國 立 交 通 大 學

資訊管理研究所

碩士論文

一個基於 MVC 架構的社交網路服務應用程式開發框架之設計與實作 — 以 Facebook 應用程式為例

The Design and Implementation of a Social Networking Service Application Framework Based on MVC Pattern — Case Studies of Facebook Application

研 究 生：黃 世 豪

指導教授：羅 濟 群 博士

中華民國 九十九 年 六 月

一個基於 MVC 架構的社交網路服務應用程式開發框架之設計與實作 — 以 Facebook 應用程式為例

The Design and Implementation of a Social Networking Service Application Framework Based on MVC Pattern — A Case Study on Facebook

研 究 生：黃 世 豪　　Student：Shih-Hao Huang

指導教授：羅 濟 群　　Advisor：DR. Chi-Chun Lo

國立交通大學

資訊管理研究所

碩士論文

A Thesis

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Information Management

June 2010

Hsinchu, Taiwan, the Republic of China

中華民國 九十九 年 六 月

一個基於 MVC 架構的社交網路服務應用程式開發框架之設計與實作 — 以 Facebook 應用程式為例

研究生：黃世豪　　指導教授：羅濟群 博士

國立交通大學資訊管理研究所

摘要

在 web 2.0 應用服務當道的今天，各項基於 web 2.0 概念的網路應用服務如雨後春筍般冒出，而其中表現最為亮眼的當屬社交網路服務（SNS）。社交網路服務商通常會開放其 API，使之成為一個開放的平台，以吸引第三方網路程式開發者在它的平台上開發應用程式，藉由群眾的力量來完善、豐富其平台的內容與服務。然而社交網路服務應用程式的環境競爭激烈，社交平台程式開發者基於搶占市場的因素，所以開發週期遠比一般的應用程式來得短。這造成許多社交網路應用程式的效能低落，也由於不易維護或重構成本過高，讓服務難以持續改善，甚為可惜！

本論文提出一個易用與高性能的社交網路服務應用程式開發框架來解決上述的問題。此框架是基於 MVC 設計樣式的架構，將程式邏輯、資料內容與顯示畫面分離，解決程式結構鬆散不易管理的問題，並使用 ORM 技術和快取機制來改進程式效能。透過這個框架，社交網路應用程式開發者可以大幅度的縮短開發的時間與維護成本，進而提供一個高品質的服務。根據實驗結果證明，在大型且複雜的社交網路應用程式中，本框架能減少 31%程式碼的行數，並提昇 2 倍的執行速度。

關鍵字：SNS、開發框架、MVC、ORM、Facebook 應用程式

The Design and Implementation of a Social Networking Service Application Framework Based on MVC Pattern － A Case Study on Facebook

Student : Shih-Hao Huang Advisor : DR. Chi-Chun Lo

Nation Chiao Tung University

Institute of Information Management

Abstract

Web 2.0 Applications are hot and popular today, and web 2.0 based services are rapid growth. The social networking service (SNS) is the most attractive service among the other web 2.0 based services. In order to attract third-party developers to develop applications on social network platform, social networking service providers open their own API. However, there are full of competitors in social network platform. Thus, the development cycle of SNS application is shorter than general applications. For the above reasons, many SNS applications have low performance and they are difficult to maintain.

Accordingly, this paper presents an efficient and easy-to-use social networking services application framework to solve above problems. This framework is based on MVC design pattern, and it separates business logic, data model, and user interface from program code to improve the structure of the program. It also use ORM technology and cache mechanism to improve the performance of SNS applications. Finally, the experimental results show that this framework can reduce number of lines of code by 31%, and double performance in large and complex SNS applications.

Keywords : SNS, framework, MVC, ORM, Facebook application

誌謝

首先要誠摯的感謝我的指導教授羅濟群老師，在研究所生涯的兩年裡，無論是平時待人處世的敦敦教誨，還是在論文寫作中不厭其煩的耳提面命，都使我受益良多，衷心感謝老師的辛苦指導。此外也要感謝我的口試委員：劉敦仁教授、古政元教授和郭更生教授，口試委員的建議十分受用，讓我的論文能夠更加的完整與嚴謹。

感謝實驗室的學長姊：斯寅、鼎元、銘家、志華學長和栩嘉、盈蓉、芝榕學姊，在論文研究中給予我許多幫助，並無私地分享許多寶貴的經驗給我，讓我在研究上少走許多歪路。另外也很感謝實驗室的同學：湘婷、元辰、致衡、冠儒和志健，無論是在學業上的同甘共苦，還是一同出遊的快樂點滴，都是我研究所期間最美好的回憶。當然，我也很感謝實驗室的學弟妹們，你們為我苦悶的碩二生活中，帶來了歡笑。我之所以能撐過去，你們功不可沒呀！

最後要感謝的是我的家人，感謝有你們在我的背後支持。無論在物質還是精神上讓我毫無後顧之憂，能全心全意的完成學業。沒有這些貴人們，也就沒有此文，由衷的感謝。

目錄

圖目錄

表目錄

第一章、緒論

本章為緒論，主要在說明本論文的研究背景、動機與研究目的，並指出現今社交網路服務開發者所面臨的問題，以及說明本論文的解決方法。最後簡單的介紹後續章節的內容。

1.1 研究背景與動機

近年來的網路發展中，以社交網路服務（SNS, Social Networking Service）平台成長得最為顯著。知名的社交平台，像是 Facebook、MySpace 與 Orkut 都擁有數以百萬的註冊會員，並且用戶成長率驚人。以 Facebook 為例，在 2009 年 7 月初時，臺灣使用者才 63 萬人註冊，但短短半年後（2010 年 3 月）已快速增長到 580 萬人次[17]。

社交平台的成長不只是因應全球化的趨勢，另一個重要的因素是在於其開放 API 給第三方網路程式開發者，讓他們可以在社交平台上撰寫網路應用程式，豐富平台上的內容與服務[9]。社交平台種種自由開放的態度和讓雙方共同受益的模式，吸引了眾多的網路開發者，讓社交平台的網路應用程式數量呈爆發性的成長。並且透過朋友間口耳相傳的群聚效應和眾多豐富、有趣的網路應用程式（服務）的加持下，讓社交平台的使用者迅速地擴增，熱潮蔓延至今。

然而社交平台是個開放的環境，在平台上的網路應用程式競爭激烈，開發者往往面臨沉重的時程壓力，所以開發週期遠比一般的應用程式來得短[1]。這造成了社交平台上充斥著許多結構鬆散且效能緩慢的應用程式，不利於開發者維護與管理。對開發者而言，如何快速的在社交平台建立高效且易於維護的應用程式，顯然是個重要的議題。

1.2 研究目的

本論文提出一個基於MVC設計樣式的社交網路服務應用程式開發框架，用以提供一種在社交平台上快速開發高效且易於維護的應用程式的解決方案。本框架的目標在於增進社交網路應用程式的三項能力：(1)改善程式結構鬆散的問題、(2) 增進開發效率、(3)提昇執行速度。這三項的說明如下：

(1)改善程式結構鬆散的問題

現今的社交網路應用程式一般的是採用 Script Language 來撰寫，常見的有：PHP、Python、Ruby 和 ActionScript(Flash)等。Script Language 的目的是希望讓程式設計師能快速的完成程式編寫的工作，所以通常設計得簡單易用，並且透過直譯器的幫助，擁有程式撰寫後無需編譯，直接執行的特點[3][11]。因此，Script Language 成為編寫網路應用程式的最佳選擇。Script Language 簡單易用、結構較為鬆散的特性，往往會造成程式設計師雖能快速的開發出所需的程式，但程式的彈性不佳，造成後續維護者的困難[6]。因此，本論文希望透過 MVC 的設計樣式來改善程式解構鬆散，缺乏彈性無法重複使用的情況。

(2)增進開發效率

雖說社交網路服務平台都會開放其 API 給外部第三方網路程式開法者使用，但想在平台上開發應用程式絕非易事。以開發 Facebook 應用程式為例，開發一個 Facebook 應用程式不算上基本的 Web 開發技術，還需額外掌握的技術如下：FBML(類似 HTML)、FBJS(類似 JavaScript)、FQL(類似 SQL)[17]。由此可見，開發一個社交服務應用程式所需的學習曲線並不低。因此，本論文提供了一個統一的程式介面，隱藏了不同技術的細節。讓開發者只需熟悉

此介面即可開發社交網路應用程式，大幅降低學習曲線，進而增進開發效率，減少開發時間。

(3)提昇執行速度

社交網路服務應用程式的 API call 通常需要進行大量的網路通訊動作，所以其是一個相對沉重的操作，如圖 1 所示。因此，本論文藉由批次處理與快取機制來減少 API call 所需的網路通訊次數，以提升程式的執行速度。

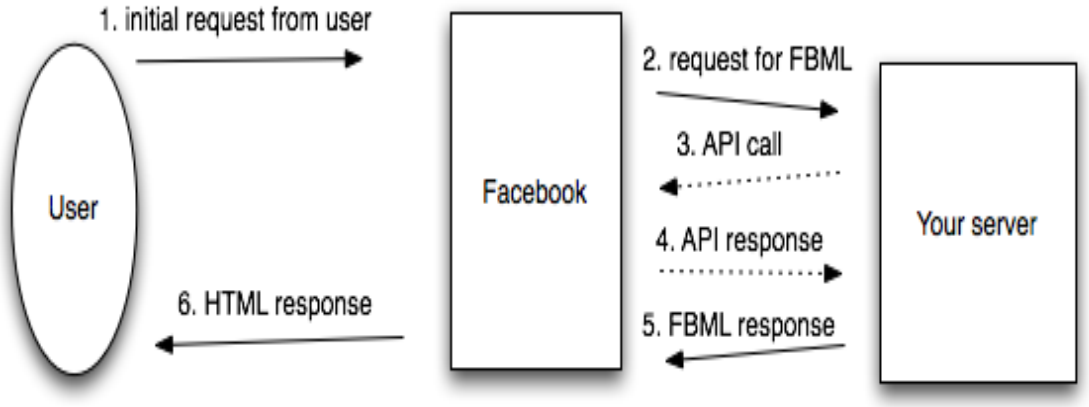

圖 1 Facebook 應用程式執行流程［17］

1.3 章節規劃

本論文的章節規劃如下：第一章為緒論，闡述研究背景、動機、目的，並概括性地描述研究的整體架構。第二章為文獻探討，概述 MVC 架構和 ORM 技術的定義與實現方法，並針對各方法之優劣進行比較。第三章為本論文提出的一個基於 MVC 架構的社交網路服務應用程式開發框架，會在此章節對

本開發框架的設計做出詳盡的解釋。第四章為實驗結果與討論，用實際的 Facebook 應用程式來驗證本框架的效果。第五章為本論文做總結，以及描述未來尚可研究的方向。

第二章、 文獻探討

在本章節裡，主要介紹及說明本論文主題相關的一些研究內容。首先會介紹 MVC 架構的定義，並說明 MVC 架構各個模組的用途與實現方式。接下來會說明使用 ORM 技術的原因與常見實現 ORM 技術的模式探討。最後會依據過去文獻的討論選出合適的快取機制方案。

2.1 MVC 架構

現今的 web-based 程式總是習慣把使用者介面(HTML、CSS、JavaScript)和伺服器端的程式(PHP、JSP、ASP)混在一起開發，所以常出現前端美工或 UI 設計人員和後端的程式設計師共同存取修改同一份檔案，因而造成衝突，並加深彼此間溝通的成本。因此這個在八十年代為編程語言 Smalltalk－80 發明的一種軟體設計模式又開始受到大家的推崇，想讓系統擺脫 HTTP 導向的開發模式，轉向較佳的軟體工程架構。

MVC 架構是軟體工程中的一種軟體架構模式，如圖 2 所示。根據[8]把軟體系統分為三個模組：事務模組(Model)、呈現模組(View)和控制模組(Controller)。它強制性的使應用程式的輸入、處理和輸出分開，藉由切割成這三個模組，將使用者介面與程式邏輯分離，讓不同的部分各司其職，使得系統的可維護性、可擴展性、靈活性以及封裝性大大提高。

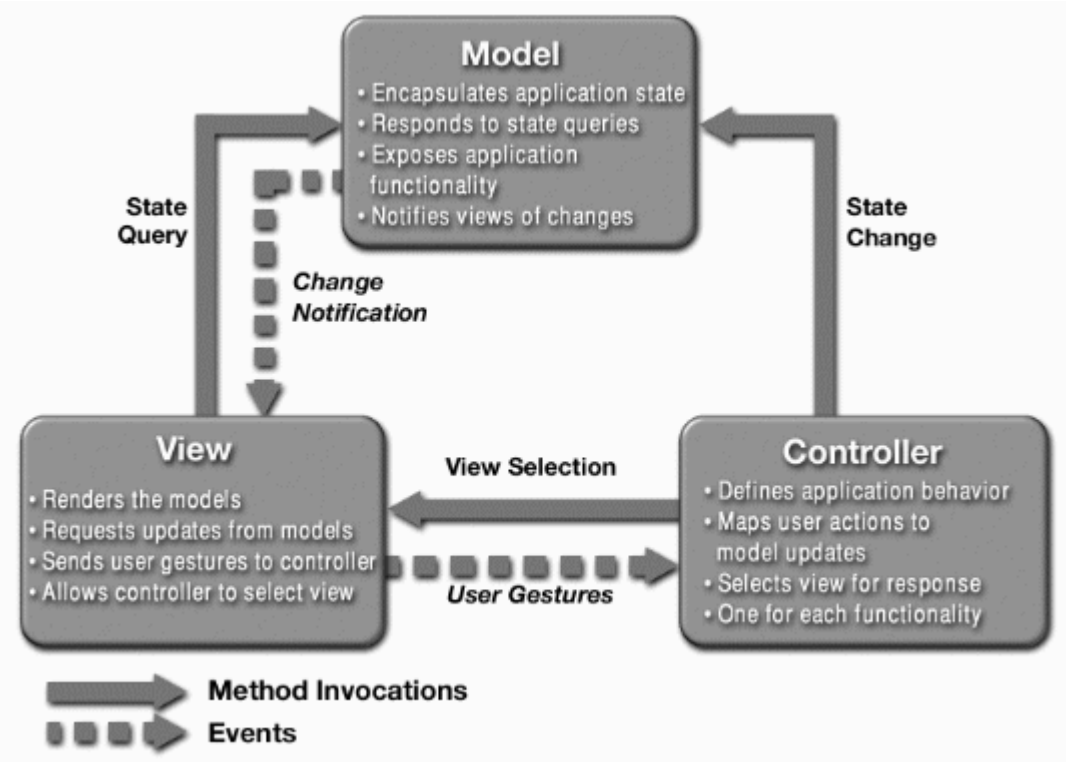

圖 2 MVC 關係圖［16］

使用 MVC 架構擁有下列的優點[5][10]：

- 能夠使用不同的呈現模組(View)來顯示相同的資料。
- 能夠輕易的改變邏輯控制。
- 提高程式碼的重用率。
- 易於後續維護與管理。
- 易於分工，例如：美工人員和程式設計師可各自負責專長的領域，針對同一專案分別開發。

2.1.1 MVC 各模組的功能

MVC 架構分成三個模組：事務模組、呈現模組、控制模組，其功能介紹如下所述。

事務模組(Model):

負責處理業務邏輯與數據操作，像是程式演算法的實作，以及對資料庫的查詢與驗證。事務模組的編寫由程式設計師負責。

呈現模組(View):

負責使用者介面的呈現，也就是使用者直接看到的畫面。通常一個系統中會有多個呈現模組，每一個呈現模組負責表達事務模組的部分狀態。當事務模組發生變化時，呈現模組能即時反應其最新狀態。

控制模組(Controller):

負責轉發程式的請求並對其進行處理。控制模組是使用者與系統之間互動的橋樑，程式的輸入(input)及輸出(output)都是由控制模組統籌處理。

簡而言之，MVC 機制是透過控制模組聯繫呈現模組和事務模組映射關係。控制模組本身不輸出任何訊息和做任何處理，它只負責將使用者的請求轉成針對事務模組的操作，和調用相應的呈現模組來顯示事務模組處理後的數據。

2.1.2 常見的控制模組設計模式

現有 web-based 的 MVC 機制在控制模組的設計上主要可分為兩類[13]：Page Controller 和 Front Controller。接著本節會簡述這兩類實現的方式並進行優劣比較。

2.1.2.1 Page Controller 模式

Martin Fowler[13]對 Page Controller 的定義為：一個負責處理特定網站頁面請求的物件。Page Controller 對網站的每一個邏輯頁面都有一個輸入控制器(Controller)，每個控制器會將負責的使用者請求聯繫起來，做出對應的處理。

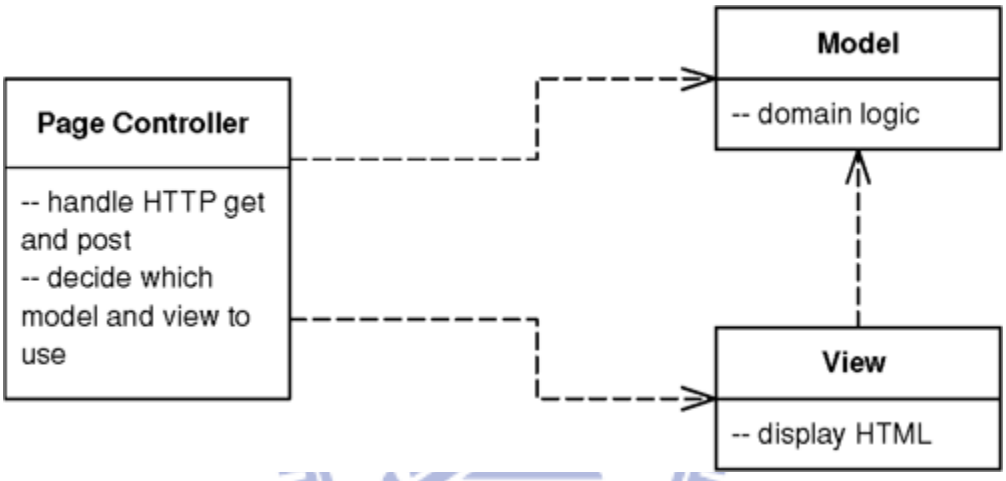

圖 3 Page Controller 的結構 [13]

如圖 3 所示，Page Controller 可接收頁面的請求、提取相關的數據、更新事務模組(Model)的內容以及向呈現模組(View)轉發請求。而呈現模組又會向對應的事務模組要求顯示的數據，三者各行其職。除此之外，Page Controller 的實現是比較容易的，因為每個 Controller 只需對特定的頁面負責，符合物件導向設計的單一責任原則[7]。

Page Controller 的主要職責有[13]：

- 解析 URL 並抽取出相關參數。
- 創建和調用相對應的事務模組來處理數據。
- 決定顯示哪一個呈現模組並把事務模組的訊息傳遞給它。

2.1.2.2 Front Controller 模式

Martin Fowler[13]對 Front Controller 的定義為：一個可以處理所有網站請求的控制模組。如圖 4 所示，Front Controller 把所有的請求通過一個處理物件(通常稱為 Handler)來集中管理，該物件可在運行時動態地進行修飾，然後把特定的請求轉發給對應的命令對象。

命令對象(Command)本身是屬於 Controller 的一部分，代表控制模組觸發的特定操作。在執行該操作之後，命令對象會選擇要使用哪一個呈現模組來顯示頁面。通常，構建此模式的框架都會使用配置文件將請求映射到操作上，以便於動態修改。因此，此模式的設計遠較 Page Controller 模式複雜。圖 5 為 Front Controller 的運作流程。

處理物件具有以下兩項職責[4]：

- 解析參數，必須接收來自 Web 伺服器的 HTTP Post 或 Get 請求，並從請求中解析相關參數。
- 選擇命令，將請求中的參數與命令對象配對，然後將控制權轉移給該命令對象，以便執行處理。

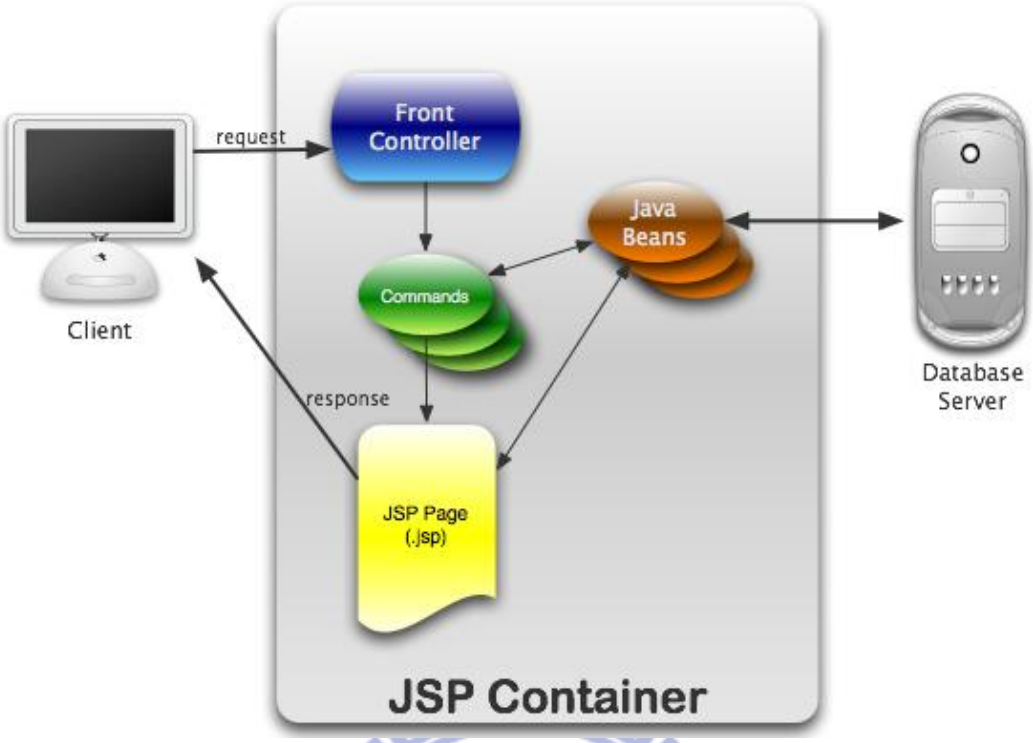

圖 4 Front Controller 的結構 [19]

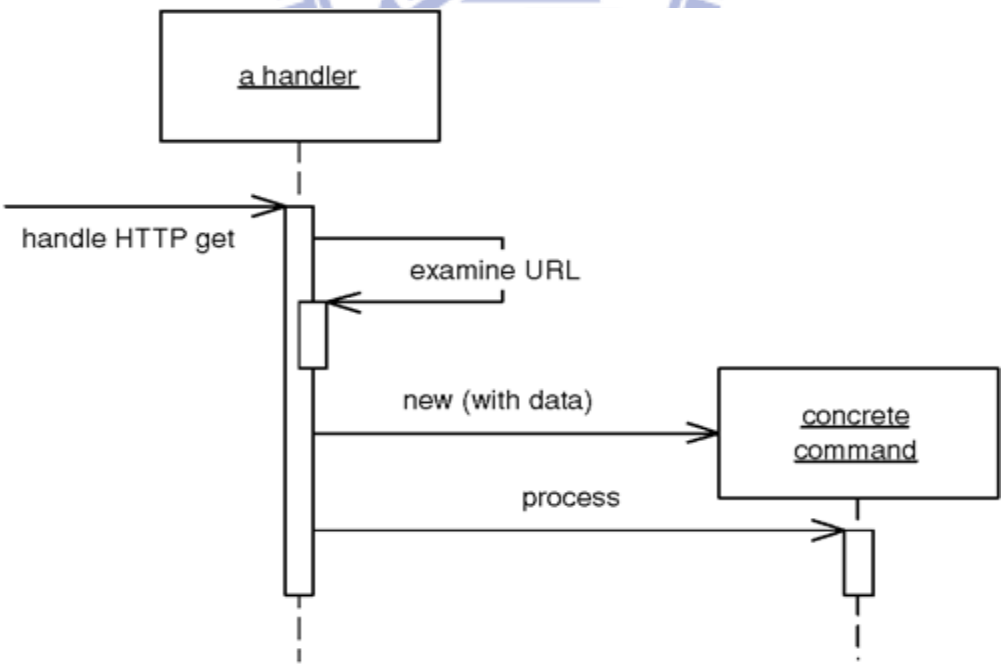

圖 5 Front Controller 的運作流程 [13]

表 1 將針對 Page Controller 和 Front Controller 的優缺點進行比較：

表 1 Controller 模式比較表

	Page Controller	Front Controller
優點	1. Controller 只進行有限範圍的處理，保持了簡單性。 2. 效能表現較佳 3. 易於實作	1. 集中化控制，易於維護與管理 2. 因集中控管之故，有較佳的安全性 3. 擴充性佳，可動態的添加命令 4. 可以將在 Page Controller 中會重複的程式碼抽取出來
缺點	1. 可能會造成 Controller 過度分散，不易維護與管理。 2. 擴充性較差 3. 由於有多個 Controllers，可能會產生重複的程式碼	1. 設計較為複雜 2. 效能表現較差

2.2 Object Relational Mapping（ORM）

ORM 技術是隨著物件導向開發方法的發展而產生的，是一種為了解決物件導向程式與關聯式資料庫存在著互不匹配現象的技術。ORM 是通過使用描述物件和資料庫之間映射的 meta data，將程序中的物件自動持久化到關聯式資料庫中。其本質上就是將資料從一種形式轉換到另外一種形式，但這也就代表在此過程中需要額外的執行開銷。然而，ORM 通常作為一種中介軟體(middleware)實現，因而存在許多機會來做優化，故在性能層面上影響不大。

ORM 技術在實現上根據[13]的分類，有 4 種常用的設計模式：(1)Table Data Gateway 模式、(2)Row Data Gateway 模式、(3)Active Record 模式 和 (4)Data Mapper 模式。

2.2.1 Table Data Gateway 模式

根據[13]定義：Table Data Gateway 是扮演一個資料庫的角色，通過它可以處理所有對資料庫表格的訪問和操作，如圖 6 所示。Table data gateway 封裝了所有操作資料庫 table 或 view 的 SQL 語法，如：select、insert、update、delete 等。當程式調用此 gateway 物件的方法即可與資料庫互動，存取資料庫的數據。

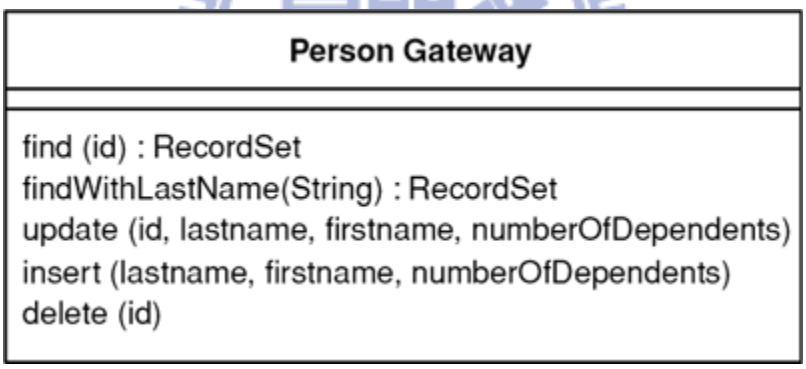

圖 6 Table Data Gateway 模式的示意圖 [13]

Table Data Gateway 的好處是提供了一個簡單的介面(interface)來存取資料庫的數據，而且它通常是無狀態的，因為它只負責資料庫數據的傳遞而已。

而使用 Table Data Gateway 需要考慮的問題是如何處理從請求中返回的多項查詢值。一種做法是返回簡單的資料結構，比如 Map；另一種做法就是使用 Data Transfer Object(一個封裝數據，純粹用於傳輸的物件)，一般建議

採用 Data Transfer Object[4]。

2.2.2 Row Data Gateway 模式

根據[13]定義：Row Data Gateway 對應資料庫中的每一筆數據記錄，它的一個實例(instance)就是一行記錄，資料庫表格中的每一欄位映射於該物件中相應的成員(field)。因此它是有狀態的，不像 Table Data Gateway 只是簡單的存取介面而已。

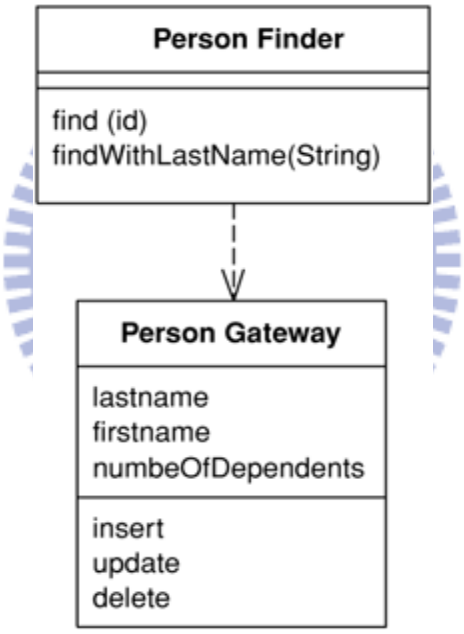

圖 7 Row Data Gateway 模式的示意圖 [13]

通常在設計 Row Data Gateway 時，會對資料庫中的每個表格對應一個查詢類別，用以返回一個 gateway 物件作為其查詢的結果，如圖 7 所示。

值得一提的是 Row Data Gateway 和 Active Record 模式十分類似，區別在於其中是否包含業務邏輯(domain logic)。Row Data Gateway 只是包含數據存取的邏輯，而沒有任何業務邏輯；而 Active Record 則包含兩者。

2.2.3 Active Record 模式

根據[12][13]定義：Active Record 包裝了資料庫表格或視圖中的一筆數據記錄，不但封裝資料庫的存取行為，也加入了該筆數據的業務邏輯，如圖 8 所示。Active Record 的資料結構應該要和資料庫表格的完全匹配，每個成員皆要對應資料庫表格中的每一個欄位。並且成員的型別要和資料庫提供的型別保持一致，並不需要在此階段手動的進行型別轉換。

Active Record 模式通常具有下列方法[12][13]：

- 透過 SQL 查詢結果建構一個 Active Record 物件
- 為插入資料庫操作預先生成一個 Active Record 物件
- 透過一個包含通用 SQL 查詢的靜態 Finder 方法，返回 Active Record 物件
- 通過 Active Record 物件更新和插入資料庫數據
- 提供 Get/Set 成員
- 可實現部分業務邏輯

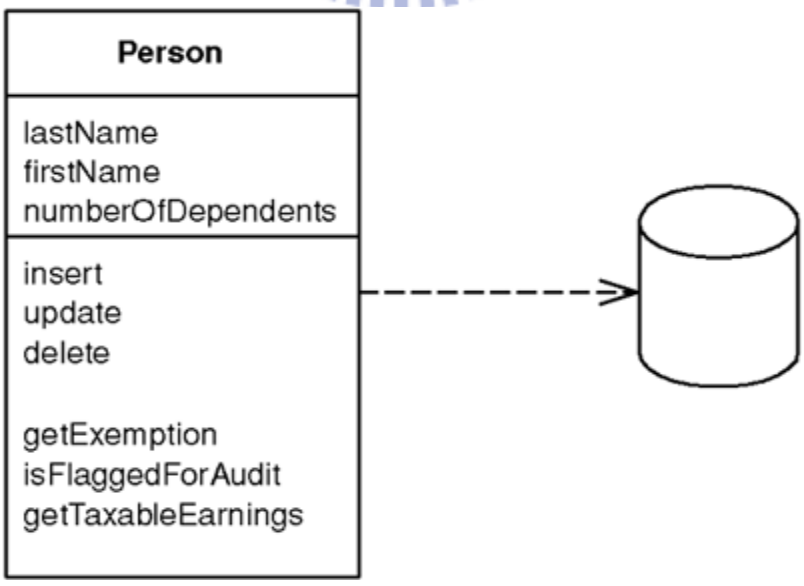

圖 8 Active Record 模式的示意圖 [13]

在業務邏輯比較簡單並且能和資料庫表格相互對應時，使用 Active Record 模式相對於 Gateway 模式更加合適。因為它包含了業務邏輯和數據存取能力，並且不需要 Data Transfer Object，減少了維護的工作量。不過 Active Record 對資料庫的耦合程度較高。

2.2.4 Data Mapper 模式

根據[13]定義：Data Mapper 是 ORM 物件和資料庫之間傳遞數據的一個中間層，使兩者保持獨立，如圖 9 所示。ORM 物件可以包含資料庫數據和業務邏輯，數據存取的邏輯由 Data Mapper 完成，這使得 ORM 物件和資料庫可以各自使用更符合自己的方式來組織數據的資料結構。Data Mapper 模式將 ORM 物件與資料庫之間的耦合程度降至最低，甚至讓使用者只需操作 ORM 物件，無需再知道 SQL 和實質的資料庫。

通常使用 Data Mapper 需要藉由外部的配置文件儲存 ORM 物件與資料庫的映射關係，例如 INI 或 XML 文件。這意味著使用此模式帶來良好的彈性之外，也增加額外的管理開銷。因此，此模式較適用於資料庫與業務邏輯映射關係複雜的大型專案[18]。

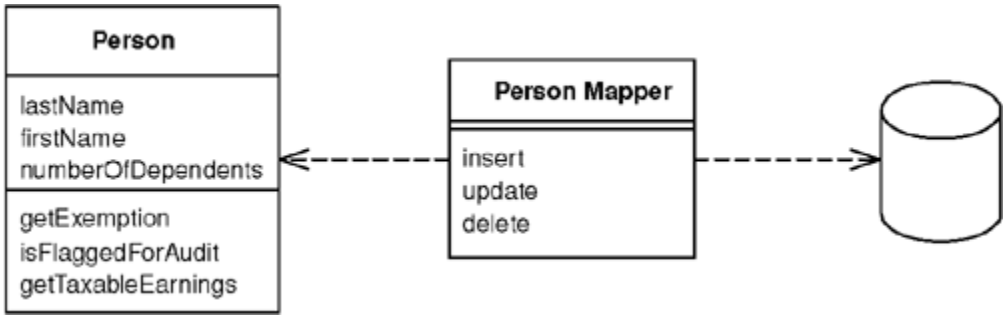

圖 9 Data Mapper 模式的示意圖 [13]

表 2 將針對 Table Data Gateway 模式、Row Data Gate 模式、Active Record 模式、DataMapper 模式等四種常見的 Object Relational Mapping 模式進行比較。

表 2 ORM 設計模式比較表

	Table Data Gateway	Row Data Gateway	Active Record	Data Mapper
複雜程度	最低	低	中	高
資料庫耦合程度	中	高	高	低
OO 程度	最低	低	中	高
維持狀態與否	無狀態	有狀態	有狀態	有狀態
是否包含業務邏輯	無	無	有	有
配置文件	通常沒有	通常沒有	可有可無	有
適用範圍	小型應用	小型應用	中型應用	大型應用

2.3 快取機制的選擇

快取機制通常是增進一個資訊系統效能表現最簡單且有效的方式，在軟體層面的快取機制設計上通常是採用基於 key-value 模式的 hash table 結構來進行資料儲存[20]。在雜湊函式的選擇上根據[15]的結果，當採用以字串為 key 時，所產生 value 的分散程度與生成速度為考量時，選擇 DJBX33A（Daniel J. Bernstein, Times 33 with Addition）演算法是最佳的選擇。其演算法如下：

$$hash(i) = hash(i-1) \times 33 + str[i]$$

如上式，DJBX33A 演算法不斷的將雜湊值乘上 33 並加上輸入字串的當前字元碼(ASCII)，直至最後一個字元。並且演算法因為選定質數 33，因此還可透過將乘法運算替換成位元運算配合加法，來提升效能。改善後的演算法[2]如下：

$$hash(i) = ((hash(i-1) << 5) + hash(i-1)) + str[i]$$

改善後的演算法在 Daniel J. Bernstein 的 Usenet Newsgroup 系統中首度使用，至現今大多數流行的雜湊函式方案(Perl、Berkeley DB、Apache、MFC、STL)皆採用此演算法。

第三章、 一個基於 MVC 架構的社交網路服務應用程式開發框架的設計

本章節會先闡述現今社交網路服務應用程式所面臨的問題，在此問題定義下，本論文提出一個基於 MVC 架構的社交網路服務應用程式開發框架。在接下來的章節中會述說本框架的設計理念，並對框架內的各個模組和快取機制做出詳盡的說明。最後簡述本框架的適用範圍與限制。

3.1 問題定義

現今社交網路服務應用程式的環境競爭激烈，社交平台程式開發者基於搶占市場的因素，所以開發週期遠比一般的應用程式來得短。但是要建構一個高品質的社交網路應用程式所需的時間不匪，要考量程式的執行效能、可維護性、重構難易度等。因此在開發社交應用程式時，其開發時程與程式的品質往往難以兼顧。而本框架想要達成的目標如下：

- 改善程式結構鬆散的問題
- 增進開發效率
- 提昇程式執行速度

因此，本論文提出一個基於 MVC 架構的社交網路服務應用程式開發框架，以 MVC 架構強制性將使用者介面與程式邏輯分離，讓不同的部分各司其職，改善程式結構鬆散的問題。並且使用 ORM 技術作為資料庫存取的中介層，加快程式開發者的開發效率。最後輔以完善的快取機制提昇社交網路應用程式的效能。

3.2 框架概述

根據[14]定義：框架(framework) 是在一個給定的問題領域內，一個應用程式的一部分設計與實現。框架是一個可重複使用的設計，它規定了應用的體系結構，協同組件之間的依賴關係、責任分配和控制流程。簡單來說，框架可以看成是一個應用程式的半成品，開發者把自己的應用程式架築在框架之上並加以擴展，即可快速地產出滿足其需求的應用程式。

本論文所提出基於 MVC 架構的社交網路服務應用程式開發框架就是一個能幫助網路開發者快速地開發社交網路服務應用程式的工具。開發者將程式建構在本框架之上，由本框架負責管理程式和 Social Networking Services(SNS)平台之間實際的 API call，並將執行結果回傳到瀏覽器，如圖 10 所示。開發者只需專注於自身業務邏輯和頁面呈現的設計，底層繁瑣的工作會由框架完成，加快開發者完成社交網路服務應用程式的速度。

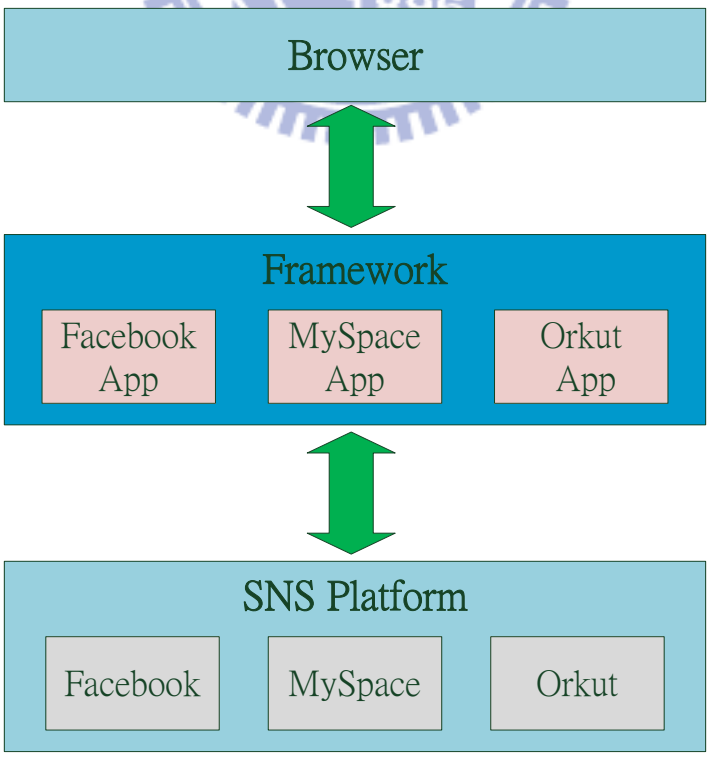

圖 10 本框架的定位

3.3 框架的設計

本論文所提出的框架如圖 11 所示，主要分成控制模組、呈現模組、事務模組、SNS API 模組和其他幫助使用者簡化開發的函式庫。

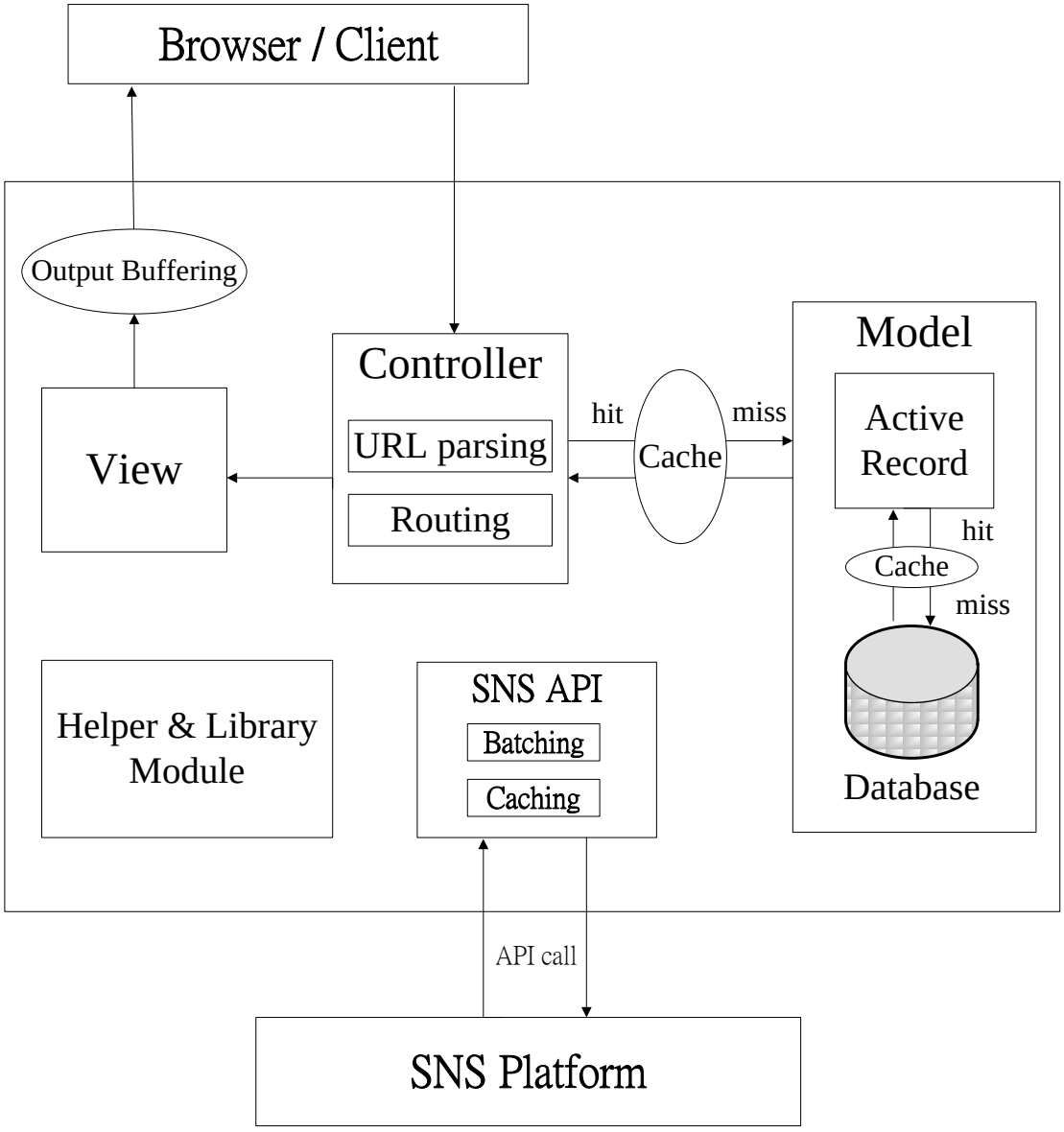

圖 11 框架的架構圖

首先由控制模組(Controller)接收來自使用者的請求，因應請求找出對應的事務模組(Model)和呈現模組(View)回應。事務模組負責業務邏輯與資料

庫數據的調用，呈現模組負責畫面的輸出，兩者各司其職，互不干擾。並且因應效能考量，在事務模組的調用提供了快取機制的加強，在呈現模組則加入緩衝機制。最後在 SNS API 模組上，採用批次處理與快取機制來減少 API call 所需的網路通訊次數，以增進效能。

接下來，此小節會針對上述的各個模組進行詳盡地介紹，並闡述本框架的快取機制。

3.3.1 控制模組(Controller)的設計

控制模組的設計是根據[4][13][19]所提出的 Page Controller 和 Front Controller 模式為基礎(Page Controller 和 Front Controller 模式請參見第二章 2.1.2 節)，針對社交網路服務平台所設計的一種介於兩者之間的混合模式，稱之為 Hybrid Controller 模式。如圖 12 所示，Hybrid Controller 模式在處理請求時，不像 Front Controller 模式般強制將所有請求都交由一個 Handler 集中控管，而是類似 Page Controller 模式將請求交由對應的 Controller 執行。在 Controller 內部運行的機制又類似於 Front Controller 模式，先將請求(URL)進行解析，從中找出對應的行動對象並執行之。

行動對象(action)是指開發者依照應用程式或服務的業務邏輯，自行撰寫的函式。在函式中，開發者可以調用相對應的事務模組來處理資料庫數據，並決定由何者呈現模組負責畫面輸出。

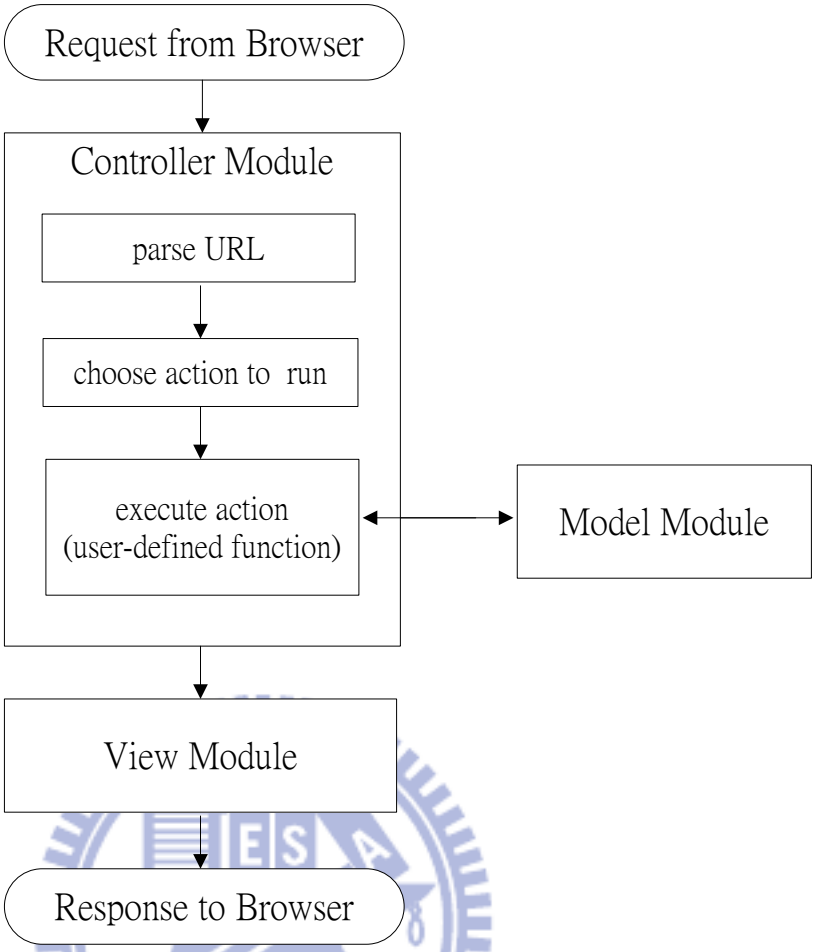

圖 12 Hybrid 模式的結構

不過在行動對象的歸屬上，Hybrid Controller 模式又與 Front Controller 模式有很大的不同。Front Controller 模式是將行動對象(或稱命令對象)與 Controller(在 Front Controller 模式稱 Handler)分離，中間透過一個抽象的介面降低行動對象與 Controller 的耦合程度，增加彈性；Hybrid Controller 模式則是將行動對象包含在 Controller 之中，如此便無 Front Controller 模式將行動對象分離後造成執行的額外開銷。並且由於本框架是基於 Script Language 開發的，所以透過其動態執行的特性，因而無需透過行動對象的分離，即可達到類似與 Front Controller 模式程度的彈性。

接著將 Hybrid Controller 模式與其他常見的 Controller 模式之差異比較整理由表 3 所列。

表 3 Hybrid Controller 模式與其他模式的比較表

	Page Controller	Front Controller	Hybrid Controller
實作難度	易	難	易
執行效能	佳	差	佳
集中控管能力	無	佳	中
擴充能力	差	佳	中
後續維護能力	差	佳	佳

3.3.2 呈現模組(View)的設計

呈現模組的設計理念則是「讓美工人員或介面設計師容易使用」為準則。此理念是基於分工的考量。在正常情況下，將程式的介面或畫面交由專業的美工人員和介面設計師，不但可取得較佳的使用者體驗，也由於分工之利，通常可有效地減少開發時程。但這些「藝術人員」往往不善於程式的編寫，更甚於對程式語言一竅不通。因此，將呈現模組包含一些程式語言的元素顯然不是一個好的主意。

然而，今日許多呈現模組是基於樣板引擎(Template Engine)實現的。樣板引擎透過解析其支援的特殊標籤(tag)，並將標籤替換成後台數據，因而達到將程式邏輯與顯示畫面分離之效。而樣板引擎往往也支援一些程式語言的

特性，例如：流程控制語法（foreach、while、if else）、函數、模板內的數學計算、正規表示式等等。這造成使用樣板引擎儼然在使用一個簡易的程式語言，似乎只有減輕程式設計師的負擔，對美工或介面設計師而言，毫無助益。

所以，本框架在設計呈現模組時不採用樣板引擎來實現。在內容渲染上採用標準的 HTML 和 CSS，在調用後台資料上使用原生 Script Language 的標籤來替換後台數據。透過這些標準標籤格式，讓美工或介面設計師可以利用視覺開發工具，無需撰寫程式即可進行頁面設計。

3.3.3 事務模組(Model)的設計

事務模組的職責是負責處理業務邏輯與資料庫的數據操作。業務邏輯會隨著企業組織的不同,有著不同的實現方式。因此根據上述事務模組的特性，在設計上採用 Design Pattern 中的 Template 模式實現。框架本身只提供事務模組的基本功能與雛形，實際的業務邏輯交由開發者自行實現。在處理資料庫數據的部份，框架提供 SQL 的包覆類別與 ORM 方案。

SQL 的包覆類別(Wrapper Class)其目的是提供一個通用的資料庫存取介面，支援不同廠商關聯式資料庫的 SQL 語法。開發者只需要使用這個通用介面即可無視不同資料庫之間的差異，輕易達成無痛的轉換資料庫，降低轉換成本。

ORM(Object Relational Mapping)的設計上，考量社交網路服務應用程式有著開發時間急迫的特性 5，決定使用 Active Record 模式(ORM 各模式的特點，請參見第二章 2.2 節)。Active Record 模式主要是透過分析資料表格的 meta data，並將分析後的資訊建構成對應此表格的 ORM 物件，物件成員

的型別和資料要與資料庫欄位保持一致。透過這個與資料庫表格完全對應的物件，取代傳統上使用 SQL 來操作資料庫的方式，其運作流程如圖 13 所示。

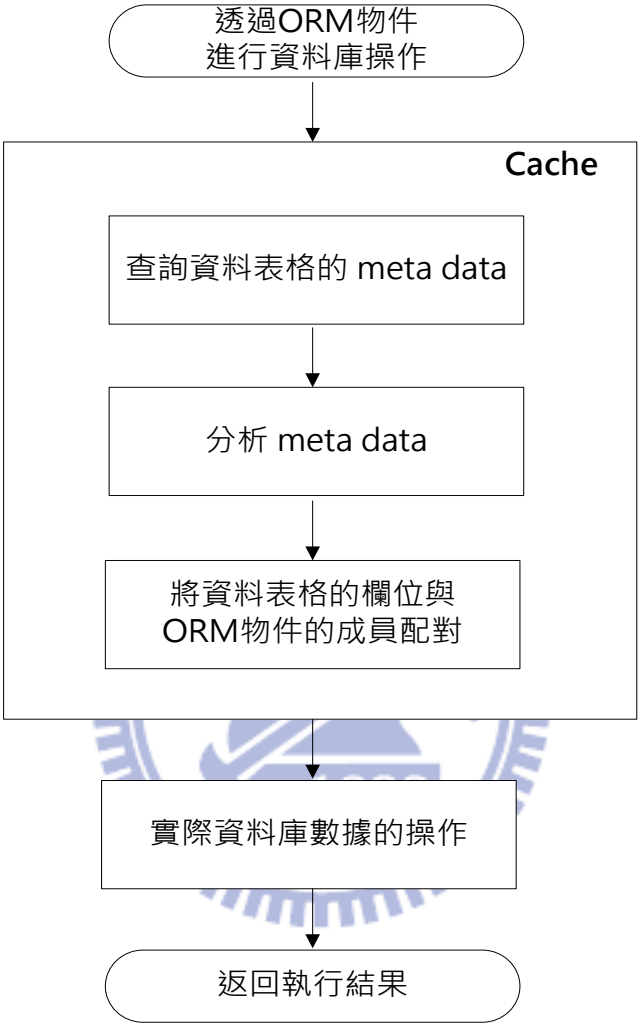

圖 13 Active Record 模式的運作流程

使用 Active Record 模式在分析和處理資料表格的 meta data 並對應成 ORM 物件的階段，明顯比傳統使用 SQL 來操作資料庫的方式多出很大的開銷。因此，本論文基於資料庫表格的 schema 是不常變動的前提下，將此階段的結果快取起來，以降低開銷，提昇效能。

3.3.4 SNS API 的設計

SNS API 設計目的是提高執行社交網路服務 API 的速度和改善其易用性。在性能部分，社交網路服務應用程式的 API call 通常需要進行大量的網路通訊動作，這使得其 API call 是個相對昂貴的操作。因此，為了解決頻繁的網路呼叫所造成的性能損失，本框架透過批次處理與快取的機制，用以降低社交網路服務 API 執行網路呼叫的頻率，提高執行的速度。

改善 API 易用性的部份，本框架包覆了原有的社交網路服務 API，用以簡化原生 API 的複雜性。以 Facebook API 為例，當使用發送訊息(feed)API 時，開發者要配合 JavaScript 來定製一個訊息骨架才可以正確使用。這造成開發者調用此 API 時，必須撰寫兩種程式語言，大大增加使用 API 難度。而本框架的 SNS API 無需使用額外的程式語言，降低 API 的使用難度，進而提昇開發效率。

3.3.5 快取機制的設計

本論文在快取機制的設計參考目前市面上流行且成熟的產品 Memcached 的機制，採用 Hash Table 作為快取的儲存結構。Hash Table 的優點在於查詢速度很快，其平均查詢時間為 O(1)，因此非常適合作為快取機制使用。而雜湊函式是決定 Hash Table 效能成敗的重要考量，因此根據[15]的比較，採用以字串為 key 時，所產生 value 的分散程度與生成速度為考量時，DJBX33A 演算法有很好的表現。因此，採用此演算法作為本論文快取機制的雜湊函式。

在碰撞處理上本論文採用鏈結法(chaining)因應，也就是當發生不同的輸

入字串卻產生相同雜湊值的情形時，將新加入的值使用額外的節點存放，用以解決碰撞問題。此外為了增加快取機制的健壯性，本論文採用 LRU(Least Recently Used)演算法作為快取置換策略(cache replacement policy)。此演算法是基於時間局部性(temporal locality)原則，也就是當一筆資料被使用後，短時間內被使用的機率極高。因此基於此原則，當快取要進行置換時，移除最近最少使用的內容可有效的增加快取命中的機率。表 4 為本論文在快取機制設計所使用的技術一覽。

表 4 快取機制設計的整理表

考量項目	實作的演算法
資料結構	Hash Table
雜湊函式	DJBX33A
碰撞處理	Chaining
快取置換策略	LRU

在快取資料是否需要同步的議題上，本論文傾向「不強制要求所有的快取資料皆需進行即時同步」。因為快取資料的同步，需要將可持久化儲存的來源資料(在網路應用程式中，通常為資料庫數據)與快取資料保持一致。此舉就會涉及到執行相對於緩慢的檔案 I/O 操作，影響快取的效能。

因此，本框架提供了兩種快取資料同步的方式：定時同步快取和即時同步快取。定時同步快取機制適用於非即時性的資料，只有在超過使用者定義的時間以後(時間單位為分鐘)，才進行資料同步動作。不然無論來源資料是否發生變動都會直接傳回快取資料；而即時快取同步機制則適用於即時性的

資料，當來源資料發生變動時，就會立即進行資料同步動作，使快取資料永遠是最新的資料。此兩者的詳細運作流程見圖 14。

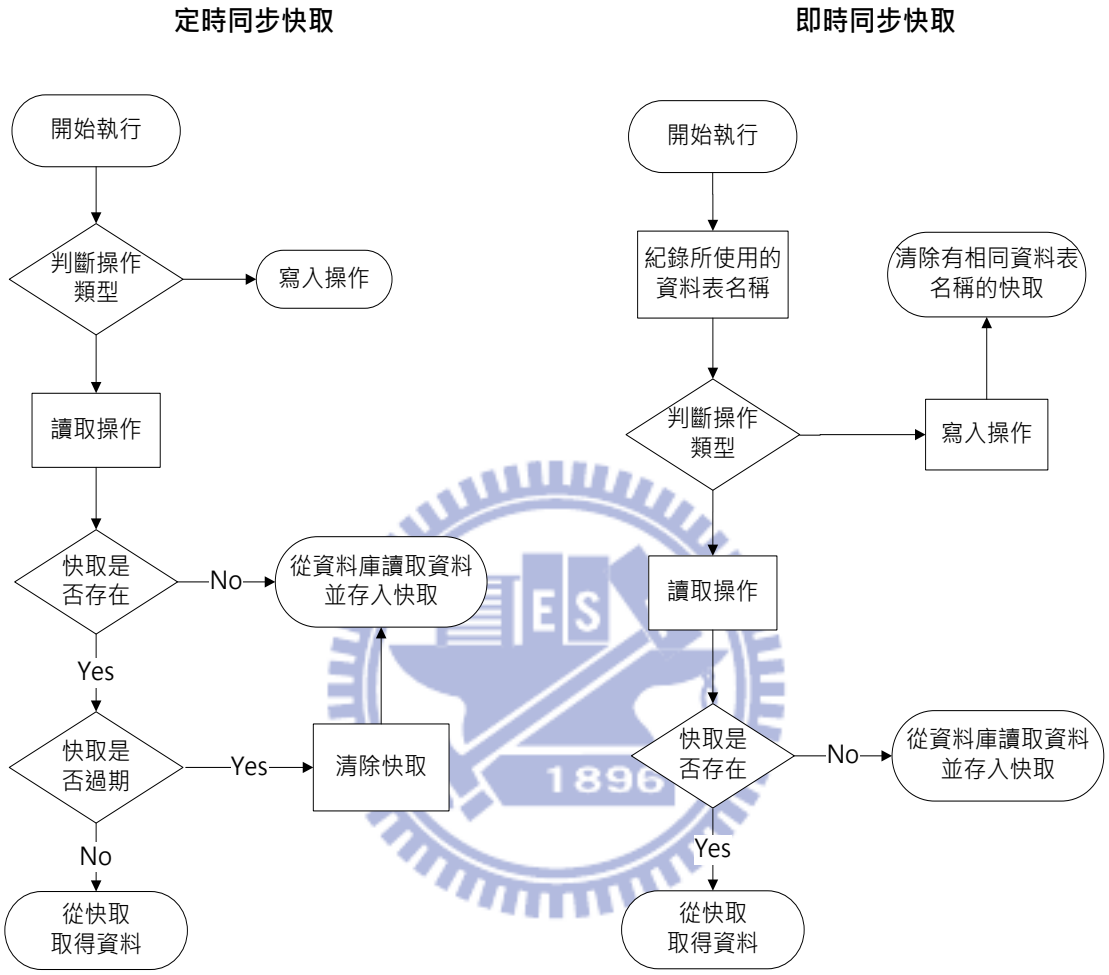

圖 14 快取機制運作流程

3.4 討論

本論文提出的框架其設計是極具彈性的，只要將社交平台的 API 添加至 SNS API 模組，即可完全適用於任何的社交網路服務應用程式。更甚者，本框架也可運用於 web-based 的應用程式，無需進行任何的修改。因此，框架

的適用性極佳。

但本框架存在一個限制，由於框架的 SNS API 模組是建構於社交網路平台的原生 API 之上。所以當社交網路平台的 API 改版或更新時，就可能會影響到 SNS API 模組的效用，嚴重點可能會造成 SNS API 無法使用。因此當社交網路平台 API 發生變動時，SNS API 模組也要隨之進行修改，以維持其正常的運作。

第四章、 系統實作與模擬結果分析

在此章節會實作本論文提出的框架，並對框架的正確性進行驗證。接著再針對 Facebook 平台的應用程式進行實驗。以不同規模的 Facebook 應用程式來衡量本框架的開發效率和系統性能。

4.1 系統實作

本框架採用 PHP 程式語言開發，依據第 3 章 3.3 節框架的設計進行實作，實作的模組如圖 15 所示。其中控制模組(Controller)、呈現模組(View)、事務模組(Model)、SNS API 模組其功能在第 3 章已有詳細的說明。Helper & Library 模組目前支援：電子郵件、日誌、分頁、資料驗證、檔案上傳、圖形驗證碼等功能，用以幫助開發者能更輕鬆的開發社交網路應用程式。

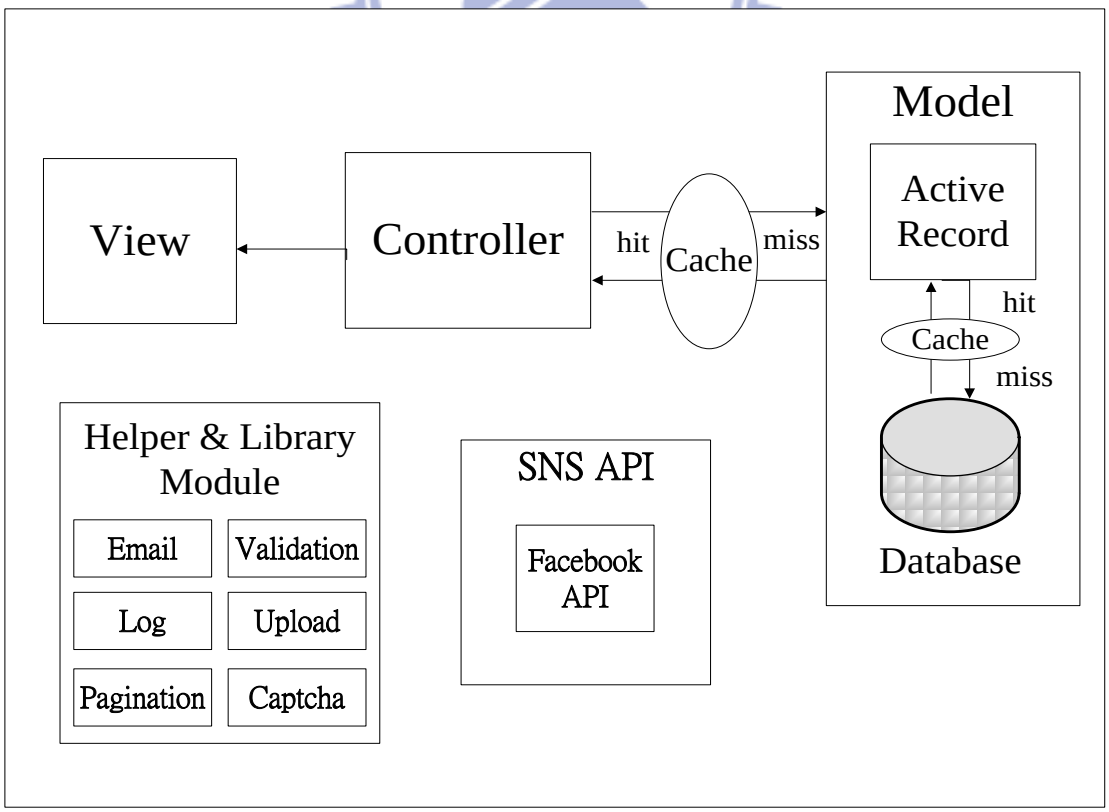

圖 15 框架實作模組一覽

4.1.1 系統流程

系統流程如圖 16 所示，首先系統從控制模組接受到使用者從瀏覽器發出的請求後，依據使用者的請求將其解析成對應的執行動作(Action)。而此執行動作是程式開發者自行定義的函式，開發者需在此處決定對應的事務模組與呈現模組為何，並可依據執行動作的即時性來決定是否需要採用快取機制，以增進程式執行速度。若是需要快取，則系統會自動將此執行動作的結果快取起來，開發者無需手動調用快取模組。

當控制模組在執行使用者自定義的動作或函式時，會取得對應事務模組的內容將其渲染至呈現模組。最後呈現模組緩衝所有需要渲染的內容至瀏覽器輸出，完整呈現使用者的請求結果。

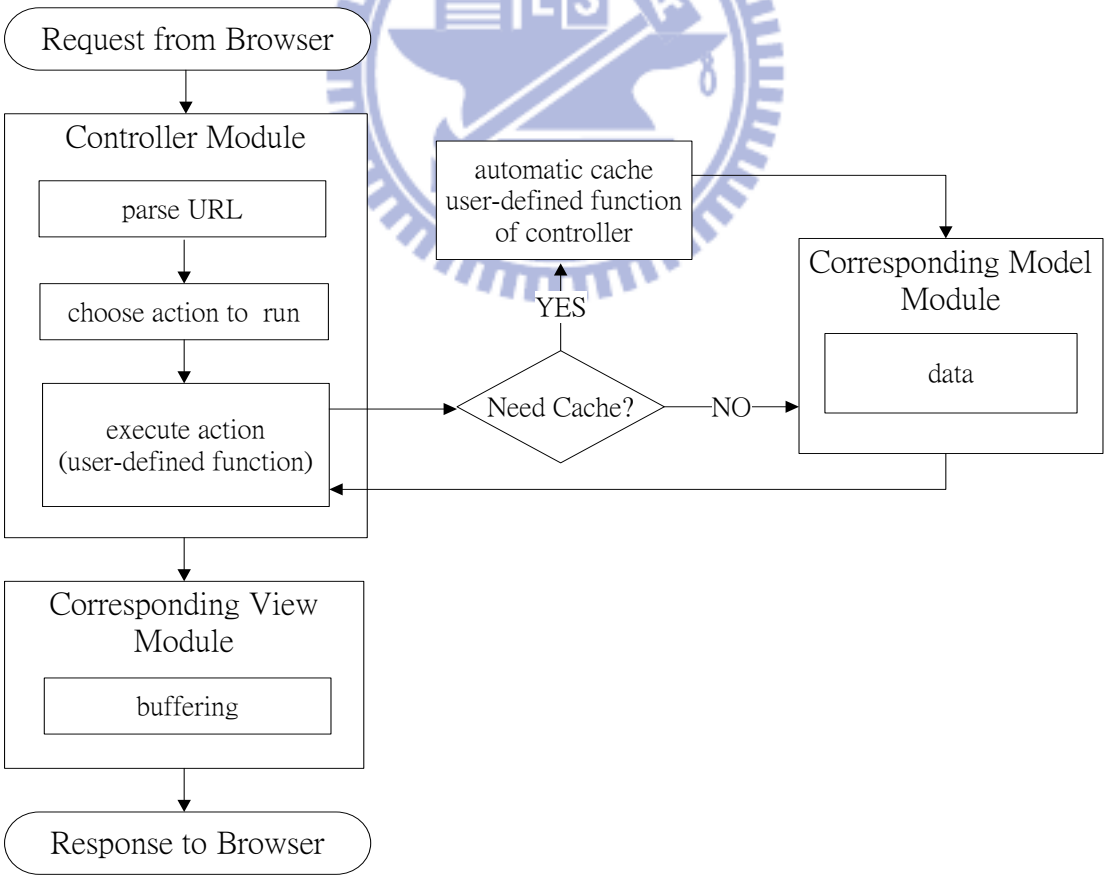

圖 16 系統流程圖

4.2 模擬結果分析

在此小節，會先對本論文所提出的開發框架進行正確性驗證，確保本框架是可信賴的。再來為了衡量本框架在實際環境中，能提昇多少社交網路應用程式的開發效率與系統性能，本論文採用 2 個實際運行中的 Facebook 應用程式以本框架進行改寫。將使用本框架的程式與原始程式進行比較分析，用以驗證本框架的效用。

4.2.1 實驗環境

Facebook 應用程式需要開發者自行建構伺服器環境與 Facebook 平台通訊，因此本實驗所需的程式皆建置於表 5 的環境中。

表 5 實驗平台環境

硬體環境	Intel Core 2 Duo CPU E8200 @ 2.66Ghz 2.67Ghz 1.87GB RAM
軟體環境	Microsoft Windows XP sp3 Apache 2.0.63 Mysql 5.0.51b Memcached 1.2.6

4.2.2 實驗評估方式

實驗評估的項目為使用本框架後能提昇多少開發效率與系統性能。衡量開發效率的指標為：程式碼行數(Lines of Code)。衡量系統性能的指標為：

回應時間(Response Time)。各項指標的定義如下。

- 程式碼行數：開發者編寫 Facebook 應用程式的程式碼總行數。此數值越小越好。
- 回應時間:使用者從瀏覽器造訪 Facebook 應用程式直至接收所有的 response 封包所需的時間。此數值越小越好。

在之後的實驗，回應時間皆是採用 1000 個不同連線的平均值。

4.2.3 實驗一 －框架正確性驗證

本框架除了 SNS API 模組外，其餘模組皆使用 PHPUnit 工具進行單元測試，確保各個模組皆正確無誤。而 SNS API 模組是完全建構於社交網路平台原生 API 上的一個包覆類別。如圖 17 所示，當使用者使用 SNA API 模組進行呼叫時，實際執行 API Call 的對象是社交網路平台原生 API。因此 SNS API 模組能保證在原生 API 無誤的前提下，正常運作。

基於上述的理由，可以保證建構於本框架上面的程式能夠正確無誤的運行。也就是說當程式發生非預期的錯誤時，此錯誤應當屬於開發者的程式邏輯錯誤，非本框架之責任。

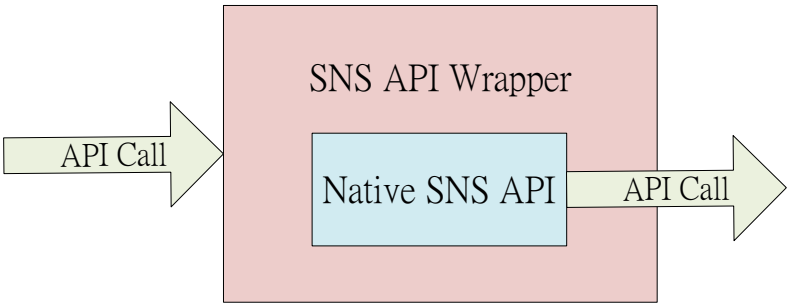

圖 17 SNS API 模組的組成

4.2.4 實驗二 － 實際案例：心理測驗

實驗一的程式是一個簡單的心理測驗，藉由判斷使用者回答的選項，給出相對應的測驗結果，見圖 18、19。而第一項實驗就是利用此簡單的程式，來驗證本框架在小型社交網路應用程式中的成效，實驗結果如表 6 所示。

其中，表 6 的回應時間是取問題頁面(圖 16)和測驗結果頁面(圖 17)的算數平均數計算而得。

圖 18 小型的 Facebook 應用程式 畫面 1

圖 19 小型的 Facebook 應用程式 畫面 2

表 6 實驗二結果

	原始程式	使用本框架後的程式
程式碼行數	158 行	146 行
回應時間	2.425 秒	2.175 秒

從實驗一的結果可看出，在小型的社交網路應用程式中使用本框架，改善的程度有限。程式碼行數從原先的 158 行縮減到 146 行，縮減程度有限。而回應時間從 2.425 秒縮短到 2.175 秒，性能雖提高了一成，但對使用者來說，並無顯著的差異。

4.2.5 實驗三 — 實際案例：GameBox

實驗二的程式是一個已經在 Facebook 上經營的小遊戲分享平台 — GameBox，見圖 20、21。GameBox 是一個可以讓 Facebook 使用者推薦喜歡的小遊戲給朋友，並對此遊戲進行討論的社交網路應用程式。第二項實驗則使用這個相對複雜的 Facebook 應用程式，來驗證本框架在大型社交網路應用程式中的成效，實驗結果如表 7 所示。

此外根據 Google Analytics 分析流量後發現，首頁是 GameBox 性能瓶頸的所在。故實驗二只有計算首頁(圖 18)的回應時間。

圖 20 大型的 Facebook 應用程式 畫面 1

圖 21 大型的 Facebook 應用程式 畫面 2

表 7 實驗三結果

	原始程式	使用本框架後的程式
程式碼行數	2621 行	1820 行
回應時間	12.701 秒	5.513 秒

從實驗二的結果可看出，在業務邏輯複雜的程式中使用本框架，表現遠比小型的社交網路應用程式出色。其程式碼行數從原先的 2621 行縮減到 1820 行，縮減幅度有 31%。而回應時間更從 12.701 秒縮短到 5.513 秒，性能提高了 130%。

第五章、 結論與未來展望

5.1 結論

現今社交平台的發展如火如荼，各個平台的成長率皆十分驚人。然而在建置社交網路服務應用程式的過程中，對於開發效率與系統性能卻難以兼顧，讓開發者甚為困擾。因此，本論文提出一個基於 MVC 架構的社交網路服務開發框架來解決此問題。

實驗結果證明，本框架對於社交網路服務應用程式的開發效率與系統性能皆有明顯的提昇，並且在大型且業務邏輯複雜的程式中，表現更為亮眼。

此外透過使用本框架來重構 GameBox 平台後發現：將程式邏輯與顯示畫面分離，能有效地改善程式結構混亂的問題。程式可讀性也大幅的提高，利於後續維護與管理。

5.2 未來展望

未來研究的方向整理如以下幾點：

(1) 增加其他的 SNS API 實作

目前本框架在 SNS API 模組方面，只有完成 Facebook API 的實作。為來可以再針對 MySpace、Orkut、Twitter 等熱門社交平台的 API 進行實作，驗證本框架在不同社交平台的成效。

(2) Open Source

本框架未來欲發佈成 Open Source Project，與全球廣大的程式開發者們一同驗證與改善本框架，使之成為一個成熟的社交網路服務應用程式開發框架。

參考文獻

[1] Cecile Demailly and Martin Silman, "The Business Impacts of Social Networking", White paper by AT&T in cooperation, Sep. 2008

[2] Daniel J. Bernstein, usenet group comp. lang. c

[3] DDudley JT, Butte AJ, and Lewitter F, "A Quick Guide for Developing Effective Bioinformatics Programming Skills.", PLoS Computational Biology, Vol. 5, Issue 12, 2009

[4] Deepak Alur, John Crupi, and Dan Malks, "Core J2EE Patterns: Best Practices and Design Strategies", 2nd edition, Pearson, 2003

[5] Diana M. Selfa, Maya Carrillo, Ma. del Rocio Boone, M., "A Database and Web Application Based on MVC Architecture", Electronics, Communications and Computers, pp. 48, 2006

[6] Diomidis Spinellis, "Java Makes Scripting Languages Irrelevant?," IEEE Software, vol. 22, no. 3, pp. 70-71, May/June 2005

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1997

[8] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", Journal of Object-Oriented Programming, pp. 26-49, Auguest/September 1988

[9] Greg Goth, "Are Social Networking Sites Growing Up?," IEEE Distributed Systems Online, vol. 9, no. 2, pp. 3, Feb. 2008

[10] Jacyntho, M., Schwabe, D., and Rossi, G., "A Software Architecture for Structuring Complex Web Applications", Journal of Web Engineering. Vol. 1, No. 1, pp. 37-60, 2002

[11] Judith Bishop and Riaan Hurter, "Competitors to Java: Scripting languages", Computer Science Department, University of Pretoria, Pretoria 0002. South Africa, 1999

[12] Kevin Marshall, Chad Pytel, and Jon Yurek, "Pro Active Record: Databases with Ruby and Rails", 1st edition, Apress, 2007

[13] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford, "Patterns of enterprise application architecture", 1st edition, Addison Wesley, 2002

[14] Michael Mattsson and Jan Bosch, "Framework Composition: Problems, Causes and Solutions", Conference on Technology of Object-Oriented Languages and Systems, pp. 203-214, 1998

[15] S. M. J. Rizvi, M. Hussain, and N. Qaiser, "Comparison of hash table verses lexical transducer based implementations of urdu lexicon", in Proc. of the Engineering, sciences and technology, student conference, 2004

[16] Developer Resources for Java Technology, available: http://java.sun.com

[17] Facebook, available: http://www.facebook.com

[18] Hibernate, available: http://www.hibernate.org

[19] JavaRanch Journal, available: http://www.javaranch.com/journal/200603/Journal200603.jsp

[20] Memchaced, available: http://memcached.org