〔46〕
- SOAP (Simple Object Access Protocol):
  是一項以 XML 為基礎的技術，主要用於定義路服務通訊的 envelope，能夠對應到 HTTP 或是其他傳輸協定，並且提供 XML 文件在網路上的傳輸，一個連續化的格式，和 RPC 互動的協定。〔46〕

### 2.2.1 UDDI

UDDI (Universal Description Discovery and Integration)起初是由微軟、IBM 和 Ariba..等幾家大型企業所共同參與推動的! 它的概念有點類似 DNS(網域名稱服務)。企業必須先向任何 IBM、HP、Microsoft..等的主機註冊 Web Services 相關資訊，有需要這些網路服務的 client 應用程式，會登入這些主機查詢有那些 web services。它也有點類似目前網路搜尋引擎的做法，在於讓使用者能快速尋找到適合的網頁內容。

UDDI 有兩個主要的部份: 註冊、搜尋。註冊的部份指的是企業發佈資訊到提供 UDDI 註冊的主機上。讓其他的企業可以搜尋及發現到它的資訊。而搜尋是另一個主要的部份，企業和個人利用 SOAP 的 API 或是其它網路服務廠商所提供的使用者介面，與 UDDI 主機互動，查詢資料。當用戶找到適當的服務後，可以直接由 UDDI Server 下載 Web Services 的 WSDL 文件內容。

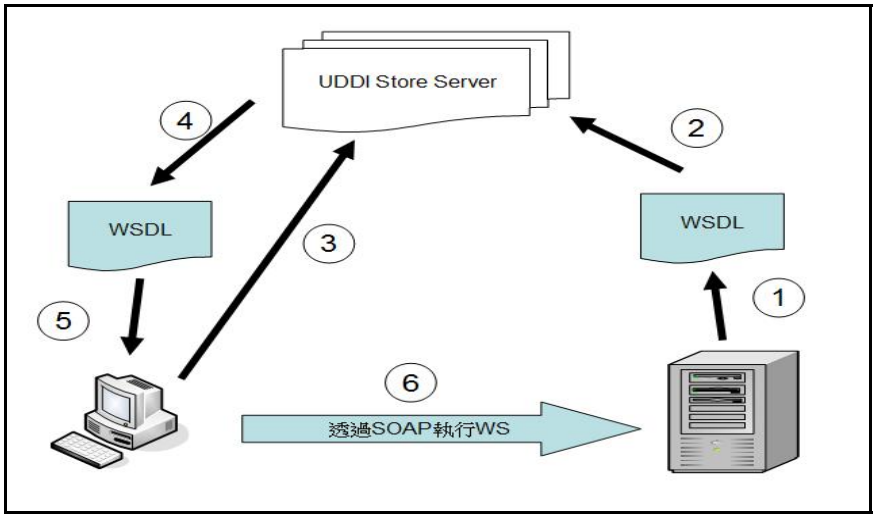



圖 9 藉由 UDDI，Client 可以找提供服務的 Server

資料來源 ： 深探網路服務 黃義焜譯

上圖顯示一個企業如何利用 UDDI 的註冊機制來註冊 Web Services 的 WSDL 資料。(1)企業首先產生 WSDL 檔案及描述 SOAP 支援的 Functions (2)接著利用 UDDI Server 提供的 API 向 Server 註冊 WSDL 及功能 (3)其它企業客戶端應用程式可透過 SOAP 向 UDDI 主機進行查詢功能 (4)UDDI 主機會依照查詢的結果，產生 WSDL schema (5)並回傳給 Client 應用程式 (6)最後，當 Client 端程式取得相對的 WSDL 文件及主機位置，就可以透過 SOAP 向遠端的 Web Services Server 發出 Request 了。

### 2.2.2 WSDL

WSDL 其實是一種 XML Schema 的格式，差別在於 WSDL 格式是經過許多大型廠商支援的標準格式。WSDL 內容如下圖所示，主要可分為以下二類定義(六個子元素)：如下圖所示 ：

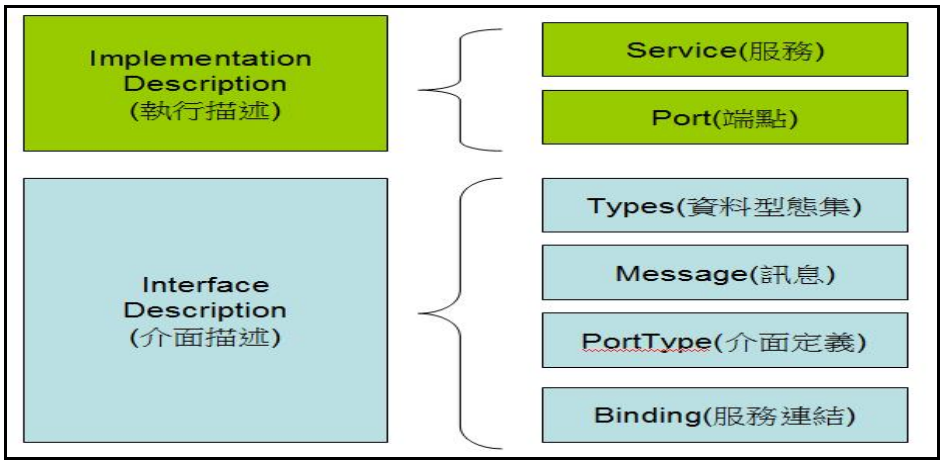



圖 10 WSDL 主要元素分類表

資料來源 ： 深探網路服務 黃義焜譯

- 介面描述(Interface Description)〔45〕：

是用來定義 WSDL 內容資料格式、傳遞溝通的訊息，以及進行作業的方式。包含以下四個元素。

- Types(資料型態)〔45〕：

  Web Services 需要定義它的輸入和輸出， 以及他們如何與其它的服務作轉換，WSDL 的型態負責處理這些事。型態可以是 XML 文件，或是文件的元素。型態一般是利用支援資料型能的 XML schema 來定義，例如 int、float、long、short、byte、string、boolean、date...等，而且可以包含複雜的型態，像是結構和陣列，也包含這些為 SOAP 定義的部份。

範例 ：

```
<wsdl:types>
<xsd:schema….>
  <xsd:element name="division">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="in0"
type="xsd:int"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="in1"
type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
…
</wsdl:types>
```

- Messages(訊息)〔45〕

  Message 代表網路服務的細節抽象概念，用於表達輸入、輸出、或錯誤等操作用訊息，每個訊息內包含 1~n 個 part 元素。在訊息段中，主要是說明有那些資訊需要傳輸，使用 XML Schema 或 Types 所提供的類型來定義整個訊息的資料結構。

範例 ：

```
<wsdl:message name="substractResponse">
    <wsdl:part name="parameters"
element="tns:substractResponse">
    </wsdl:part>
  </wsdl:message>
```

- PortType(介面定義)〔45〕：

  主要是定義一連串傳送收的操作(Opteration)，並對應要傳送的訊息，可視為整個網路服務的縮影，它是用抽象的方式來指出這個 web service 能提供什麼功能，因此當做為檢視 WSDL 文件的起點。其依據 input/output message 組合不同，可區分四個主要的傳輸方式(Solicit-Response、Notification 尚未支援)，如圖下所示 ：

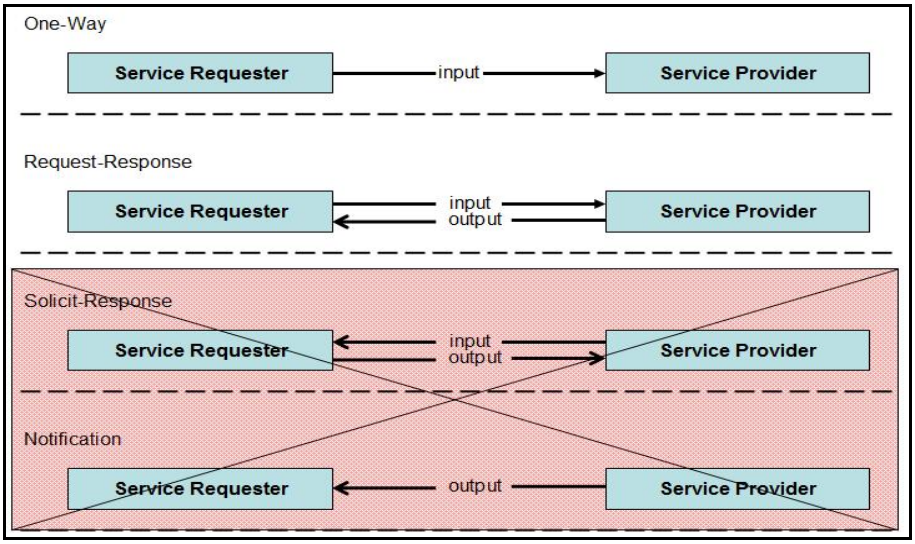



圖 11 WSDL Port Type 型式

範例 ：

```
<wsdl:portType name="AdvancedMathPortType">
    <wsdl:operation name="division">
<wsdl:input name="divisionRequest"
message="tns:divisionRequest">
    </wsdl:input>
      <wsdl:output name="divisionResponse"
message="tns:divisionResponse">
    </wsdl:output>
    </wsdl:operation>
</ wsdl:portType>
```

■ Binding(連結)

用於定義服務傳輸協定及相關資料型態，清楚定義由 Port-Type 及 Message 所提到的資料元素部份。簡單來說，就是用來定義服務傳輸協定、相關資料型態及提供之 Operations。Binding 因為使用的協定而有三種不同的方式 (SOAP Binding、HTTP Get/Port Binding、MIME Binding)。下章節會介紹什麼是 SOAP..

範例 ：

```
<wsdl:binding name="AdvancedMathHttpBinding" …>
    <wsdlsoap:binding style="document" …/>
    <wsdl:operation name="division">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="divisionRequest">
        <wsdlsoap:body use="literal"/>  </wsdl:input>
      <wsdl:output name="divisionResponse">
        <wsdlsoap:body use="literal"/>  </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

- 執行描述(Implementation Description)〔45〕:

主要是說明如何執行由服務介面所提供的服務。包含以下二個元素。

- Service(服務)

  Service 具體描述一個網路服務所有可能之作業端點的細節。服務允許特定的 Client application(遠端應用程式)，依據 Web Service 所提供的 Service 進行互動。

- Port(服務端點)

  Port 是 Service 元素的子元素，它不能單獨存在。Port 主要是用來描述資料傳輸時，提供服務的 end point 位置(URL)，因此每個 port 包含了唯一的協定及網址。每個 port 代表外界可以和 service 此溝通的一個進入點。

範例 :

```
<wsdl:service name="AdvancedMath">
    <wsdl:port name="AdvancedMathHttpPort"
binding="tns:AdvancedMathHttpBinding">
      <wsdlsoap:address
location="http://localhost:7001/WebServicesToEJBApp/Advanced
Math"/>
    </wsdl:port>
  </wsdl:service>
```

### 2.2.3 SOAP(Simple Object Access Protocol)

簡單物件取存協定(SOAP)，簡單來說就是在網際網路環境之下將溝通方式標準化的通訊規範。它是一個獨立的訊息，可以獨自運作在不同的作業系統與網路上面，例如在微軟的 Windows 或 Linux 的建構下運作，並可以使用各種不同的通訊方式來傳輸，例如 SMTP、MIME，或是 HTTP 等。但大部份是以 HTTP 傳輸協定來傳遞 XML 文件格式的內容。只要 Client 端能夠製作及傳送 SOAP 訊息給 Server 端，而 Server 端也能夠解讀 Client 端所發出的訊息，進而執行其需求並做出回應，即可以透過 SOAP 完成動作。

SOAP 的主要用途是使用在分散式環境下，不同平台之間可以由文字格式(Text-based)方式溝通，使用應用系統可以互相溝通。換言之，SOAP 作用是編譯網路服務所需的要求或回應後，再將編譯後的訊息送出到網路，簡單來說就是應用程式和用戶之間傳輸資料的一種機制。在本篇論文裡，SOAP 層的傳輸機制，是交由 VB 及 JAVA AP 層來負責。

SOAP 的 XML Schema 架構可分為 Envelope、Header、Body 三個部份；其組織架構是與 XML 的語法相結合應用。以下範例分別是 Web Services 借由 SOAP 發出的 Request 及 Response XML 格式內容 :

範例 :

```
<S:Envelope …>
```

```
    <S:Body>
        <a:add>
         <a:in0 XI:type="XS:int">1</a:in0>
         <a:in1 XI:type="XS:int">2</a:in1>
        </a:add>
    </S:Body>
</S:Envelope>
```

## 2.3 Object-Oriented Web Services 介紹

介紹完 Web Services 後，接下來就要介紹何謂 Object-Oriented Web services。所謂 Object-Oriented Web services(物件導向網路服務)，就是藉由物件導向的設計方法，先設計物件後，再將物件轉換件 WSDL 文件的概念，也就是所謂的 Contract Last 的方法。

上章節中，我們知道 Web Services 最重要的一個步驟是產生 WSDL 文件內容，我們可以想像一下，如果能將 Binding 定義成自動透過 SOAP 呼叫 EJB 的方法，並且 EJB Method 的參數及回傳的 Java 物件定義 XML 格式內容，如此一來便能達到使用 XML 來傳來傳遞資料了(圖 12)。

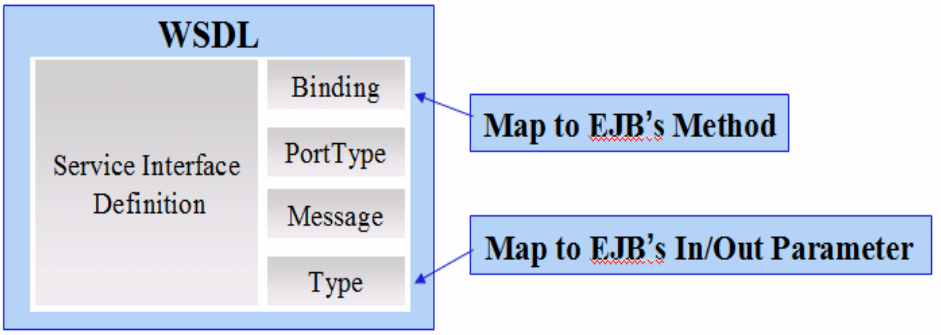



圖 12 Object-Oriented Web Services vs. EJB 的概念

但是 JAVA 及 EJB Interface 物件導向資料結構的複雜度，如果要手動產生出 WSDL 文件中的 Binding (Web Service 提供的 SOAP Method)，及 Types(xml 資料結構)的定義，反而會讓開發人員怯步。所以只能借助 OO-WS 的技術來產生 WSDL 的文件。

一般公司或企業通常將 Web services 應用於電子商務以及 B2B 與 EAI 系統中，而資訊人員而在設計這類型專案時，通常會先專注在 Web services 介面的定義以及溝通方式，接著才依據這 XML 內容來開發 JAVA 或是 COM 的程式。 這是所謂的 Contract-first 的開發流程。反之若開發人員先設計資料交換的方法及物件，就是 Contract Last 的作法。目前在 JAVA 的開發環境裡，目前有幾種以 Contract Last 做法為基礎的技術。

### 2.3.1 JSR-181

JSR-181(Java 平台網路服務 Metadata)是由 BEA 提出的用於加速 Web Services 開發的一種基於 Java annotation 的程式設計模式，能協助軟體開發人員在 Java 平台上更快速的佈建網路服務，進而有效降低企業軟體成本與複雜度〔43〕。只要編寫程式時加入 annotation 就可以制定 WSDL，消息產生屬性，安全認證方式。目前已被納入到 J2EE 1.5 標準。範例如下 ：

範例〔43〕：

```
@WebMethod(operationName="toLower")
public String toLowerCase(String lowerReqString) {
    String lower = lowerReqString.toLowerCase();
    System.out.println("toLowerCase Request: " + lowerReqString);
    System.out.println("toLowerCase Reply: " + lower);
   return(lower);
}
```

### 2.3.2 POJO

POJO(Plain Old Java Object)，又稱簡單 JAVA Object。這個關念有點類似 JAVA 的 VO or TO。簡單來說就是只定義屬性，不具任何程式邏輯的 Java Object。藉由 POJO 的而定義 JAVA Object 可被用來轉換成 WSDL 裡的 data type。範例如下 ：

範例:

```
public class EmployeeEai implements java.io.Serializable {
   public String empid;
   public String getEmpid() {
        return this.empid;}
    public void setEmpid(String empid) {
        this.empid = empid;}
}
```

Object-Oriented Web Services 的優點是什麼呢?我們可以想像一下，如果系統同時需要多個且內容複雜的 WSDL 介面，且每個 WSDL 又由許多內容時相似時，對系統設計人員必定是一大負擔。但若換個角度想考，現在專案的開發都強調物件導向的設計理念，若設計人員用物件導向的角度來設計資料交換的內容，先定義物件，再由物件內容來產生對應的 WSDL 文件，這樣不但保有原來物件導向程式設計的優點，也可以減少許多因為修改 WSDL 時造成的困擾。許多物件導向設計上的優點，在 Web services 上仍然可以使用的。例如 Web services 繼承的應用…等。如下範例 ：

範例 ：

假設我們實作了一個 Web services 具有加/減法的功能。首先我們必須建立一個 Java Class，這個 Class 必須是 Public，在 Class 中宣告 add 及 subtract 兩個 public method，接下來我們只需要透過一些輔助程式就將這些定義為 public method 及 class 轉換成 WSDL 文件。

但值得注意的是，JAVA 程式是具有繼承、封裝的導向設計的概念，因此，若有些 method 是不具有提供外部存取的需求時，我們可以將這些程式或類別宣告為內部使用(private)的 method。就如同下方程式碼中的 Public int Add(int n1, int n2) 方法，若將這個 Method 移除或是將 public 改成 private，那麼這個 method 就不會出現在 WSDL 服務描述中，這時候 Add 方法，只能提供給內部或其他程式、類別，以物件方法呼叫的方式使用，當然，若將 Public 改為 Private 則只有物件內部可以使用了。

Math.java

```
package org.edu.nctu.webservice.interfaces;
public interface Math {
     public int add(int n1，int n2) ;
     public int subtract(int n1，int n2);
}
```

另外，設定我們設計另一個 Class(AdvancedMath.java)繼承了剛才的 Math.java，那麼由 AdvancedMath.java 所產生的 WSDL，也具有 Math.java 的 SoapMethod 的功能。

AdvancedMath.java

```
package org.edu.nctu.webservice.interfaces;
public interface AdvancedMath extends Math{
     public int multiplication(int a，int b);
     public int division (int a，int b) ;
}
```

```
- <wsdl:binding name="AdvancedMathHttpBinding" type="tns:AdvancedMathPortType">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="division">
      <wsdlsoap:operation soapAction="" />
    + <wsdl:input name="divisionRequest">
    + <wsdl:output name="divisionResponse">
    </wsdl:operation>
  + <wsdl:operation name="add">
  - <wsdl:operation name="subtract">
      <wsdlsoap:operation soapAction="" />
    + <wsdl:input name="subtractRequest">
    + <wsdl:output name="subtractResponse">
    </wsdl:operation>
  + <wsdl:operation name="multiplication">
  </wsdl:binding>
```

# 第三章 OO-WS vs. JCOM的差異性實驗及分析

## 3.1 簡介

在第二章我們分別介紹過 JCOM 及 OO Web Services 架構及運作方法，但為了能真正了解 WS vs. JCOM 之間的優劣及差異性，我們分別實作了以下幾種資料交換架構的測試。

（一） 以 JCOM - Zero Client 模式為中介軟體的實驗

（二） 以 JCOM Native Out Process + Late Binding 模式為中介軟體的實驗

（三） 以 JCOM Native In Process + Early Binding 模式為中介軟體的實驗

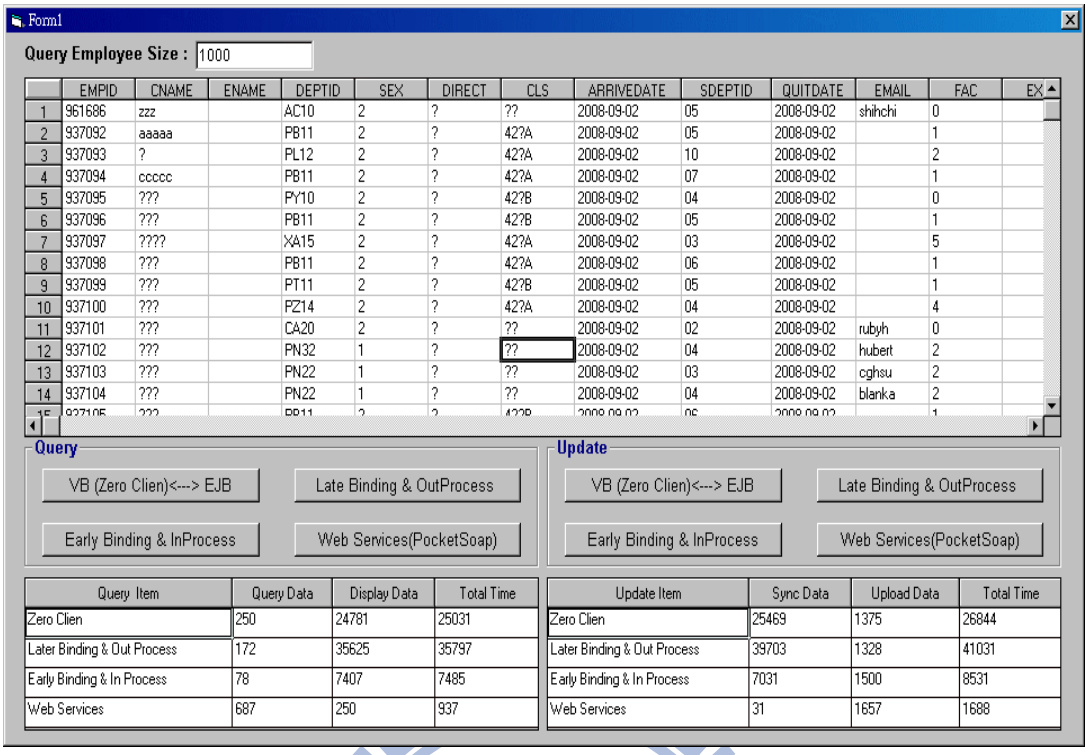
（四） 以 Object-Oriented Web Service 為中介軟體的實驗

該實驗是假設某公司要求開發人員必須開發一個查詢/修改員工基本資料的介面為範列。軟體架構是以 Windows 系統平台為執行環境，以 Microsoft Visual Basic 6 開發客戶端應用程式透過 Intranet 向遠端 Weblogic Server 上的 EJB 查詢所有公司員工基本資料，並將查詢結果回傳至使用者介面。操作人員可以直接由介面修改員工基本資料後，並儲存至資料庫。

## 3.2 系統設計

### 3.2.1 使用者操作介面設計

使用者操作介面說明下圖 13 所示。最上方的 Text 為查詢資料筆數輸入畫面，可用來設定查詢資料筆數。下方的 table list 則是呈現的員工資料，使用者可以直接在 table 裡修改資料。中間區域有許多 Button 的，左方 Query 區塊內有四個 buttons，分別透過四種不同的架構查詢資料; 左方的 Upload 區塊內同樣有四個 buttons，則是透過不同的架構回傳資料。左下方的 table 則呈現的是每個架構下查詢資料所需花費的時間:右下方的 table 則呈現每個架構上傳資料所需花費的時間。使用者依據不同的測試架構查詢/修改員工資料時，系統會將所花費的時間記錄在下方的 table 中。




| | EMPID | CNAME | ENAME | DEPTID | SEX | DIRECT | CLS | ARRIVEDATE | SDEPTID | QUITDATE | EMAIL | FAC | EX |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 961686 | zzz | | AC10 | 2 | ? | ?? | 2008-09-02 | 05 | 2008-09-02 | shihchi | 0 | |
| 2 | 937092 | aaaaa | | PB11 | 2 | ? | 42?A | 2008-09-02 | 05 | 2008-09-02 | | 1 | |
| 3 | 937093 | ? | | PL12 | 2 | ? | 42?A | 2008-09-02 | 10 | 2008-09-02 | | 2 | |
| 4 | 937094 | ccccc | | PB11 | 2 | ? | 42?A | 2008-09-02 | 07 | 2008-09-02 | | 1 | |
| 5 | 937095 | ??? | | PY10 | 2 | ? | 42?B | 2008-09-02 | 04 | 2008-09-02 | | 0 | |
| 6 | 937096 | ??? | | PB11 | 2 | ? | 42?B | 2008-09-02 | 05 | 2008-09-02 | | 1 | |
| 7 | 937097 | ???? | | XA15 | 2 | ? | 42?A | 2008-09-02 | 03 | 2008-09-02 | | 5 | |
| 8 | 937098 | ??? | | PB11 | 2 | ? | 42?A | 2008-09-02 | 06 | 2008-09-02 | | 1 | |
| 9 | 937099 | ??? | | PT11 | 2 | ? | 42?B | 2008-09-02 | 05 | 2008-09-02 | | 1 | |
| 10 | 937100 | ??? | | PZ14 | 2 | ? | 42?A | 2008-09-02 | 04 | 2008-09-02 | | 4 | |
| 11 | 937101 | ??? | | CA20 | 2 | ? | ?? | 2008-09-02 | 02 | 2008-09-02 | rubyh | 0 | |
| 12 | 937102 | ??? | | PN32 | 1 | ? | ?? | 2008-09-02 | 04 | 2008-09-02 | hubert | 2 | |
| 13 | 937103 | ??? | | PN22 | 1 | ? | ?? | 2008-09-02 | 03 | 2008-09-02 | cghsu | 2 | |
| 14 | 937104 | ??? | | PN22 | 1 | ? | ?? | 2008-09-02 | 04 | 2008-09-02 | blanka | 2 | |

| Query Item | Query Data | Display Data | Total Time |
|---|---|---|---|
| Zero Clien | 250 | 24781 | 25031 |
| Later Binding & Out Process | 172 | 35625 | 35797 |
| Early Binding & In Process | 78 | 7407 | 7485 |
| Web Services | 687 | 250 | 937 |

| Update Item | Sync Data | Upload Data | Total Time |
|---|---|---|---|
| Zero Clien | 25469 | 1375 | 26844 |
| Later Binding & Out Process | 39703 | 1328 | 41031 |
| Early Binding & In Process | 7031 | 1500 | 8531 |
| Web Services | 31 | 1657 | 1688 |

圖 13 操作介面

### 3.2.2 網路架構設計

本實驗是模擬企業內的應用系統，所以該實驗的網域是屬於企業內部網路。為了能真正模擬使用者實際操作，實驗中準備了二台電腦。一台為模擬 Server 主機，其中安裝了 Weblogic 及 MySQL Database。另一台則模擬 Client 端，安裝 Visual Basic 及 Visual C++。如下圖所示，本實驗會借由 Client 端 PC，分別以四種不同的架構執行 Server Side 的 EJB，而 EJB 向 MySQL 查詢資料後，回傳給 Client 端 PC。

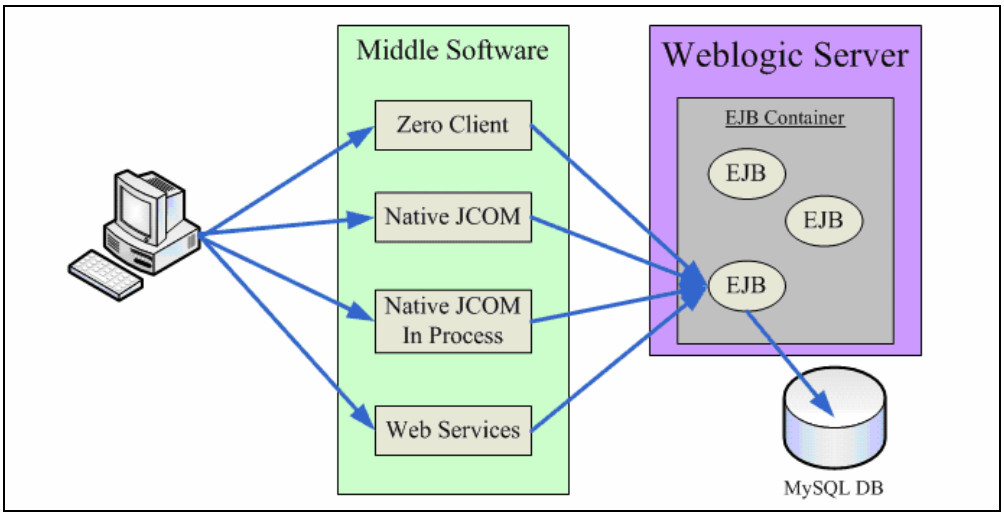



圖 14 網路架構圖

### 3.2.3 系統模組

實驗中的系統模組，採用 EJB 3 + Spring + Hibernate 的方式來實作 MVC 架構。


圖 15 Sequence Diagram

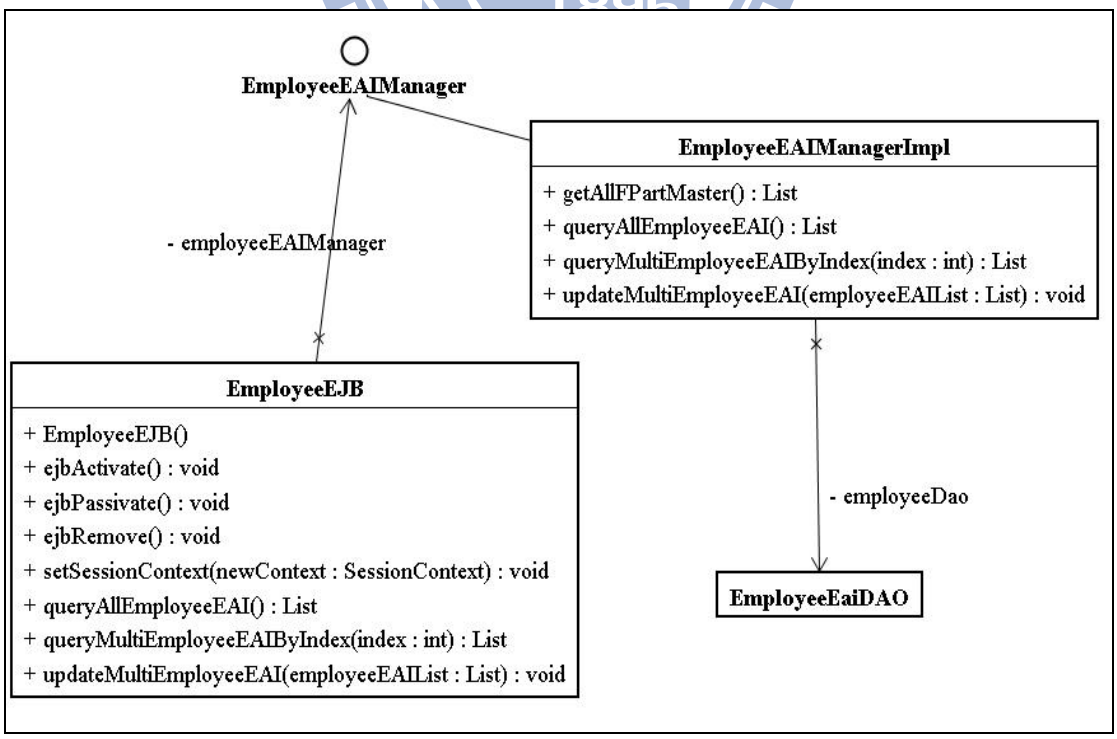



圖 16 Class Diagram

### 3.2.4 資料庫

本實驗的資料庫是 MySQL 5，在資料庫中，新建了一個 TABLE，以下是該 TABLE 的 Domain Objects：

Table Name: Emyloyee_EAI

| Primary Key | Column Name(C) | Type | Length | Description |
|---|---|---|---|---|
| V | EMPID | VarChar | 6 | |
| | CNAME | VarChar | 10 | |
| | ENAME | VarChar | 26 | |
| | DEPTID | VarChar | 6 | |
| | SEX | int | 5 | |
| | DIRECT | VarChar | 2 | |
| | CLS | VarChar | 10 | |
| | ARRIVEDATE | date | | |
| | SDEPID | VarChar | 2 | |
| | QUITDATE | date | | |
| | EMAIL | VarChar | 20 | |
| | FAC | VarChar | 1 | |
| | EXT | VarChar | 20 | |
| | SECRETRAY | VarChar | 10 | |
| | ACCOUNT_CREATED_FLAG | VarChar | 1 | |

## 3.3 實作

### 3.3.1 環境變數及前置作業

環境設定

- Server Side (CPU : C2D 2.53G):
  - Windows XP(sp2)
  - Weblogic Server8.15(JDK1.4)
  - XFire + Spring Framework
  - Hibernate 3
  - MySQL 5
- Client Side (CPU : P4 2.8G):
  - Windows XP(sp2)

- Weblogic Server library & JDK_1.4
- Visual Studio 6
- VaSpread3
- PacketSOAP 1.5.4

### 3.3.2 實作JCOM Zero Client模型:

這是仿照第 2.1.3 小節的論述實作出的實驗這是最簡單、容易實做的架構，在開發的過程中。系統開發人員只需設定 Weblogic Server JCOM 即可。Client Side 及 VB6 不需做任何其他的設定。圖 17 是 Zero Client 的資料流程圖，圖 18 是 Zero Client 的架構圖 ：

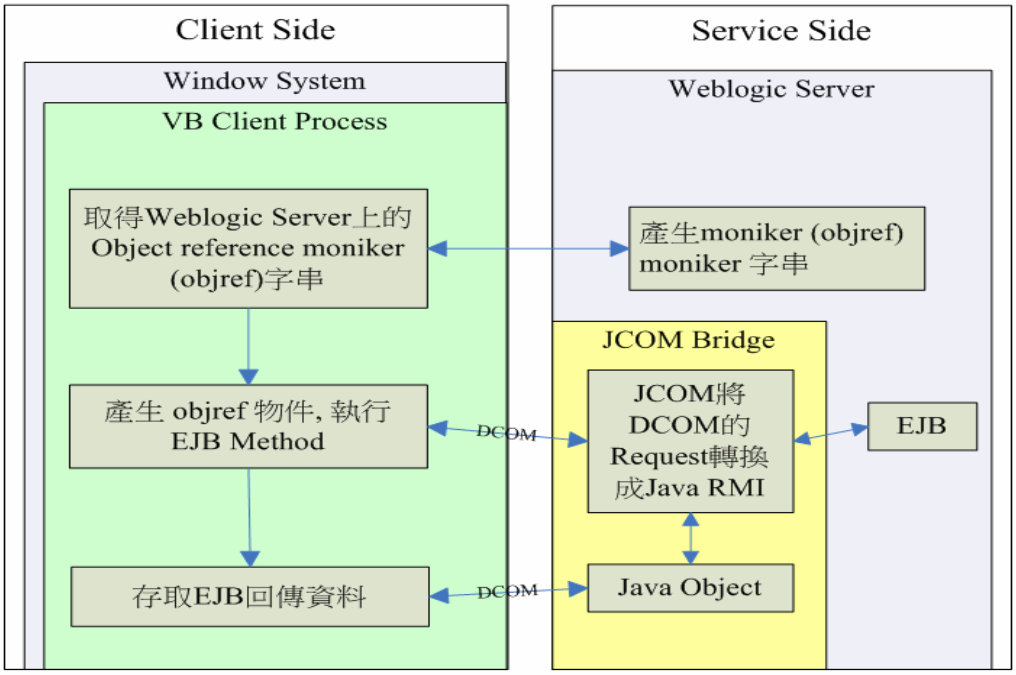



圖 17 Zero Client 資料流程圖

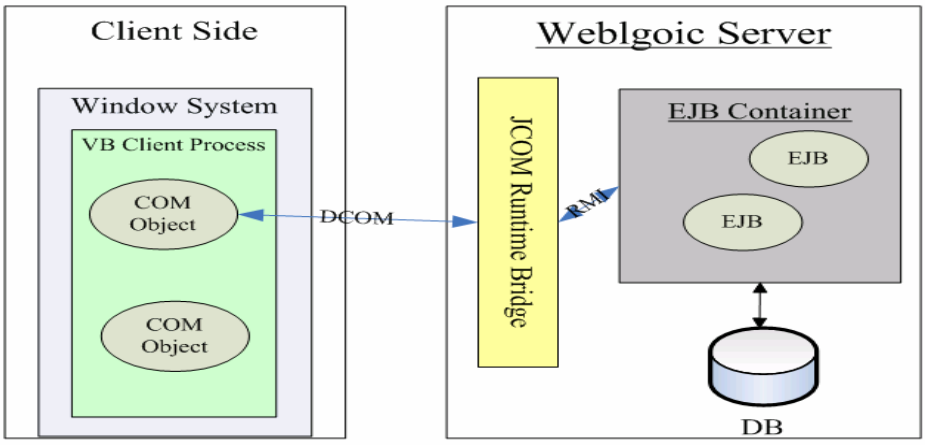



圖 18 Zero Client 的架構圖

開發步驟如下 ：

Step 1. 設定打開 Weblogic JCOM Listener 。

Step 2. 設定 JCOM <-> EJB Interface class 的存取權限。

Step 3. 藉由 Weblogic 提供的 URL 取得 objref 的字串，取得與 Weblogic JCOM 的 Object Reference Moniker，並由此物件和 JCOM 連絡。

Step 4. 實作 VB 呼叫遠端 EJB 的部份程式碼 ：

```
g_strMonikerID = "objref:TUVPVwEAAAAABAIAAAAAAMAAAAAAAAB…"
Set g_objRefMoniker = GetObject(g_strMonikerID)
'系統開發人員只需設定 Weblogic Server JCOM 即可,
Set mdCommonQuerySBHome = g_objRefMoniker.get(g_strWlServerName +
":jndi:" + g_SBJNDIName)
Set mdCommonQuerySB = mdCommonQuerySBHome.Create
Set allEmpList =
mdCommonQuerySB.queryMultiEmployeeEAIByIndex(CInt(g_queryCount))
```

### 3.3.3 實作JCOM Native Out Process + Late Binding 模型 ：

這是仿照第 2.1.4 小節的論述實作出的實驗。這模式比較複雜的部份在於必須新增一個 EJB Agent 做為 EJB Remote Client Class，並且將整個 JCOM JVM 移植到 Client Side。

軟體架構如圖 ：

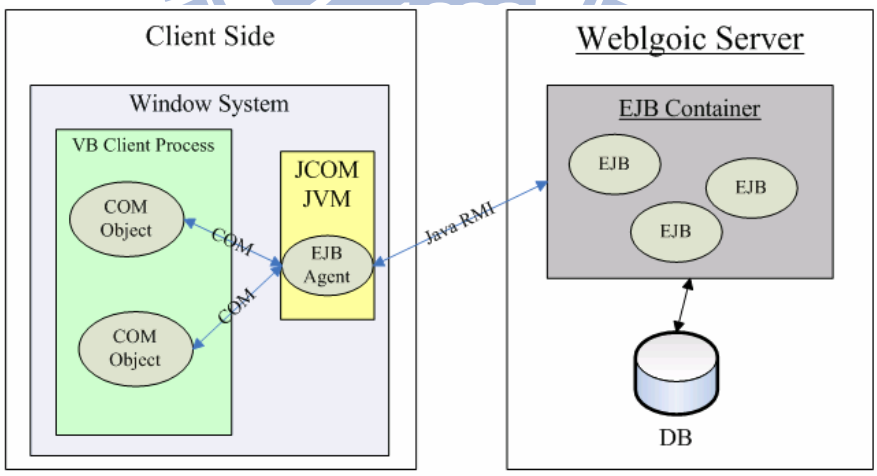



圖 19 JCOM Native Out Process + Late Binding 的架構圖

實作步驟如下 ：

Step 1. 設定打開 Weblogic JCOM Listener 。

Step 2. 設定 JCOM <-> EJB Interface class 的存取權限。

Step 3. Copy JCOM 相關程式到 Client 端，並且在 Client 端執行 JCOM Runtimes 程式。

Step 4. 在 Client 端註冊相對的 JCOM JVM 後，即可以取得對應

Java moniker Object。

Step 5. 實作 VB 呼叫遠端 EJB 的部份程式碼 ：

```
Set jcomEjbAgent =
GetObject("NativeOutProcess:com.tcmb.wrapper.JComEJBAgent")
    Set homeRemote = jcomEjbAgent.getEmployeeEJBRemote()
    Set allEmpList =
homeRemote.queryMultiEmployeeEAIByIndex(g_queryCount)
```

### 3.3.4 JCOM Native In Process + Early Binding 模型

據官方說法，這個方法是 JCOM 執行速度最快的一種架構，但是必須花費最多的步驟。和上個模式不同的地方，除了上個架構所有的步驟外，開發人員還必須先將 EJB 的 Interface 轉換 IDL[註 1]，再將 IDL 轉換成 TLB[註 2]檔，才能載入 COM Reference library。

軟體架構如圖 ：

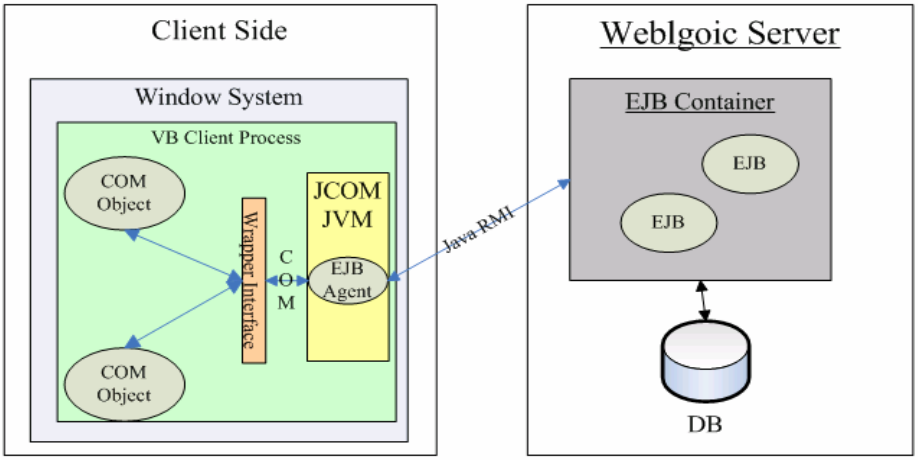



圖 20 JCOM Native In Process + Early Binding 的架構圖

實步驟如下 ：

Step 1. 在 Weblogic Server Console 裡，設定打開 JCOM Listener 。

Step 2. 設定 EJB Interface class 的存取權限。

Step 3. 將 EJB Interface 及相關 Java Class 透過 java2com 的程式轉　　換成 IDL 及 Java Wrapper。

Step 4. 透過 Microsoft tools，將 IDL 檔轉換成 TLB 檔，並且載入 VB Reference Library 裡。

Step 5. 在 Client 端上執行 JCOM Runtimes 程式。

Step 6. 最後再 Client 端註冊相對的 JCOM JVM，才能取得對應 Java moniker Object。

Step 7. 實作 VB 呼叫遠端 EJB 的部份程式碼 ：

```
Dim employeeEJBRmote As
EmployeeEJBTLB.OrgEduNctuProjectInterfacesEmployeeEJBRemote
    Set employeeEJBRmote = jcomEjbAgent.getEmployeeEJBRemote
    Set employeeList =
employeeEJBRmote.queryMultiEmployeeEAIByIndex(g_queryCount)
```

### 3.3.5 實作Object-Oriented Web Service 模型

為了方便能動態產生 WSDL 文件，我用了 Open Source framework (XFire) 做為 Web Service 的平台及動態產生 WSDL 文件的技術。而 Client 端應用程式，則是採用 PocketSOAP 軟體將 WSDL Schema 轉成 VB 的 Library PocketSOAP 解析 WSDL 文件後會產生三種 vb class：(1)Type Object class: 是由 WSDL Types Elements 所產生。(2)SOAP Proxy Object class 由 WSDL SOAP Elements 所產生。(3)Serialize Object class 用來解析/組譯 XML。軟體架構如圖 ：

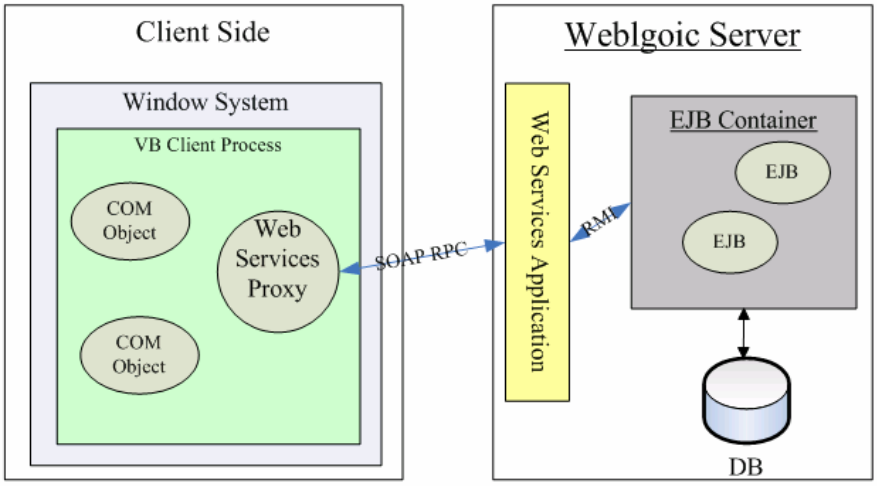



圖 21 JCOM Native In Process + Early Binding 的架構圖

實作步驟如下 ：

Step 1. Weblogic Server 上新增一個 Web Application，並且載入 XFire Framework。

Step 2. 將已存在 EJB Remote Interface，透過 Xfire Framework 轉換成公開的 Web Service WSDL。
利用 PocketSOAP 軟體，將 WSDL Schema 轉換成 VB Class 並且載入 VB Reference Library 中。

Step 3. VB 程式開發人員可以透過 PocketSoap 所產生的物件，直接呼叫遠端的 EJB，但是經由 PocksetSoap 回傳的 VB 物件在使用上會和原 Java 物件有所不同。

Step 4.　實作 VB 呼叫遠端 EJB 的部份程式碼 ：

```
Dim wsProxyEngine As New EmployeeEJBWebservicesPo
    Dim emps() As EmployeeEai
    startTime = GetTickCount()
    wsProxyEngine.url =
"http://Server:7001/WebServicesToEJBApp/EmployeeEJBWebService"
emps =
wsProxyEngine.queryMultiEmployeeEAIByIndex(g_queryCount).Employ
eeEai
```

## 3.4 實驗結果分析：

這個小節會依據實驗所得到的結果進行不同角度的分析，以下數據是測試後的結果。

首先，我們分別針對每個模型執行查詢了 1000 筆員工資料，並且秀出員工的 14 個屬性資料，也就是會有 1000 * 14 個欄位資格需要呈現。

- 在網路流量，我們分別查得其所需的網路流量如下 ：

  Zero Client Mode: 14.1195 / 5.4641 MB

  Native In/Out Mode: 0.1212 MB

  Web Services Mode: 1.0564 MB

- 執行總時間 ：

| Query Item | Query Data | Display Data | Total Time |
|---|---|---|---|
| Zero Clien | 250 | 24781 | 25031 |
| Later Binding & Out Process | 172 | 35625 | 35797 |
| Early Binding & In Process | 78 | 7407 | 7485 |
| Web Services | 687 | 250 | 937 |

| Update Item | Sync Data | Upload Data | Total Time |
|---|---|---|---|
| Zero Clien | 25469 | 1375 | 26844 |
| Later Binding & Out Process | 39703 | 1328 | 41031 |
| Early Binding & In Process | 7031 | 1500 | 8531 |
| Web Services | 31 | 1657 | 1688 |

| Server query & generate xml & transfer | Client side proxy pasing xml time |
|---|---|
| 438 | 249 |

圖 22 各模型執行時間

圖 22 中的 Query Data 欄位所表示的是 client 端呼叫 Server 執行 EJB 所需的時間，而 Display Data 表示的是將 EJB 回傳的值顯示在 UI 上所需的時間，Total Time 則是總花費的時間。另外，我們也針對 web services 的 query data 的部份，區分為二段時間(1) Server query & Generate xml & Transfer date.(2) Client Side parsing xml to VB Object.。各模型花費最多時間依序是 JCOM Native Out Process + Late Binding Mode > Zero

Client Mode > Native In Process + Early Binding > Web Services Mode。

- Client Side CPU 使用狀況 :

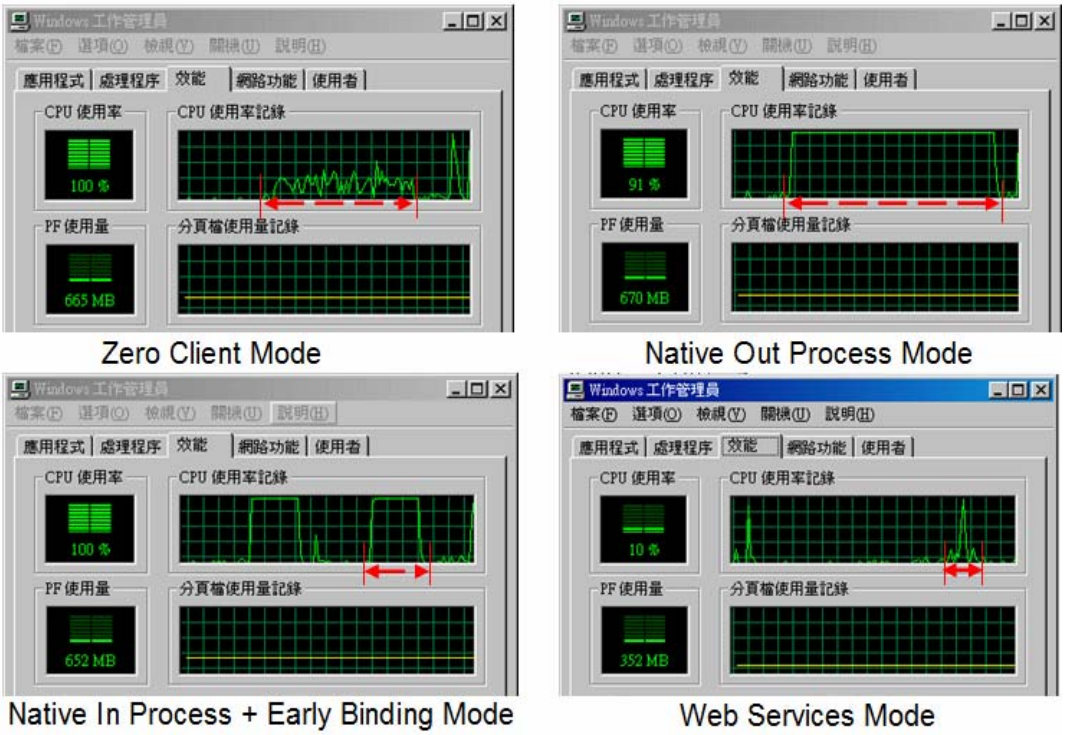



圖 23 各種模式 Client Side 的 CPU 運作狀態

- Weblogic Performance Monitor :

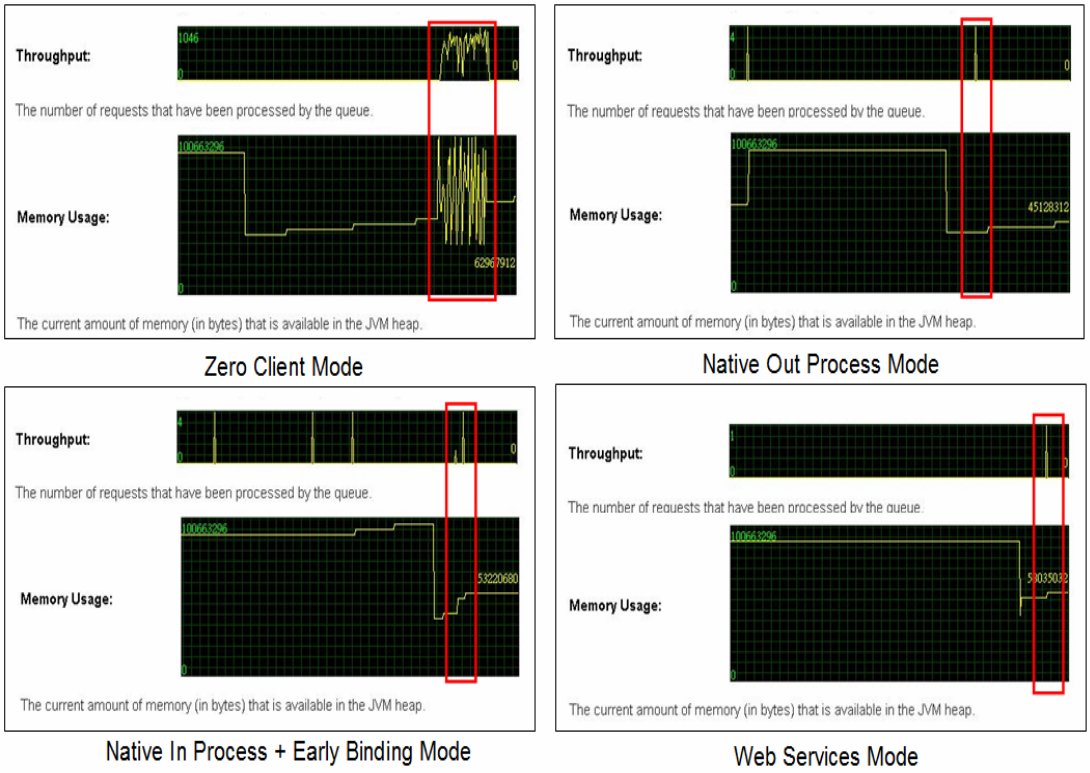



圖 24 各模型的 Weblogic Performance Monitor

各模型的 Weblogic Performance Monitor 的狀態依序是 ：Zero Client Mode > JCOM Native Out Process + Late Binding Mode ≈ Native In Process

+ Early Binding ≈ Web Services Mode。

### 3.4.1 執行效率(速度)分析

由圖 22 的執行結果，我們發現 Zero Client Mode 只花費不到一秒的時間就和 Server 取得物件，但是由於該模式是採用動態取得物件資料，也就是每當取得員工某個屬性資料時，便會透過 DCOM 向 Server 取得，這也證明了為何 Zero Client Mode 花費非常多的時間在呈現資料上。而由圖 22 我們得知四種模型中，執行速度最快的是 Web Services 架構。這個結果證明也驗證了本論文的可行性及目的。附錄一是每種架構各執行十次的實際執行時間。

至於為何 web services 和 In Process 之間的執行差距會有如此大的差距呢?我們後來分析的結論如下圖 25 所示。若 Client 端 VB 透過 JCOM Bridge 取得某一 JAVA 的資料，必須先透過 COM/DCOM 將 Request 發送到 JCOM Bridge(步驟 1)，然後 JCOM Bridge 將 COM Request 找到對應的 COM Proxy (步驟 2)，再藉由 COM Proxy 找到相關的 JAVA Class(步驟 3)，最後取得 Object value 後再回傳到 client side(步驟 4)。

可想而知，一旦大量透過 JCOM 取得 JAVA 物件的值，就等同於一直重覆步驟 1,2,3,4…，這就是為什麼 JCOM 在處理大量資料交換時效能不佳的因。

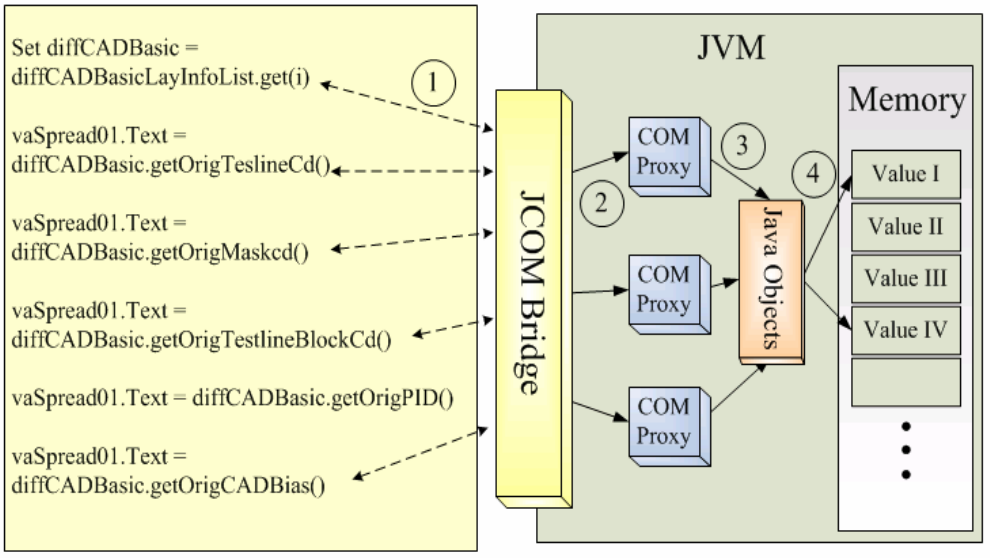



圖 25 VB 透過 JCOM Bridge 取得 JAVA 資料流程圖

另外，讓我們感到意外的是原本以為 Zero Client Mode 所花費的時間應該會是最多，但是結果反而是 Native Out Process + Late Binding 花費的時間最多，而且 CPU 使用度最高。歸究原因可能是因為 JCOM 的架構，分成二個步驟 ：

1. 利用 JCOM JVM，將 JAVA 資料轉換成 COM 物件。
2. 使用 DCOM/COM 進行資傳輸。

Native 模式雖然將 JAVA 物件載入客戶端，減少 Client <-> Server 之間的來回次數，但在本機端進行物件轉換動作及透過 COM 方式進行資料交換必須使用大量 CPU 時間，反而導致效能不好。反而是 Zero Client Mode 因為將 JAVA <-> COM Object 交換動作，交由 Server 執行，所以速度比較快。

至於 Native In Process + Early Binding 模式和 DOM 模式相比較，雖然同樣是將 JAVA 物件載入 Local Side 來處理，但是它先將 JAVA 物件透過 IDL 方式定義在 COM+ DLL Library 中，並且載入了 JCOM JVM DLL，所以整個資料交換動作都屬於同一 Process，所以執行速度較快。

但就 In Process Mode 和 Web Services 的比較，結果讓人非常驚訝。Web Services 花費的時間，是 In Process 的二分之一不到，若資料量愈大，差距愈明顯。而且 Web Services 所花費的資源，無論是 Server Side 或是 Client Side，都非常少。實驗前我們擔心 Web Services 在處理 XML 的 Parser 及網路流量過重會導致資料交換過慢。但在這個實驗中，XML 處理的速度似乎遠遠超小於 JAVA 與 COM 物間轉換的時間。

為了證明本論文的論點是否真的可行，我們亦針對上傳功能進行測試。該實驗是一次將 1000 筆資料回傳到 server side 進行儲存資料庫動作。結果如同圖 22 右方數值所示，web services 總花費時間也是最少的。

另外，由圖 23 及 24 中我們可以看到各模型所花費的 Client Side CPU 的使用度及 Server Side Weblogic Monitor 的 throughput 的使用狀況。Zero Client Mode 的 CPU 使用率雖然大約 50%左右，但是它是將資料轉換的動作交由 Weblogic server 執行，相對的必須花費 server side 的資源。而 JCOM Native OUT/IN Process，則是會佔用 CPU 100%的使用率。不難想像若 Client Side 若有其他的應用程式正在執行，必定會使得執行效能變差。而 Web Services 所耗費 Server 及 Client Side 的資源就相對減少許多，這也證明了 Web Services 取得 JCOM 的可行性。

### 3.4.2 Multi-Users 執行效能分析

以開發的複雜度來看，最簡單的架構毫無疑問的是 Zero Client 架構。但是 Zero Client 的執行效能不好，所以本章節只針對 Web Services Mode 及 Native In Process + Early Binding 進行比較。

通常應用系統不可能只有一位使用，若同一時間有多位使用者執行程式時，Web Services 是否還能保持這麼好的效率呢? 接下來我們將模擬多人同時執行查詢大量資料，並且針對結果進行分析。該實驗只針對 in Process 及 Web Services 兩方法進行實驗。實驗過程中，我們準備四台 Client PC，並且於 PC 上安裝了 Weblogic JCOM bridge 及 JDK1.4。

為了放大數值的差異，我們預設是查詢 10,000 的員工資料。在測試時，

我們先記錄每台 PC 各別的執行時間，然後再測試 4 台 PC 同時執行所花費的時間.結果如下方 4 個表格。

表 1 Execution Time of Web Services Single User

| Web Services-(Single-User) | | | | |
|---|---|---|---|---|
| PC No. | Server query & Generate XML & Transfer Time | Client side proxy pasing xml time | Display Time | Total Time |
| No. 1 | 3141 | 9094 | 2109 | 14344 |
| No. 2 | 3938 | 19485 | 2428 | 25851 |
| No. 3 | 3297 | 20670 | 4574 | 28541 |
| No. 4 | 3766 | 15964 | 3324 | 23054 |

表 2 Execution Time of Web Services Multi User

| Web Services-(Multi-User) | | | | |
|---|---|---|---|---|
| PC No. | Server query & Generate XML & Transfer Time | Client side proxy pasing xml time | Display Time | Total Time |
| No. 1 | 6781 | 13453 | 2741 | 22975 |
| No. 2 | 10938 | 18470 | 2280 | 31688 |
| No. 3 | 10297 | 22265 | 2157 | 34719 |
| No. 4 | 8688 | 8828 | 4725 | 22241 |

表 3 Execution Time of In Process Single User

| JCOM In Process-(Single-User) | | | |
|---|---|---|---|
| PC No. | Query EJB & Transfer Time | Display Time | Total Time |
| No. 1 | 813 | 596781 | 597594 |
| No. 2 | 953 | 889203 | 890156 |
| No. 3 | 906 | 1265891 | 1266797 |
| No. 4 | 1015 | 1041382 | 1042397 |

表 4 Execution Time of In Process Multi User

| JCOM In Process-(Multi-User) | | | |
|---|---|---|---|
| PC No. | Query EJB & Transfer Time | Display Time | Total Time |
| No. 1 | 1547 | 654849 | 656396 |
| No. 2 | 2359 | 907107 | 909466 |
| No. 3 | 1672 | 1499113 | 1500785 |
| No. 4 | 2656 | 1049737 | 1052393 |

將上述資料整理後，我們可以得到以下兩個表格圖 26、27，其中圖 26 的左邊是 Web Services Single-User and Multi-User 的 Call EJB + Data Transfer 的平均執行時間，而右邊則是 JCOM in Process 的平均執行時間。由圖中我們可以很清楚看出，Web Services Mode 的 Server 的執行時間，會因為同一時間執行人數的多寡而產生較大的落差。這是可以理解的結果，

因為 Web service 必須先將資料轉換成 xml format，再將 xml 透過 HTTP protocol 傳送到 client 端。一旦同時間多人使用的話，server 的 loading 就會增加，執行時間也會增加。而 JCOM 的架構，Server 只需負責查詢資料，然後將資料透過 RMI protocol 傳送到 client JCOM-Bridge，JCOM 對 Server 的 loading 遠比 Web Services 小。

看到目前為止，也許大家會開始質疑 Web Service 替代 JCOM 的可行性. 但接下來的圖 27 或許可以讓大家放心。圖 27 左邊是 Web Services Single-User and Multi-User 平均的 Total execution time，而右邊則是 JCOM in Process 平均 Total execution time. 雖然 Web services 因為 4 位使用者同時執行而平均執行時間增加了 6 秒，但平均才花費 29 秒左右的時間。而 JCOM in Process 雖然前後只增加 2 秒，但是平約執行時間是 967.2 秒，約 16 分鐘，兩者相差 32 倍.所以 Web Services 在多人使用的情況下，進行大量資料交換時，仍然是比 JCOM 好。

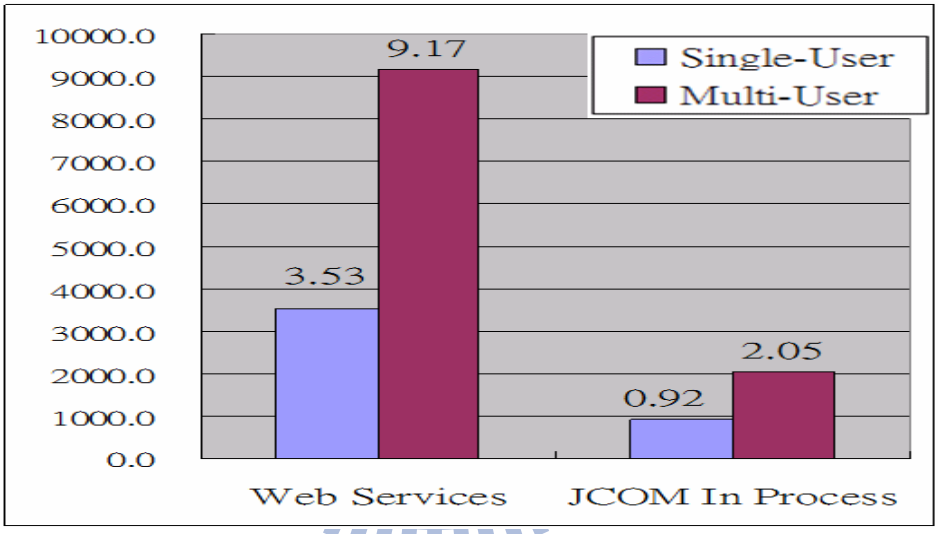



圖 26 100M bandwidth execution time

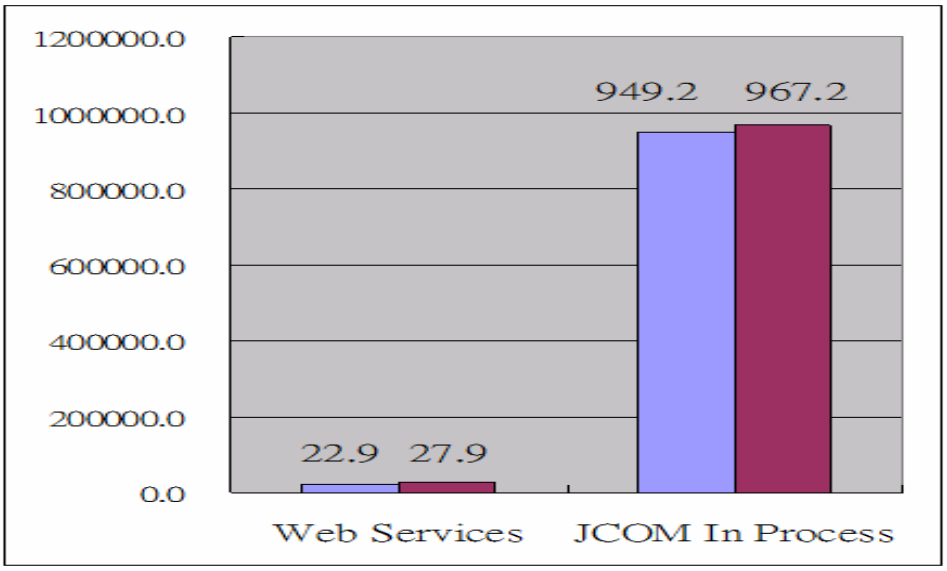



圖 27 100M bandwidth execution time

### 3.4.3 開發複雜度分析

表 Z 中是 JCOM vs. Web Services 實做步驟的比較表，可分為 Server 端的實作步驟及 Client 端的實作步驟。

Server Side:

- Native JCOM：
  每個 Client Side 必須需安裝 JDK1.4，並且將 Weblogic 相關的 jar 檔放入相對的位置。實作時必須在 client side 新增 EJB Agent 的 java code，並藉由這個 Agent 執行遠端 EJB，而須 JCOM 必須設置在 client side。
- Xfire：
  必須建置一個 Web Application 用來發佈、接收及傳送 Client 的 Request/Response。

Client Side:

- Early Binding vs. WSDL Wrapper：
  JCOM Early Binding 的實作，必須經由二個步驟。先藉由 Weblogic 提供的 tool 將 EJB interface 轉換成 IDL 文件，然後再由 Microsoft C++ tool 將 IDL 轉換成 TLB 檔。
- Web Services：
  只需藉由 PocksetSOAP 所提供的 tool 將 WSDL 文件轉換成 VB Class Object。

綜合以上兩點來看，兩者的開發複雜度差不多。但就實際企業的情況而言，企業裡的電腦不可能只有一台，也就是 Client 不只有一個。若採用 JCOM Native 應用在企業系統的話，可想而知，必定會增加開發人員不少的工作量(JCOM 必須額外安裝 JDK，及 JCOM Libraries，並且必須安裝 JCOM JVM)。所以本篇論文認為 JCOM Native 的開發複雜度 > Web Services。

至於軟體程式碼的部份，雖然 JCOM 提供了直覺式使用 java 資料結構的方法，但是 web service 經由 3-party tools 的幫忙，也能將 WSDL 文件轉換成 Client object，一樣能直接使用 java 的資料結構，只是使用的方法必須依照 VB 的語法(例如：迴圈的使用方法必須是 For index=0 To UBound(empList))。

表 5 JCOM vs. Web Services 實做步驟比較

| | Zero Client Mode | Early Binding + In Process Mode | Web Services Mode |
|---|---|---|---|
| Server Side | 1. Enable COM calls on the server listen port.<br>2. Grant access to server classes to COM clients.<br>3. Obtain an object reference moniker (ORM) from the WebLogic Server ORM servlet | 1. Generate EJB interface wrappers and an IDL file with the java2com tool. | 1. Create XFire web Services application on Weblogic Server<br>2. Setup XFire configuration file (web Services name and EJB remote proxy) |
| Client Side | | 1. Install JDK 1.4 & JCOM JVM.<br>2. Create a java agent class to access remote EJB<br>3. Compile IDL and get TLB file.<br>4. Register the type library and set the JVM it will service. | 1. Transfer WSDL Schema to VB Class by 3th -Party tool. |
| | Dim employeeEaiVo As Object | Dim employeeEaiVo As OrgEduNctuProjectDaoEmployeeEai | Dim employeeEaiVo As EmployeeEai |
| | For i = 0 To (EmpList.Size() -1 )<br>…<br>Next | For i = 0 To (EmpList.Size() -1 )<br>…<br>Next | For index = 0 To UBound(empList)<br>…<br>Next |

### 3.4.4 系統耦合度分析

我們知道 JCOM 整合在 Weblogic Server 上的一個軟體，它必須和 Weblogic 綁在一起才能運作，而且只支援 JAVA 與 COM+之間的連結。而 Web Services 則是一個公開的標準，各系統也有提供相關的技術支援。所以就系統耦合度高低而言，Weblogic Server 較 JCOM 來的鬆散許多。

### 3.4.5 網路頻寬分析

上述的論述是依據企業內部的網路架構，並不設定任何的網路頻寬。但是本論文認為網路頻寬流量的大小佔整個實驗一個非常重要的因素，所以特地將其獨立出來分析。下表是我們設定幾組 Server 主機的頻寬並且分別測試每個模型後所得到的結果 ：

表 6 256K 網路頻寬下執行結果

| 頻寬限制 ： 256K | | | 毫秒 |
|---|---|---|---|
| Query Item | Query Data | Display Data | Total Time |
| Zero Client | 63 | 340875 | 340938 |
| Out Process | 4046 | 41813 | 45859 |
| In Process | 4063 | 9500 | 13563 |
| Web Services | 38984 | 234 | 39218 |

表 7 512K 網路頻寬下執行結果

| 頻寬限制 : 512K | | | 毫秒 |
|---|---|---|---|
| Query Item | Query Data | Display Data | Total Time |
| Zero Client | 79 | 346125 | 346204 |
| Out Process | 2016 | 41172 | 43188 |
| In Process | 1735 | 9344 | 11079 |
| Web Services | 17906 | 125 | 18031 |

表 8 1M 網路頻寬下執行結果

| 頻寬限制 : 1M | | | 毫秒 |
|---|---|---|---|
| Query Item | Query Data | Display Data | Total Time |
| Zero Client | 63 | 321390 | 321453 |
| Out Process | 1109 | 35547 | 36656 |
| In Process | 922 | 9234 | 10156 |
| Web Services | 8641 | 281 | 8922 |

表 9 5M 網路頻寬下執行結果

| 頻寬限制 : 5M | | | 毫秒 |
|---|---|---|---|
| Query Item | Query Data | Display Data | Total Time |
| Zero Client | 31 | 119406 | 119437 |
| Out Process | 360 | 35812 | 36172 |
| In Process | 265 | 9469 | 9734 |
| Web Services | 2078 | 219 | 2297 |

表 10 10M 網路頻寬下執行結果

| 頻寬限制 : 10M | | | 毫秒 |
|---|---|---|---|
| Query Item | Query Data | Display Data | Total Time |
| Zero Client | 31 | 68781 | 68812 |
| Out Process | 282 | 36812 | 37094 |
| In Process | 156 | 9562 | 9718 |
| Web Services | 1563 | 218 | 1781 |

表 11 100M 網路頻寬下執行結果

| 頻寬限制 : 100M | | | 毫秒 |
|---|---|---|---|
| Query Item | Query Data | Display Data | Total Time |
| Zero Client | 47 | 26125 | 26172 |
| Out Process | 172 | 33922 | 34094 |
| In Process | 78 | 9234 | 9312 |
| Web Services | 515 | 235 | 750 |

表格中[Query Item]欄位為測試模型名稱、[Query Data]欄位為 Client 向 Server 取得資料所花費的時間、[Display Data]欄位是將資料顯示在畫面所花費的時間、[Total Time]為[Query Data]+[Display Data]的總和。

以下圖 28 是依據上述各測試頻寬數據所產生的折線圖。

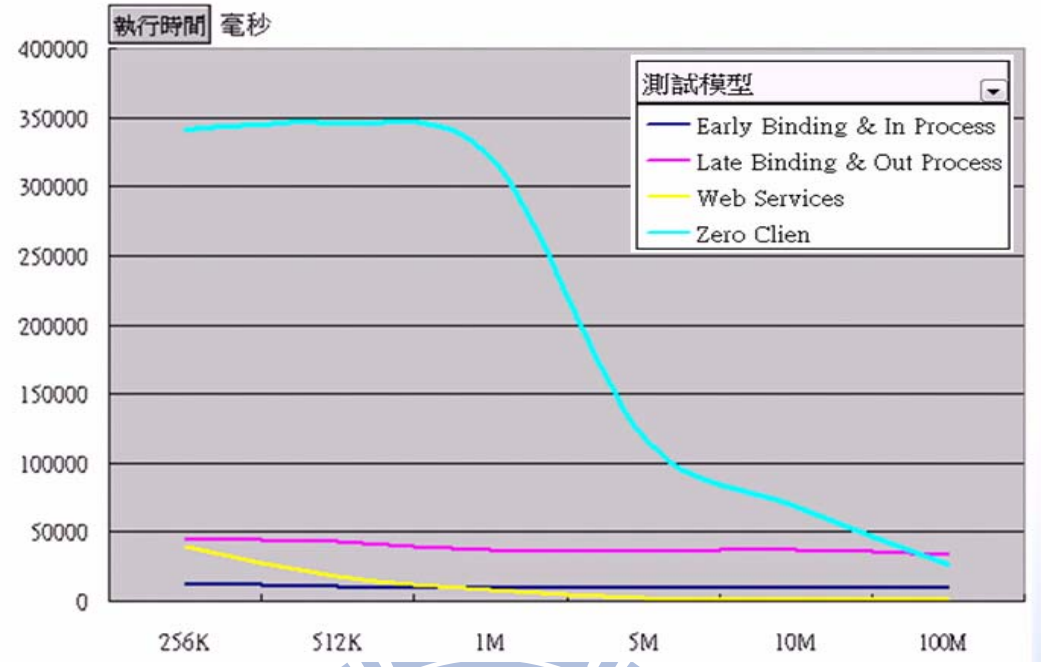



圖 28 各測試模型對應頻寬的執行時間折線圖

以上，我們發現幾個特徵：

1. Zero Client Mode，需要非常大的頻寬才能縮短時間。這表示同時段若有其他系統正在使用網路，變會發生排擠的現象。另外我們也發現 Zero Client Mode 執行時會導致 Weblogic Server 的效能非常的差，推測應該是在處理 Zero Client 的 Request。
2. Late Binding & Out Process Mode 的架構受網路頻寬因素的影響最少，但平均效能是最差的。
3. 當頻寬夠>1M 時，Web Services 的執行速度就會比 Early Binding + In Process 來的快。

另外，為了更進一步比較 In Process 及 Web Services 兩種架構的差異性，我們又進行以下測試。我們預設將頻寬固定在 1M bits，針對查詢員工筆數不同所花費的時進行比較。

表 12 中的第一欄位記錄著查詢員工筆數及 Web Services/Out Process/In Process 的網路流量，第二欄位則是測試項目名稱，第五欄執行花費總時間。分析表 12 裡的資料後，得到圖 29。由圖 29 中，我們發現當查詢的資料量不多，In Process 模式的執行效能比較好。但隨著查詢的資料愈來愈多，In Process 花費在資料轉換(JAVA and COM)的時間，愈來愈接近網路傳送 XML 所花費的時間，當資料量超過某一界線 In Process 的效能就不如 Web Services 模式了。例如: 當使用者查詢一千筆資料時，各模式的網路流量分別是(1.0641M/0.1212M)，所花費的時間分別是(9.165 秒/10.016 秒)，二者之間的差距不到一秒。但是若查詢 1500 筆資料時，各模式的網路流量分別是(1.5857M/0.1821M)，所花費的時間分別是(13.578 秒/26.782 秒)，差距有 13

秒之多。我們得到以下資料，XML Parsing Rate 大約是 8.844(秒/千筆)，JCOM Data Exchanged Rate 大約是 23.625(秒/千筆).由此我們更可以確定在 1Mbits 以上的頻寬中，Web services 的方法比 JCOM Native In Process 更有效率，也比其他 JCOM 模式來的好。

表 12 1M 網路頻寬下查詢不同筆數資料的執行結果

| 查詢筆數(網量流量) | 測試項目 | 取得資料 | 顯示資料 | 總時間 |
|---|---|---|---|---|
| 100 Records (0.0137M) | Early Binding & In Process | 1 | 250 | 251 |
| 100 Records (0.106M) | Web Services | 1203 | 47 | 1250 |
| 200 Records (0.0257M) | Early Binding & In Process | 140 | 453 | 593 |
| 200 Records (0.2117M) | Web Services | 1906 | 94 | 2000 |
| 500 Records (0.0613M) | Early Binding & In Process | 438 | 1625 | 2063 |
| 500 Records (0.5283M) | Web Services | 4406 | 141 | 4547 |
| 1000 Records (0.1212M) | Early Binding & In Process | 922 | 7797 | 8719 |
| 1000 Records (1.0564M) | Web Services | 8594 | 297 | 8891 |
| 1300 Records (0.1572M) | Early Binding & In Process | 1203 | 18469 | 19672 |
| 1300 Records (1.3735M) | Web Services | 11422 | 265 | 11687 |
| 1500 Records (0.1821M) | Early Binding & In Process | 1532 | 25250 | 26782 |
| 1500 Records (1.5857M) | Web Services | 13250 | 328 | 13578 |

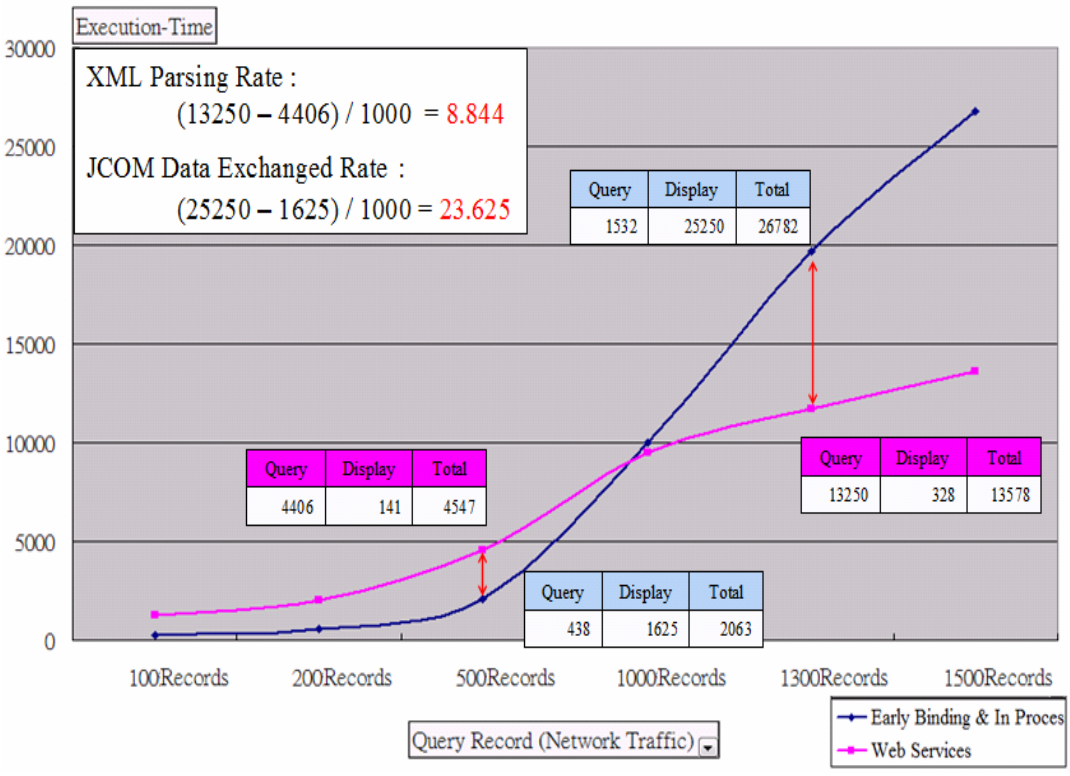



圖 29 Network traffic vs. Execution-Time

# 第四章 實際專案導入

## 4.1 專案介紹

該案例是台灣某半導體公司專案開發時的實際案例。專案主要功能是提供工程師線上查詢及修改製程技術文件的應用程式。當工程師想要修改某個版本的製程技術文件後，必須進入該系統將新版本資料與來源版本文件進行比對，並且詳細描述前後版本之間的差異性及理由並產生 EXCEL 檔。系統經過 E-MAIL 告知主管及各廠區相關負責人進行文件生效的簽核流程。簽核人必須進入該系統審核文件內容後，決定是否同意新版技術文件的是否能生效。圖 30、31 分別是專案其中一個 Use Case 的 Activity Diagram 及 Use Case Diagram。

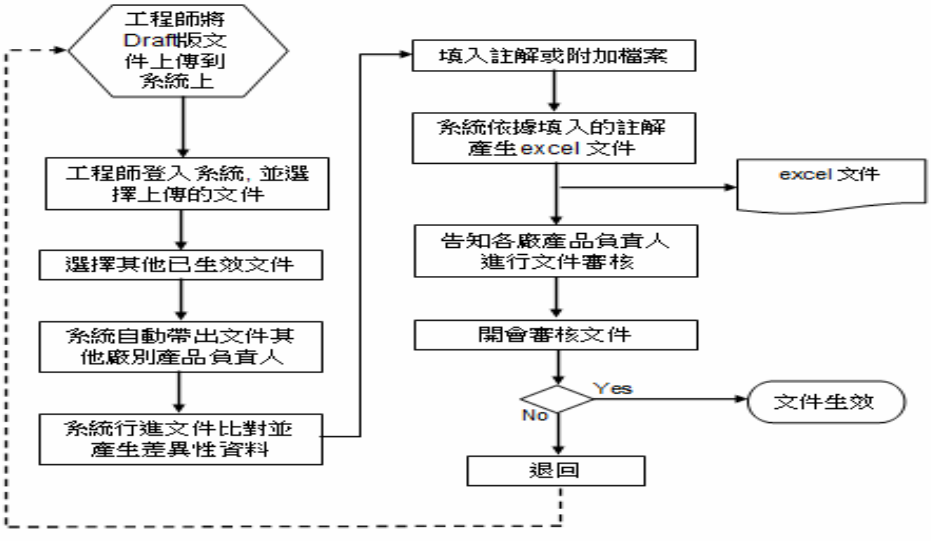



圖 30 專案流程圖

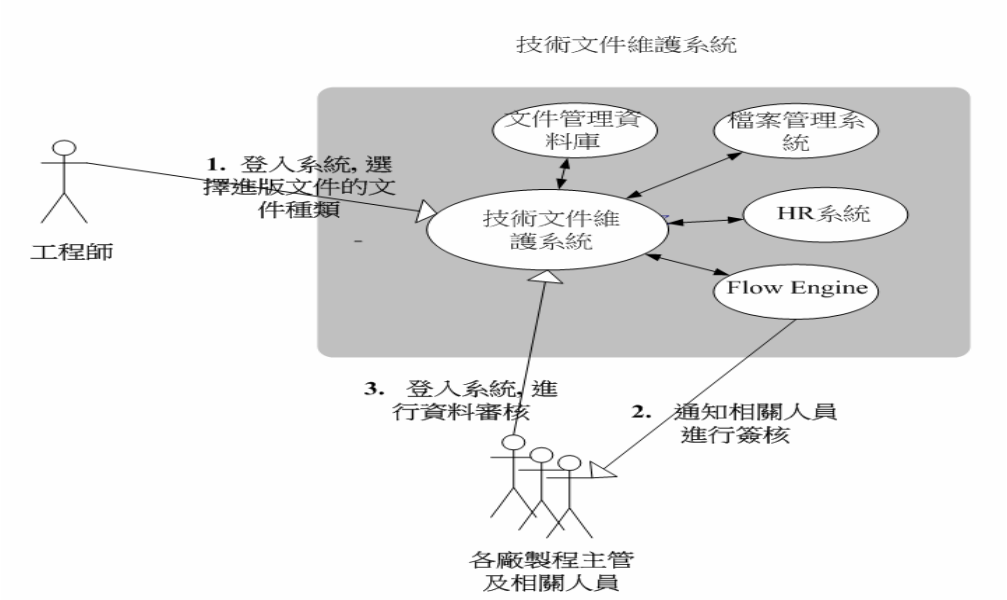



圖 31 Use Case Diagram

## 4.2 系統程式架構

圖 32 是該公司軟體基本架構,圖中紅色線上方的流程就是 JCOM 為中間軟體的架構 ：

1. 為了防止企業資料外洩、員工權限管控及應用程式版本管理，該公司導入 Citrix Server 的架構。使用者必須先透過個人 PC 登入 Citrix Server 進行身份確認後，才能在 Citrix Server 上使用各種應用程式程式。
2. 在 Citrix Server 上，以 Visual Basic 6 程式語言開發客戶端應用程式，VB 程式必須透過 JCOM 呼叫遠端 Weblogic Server 上的 EJB(使用 Stateless Session Bean + Hibernate ( Object / Relational Mapping))，達成 3-Tier architecture。
3. Hibernate ：EJB 透過 Hibernate 及 Connection pool 向資料庫取得資料。

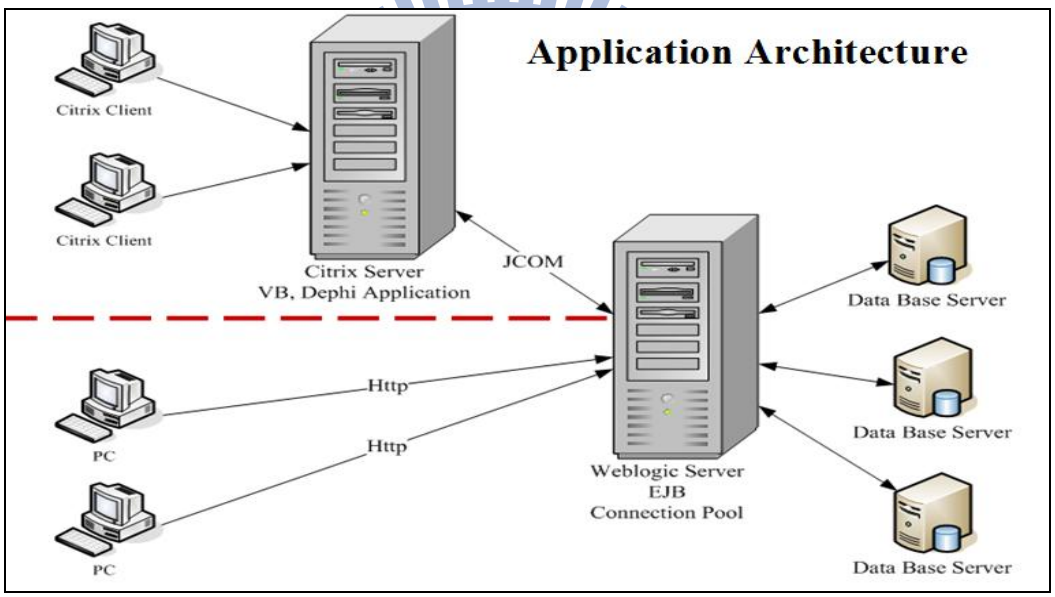



圖 32 應用程式基本架構

## 4.3 專案開發瓶頸及解決方法

由於在 Citrix Server 上的 VB 程式必須透過 JCOM(Zero Client 模式)呼叫遠端 EJB。取得前後版次的技術文件後才能在 Client 端的中進行比對，並且提供輸入註解及檔案的操作介面。

後來使用者反應文件比較速度太慢，完整比對一份資料需要花費 2~5 分鐘。後來開發人員深入分析為何會如此慢的原因，主因就是 JCOM 在處理大量資料交換時的效能太差。

雖然系統開發人員在開發過程中已發現這個問題,但一開始大家只單純的以為是開發環境主機效能不好,只要程式上到正式主機上就不會有這個問題了,所以並未嘗試解決這個的問題。直到程式開發完成,進入 UAT 階段時,才發現問題依然存在。不但如此,還因為該系統佔據了主機的其他資源,使得其他應用程式也無法順利執行。

於是系統開發人員想出了二個解決辦法來解決這個問題 :

1. 3-Ties to 2-Ties : 以往開發時遇到這種情況時,直覺是修改三層式架構,改用 Client-Server 的方式,由 VB 端直接存取資料庫。但這方法等於是把 EJB 端的程式全部重新用 VB 開發一遍,需花費許多人力,時間及資源。
2. Web Service 取代 JCOM : 以 Web Service 做為中介軟體,取代原來 JCOM (藉由 XFire Framework 直接將開發完成的 EJB Remote Interface,轉成 WSDL 文件,再由 PocketSOAP 將 WSDL 文件轉成 VB 的 Object)。圖 33 是該公司原來所使用的 JCOM 中介軟體架構,其中的藍色方塊是 JCOM 所處位置。而圖 34 則是用 OOWS 取代 JCOM 後的架構圖。

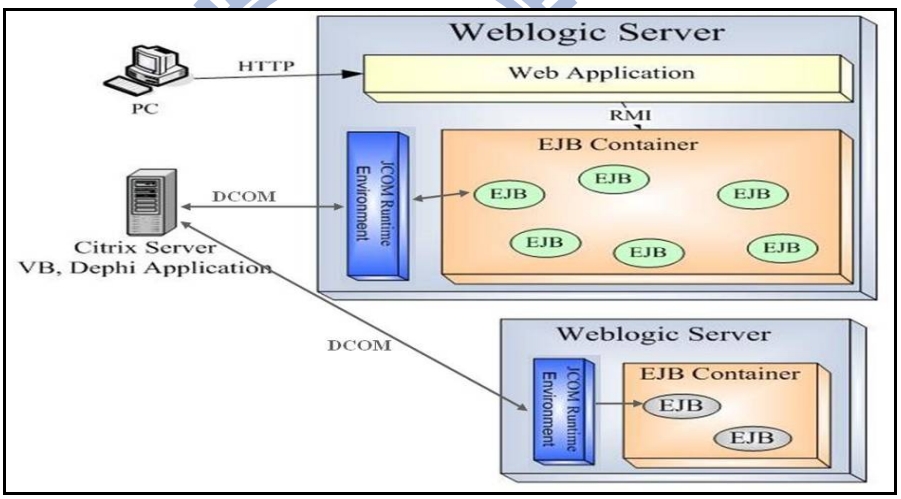



圖 33 JCOM 軟體架構

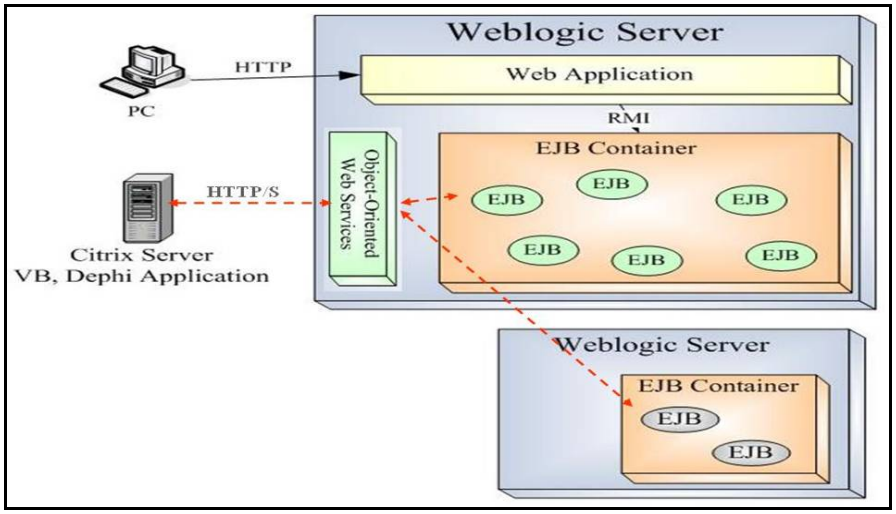



圖 34 採用 Web Service 取代 JCOM 軟體架構

## 4.4 導入 Web Services ：

為了減少導入的時間，開發人員針對幾個需要大量資料交換的動作進行 Web Services 取代 JCOM。該專案導入的過程可分成二階段來說明。

### 4.4.1 如何產生EJB對應的WSDL文件

首先，第一階段是必須產生 EJB 對應的 WSDL 文件。該專案採用的 XFire Open Source Framework API。XFire 它除了支援 JSR181 的標準外，也支援 JAVA POJO(Plain Old Java Object)的技術。所以開發人員只需要透過 XFire Configuration File 的設定，將 Web Services 指定到 EJB Remote Interface，它就會使用 POJO 的技術讀取 java class，並產生對應的 WSDL 文件。圖 35、36 該專案中部份程式碼，透過 XFire Configuration File 設定產生對應的 WSDL 文件

```
public interface MaintenanceTLService {

    public String[] getActiveTLByT3Code(String queryTl3Code) throws EJBException ;
    public boolean checkGDSFileIsReady(String testlineCode) throws EJBException ;
    public TiTcmbTestLine runCompareOneFunction(String testlineCode, String refTest
    public TiTcmbTestLine queryTestlineByTestlineCode(String testlineCode) throws E
    public List runCompareTwoCADBasicLay(List diffCADBasicLayInfoList) throws EJBEx
    public List runCompareTwoCADDataLayInfo(List diffCADDataLayInfoList) throws EJB
    public boolean runSummaryFunction(TiTcmbTestLine tiTcmbTestLineVo) throws EJBEx
    public boolean saveOrSubmitAction(TiTcmbMaster tiTcmbMaster) throws EJBExceptio
}
```

```
- <wsdl:binding name="MaintenanceTLServiceHttpBinding" type="tns:MaintenanceTLServicePortType">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  + <wsdl:operation name="runCompareOneFunction">
  + <wsdl:operation name="runCompareTwoCADDataLayInfo">
  + <wsdl:operation name="checkGDSFileIsReady">
  + <wsdl:operation name="queryTestlineByTestlineCode">
  + <wsdl:operation name="saveOrSubmitAction">
  + <wsdl:operation name="runCompareTwoCADBasicLay">
  + <wsdl:operation name="runSummaryFunction">
  + <wsdl:operation name="getActiveTLByT3Code">
  </wsdl:binding>
```

圖 35 將 EJB 的 Interface 轉換成 WSDL 的 Binding

```
public class DiffAttachfile {
    public String filename;
    public String fileloacation;
    public String fileId;
    public String getFilename() {
    public void setFilename(String filename) {
    public String getFileloacation() {
    public void setFileloacation(String fileloacation) {
    public String getFileId() {
    public void setFileId(String fileId) {
}
```

```
<xsd:complexType name="DiffAttachFile">
   <xsd:sequence>
      <xsd:element minOccurs="0" name="fileName" nillable="true" type="xsd:string"/>
      <xsd:element minOccurs="0" name="fileLoacation" nillable="true" type="xsd:string"/>
      <xsd:element minOccurs="0" name="fileId" nillable="true" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
```

圖 36 Java 物件轉換成 WSDL 的 Types

圖 37 是該專案 XFire 的流程圖。當 Web Application 接收到來自客戶端的 SOAP Request 時，Servlet 會藉由 xfire framework 將 soap 及 type 的 xml 轉換成 java 的物件，並且透過 java RMI 呼叫遠端 EJB。當 EJB 回傳資料時，XFire 會再將 java 物件轉換成 xml 格式回傳給 client 端。

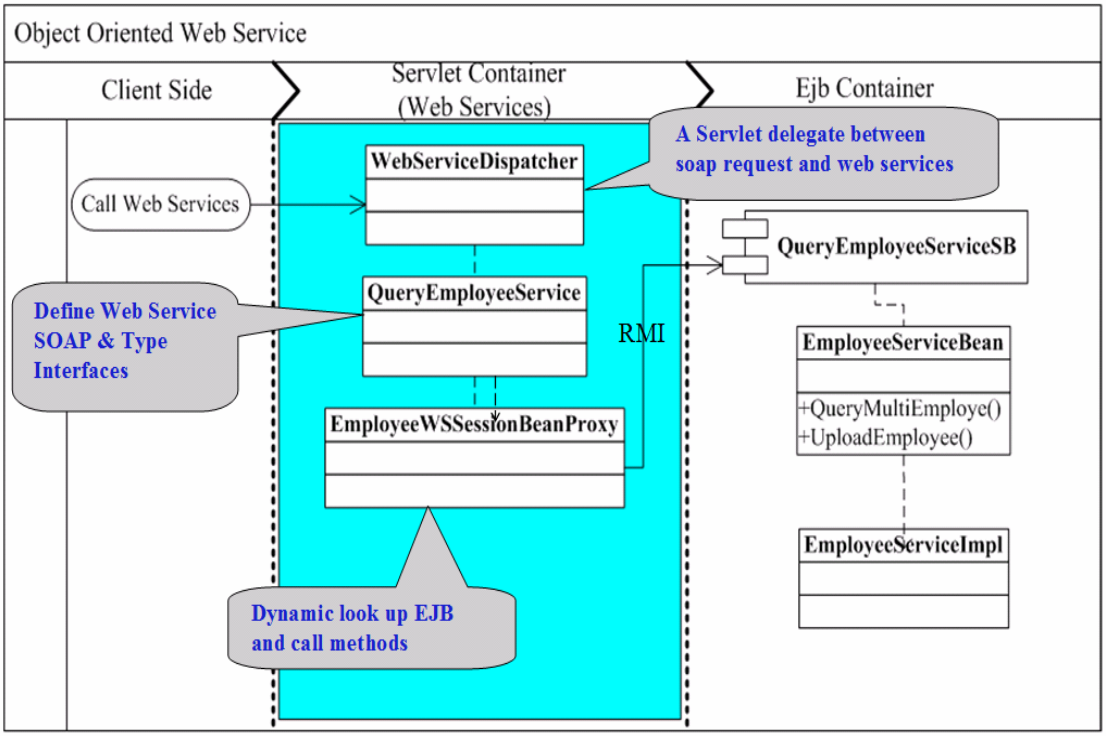



圖 37 XFire + Spring Framework Workflow

### 4.4.2 如何將VB程式由JCOM轉換成Web services

第二步驟是 Client 端 VB 的程式如何由 JCOM 模式轉換成 Web Services SOAP 的模式。由於該公司是仍是採用 VB 作為客戶應用程式的開發工具。所以的 Microsoft 推出的工具無法使用，必須透過第三方軟體 PocketSOAP 才能支援 VB。PocketSOAP 是一 Freeware API(如圖 38)，可將 WSDL Context 轉換 VB 的 Class object。它主要產生的 VB Object 可以分成三類 ：

1. Type Object ： 單純的 OO 物件，傳送 Web Service 所須的內容，對應於 WSDL Types Elements。
2. SOAP Proxy Object ： 用來負責和 Web Services 傳送及接收的物件。
3. Serialize Object ： 用來產生 xml 及將 xml 轉換成 VB Object 的 class。

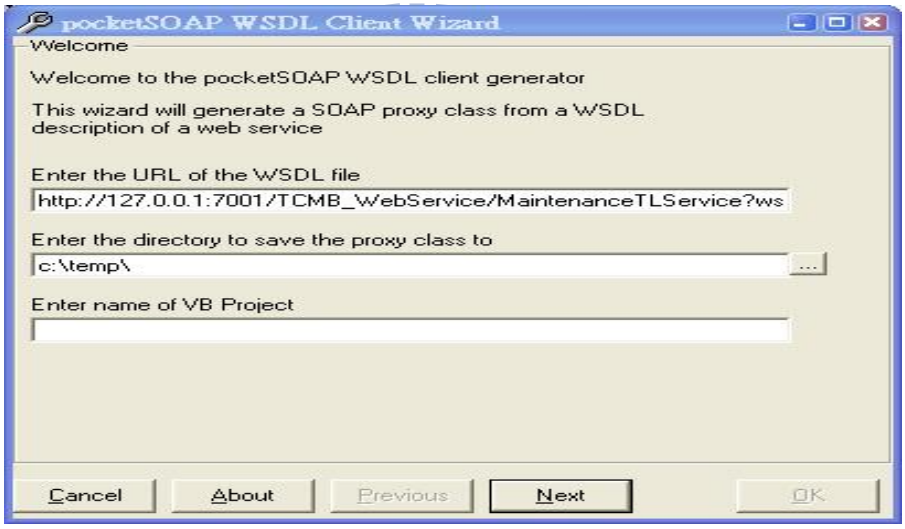



圖 38 PocketSOAP 工具的截圖

最後開發人員只需將產生的物件載入 VB library，並且修改原本使用 JCOM 時的程式碼即可。範例如下 ：

| | | |
|---|---|---|
| Dim employeeEaiVo As Object | | Dim employeeEaiVo As EmployeeEai |
| vaSpread01.Text = employeeEaiVo.getEmpid() | > | vaSpread01.Text = employeeEaiVo.empid |
| For i = 0 To (EmpList.Size() -1 )<br>…<br>Next | | For index = 0 To UBound(emps)<br>…<br>Next |

## 4.5 專案分析 ：

表 13 是我們針對 Compare(文件比較)，Summary(產生 Excel 檔)

以及 Save(存儲)等幾個需要進行大量資料交換的步驟進行追蹤記錄而產生的一個報表。

其中黃色代表的是採用 JCOM Zero Client 做為中間軟體的的執行時間，綠色則是以 Web Services 為架構的執行時間。由此圖中，我們可以清楚看到採用 Web Service 的方法，比較原本使用 JCOM 的架構所需花費的時間節省了 5~8 倍。

表 13 JCOM & Web Services executed time log table

| RUN_STAGE | JCOM | STAR_TIME | END_TIME | OOWS | |
|---|---|---|---|---|---|
| Compare 1 | 120684 | 2008/4/23 04:12:22 | 2008/4/23 04:12:46 | 23625 | 5.10832 |
| Summary | 120944 | 2008/4/23 04:20:46 | 2008/4/23 04:21:08 | 22750 | 5.31622 |
| SaveAsDraft | 241723 | 2008/4/23 04:33:08 | 2008/4/23 04:33:45 | 36319 | 6.65555 |
| Compare 1 | 120476 | 2008/4/23 04:55:45 | 2008/4/23 04:56:06 | 21500 | 5.60353 |
| Summary | 121345 | 2008/4/23 05:04:16 | 2008/4/23 05:04:35 | 18750 | 6.47173 |
| SaveAsDraft | 242423 | 2008/4/23 05:16:25 | 2008/4/23 05:17:02 | 37268 | 6.50486 |
| Compare 1 | 120089 | 2008/4/23 05:39:32 | 2008/4/23 05:39:58 | 25375 | 4.73257 |
| Summary | 120773 | 2008/4/23 05:47:58 | 2008/4/23 05:48:24 | 26250 | 4.60088 |
| SaveAsDraft | 240959 | 2008/4/23 06:00:24 | 2008/4/23 06:01:09 | 45636 | 5.28002 |
| Compare 1 | 180084 | 2008/4/23 06:23:09 | 2008/4/23 06:23:31 | 21625 | 8.32758 |
| Summary | 180408 | 2008/4/23 06:31:26 | 2008/4/23 06:31:55 | 29375 | 6.14155 |
| SaveAsDraft | 360654 | 2008/4/23 06:43:44 | 2008/4/23 06:44:29 | 44116 | 8.17513 |
| Compare 1 | 60021 | 2008/4/23 07:06:19 | 2008/4/23 07:06:32 | 13000 | 4.617 |
| Summary | 60689 | 2008/4/23 07:14:47 | 2008/4/23 07:15:01 | 14875 | 4.07993 |
| SaveAsDraft | 121519 | 2008/4/23 07:27:01 | 2008/4/23 07:27:22A | 20638 | 5.88812 |

# 第五章 結論與未來研究

## 5.1 結論

首先，本篇論文的研究的並非強調 Web Services 可以完全取代 JCOM/DOM 的架構。在許多情況下 JCOM/DCOM 仍是有其優勢。例如若須要透過網路執行遠端應用程式，並且不需要進行大量資料往返的系統時，Zero Client Mode 就比 Web Service 來的方便許多。

而本研究目的是希望以 Object-Oriented Web Service 架構，解決 JCOM/DCOM 在處理大量資料交換時效能不佳的問題，並且提出一套真正方便、有效率的實作方法，可減少企業導入 Web Services 可能會遇到問題。在本篇論文的實驗結中，可以很明顯的看出來，Web Service 的執行效能在大量資料交換時，的確比 JCOM 來的好。

過去幾年，所以許多大型企業為了解決 JCOM/DCOM 效能的問題，情願花多上 2~3 倍的人力及時間去開發新的 Web 化系統，也不願導入 Web Services。我們歸究其主要的原因有二:

1. WSDL 的定義太複雜以及需要花費許多時間在處理 XML 往返資料上。許多企業資訊人員一談到要將 Object-Oriented (物件導向)的程式改成用 XML Schema 結構表示時，往往花費許多腦力，以致不想使用，甚至完全排斥它。
2. 資訊安全的問題。

但是就目前的技術成熟度而言，這些都已不再是問題。藉由本篇論文的實驗結果，證明了 Web Service 的執行效能在大量資料交換時，的確比 JCOM 好。並且證明了 Web Services 只要是有好的輔助工具，它不再是如此的複雜難以使用。

而 Web Service 安全上的疑慮，雖然這不在本篇討論的範圍內，但目前也已經有許多解決方法來防止資料外洩(例如:HTTPS、XML 資料壓縮、資料加密、身份驗證…等)，甚至是將 Web Application 的安全管理機制套用在 Web Services 上都是可行的。

## 5.2 未來研究

本研究所提出的架構是透過 Web Services 直接將 EJB 轉換 WSDL 文件。但企業內一定會有許多不同的系統，若未來每個系統，甚至是和外包廠商之間，都將利用 Web Services 做為資料交換的平台，可想而知 Client Application、Web Services 及 EJB 之間的關係將會非常複雜。因此未來研究的重點會著重於如何管理 Client Application、Web Services 及 EJB 之間的關係以及安全性的定義。

另外，實驗中的 Web Services 架構，Server 在產生 xml content 時，會造成 Server 的 loading 的增加，若有非常大量使用者同時執行時，是否會讓 server 太忙碌而導致 web services 的效率不佳或許是值得研究的方向。

若有些企業必須針對傳輸內容有保密或是壓縮的需要時，那 Web Service 是否還能有這麼好效能，也是另一個可以研究的方向。

# 參考文獻


[1]. Gustavo Alonso, Fabio Casati, Harumi Kuno, et al., "Web Services Concepts, Architectures and Applications", 2004

[2]. B. Medjahed et al., "Infrastructure for E-Government Web Services", Internet Computing, IEEE CNF, vol. 7, no. 1, pp.58 - 65, 2003

[3]. K. Gottschalk, S. Graham, H. Kreger, et al., "Introduction to Web Services Architecture," IBM System Journal, vol. 41, no. 2, pp.170-177, 2002

[4]. Goth, G., "News: data-driven enterprise - slouching toward the semantic Web", Distributed Systems Online, IEEE JNL , Volume 7, Issue 3, March 2006

[5]. Davis, A., Du Zhang, "A comparative study of DCOM and SOAP", Multimedia Software Engineering Proceedings. Fourth International Symposium on , IEEE CNF, pp.48 - 55 , Dec. 2002

[6]. SeokHyun Yoon, DongJoon Kim, SangYong Han, "WS-QDL containing static, dynamic, and statistical factors of Web services quality", IEEE CNF, pp.808 - 809 6-9 July 2004

[7]. Donglai Zhang, Coddington, P., Wendelborn, A., "Binary Data Transfer Performance over High-Latency Networks Using Web Service Attachments", e-Science and Grid Computing, IEEE CNF, pp.:261 - 269, 10-13 Dec. 2007

[8]. Parsa, S., Ghods, L., "A new approach to wrap legacy programs into web services", Computer and Information Technology, IEEE CNF, pp.442 - 447, 24-27 Dec. 2008

[9]. Magdalenic, Ivan; Vrdoljak, Boris; Skocir, Zoran; "Towards Dynamic Web Service Generation on Demand", Software in Telecommunications and Computer Networks, IEEE CNF, pp.276 - 280 , Sept. 29 2006-Oct. 1 2006

[10]. He Guo; Chunyan Guo; Feng Chen; Hongji Yang;, "Wrapping Client-Server Application to Web Services for Internet Computing", Parallel and Distributed Computing, Applications and Technologies, IEEE CNF, pp.366 - 370, 05-08 Dec. 2005

[11]. Curbera, F. et al., "Unraveling the Web Service Web. An Introductionto SOAP, WSDL, and UDDI", IEEE Internet Computing, vol. 6, no. 2, pp. 86-93, Mar./Apr. 2002

[12]. Lin, J., Taso, H., and Chu, Y., "Object-Oriented Analysis and

Designof Web-Based Information Systems", IEEE Engineering of ComputerBased Systems, 2001, pp. 68-75.
[13]. "Web Services Conceptual Architecture 1.0", IBM Software Group, 2001
[14]. Stal, M., "Web Services：Beyond Component-Based Computing," CACM, Vol. 45, pp.71-76. 2002
[15]. Jiang, J.; Systa, T., "UML-based modeling and validity checking of Web service descriptions", Digital Object Identifier, Web Services, 2005. ICWS 2005. Proceedings., IEEE CNF, page:460, 11-15 July 2005
[16]. Zhang, C. Jacobsen, H.-A., "Refactoring middleware with aspects", Parallel and Distributed System, IEEE CNF, pp.1058 - 1073, Nov. 2003
[17]. Binder, W. Hulaas, J. Moret, P., "A Quantitative Evaluation of the Contribution of Native Code to Java Workloads", Workload Characterization 2006 IEEE International Symposium on, IEEE CNF, pp.201 - 209, 25-27 Oct. 2006
[18]. Newcomer E., Lomow G. (2007). "Understanding SOA with WebServices." Pearson Education, Inc.
[19]. Brenner M. R., Unmehhopa M. R. "Service-Oriented Architectureand Web Services Penetration in Next-Generation Networks", Bell LabsTechnical Journal, Vol.12, No.2, pp.147-160. (2007)
[20]. Steve Vinoski , "Web services Integration Models", IEEE Internet Computing, pp.89-91, May/June 2002.
[21]. Zhuang-Ye Liu；Zheng Yao，"基于 Web 服務的教師管理系統的設計與實現"，中國科學院研究生院學報，pp.127-131, 26 卷 1 期, 01/01/2009
[22]. 楊仁達，「物件導向技術與標準」，資訊與電腦，pp.136-141，05/1996
[23]. Roy, J., Ramanujan, A., "Understanding Web Services", IT Professional, Vol. 3, pp. 69-73, 2001
[24]. Wei Yu;Ming Cai，「基于 Web 服務的遠程數據訪問」，江南大學學報(自然科學版)，pp.536-538，5 卷 5 期，10/2006
[25]. Jia-Rui Chen;Guo-Yong Cai,「基於擴展 WSDL 變異的 Web 服務測試方法」, 計算機應用，pp.1725-1728，27 卷 7 期，07/2007
[26]. Chi Zhang;Guang-Jun Huang;Jian Wu,等，「分佈式組件與 Web 服務集成技術研究」，微電子學與計算機，23 卷 3 期，03/2006
[27]. Ren-Jin Zhang, Bin Liu，「Application of Component in Web Services Application Integration」，廣西師範大學學報(自然科學版), pp.144-147, 25 卷 4 期，12/2007
[28]. 林章鈞，「開啟 EAI 技術新頁的 Web Services」，資訊與電腦，pp.30-33, 09/2002。

[29]. 張思源，「企業應用 Web 服務之策略」，財團法人資訊工業策進會，資訊與電腦雜誌出版, 2002
[30]. 簡西村, "Web Services 應用與發展」，資訊與電腦，pp.73-77，08/2002
[31]. 簡西村，「網路服務化：Web Services 技術與應用"，資訊與電腦，pp.86-92，10/2002
[32]. 劉遠威、黃雯汝, "Web Services 帶來整合革命", 資訊與電腦, 261 期 p:19-25, 2002.04
[33]. 簡瑞炤, "以 XML 標準透過 Web Service 達到異質資料庫資料交換機制", 亞洲大學資訊工程學系碩士論文, 2006
[34]. W3C, World Wide Web Consortium Website, http://www.w3.ogr
[35]. Bea Weblogic, http://edocs.bea.com/wls/docs92/jcom/overview.html
[36]. Universal Description, Discovery, and Integration(UDDI), http://uddi.org
[37]. Web Services Interoperability(WS-I), http://www.ws-i.ogr
[38]. 談 Web services 的物件導向應用, http://www.microsoft.com/taiwan/msdn/columns/dotNETResearch/objapp_01.htm
[39]. XFire, http://xfire.codehaus.org/
[40]. Paul Muschamp. An introduction to Web Services [J].BT Technology Journal.2004,22(1).
[41]. Vassilis Kapsalis,onstantinos Charatsis,Manos Georgoudakis, Efstratios Nikoloutsos,George Papadopoulos.A SOAP-based system for the provision of e-services[J].2004,（26）:527～541.
[42]. Martin Tsenov.WAN communication using SOAP protocol[J]. Interbational Conference on Computer Systems and Technologies:e-Learning. 2003，406～410.
[43]. BEA - Java Specification Request (JSR)181 http://www.bea.com.tw/techdoc/05news/techdoc/01news_050418.htm
[44]. Eric Newcomer 著，深探網路服務，黃義焜譯，培生教育出版，台北，民國九十一年。
[45]. 戚玉樑、彭淑芸、張琪瑩、賴德優，Web Services 探索與應用，全華科技圖書，民國九十二年
[46]. 戚玉樑，網際服務技術導論，全華科技圖書，民國九十三年

# 註　　解

[1]. IDL : Interface Definition Language，介面定義語言。

[2]. TLB : TLB 是一種 OLE(或 ActiveX)定義檔案，它包括常數、界面(Interface)、類等的定義。你可以在 VB 的集成環境的 Project|Reference 中將 TLB 檔案加入項目，然後在 Object Browser 中看到該檔案中包括哪些常數、界面、類，而每個類又包括什麼方法和屬性。微軟提供的各種 SDK 中通常包括一個或數個 TLB 檔案以方便編程。你也可以製作 TLB 檔案，首先編寫一個 ODL 檔案(在 VC++的幫助中有語法說明)，然後使用 MKTYPLIB(在 VB 光碟上有)編譯生成 TLB 檔案。類似的檔案，還有 Office 所提供的 OLB 檔案。

[3]. JSR : Java Specification Request。

# 附　錄　一

| 測試項目 | 取得資料 | 顯示資料 | 總時間 |
|---|---|---|---|
| Web Services | 500 | 266 | 766 |
| Web Services | 657 | 250 | 907 |
| Web Services | 516 | 250 | 766 |
| Web Services | 531 | 266 | 797 |
| Web Services | 531 | 297 | 828 |
| Web Services | 547 | 250 | 797 |
| Web Services | 579 | 296 | 875 |
| Web Services | 547 | 328 | 875 |
| Web Services | 500 | 266 | 766 |
| Web Services | 547 | 250 | 797 |
| Early Binding & In Process | 140 | 5719 | 5859 |
| Early Binding & In Process | 94 | 5656 | 5750 |
| Early Binding & In Process | 78 | 5484 | 5562 |
| Early Binding & In Process | 94 | 5437 | 5531 |
| Early Binding & In Process | 94 | 5484 | 5578 |
| Early Binding & In Process | 94 | 5297 | 5391 |
| Early Binding & In Process | 78 | 5328 | 5406 |
| Early Binding & In Process | 78 | 6719 | 6797 |
| Early Binding & In Process | 93 | 5297 | 5390 |
| Early Binding & In Process | 78 | 5469 | 5547 |
| Early Binding & In Process | 79 | 6890 | 6969 |
| Later Binding & Out Process | 157 | 26843 | 27000 |
| Later Binding & Out Process | 172 | 26593 | 26765 |
| Later Binding & Out Process | 141 | 26937 | 27078 |
| Later Binding & Out Process | 156 | 27781 | 27937 |
| Later Binding & Out Process | 156 | 26657 | 26813 |
| Later Binding & Out Process | 125 | 26422 | 26547 |
| Later Binding & Out Process | 172 | 26578 | 26750 |
| Later Binding & Out Process | 140 | 26766 | 26906 |
| Later Binding & Out Process | 141 | 28078 | 28219 |
| Later Binding & Out Process | 140 | 27329 | 27469 |
| Zero Clien | 31 | 18906 | 18937 |
| Zero Clien | 47 | 21375 | 21422 |
| Zero Clien | 47 | 19640 | 19687 |
| Zero Clien | 31 | 19125 | 19156 |
| Zero Clien | 47 | 19953 | 20000 |
| Zero Clien | 125 | 18516 | 18641 |
| Zero Clien | 31 | 19454 | 19485 |
| Zero Clien | 47 | 19609 | 19656 |
| Zero Clien | 47 | 18219 | 18266 |

毫秒