私立東海大學資訊工程與科學研究所

碩士論文

指導教授：周忠信

可用於發展網路服務導向應用系統的分享記憶服務

SMS – A Share Memory Service for Web Service Oriented Applications

研究生：陳瑞盟

中華民國九十二年七月二十八日

摘要

網路服務導向應用系統，可以視為是實作網際網路應用的主要解決方案之一。使用網路服務的主要優點，在於它們被視為是藕合度鬆散、可以動態繫結、擁有美好溝通互動的元件。然而，在實作一個網路服務時，如果它的狀態以及資料需要和其他網路服務彼此互相流通、分享時，則會變的較為複雜。在這篇論文裏，我們提出一個稱為分享記憶服務的新服務 SMS。它提供一個可以讓網路服務導向應用系統裏的網路服務，彼此間得以共享 session 內容以及分享資料的機制。此類建構在分享記憶服務上的網路服務導向應用系統，稱之為 SMS 應用系統。本研究利用 Java 技術來實證 SMS 的可行性。我們提供一組發展 SMS 應用系統所需的 API。同時為簡化程式寫作，我們也提供五個 Java 虛擬指令。透過這五個 Java 虛擬指令，發展 SMS 應用系統時可以不需直接使用 API。

關鍵字：網路服務、簡單物件存取協定、網路服務導向應用系統、分享記憶服務、SMS 應用系統。

Abstract

Web service oriented application can be considered as the next evolution technology for implementing Internet applications. The major advantage of using web services is that they can be considered as loosely coupled dynamically bound components with great interoperability. However, it becomes complicated to implement web services when their states or data are required either to be passed or to be shared by the other services. In this paper, we proposed a framework named Share Memory Service (SMS). It provides a mechanism for the services in a web service oriented application to share session context and their data for one another transparently. Web service oriented applications built upon SMS are called SMS applications. In this research, we use Java technology to verify our proposed solution. We provide a set of API for developing SMS applications. Furthermore, in order to simplify the coding effort, we also provide five new Java pseudo-instructions. With these new instructions, developers need not use the API directly when developing an SMS application.

Keywords: web services, SOAP, web service oriented applications, share memory service, SMS applications

誌謝

在研究所的研究生涯，感謝指導教授周忠信博士，在學業及生活上悉心指導與教誨。周老師除了在專業理論方面傾囊相授之外，對於研究學問的方法與態度更是學生的學習典範。於此致上由衷的感謝與敬意。

感謝口試委員謝金雲老師與鄭有進老師，在百忙之中撥冗參加，並不吝提出指導，使論文內容更加完整。

感謝我媽媽及家人親友的全力支持。感謝賴俞、宗錫、佳瑋、勃甫等實驗室所有學弟的協助。謝謝瑞男、天麒、詣靈、志斌、東興、盈仁、柏任、琇緯、正隆、大中、依恒等一群好友的加油鼓勵。

謹以此論文獻給我辛勞的媽媽及先父。

目次

圖目錄

第一章　緒論

1.1 研究動機及目的

在以往的分散式計算技術中，如 CORBA[2]和 Java RMI[1]等，相互合作的元件間彼此之藕合度極高。影響所及的最大缺點之一，就是當任何一個元件做了些微改變，都有可能造成與其互相依賴的其他合作元件無法正常運作。而這對於系統的再重複使用是很大的限制。因此目前分散式應用程式的發展趨勢之一，已經不再傾向於發展藕合度高的系統 [7, 11-13]。取而代之的 *web services* 這個被視為藕合度鬆散 (*loosely coupled*)、可以動態繫結的新技術，將在發展新一代的分散式應用程式中佔有一席之地[3, 5, 7, 8, 11-13]。

Web service 的概念是利用統一資源辨識碼 (*URI, Uniform Resource Identifier*)來辨識應用程式。而應用程式的介面和繫結可以透過 XML 來定義、描述和發現，並且可以和其他透過 *Simple Object Access Protocol* (*SOAP*) [6]使用 XML 為基礎來傳遞訊息的服務直接互動。使用 web services 的好處在於它們可以讓應用程式的各個功能透過標準的方法在網際網路 (Internet)上流通。以往一般的應用程式都必須遵造設計者自己制定的嚴格協定才可以啟動，而現在的應用程式可以遵循 Internet 的相同架構來操作使用。更進一步來說，應用程式可以彼此溝通合作，不需要理會它們是用什麼程式語言撰寫、它們是在哪個作業平台上發展或是它們自己內部是使用何種協定。

Web Service oriented Application (*WSoA*)是一組彼此相互合作，完成一個特定任務的 web services。我們把在 WSoA 中，web services 的呼叫順序稱之為 *service flow*。相較於以往傳統應用程式的發展流程，在發展 WSoA 時，透過大家都同意的規格，發展者可以動態的整合所需要的 web services，將相關的 web services 佈署在一起，形成應用程式。事實上，我們可以將 WSoA 視為是實作網際網路應用的主要解決方案之一。毫無疑問的，WSoA 將是軟體架構的下一波進化。

我們在先前的段落中提到過，使用 web services 的主要優點在於它們被視為藕合度鬆散、可以動態繫結、擁有美好溝通互動的元件。但是這個優點在考慮到以下這二個問題時卻可能造成實作上的困難。第一個問題是關於在一個 service flow 中，如何將狀態

值 (state)和 session 的內容在不同的 web services 中互相流通。第二個問題是關於在 WSoA 中，web services 彼此分享資料的需求。事實上，在實作一個 web service 時，如果它的資料需要和其他 web services 彼此互相流通、分享，則會變的很複雜。而且如果愈多的資料需要流通、分享，則 SOAP 訊息將要創造愈多的參數來記載這些資料。一般而言，當 SOAP 訊息內容愈多，處理起來將愈不方便。

除了以上所提的二個有關於實作方面的問題，從程式設計的觀點來看，如果在 WSoA 中可以使用分享的資料，將可以大大的簡化系統設計的流程。因此，在這篇論文中，我們提出了一個稱之為 *Share Memory Service* (*SMS*)的服務，它提供了讓 WSoA 裏的 web services 可以很簡單的傳遞 session 的內容、分享彼此的資料的機制。我們將建構在 SMS 之上的 WSoA 稱之為 *SMS applications*。利用 SMS 架構來發展 WSoA 的優點是很顯而易見的。舉例來說，如圖 1.1 左所示，在 WSoA 裏有 n 個 web services，分別是 WS_1、WS_2、、、WS_n。如果這 n 個 services 間要彼此互相共享狀態或資料時，它的傳輸機制之複雜度將是 $O(n^2)$。但是同樣的情況，如圖 1.1 右所示，SMS application 的複雜度將只有 $O(n)$。

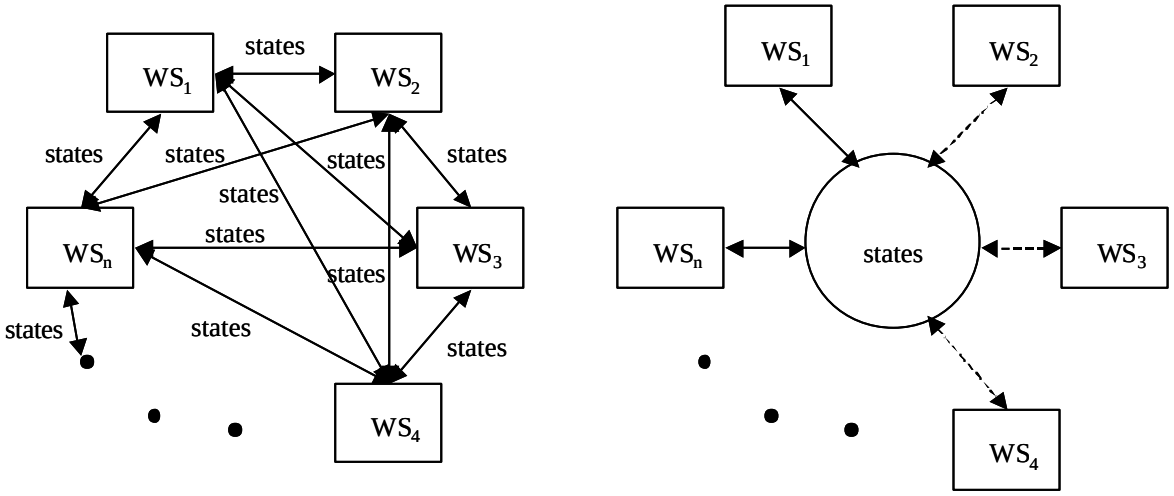


圖 1.1 一般 WSoA(左)和 SMS application(右)的差別示意圖

在這篇論文裏，我們將提供一組撰寫構成 SMS application 之 web services 所需的 API。我們利用 Java 的技術來實證我們的解決方案確實可行。更進一步地，為了簡化程式寫作，我們也提供了五個 Java 虛擬指令。透過這五個 Java 虛擬指令，發展 SMS 應用

系統時可以不需直接使用 API。

1.2 章節內容

本論文結構如下：在第二章中我們將介紹 web services 的概念。第三章則介紹 SMS 的架構及其元件說明。第四章說明我們設計的 API 及虛擬指令，並對 WSoA 和 SMS application 進行分析比較。第五章則是本論文的結論。

第二章　背景資料

Web service 技術是建立在現存的標準之上，如 *HTTP*, *XML* (*Extensible Markup Language*), *SOAP* (Simple Object Access Protocol), *WSDL* (*Web Services Description Language*) and *UDDI* (*Universal Description, Discovery and Integration*) [4, 6, 8-10]。為了能更深入了解接下來的章節內容，我們簡單說明 web service 的概念。

2.1 Web Services

Web service 可被視為是一個描繪可透過網路操作運算的介面。利用標準的正式 XML 符號來描繪 web service 所提供的服務。它包含了對客戶端或對其他想和這個 web service 合作的 web services 所有相關細節。要描繪一個 web service，基本上必須包含訊息格式、傳輸協定以及這個 web service 的所在位置。Web service 架構是建立在三個角色—服務提供者 (Service Provider)、服務註冊者 (Service Registry) 和服務要求者 (Service Requestor)彼此間的三個運作，分別是—發表 (Publish)、尋找 (Find)和繫結 (Bind)。圖 2.1 表示 web service 的運作機製。我們簡單說明如下：

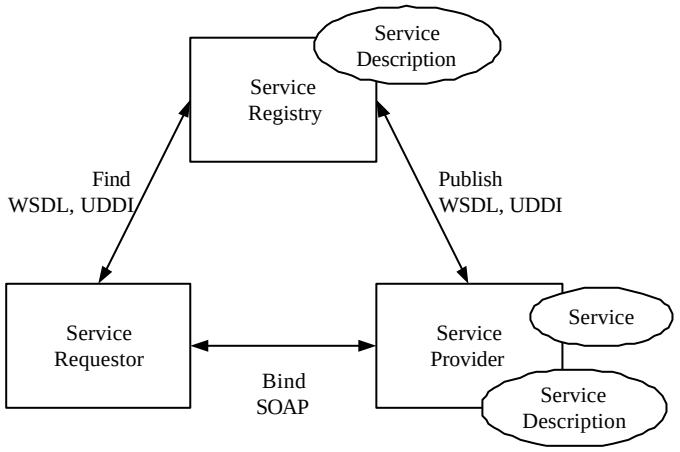


圖 2.1 Web service 架構的角色和運作

- 服務提供者 (Service Provider)：服務提供者指的是 web service 的擁有者。它必須把 web service 的定義以及如何和它協同合作的資訊用 WSDL 描述清楚上傳給服務註冊者，才能讓服務要求者找得到這 web service。
- 服務要求者 (Service Requestor)：服務要求者指的是客戶端程式。它一開始透

過服務註冊者找到符合要求的 web service，然後向這個 web service 的服務提供者要求服務。

- 服務註冊者 (Service Registry)：服務註冊者負責存放 web service 的定義以及與此 web service 合作的所需資訊，並提供服務要求者查詢之。它同時也負責管理服務提供者和服務要求者二者間的互動。
- 發表 (Publish)：這個操作讓服務提供者可以將 web service 的定義以及如何和它協同合作的資訊發表到服務註冊者，如此服務要求者才能找到需要的 web service。
- 尋找 (Find)：這個操作讓服務要求者可以詢問服務註冊者是否有符合要求的 web service。
- 繫結 (Bind)：這個操作讓服務要求者可以和它查詢到的 web service 進行聯繫、呼叫。

2.2 Simple Object Access Protocol (SOAP)

？述完 web service 的基本概念後，我們接著介紹另一個和研究主題息息相關的 SOAP。SOAP 是一個建立在 XML 上，用來在分散式環境中交換結構化資訊的輕量級(*lightweight*)協定。SOAP 本身並不定義任何關於程式語言的語法或實作，相對的，它定義一個藉著模組化的包裝模組來表達應用程式語義的機制，以及將模組內資料編碼的編碼機制。SOAP 主要包括三個部份：

- SOAP 信封：陳述包含之訊息、誰該負責處理、應該如何處理以及哪些內容是可有可無或是一定要有。
- SOAP 編碼規則：定義一組編碼規則來表示應用程式本身定義的資料型態實體。
- SOAP RPC 表示法：定義用來表示遠端程序呼叫的請求和回應之間的互動方式。

圖 2.2 為有附加檔案的 SOAP 訊息階層架構示意圖。

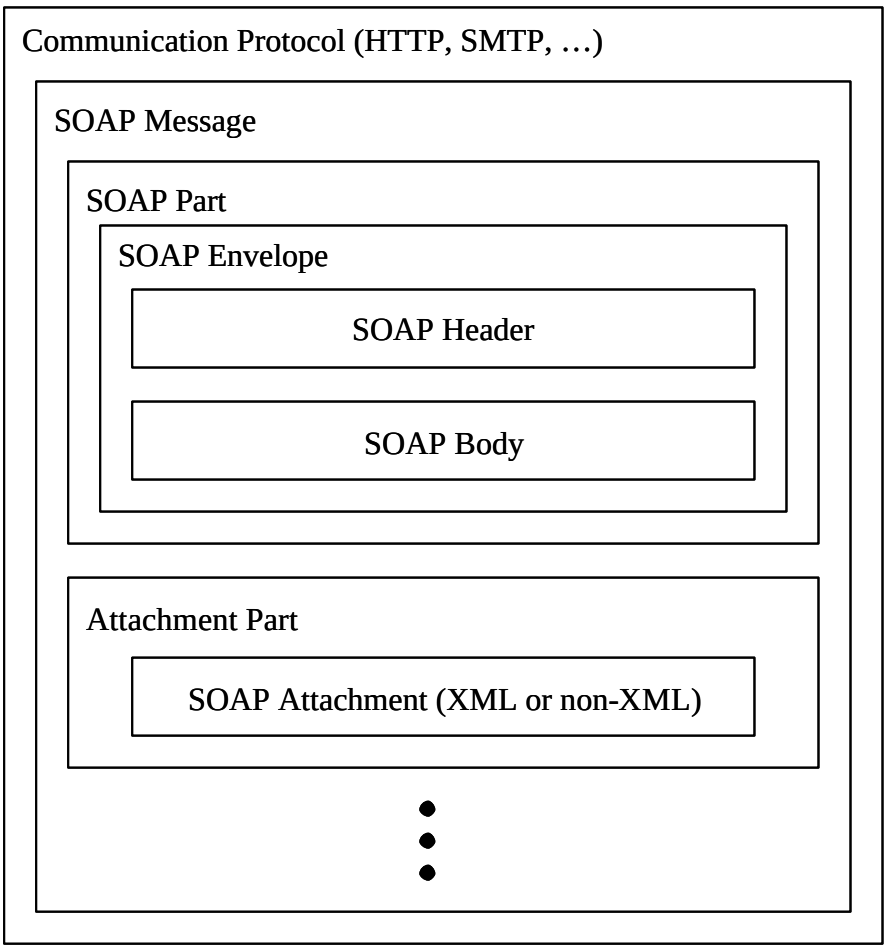


圖 2.2 有附加檔案的 SOAP 訊息示意圖

了解 web services 和 SOAP 的定義後，下章開始介紹我們的解決方案。

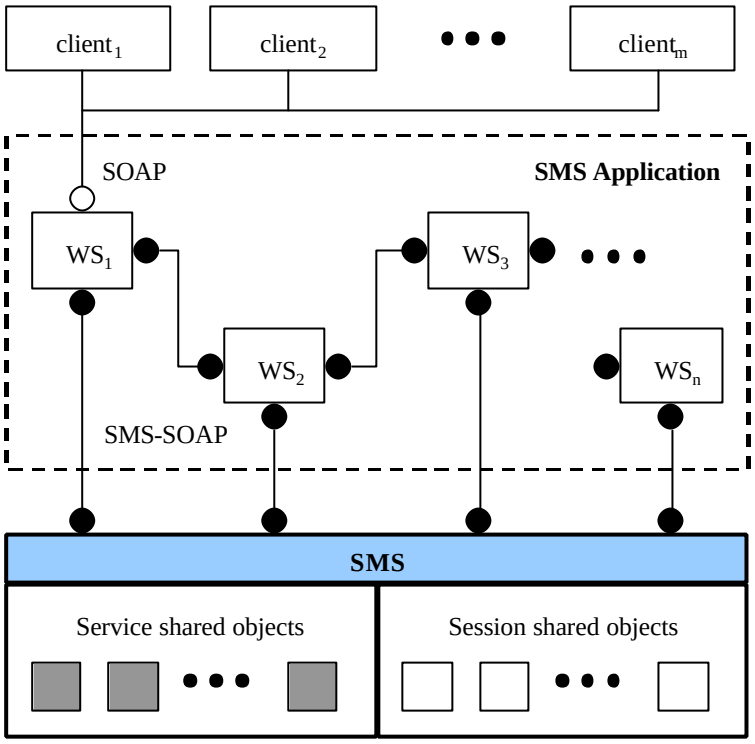
第三章　SMS 架構設計

3.1 系統架構概觀

SMS 架構是建立在 web service、SOAP 等與平台、語言無關的技術上。它的基本概念是想讓每個 web service 都可以自行定義它自己想要和其他 web services 所共同分享的資料，並且將這些分享資料所代表的意義及相關資訊發表於服務註冊者。想要一起合作的 web services 可以透過查詢得知分享資料所屬的 web service 及所代表的意義等相關資訊，然後經由簡單的宣告參考，就可以很方便、直接的存取這些分享資料來完成某些特定的工作。也就是說，SMS 架構提供了一個分享資料的儲存空間，讓所有想要提供或取用分享資料的 web services 只要遵照我們所提供的 API 或虛擬指令來撰寫，就可以無縫地 (seamlessly)和我們的 SMS 系統合作。

從程式設計的觀點來看，如此一來，程式設計師要撰寫一個 WSoA 不再需要自己從頭到尾撰寫所有的程式碼，而是可以先透過查詢機制，尋找看看是否有符合自己需求的 SMS web services 及其分享資料，然後加以利用。而如果日後發現更有效率的 SMS web services 也可以很簡單的替換掉舊的 SMS web services，大大地減少了程式發展所需的人力和時間。

SMS 架構如圖 3.1 所示，它包含了三個主要的部份，它們分別是 SMS、SMS web services 和 *SMS-SOAP*。我們在接下來的章節分別對它們做進一步的描述。



WS_1, WS_2, …, WS_n 代表 SMS web services.
傳遞連結線的實心圓端點代表透過 SMS-SOAP 傳遞訊息.

圖 3.1 SMS 架構

3.1.1 Share Memory Service

Share Memory Service 是整個 SMS 架構的核心部份。它負責處理 SMS web services 透過 SMS-SOAP 所傳遞的請求訊息，確認訊息的合法性，並將正確的分享資料透過 SMS-SOAP 回覆給 SMS web services。

SMS web services 所定義的分享資料實際上是儲存在 SMS 的記憶體空間。因此，SMS web services 必須透過 SMS-SOAP 統一向 SMS 發出請求訊息來存取所需的分享資料。為了能夠正確且快速的回應每一個來自 SMS web services 請求訊息，SMS 分二部份來達成這個目標，一是驗証請求訊息的合法性，一是正確且有效率的管理本身的記憶體空間。

當接收到 SMS web services 的請求訊息時，SMS 根據先前佈署的資訊來驗証請求訊息的合法性和存取分享資料的權限。查看請求訊息是否真的是由正確的 SMS web services 所發出，以避免有心人士傳送假訊息來竊取 SMS web services 的分享資料。在

確認完訊息的合法性之後，SMS 接著必須判斷發出請求訊息的 SMS web services 是否具有存取這些分享資料的權限。透過這些處理機制，不但可以增加系統的安全性，也同時減少了不必要的存取動作，增加系統效能。

基於分享資料的共享範圍不同，我們將儲存在 SMS 記憶體空間的分享資料進一步區分為二種類型：一種是只能被呼叫此應用程式的客戶端本身所存取的分享資料，我們稱之為 *Session shared objects*；一種是能被不同的客戶端存取的分享資料，我們稱之為 *Service shared objects*。SMS 可以同時服務不同的客戶端，每個客戶端都有屬於它自己的分享資料的實體，即在 SMS application 的執行過程當中，每個客戶端都有自己專屬的 Session shared objects，方便 SMS web services 獲得分享資料，但又不致於被其他同時執行的客戶端掠取。

儲存於 SMS 的分享資料可能同時有多個 SMS web services 要求存取，因此 SMS 必須提供一個負責資料同步化的機制，確保每個 SMS web services 都能獲得正確的分享資料。我們會在接下來的章節討論如何實作 SMS，達到上述的目標。

3.1.2 SMS web Services

SMS web services 是指透過 SMS 架構的支援可以跟其他 SMS web services 分享彼此的狀態和資料的 web services。一個可以被分享使用的資料首先必須由擁有它的 SMS web service 宣告；然後在佈署階段，SMS 根據 SMS application 所提供的佈署資訊，規劃適當的記憶體空間來儲存執行時期的分享資料；最後想要存取這些分享資料的 SMS web services 也必須在程式中宣告所要參考的分享資料，才能真正擁有存取這些分享資料的權限。在我們所提出的 SMS 架構中，為了簡化 SMS web service 呼叫另一個 SMS web service 的分享資料的互動過程，SMS web services 彼此之間的呼叫是透過 SMS-SOAP 來完成的。我們緊接著介紹 SMS-SOAP。

3.1.3 SMS-SOAP

SMS-SOAP 是一個 SOAP 協定，它讓 SMS web services 彼此可以互相溝通，並且讓 SMS 知道 SMS web services 所欲存取的分享資料為何，以達成資料分享的目的。因為在 SMS 架構之下，當 SMS web service 呼叫另一個 SMS web service 時，或是 SMS web services 想向 SMS 存取分享資料時，一些 SMS 所需要的資訊，諸如 session 等等的資訊

將會被封裝進 SMS-SOAP 的請求或回覆訊息。這些資訊是必須的，因為它可以幫助 SMS 區分是哪一個 SMS web service 觸發了 SMS，讓 SMS 可以對這些請求訊息做進一步的處理。我們將在下一個章節進一步的介紹 SMS-SOAP 的設計。

3.2 分享資料的生命週期

SMS 所提供的記憶體空間並非無窮無盡，因此我們必須善用有限的記憶體空間，讓每個呼叫 SMS application 的客戶端都可以順利執行。基本上，SMS 架構提供了三種生命週期來管理分享資料的，它們分別是：

- NEVER_SWAP：

如果某個分享資料會被大部份的 SMS web services 時常存取的話，可以考慮宣告成 NEVER_SWAP，這樣一來，當 SMS 發現記憶體空間不足而呼叫 *Memory Swapper* 這個在 SMS 架構裏負責置換記憶體的管理元件時，不會馬上置換這種常常會用到的分享資料所佔用的記憶體，以減少 I/O 動作的次數。這種分享資料所佔用的記憶體空間必須由定義宣告的它的 SMS web service 呼叫 release()這個方法後，Memory Manager 才會去釋放它所佔用的記憶體空間。

- DEFAULT_SWAP：

當分享資料被宣告成 DEFAULT_ SWAP 時，Memory Manager 會根據分享資料的寫入時間和預設的租借時間來判斷這塊記憶體空間的租賃合約是否到期，而決定是否要置換或釋放此一記憶體區塊。

- LRU_SWAP：

當分享資料被宣告成 LRU_SWAP 時，它所佔用的 SMS 記憶體空間會因 Memory Manager 發現它是最近最少用到的分享資料而被置換或釋放。

我們利用 JAVA 的技術來實作出我們的系統架構，所有的 API 和範例都是透過 JAVA 來完成的。

3.3 系統元件介紹

圖 3.2 是 SMS 的元件圖。它們分別是 Session Controller (SC)、Access Controller (AC) 、Memory Binder (MB)和 Memory Manager(MM)。我們接下來介紹它們的功能性：

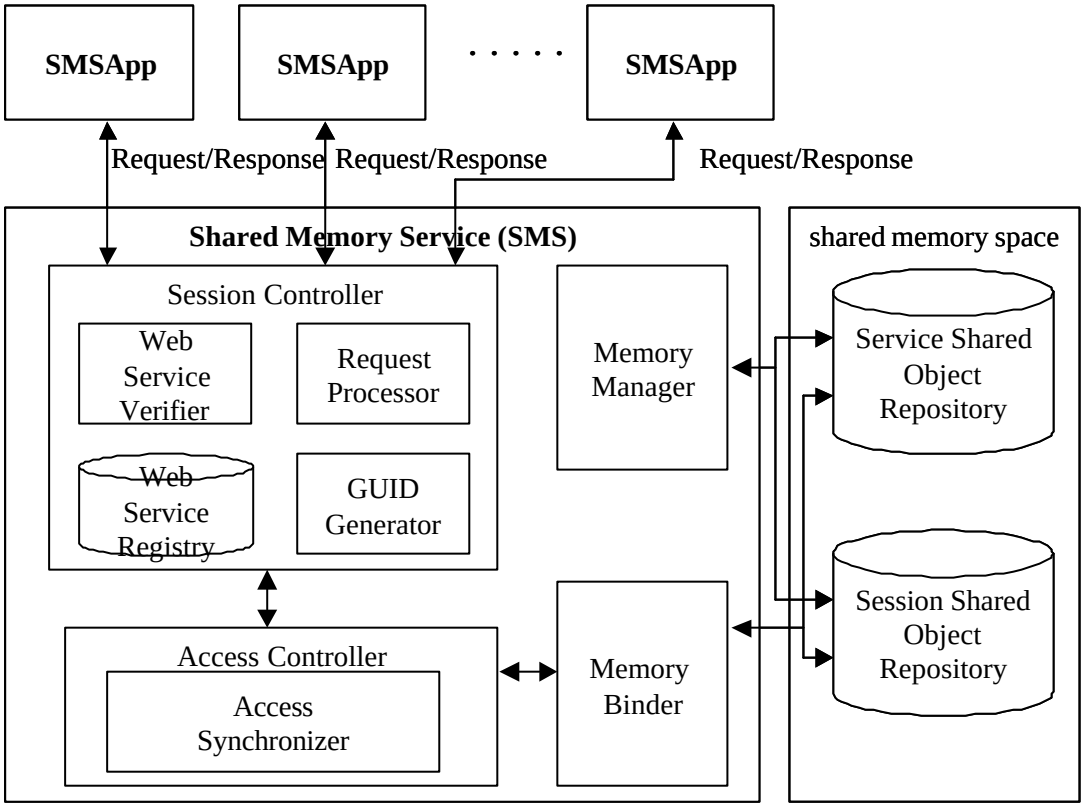


圖 3.2 SMS 元件圖

3.3.1 Session Controller介紹

SC 的主要功能為負責處理 SMS web services 透過 SMS-SOAP 所發出的請求訊息。它必須判斷請求訊息的合法性，並且從請求訊息中萃取出所有 SMS 將會用到的相關資訊，才能正確無誤的存取到分享資料。當 SMS 接收到 SMS web services 所發出的請求訊息後，由 Request Processor 負責解譯請求訊息，萃取發出訊息的 SMS web service 的 IP、Session ID、欲存取的分享資料名稱等資訊，提供給其他元件使用。根據佈署資訊，SMS 會將構成 SMS application 的 SMS web services 的相關資訊儲存於 Web Service Registry，因此 Web Services Verifier 可以比較 Web Service Registry 所儲存的資料和發出請求訊息的 SMS web service 的 IP，來確認這個請求訊息是否由正確的 SMS web service 所發出，以避免有心人士傳送假訊息來竊取分享資料。如果 SMS web service 是第一次

呼叫 SMS，GUID Generator 將會給這個 SMS web service 一個唯一的 Session ID。SMS web services 可以使用 SOAPMessageProcessor 這個 API 將這個 Session ID 透過 SMS-SOAP 在整個 service flow 中傳遞，因此，SMS 可以根據 Session ID 來判斷目前請求訊息是來自於哪個客戶端所呼叫的 SMS application。Request Processor 會將 Session ID 和欲存取的分享資料名稱傳遞給 AC 和 MB，開始對 SMS 記憶體進行存取分享資料的動作。

當 SMS application 在發展之際，同時也必須撰寫一份關於佈署資訊的 XML 檔案。我們會在稍後的章節介紹我們設計的 DTD 格式。SMS 會分析這個檔案，並且將內容資訊存入 Web Service Registry。Web Service Registry 主要記錄如下的資料：一、SMS application 的名稱和其 URI。二、構成 SMS application 的所有 SMS web services 的名稱和其 URI。三、分享資料的名稱、別名、型態、共享範圍、所屬的 SMS web service 和擁有存取權限的 SMS web services。

GUID Generator 利用 SMS 的 IP 位址和 UID 相結合的字串製造出一個全球獨一無二的字串來當 Session ID。所謂的 UID 它是根據三個數據所構成，第一個是可以唯一代表產生這個 UID 之 VM 的數字；第二個是 VM 產生 UID 的時間；第三個是為了防範萬一第一個數字和第二個時間都一樣時，可以用來加以判斷的數字。很明顯的，只要 VM 需要超過百萬分之一秒來重新啟動，而且系統時間不會被往回設定，那麼在這個 VM 上所得的任何一個 UID 都不會相同。因此，當我們將 VM 所在的 IP 位址和 UID 相結合所產生的字串絕對是一個全球獨一無二的字串，也因此很適合來當做我們在 SMS web services 中傳遞的 Session ID。

3.3.2 Access Controller 介紹

由於 SMS web services 將分享資料儲存在 SMS 的記憶體空間，可預見的，當某個 SMS web service 在存取分享資料時，另一個 SMS web service 有可能同時也會參考到同一個分享資料。這時候，SMS 必須確保參考到同一個分享資料的所有 SMS web services 都能正常的運作，得到正確的分享資料。AC 的主要功能就是確保正在存取分享資料和物件的第一個 SMS web service 能順利完成整個 transaction，不會被中斷；同時，也確保所有欲存取同一個分享資料的其他 SMS web services 在第一個 SMS web service 完成

存取動作之後，能依序存取到正確的分享資料。也就是說，當存取分享資料時，Access Synchronizer 負責處理 concurrency control 這方面的問題，確保分享資料的正確性。我們利用 Java 的監視器 (monitors)機制來達成同步的效果，當某個 SMS web service 捉住了某個分享資料和物件的監視器之後，其他的 SMS web services 便無法對該分享資料進行存取，防止兩個以上的 SMS web services 同時存取某一個分享資料。

3.3.3 Memory Binder 介紹

MB 主要是負責整個 SMS application 分享資料的存取。當 SMS web services 宣告分享資料時，SMS 會根據分享資料的共享範圍、資料型態來決定這個分享資料所在區塊和所需的記憶體大小，並且將初始值載入，等待其他 SMS web services 來取用。如前所述，當一個請求訊息經過 SC 的萃取確認和 AC 的存取確認後，表示 SMS 已經對記憶體做好 concurrency control，因此 MB 可以進行真正的存取動作。

針對每一個呼叫 SMS application 的客戶端，如前所述，SMS 會給定一個獨一無二的 Session ID 以資鑑別，同樣的，在 SMS 記憶體中也是以 Session ID 為索引，給每個客戶端一塊記憶體空間，如圖 3.3 所示。因此，MB 第一步驟便是根據 SC 所傳送過來的 Session ID 知道要去那個記憶體區塊尋找分享資料。每一個客戶端所分配到的記憶體空間，實際上就是一個 Hashtable，因此，MB 的第二步驟便是根據分享資料的分享範圍來判斷是要在這個 Session ID 的記憶體區塊尋找 Session shared object 或是去專門存放 Service shared objects 的記憶體區塊尋找。接著透過分享資料的名稱快速的在 Hashtable 上找到實際的對應值，做進一步的讀取、儲存等動作。

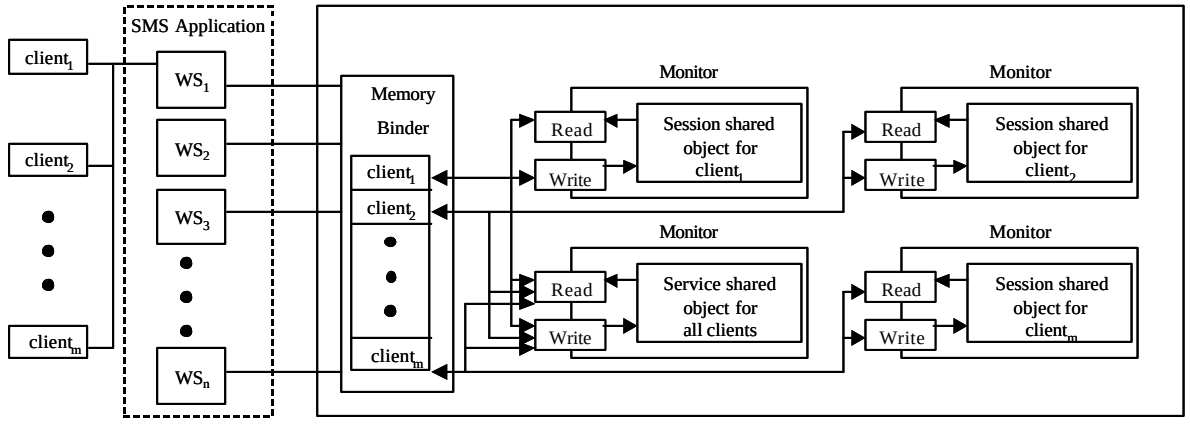


圖 3.3 Memory Binder 示意圖

根據以上的說明，我們將 SMS web service 第一次向 SMS 發出寫入分享資料之請求訊息到 SMS 回傳寫入完成之回覆訊息的整個互動過程顯示於圖 3.4 的循序圖中。

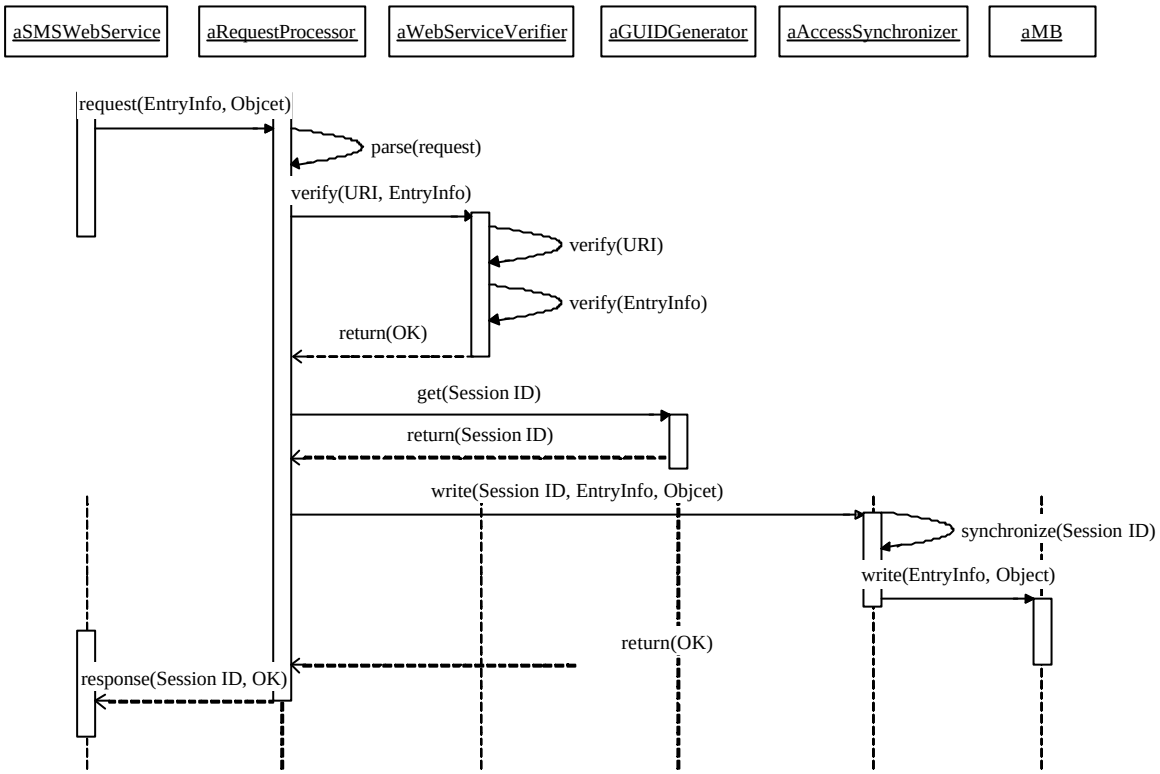


圖 3.4 SMS web service 第一次寫入分享資料之循序圖

3.3.4 Memory Manager 介紹

MM 主要是負責置換 SMS 的記憶體空間。如前所提，由於 SMS 的記憶體空間並非取之不盡、用之不竭，因此為了有效的利用記憶體空間，我們設計了三種生命週期來管理分享資料，它們分別是 NEVER_SWAP、DEFAULT_SWAP 和 LRU_SWAP。而 MM 啟動的時機有二個，其一是 SMS 發現記憶體空間的使用比率超過預設值，其二是 SMS 每隔一段固定時間。二個機制同時並行，置換掉一些比較不常用的分享資料，讓更多客戶端可以享受服務。

當 MM 進行記憶體置換時，首先會查看宣告成 LRU_SWAP 的分享資料是否可以被置換，如果記憶體空間還是不夠時，會進一步檢查宣告成 DEFAULT_SWAP 的分享資料的租賃合約是否到期，可否被置換。雖然 NEVER_SWAP 分享資料的記憶體空間在 SMS

web services 呼叫 release()之前不會被 Memory Manager 所置換，但也可能因此造成 SMS 記憶體空間的不必要浪費，所以最好還是根據分享資料的適用特性，選擇適當的管理方式。

3.4 佈署資訊

這一節介紹我們為 SMS application 佈署資訊所設計的 DTD，其格式如圖 3.5 所示。SMS application 在發展時，必須依照這份 DTD 撰寫 XML 檔案，主要目的就是讓 SMS 可以根據這份 XML 檔案得知構成整個 SMS application 的 web services 及分享資料的相關細節，記錄於 Web Service Registry，當 SMS application 正式呼叫 SMS 時，SMS 的各元件就可以直接查詢 Web Service Registry 完成正確的存取控管。

文件一開始先說明 SMS application 的基本資訊，包括了 SMS application 的名稱、SMS application 的 URL、由哪些 SMS web services 構成以及呼叫的 SMS 的名稱資料。接著在描述 SMS web service 時，除了說明 web service 的名稱和 URL 之外，最重要的就是必須針對自己所定義或是所參考的分享資料做進一步的描述。

在描述自己所定義的分享資料時，除了名稱、流通於 SMS application 的別名、資料型態之外，還要說明它是屬於 Service shared object 還是 Session shared object，其他的 web services 有沒有讀取或是寫入的權限。至於描述參考的分享資料時，就只要說明名稱和參考的別名即可。

最後在說明 SMS 時，我們利用 variable-catalog 將所有的分享資料分成二大類來描述，一是 Service shared object，一是 Session shared object，讓 SMS 可以快速的查出分享資料的分享範圍。

```
<?xml version = “1.0” standalone=”yes”?>
<!DOCTYPE DOCUMENT [
<!ELEMENT DOCUMENT (SMSApp)>
<!ELEMENT SMSApp (SMSApp-name, SMSApp-url, description?,web-service+,SMS)>
<!ELEMENT SMSApp-name (#PCDATA)>
<!ELEMENT SMSApp-url (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT web-service (web-service-name, web-service-url, description?,
variable-def*, variable-ref*)>
<!ELEMENT web-service-name (#PCDATA)>
<!ELEMENT web-service-url (#PCDATA)>
<!ELEMENT variable-def (variable-name, variable-type, description?, variable-scope,
variable-persistence, variable-alias, variable-permission+)>
<!ELEMENT variable-name (#PCDATA)>
<!ELEMENT variable-type (#PCDATA)>
<!ELEMENT variable-scope (Service | Session)>
<!ELEMENT Service EMPTY>
<!ELEMENT Session EMPTY>
<!ELEMENT variable-persistence (Y | N)>
<!ELEMENT Y EMPTY>
<!ELEMENT N EMPTY>
<!ELEMENT variable-alias (#PCDATA)>
<!ELEMENT variable-permission (permission-value, allowed-web-service+)>
<!ELEMENT permission-value (R |W)>
<!ELEMENT R EMPTY>
<!ELEMENT W EMPTY>
<!ELEMENT allowed-web-service (#PCDATA | ALL)>
<!ELEMENT ALL EMPTY>
<!ELEMENT variable-ref (variable-name,variable-alias)>
<!ELEMENT SMS (SMS-name, description?, variable-catalog)>
<!ELEMENT SMS-name (#PCDATA)>
<!ELEMENT variable-catalog (service-variable*, session-variable*)>
<!ELEMENT service-variable (variable-alias, web-service-name, variable-name)>
<!ELEMENT session-variable (variable-alias, web-service-name, variable-name)>
]>
```

圖 3.5 SMS application **佈署資訊** DTD

3.5 總結

我們所提出的 SMS 架構，基本上是希望提供資料分享的架構，並且可以和現行的 web services 共同合作。在圖 3.1 中，空心圓端點代表標準的 SOAP 訊息，實心圓端點代表 SMS-SOAP。SMS application 的客戶端可以透過標準的 SOAP 訊息來呼叫 SMS application，也就是說不用大幅修改原先的程式碼便可以呼叫 SMS application，沒有整合上的問題。

第四章　SMS 實作

4.1 SMS API 介紹

在 SMS 架構下，我們提供了一組 Java SMS API 來幫助程式設計師發展 SMS web services，請參照附錄 A。我們簡略的描述這些 API 的主要功能。

SOAPMessageProcessor 這個 API 負責創造與處理用來跟 SMS 及 SMS web services 互相溝通的 SMS-SOAP 訊息。setSession(SOAPMessage,SOAPSession)將原本的 SOAP 訊息加入和程式運作息息相關的 SOAPSession 資訊，形成 SMS-SOAP 訊息。GetSession(SOAPMessage)則是從 SMS-SOAP 訊息萃取出 SOAPSession，讓 SOAP API 做進一步的處理。

SOAPSession 這個 API 是用來將 session 的相關資料封裝，然後再由 SOAPMessageProcessor 將它加入 SOAP 訊息。session 在建立的時候，SMS 會設定創造時間和預設的存活時間，以便根據到期時間決定要不要保留此 session 相關的分享物件，避免很久沒用的分享資料繼續佔用 SMS 的分享記憶體空間。利用 GetSessionID() 獲取由 SMS 所給定的獨一無二的 Session ID。而 getCreatedTime()和 getExpiredTime() 分別傳回 session 的創造時間和到期時間。

EntryInfo 這個 API 用來建構分享資料實體的相關資訊 以 EntryInfo(String objName, String class, String webServiceName, Int memoryManagementType, long leaseTime)這個建構式為例，參數所代表的意義分別是分享資料的物件名稱、分享資料的類別名稱、分享物件所屬的 SMS web service 名稱、分享資料的生命週期模式，分享資料的租借時間。SMS 就是根據 EntryInfo 和 Session ID 找到正確的記憶體空間。上述各個參數的實際值或預設值可以分別利用 getObjectName()、getClassname()、getWebServiceName()、getMemoryManagementType()、getLeaseTime()取得。

SMSContextFactory 這個 API 是用來產生 SMSContext。SMS web service 利用 SMSContext 提供的 lookup(String SMSName)這個 method 找到名稱為 SMSName 的 SMS，然後可以進一步的與其合作。

SharedMemoryService 這個 API 提供 read(String sessionID, EntryInfo info)和 write(String sessionID, EntryInfo info, Object obj)這二個 method 讓 SMS web service 讀取和寫入儲存於 SMS 記憶體空間的分享資料。

我們也提供三個例外處理。SMSInternalException 表示 SMS 執行時期的例外。ReadException 表示分享資料和物件目前無法讀取。WriteException 表示分享資料和物件目前無法被寫入。

4.2 SMS 虛擬指令

為了簡化程式寫作，我們也提供了五個 Java 虛擬指令，分別為 GetSOAPSession、SetSOAPSession、ExternDef、ExternRefR 和 ExternRefW。透過這五個 Java 虛擬指令，發展者將可以不須要直接使用到 API，可以更簡單的撰寫出 SMS web service。我們接下來介紹如何使用這五個 Java 虛擬指令以及轉換後的程式碼為何。

- GetSOAPSession 是用來從 SOAP 訊息萃取 session 資訊。寫法如下所示：

```
GetSOAPSession request_message;
```

轉換後的程式碼為：

```
SOAPMessageProcessor processor = new SOAPMessageProcessor();
SOAPSession _SESSION = processor.getSession(request_message);
```

- SetSOAPSession 是用來將 session 資訊嵌入 SOAP 訊息。寫法如下：

```
SetSOAPSession response_message;
```

轉換後的程式碼為：

```
SOAPMessageProcessor processor = new SOAPMessageProcessor();
processor.setSession(response_message, _SESSION);
```

- ExternDef 是用來定義 Session shared objects 和 Service shared objects。如果是定義 Session shared object，則在 ExternDef 這個虛擬指令之後加上 Session shared object 的資料型態以及 Session shared object 的名稱，並在最後將其在整個 SMS application 流通的別名用角括號括起來。如下所示：

```
ExternDef Object objA <aliasA>;
```

轉換後的程式碼為：

```
try{
SharedMemoryService _service = SMSContext.lookup(SMSName);
EntryInfo info = new EntryInfo(“objA”, “Object”);
_service.write(_SESSION, info, objA);
}catch(Exception e){}
```

如果是定義 Service shared object**，則在** ExternDef**這個虛擬指令之後必須先加上這個** SMS web service**的名稱，並且用方括號括起來，然後加上** Service shared object**的資料型態以及** Service shared object**的名稱，在最後將其在整個** SMS application**流通的別名用角括號括起來。要特別注意的一點是，為避免混淆，在同一個** SMS application**中，所有的別名名稱必須是唯一的。**

```
ExternDef [WebServiceName]Object objB <aliasB>;
```

轉換後的程式碼為：

```
try{
SharedMemoryService _service = SMSContext.lookup(SMSName);
EntryInfo info = new EntryInfo(“objB”, “Object”,
“WebServiceName”);
_service.write(_SESSION, info, objB);
}catch(Exception e){}
```

- **當** SMS web services **想要讀取分享資料，它必須使用** ExternRefR **這個虛擬指令。不論是要參考** Session shared object **或是** Service shared object**，寫法都是在** ExternRefR **這個虛擬指令之後加上分享資料的資料型態及名稱，在最後將其在整個** SMS application **流通的別名用角括號括起來。如下所示：**

```
ExternRefR Objcet objA <aliasA>{
   objA.invoke();
}
```

轉換後的程式碼為：

```
try{
SharedMemoryService _service = SMSContext.lookup(SMSName);
EntryInfo info = new EntryInfo(“objA”, “Object”);
Object objA = (Object)_service.read(_SESSION, info);
objA.invoke();
}catch(Exception e){}
```

- **當** SMS web services **想要寫入或更新分享資料時，它必須使用** ExternRefW **這個虛擬指令。不論是要參考** Session shared object **或是** Service shared object**，寫法都是在** ExternRefW **這個虛擬指令之後加上分享資料的資料型態及名稱，在最後將其在整個** SMS application **流通的別名用角括號括起來。如下所示：**

```
ExternRefW Objcet objA <aliasA>{
   objA.invoke();
}
```

轉換後的程式碼為：

```
try{
SharedMemoryService _service = SMSContext.lookup(SMSName);
EntryInfo info = new EntryInfo(“objA”, “Object”);
Object objA = (Object)_service.read(_SESSION, info);
objA.invoke();
_service.write(_SESSION, info, objA);
}catch(Exception e){}
```

本節最後以一個程式片段為例子，讓程式設計師可以進一步的了解如何利用我們所提供的虛擬指令來撰寫程式。

```
public void onMessage(SOAPMessage message) {
BillDoc bill = null;
GetSOAPSession request_message;
bill = new BillDoc();
ExternRefeR UserInfo userInfo <aliasuserinfo>{
bill.setUserInfo(userInfo);
```

```
}
ExternRefeR ShoppingCart userShoppingCart
<aliasusershoppingcart>{
bill.setCartInfo(userShoppingCart);
}
SetSOAPSession response_message;
}
```

轉換後的程式碼為

```
public void onMessage(SOAPMessage message){
BillDoc bill = null;
SOAPMessageprocessor processor = new SOPAMessageprocessor();
SOAPSession _SESSION = processor.getSession(request_message);
bill = new BillDoc();
try{
SharedMemoryService _service0 = SMSContext.lookup(SMSName);
EntryInfo info = new EntryInfo(“userInfo”, “UserInfo”);
UserInfo userInfo = (UserInfo)_service.read(_SESSION, info);
bill.setUserInfo(userInfo);
}catch(Exception e){}
try{
SharedMemoryService _service1 = SMSContext.lookup(SMSName);
EntryInfo info = new EntryInfo(“userShoppingCart”,
“ShoppingCart”);
ShoppingCart userShoppingCart =
(ShoppingCart)_service.read(_SESSION, info);
bill.setCartInfo(userShoppingCart);
}catch(Exception e){}
SOAPMessageprocessor processor1 = new SOPAMessageprocessor();
processor.setSession(_SESSION,response_message);
}
```

4.3 分析比較

SMS 架構的主要立意就是將分享資料集中管理，讓想要存取分享資料的 web services 可以很方便的直接存取所需資料，而不是透過其他的 web services 才輾轉得到。

我們分別利用一般的 web services 和 SMS web services 發展出一個專門傳輸列印文字檔案內容的 WSoA 和 SMS application。對於 WSoA，第一個 web service 負責取得文字檔案內容並且設定要透過幾個 web services 傳遞之後，再由最終的 web service 將文字檔案內容列印出來。介在第一個和最終的 web services 之間的所有 web services 都只是單純的瀏覽文字檔案內容然後傳往下一個 web services。對於 SMS application，第一個 SMS web service 負責取得文字檔案內容，將其寫入 SMS 當成分享資料，並且設定要透過幾個 SMS web services 傳遞，再由最終的 SMS web service 向 SMS 取出文字檔案內容，加以列印。所有 SMS web services 之間只傳遞用來識別的 Session ID 等相關資訊，而不需傳遞文字檔案本身的內容。

分析 WSoA 和 SMS application 完成整個傳輸列印工作這中間所花費的時間，主要可以區分為傳輸時間和資料處理時間。傳輸時間是指某個 web service 將資料傳遞到另一個 web service 所花的時間。資料處理時間是指從 web service 接收到資料，對資料內容加以處理，到準備傳遞給下一個 web service 前所花的時間。

我們針對文字檔案內容其資料量的多寡以及 web services 數目的多寡這兩個因素加以分析比較，WSoA 和 SMS application 在同樣的條件之下，完成傳輸列印任務，其總花費時間、資料處理時間和傳輸時間的增加幅度。就所得的數據、圖表證明 SMS 架構的特性。

首先，我們固定透過十個網路服務，改變傳遞的資料量，分別為 20K、40K、、、100K 時，WSoA 和 SMS application 完成的時間如圖 4.1 所示。而花在資料處理的時間如圖 4.2 所示。花在傳輸上的時間如圖 4.3 所示。由圖 4.1 可以發現當傳遞的資料量越來越大量時，兩者完成任務的時間差就越來越大。由圖 4.2 發現 SMS application 在資料處理上並沒有明顯的時間優勢，這主要是因為我們多花了部份時間在驗證網路服務的正確性以及是否有權限存取分享資料。但 SMS application 整個花在資料處理時間並不會因此而比 WSoA 遜色。由圖 4.3 可以很明顯的看出 SMS application 的優點。同樣是透過十個網路服務，當資料量大時，由於 SMS application 將資料存入 SMS，只傳遞一些必要的識別資訊，因此花在傳輸上的時間很固定，不會像 WSoA 因資料量大而暴增。

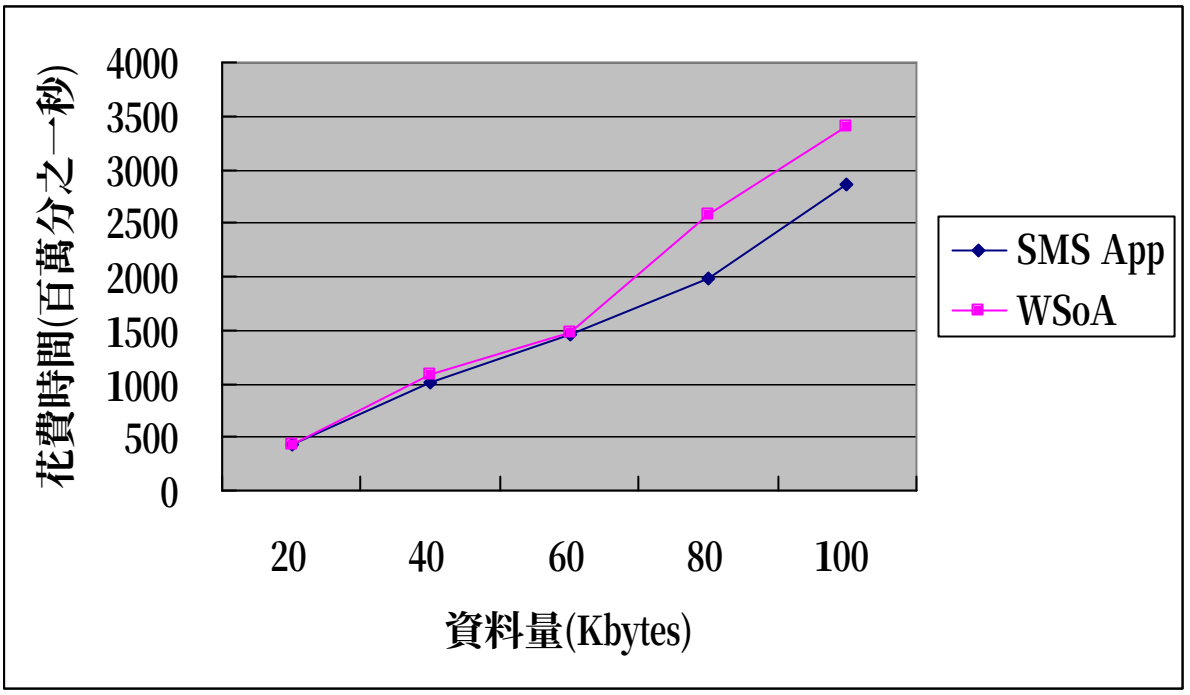


圖 4.1 SMS App 與 WSoA 透過十個網路服務傳遞資料花費時間比較圖

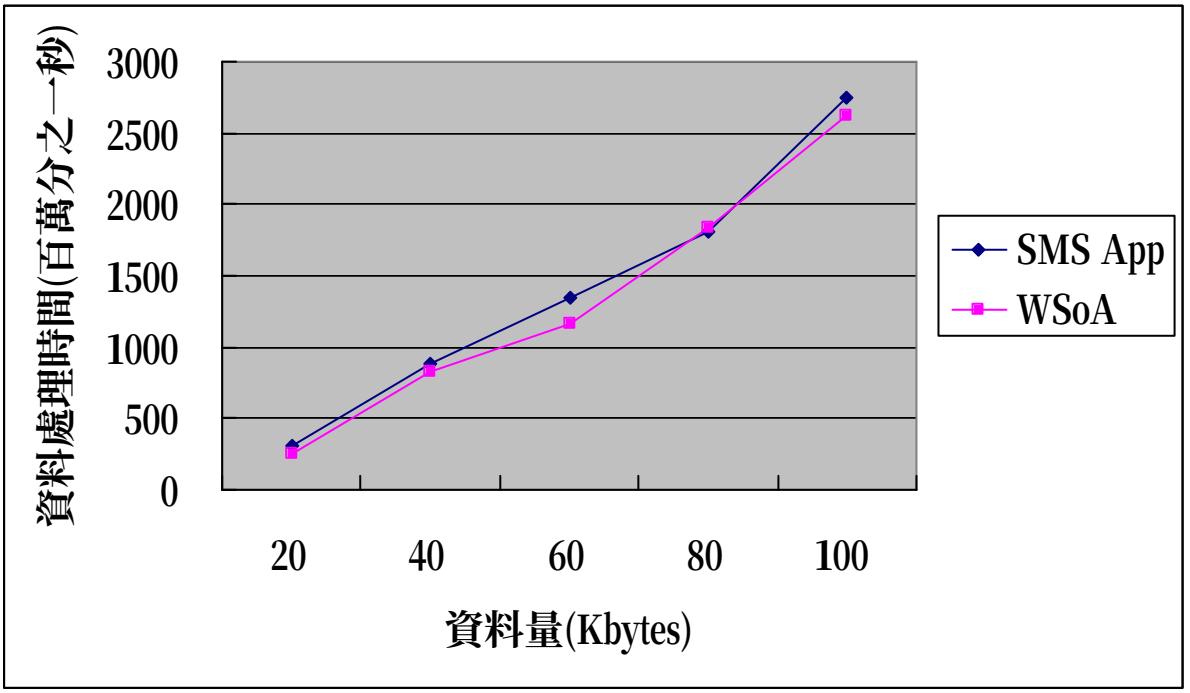


圖 4.2 SMS App 與 WSoA 透過十個網路服務傳遞資料的資料處理時間比較圖

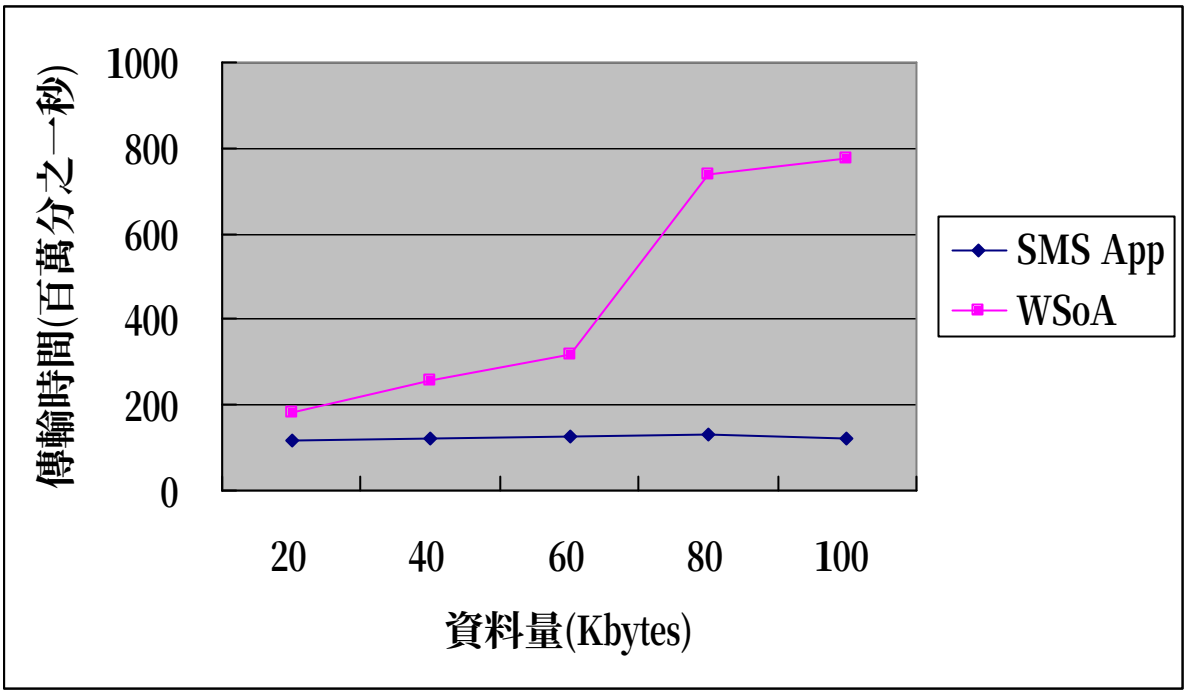


圖 4.3 SMS App 與 WSoA 透過十個網路服務傳遞資料的傳輸時間比較圖

再來，我們針對同樣的資料量，以 100Kbytes 為例，分別透過十個、二十個、、、五十個網路服務傳遞，比較 WSoA 和 SMS application 完成任務所需的時間。完成的時間如圖 4.4 所示。而花在資料處理的時間如圖 4.5 所示。花在傳輸上的時間如圖 4.6 所示。由圖 4.4 可以發現 SMS application 相較於 WSoA，佔有絕對的時間優勢。因為觀察圖 4.5 可以發現，當網路服務數目增多時，由於 WSoA 的每個網路服務均需對傳遞的資料進行判讀，所以花在資料處理時間終究會因為網路服務數目的增加而大於 SMS application 花在資料處理的時間。分析圖 4.6，由於 SMS application 將欲傳遞的資料視為分享資料儲存在 SMS，網路服務彼此之間只傳遞一些必要的識別資訊，因此雖然網路服務數目增加，SMS application 花在傳輸的時間上的增加量不會急遽增加。反觀 WSoA，資料必須經過每個網路服務接收、傳遞，因此在資料量大、網路服務數目大的情況之下，傳輸時間會大量增加。

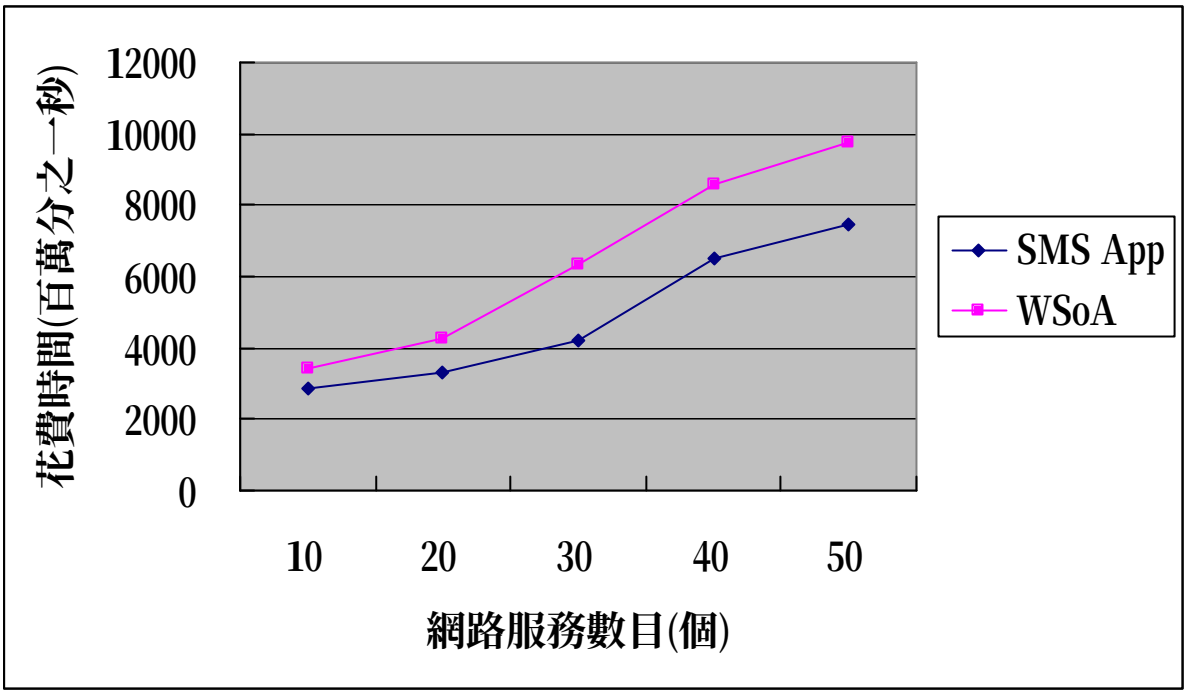


圖 4.4 SMS App 與 WSoA 傳遞 100Kbytes 資料的花費時間比較圖

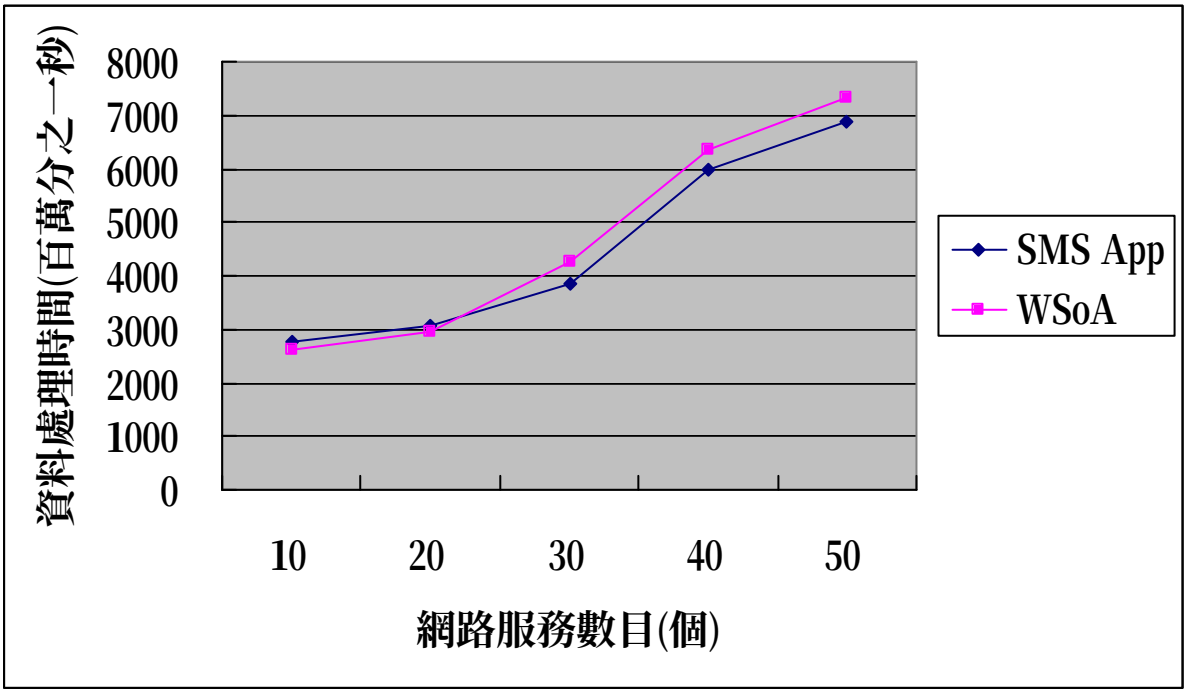


圖 4.5 SMS App 與 WSoA 傳遞 100Kbytes 資料的資料處理比較圖

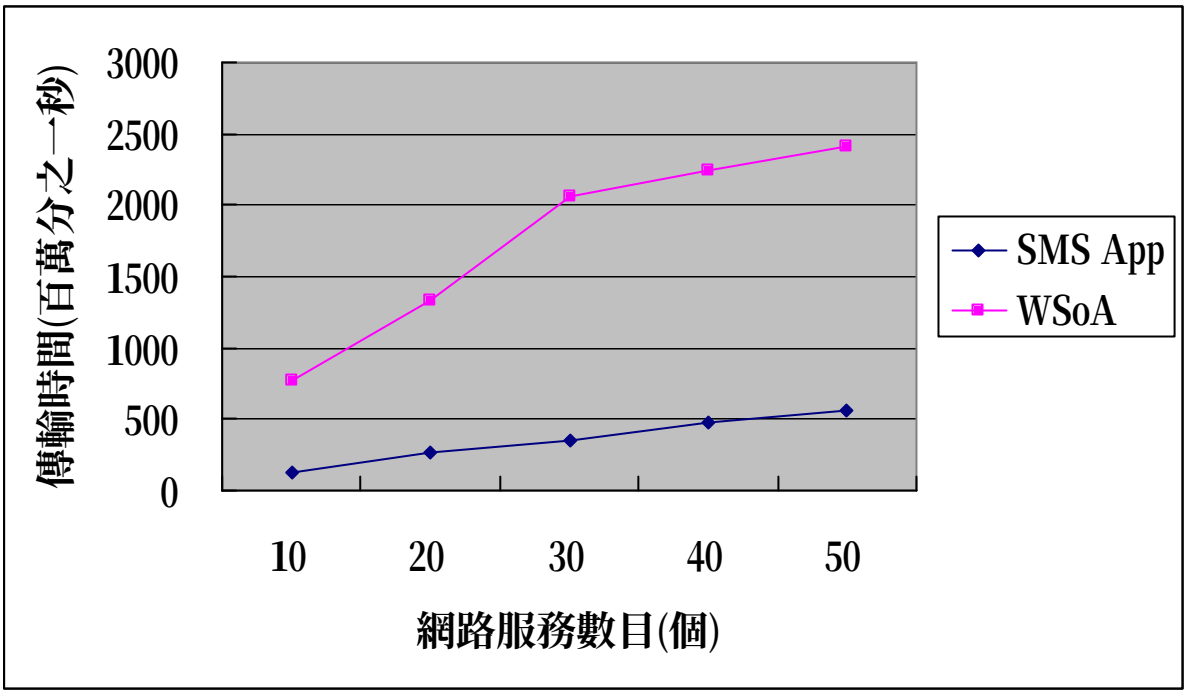


圖 4.6 SMS App 與 WSoA 傳遞 100Kbytes 資料的傳輸時間比較圖

4.4 總結

在經過實際案例測試後，可以很明顯的發現到 SMS application 相較於一般 WsoA 的優點。SMS 負責提供分享記憶體，做好存取控制，網路服務發展者在撰寫程式時可以很方便直觀的存取分享資料。從程式設計的觀點來看，程式碼不但更為簡潔，也縮短了程式發展所需時間。而且當網路服務數目越大，來回傳遞的資料量越大，SMS application 所獲得的時間優勢也就越大。

第五章　結論及未來展望

5.1 結論

在這篇論文裏，我們提出了一個發展網路服務導向應用系統的新架構。透過 Share Memory Service (SMS)架構，程式設計師可以很簡單利用我們提供的 SMS API 或是虛擬令來設計、實作出網路服務導向應用系統。另一方面也更容易引用、整合別人發展出來好用的網路服務，不用從頭到尾自己獨立發展整個網路服務導向應用系統。因此，不但能大幅降低程式設計所需的人力和時間，而且也更符合現今應用系統傾向藕合度鬆散、動態繫結的要求。

事實上，當越多的 web services 組成 WSoA，彼此互相共享狀態或資料時，越能看出 SMS 架構的優點。因為 SMS 架構的基本目標之一就是為了大幅地降低 web services 彼此分享資料的複雜度。在最差的情況下，可以將時間複雜度由 $O(n^2)$降低為 $O(n)$。

SMS 架構的另一個優點就是現行的 web services 不需要做太多的修改就能和 SMS 合作。也就是說 SMS application 可以和現今的 web services 並行不悖，互相合作。

5.2 未來工作

我們目前發展的 SMS 架構傳遞的分享資料其資料型態是基本的資料型態，如字串、整數等等。我們希望未來進一步透過序列化的技術，能夠傳遞更複雜的物件實體。另一方面，我們希望能發展更多介面化的工具，讓程式發展者可以更簡單的撰寫程式和佈署資訊。

參考資料

[1] "Java Remote Method Invocation," URL:http://java.sun.com/products/jdk/rmi/, 2003

[2] "Welcome To The OMG's CORBA website," URL:http://www.corba.org, 2003

[3] A. Bosworth, "Developing Web services," Proceedings of 17th International Conference on Data Engineering, 2001, pp.477-481.

[4] Ariba, IBM, Microsoft, "UDDI: White papers," URL:http://www.uddi.org/whitepapers.html, 2001.

[5] C. Adam, "Why Web Services," URL:http://www.webservices.org/index.php/article/articlestatic/75, 2002.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1",URL: http://www.w3.org/TR/2000/NOTE-SOAP-20000508, 2000.

[7] D. Gisolfi, "Web services architect: Part 1 An introduction to dynamic e-business," URL: http://www-106.ibm.com/developerworks/webservices/library/ws-arc1/, 2001.

[8] E. Castro-Leon, "A perspective on Web Service," URL:http://www.webservices.org/index.php/article/articleprint/113/-1/3/, 2002.

[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," URL:http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[10] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE INTERNET COMPUTING*, Vol. 6, Issue 2, 2002, pp.86-93.

[11] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)," URL:http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, 2001.

[12] IBM Web Services Architecture Team, "Web Service architecture overview The next stage of evolution for e-business," URL:http://www-106.ibm.com/developerworks/webservices/library/w-ovr/, 2000.

[13] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web services architecture," *IBM SYSTEMS JOURNAL*, Vol. 41, NO 2, 2002, pp.170-177.

附錄 A. SMS API

SOAPSession

+getSessionID():String
+getCreatedTime():String
+getExpiredTime():String

EntryInfo

+EntryInfo(String objName,String classType)
+EntryInfo(String objName,String classType,
Int memoryManagementType, long leaseTime)
+EntryInfo(String objName,String classType,
String webServiceName)
+EntryInfo(String objName,String classType,
String webServiceName,
Int memoryManagementType, long leaseTime)
+getObjectName():String
+getClassName():String
+getWebServiceName():String
+getMemoryManagementType():int
+getLeaseTime():long

SMSInternalException

ReadException

WriteException

SOAPMessageProcessor

+getSession(SOAPMessage): SOAPSession
+setSession(SOAPMessage,SOAPSession):void

SharedMemoryService

+read(String sessionID, EntryInfo info):Object throws
ReadException, SMSInternalException
+write((String sessionID, EntryInfo info,Object obj)
throws WriteException, SMSInternalException

SMSContext

#init(Hashtable env):void
+getEnvironment():Hashtable
+lookup(String SMSName): SharedMemoryService

SMSContextFactory

+createContext():SMSContext
+static getInstance():SMSContextFactory