

雲端服務與 Android 軟硬整合之路

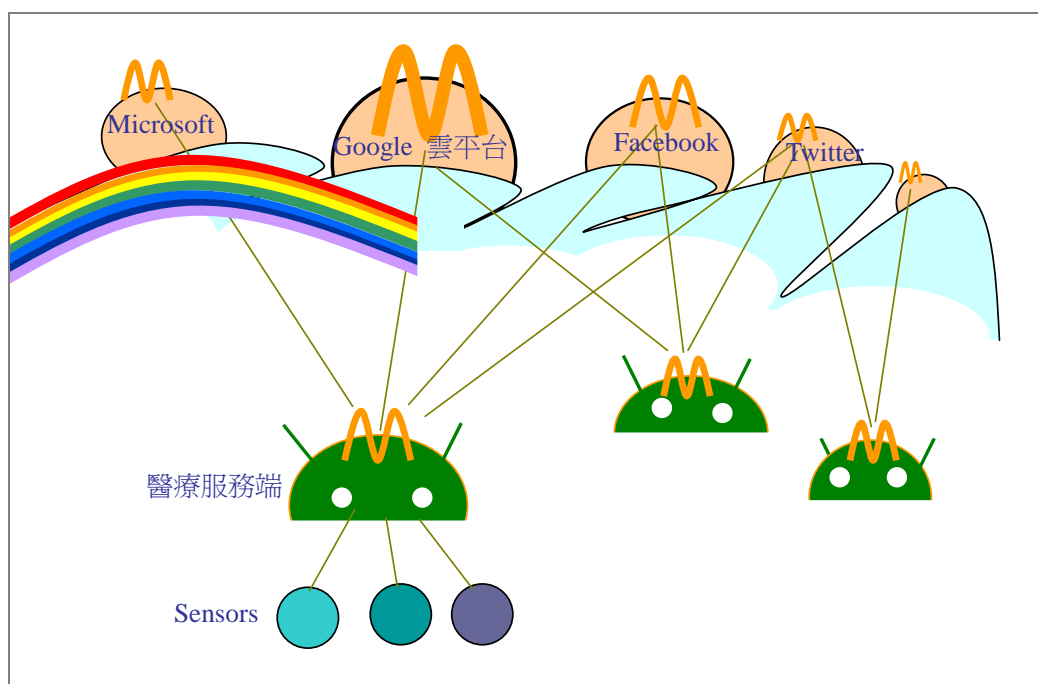
走進巴黎鐵塔看不見巴黎鐵塔
走進雲裡看不見彩虹
從軟硬整合看見雲層上的彩虹
請您來分享彩虹之美

0. Android 整合之路

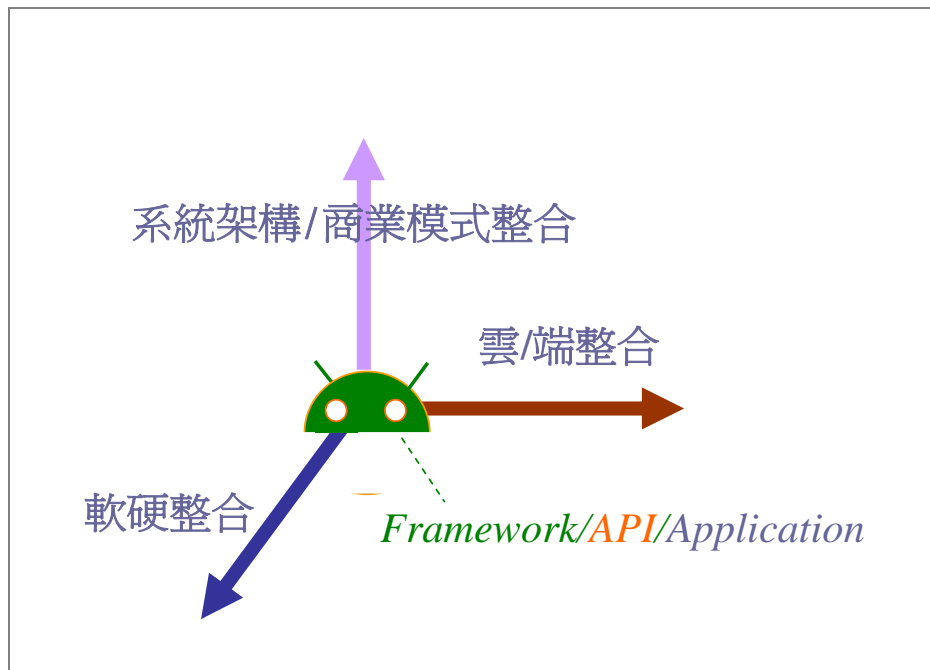
在眾雲成群下，Android 成為移動端的一枝獨秀，也是進軍雲產業的主要渠道。

云想衣裳花想容
Android 端里映彩虹

By 李白 & 高煥堂

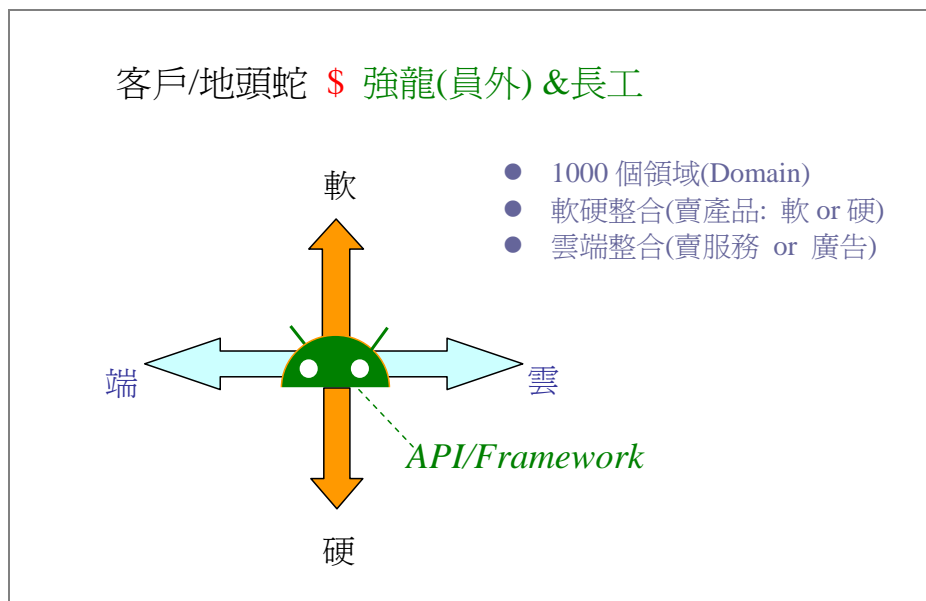


從移動端看眾雲成群



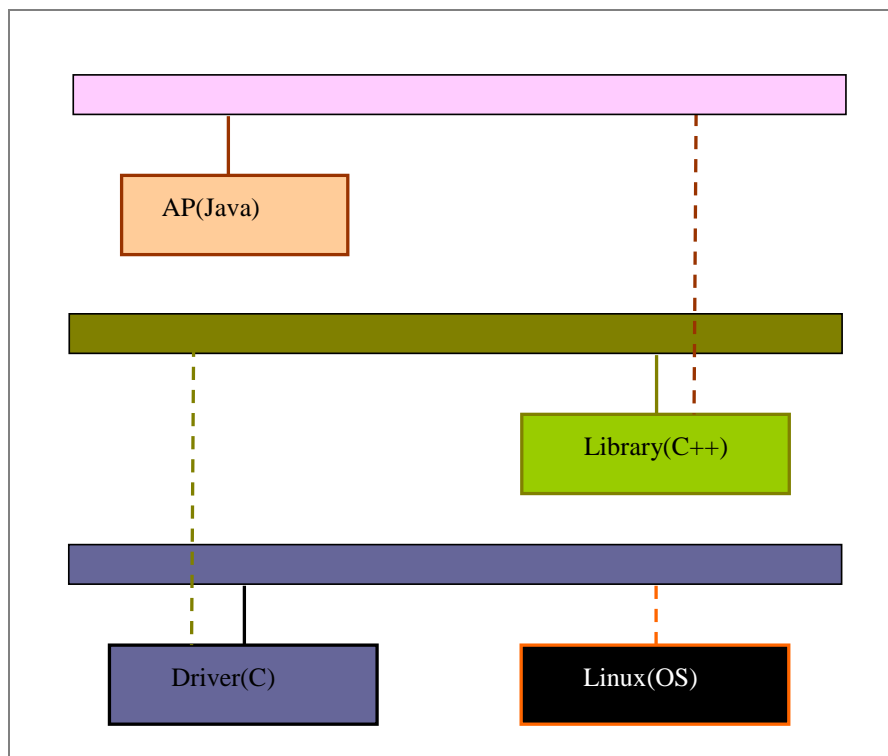
Android 的整合之路的三個面向

在眾強龍之下，”小強龍”有無限廣大的天空。



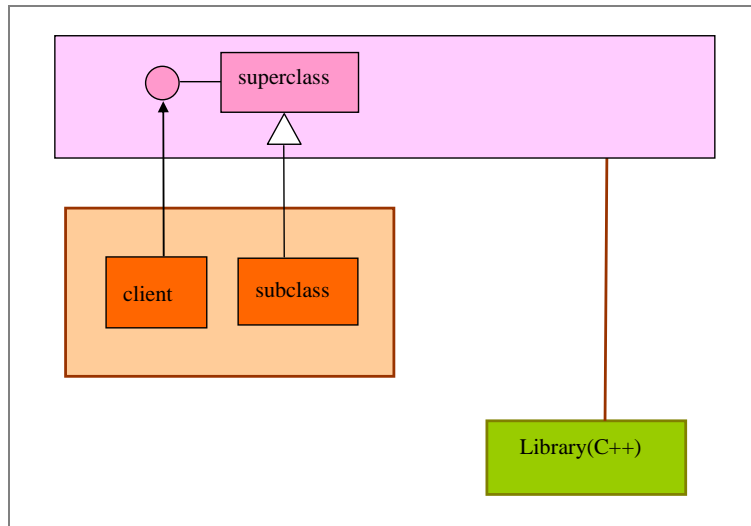
人人有成爲”小強龍”美好機會

在眾星球上建立星際連鎖商店，各商店的框架是一致的。



一致的架構

一致的架構來自簡單元素的無限組合。



簡單的基本元素

1. 雲與端整合的關鍵：API

1.1 雲端服務發展新焦點：

API(Application Programming Interface)

海峽兩岸 IT 硬件產業是舉世聞名的，基於這項產業優勢，進而開發各層級(Layer)、各領域(Domain)的軟件介面是具有極大商機的。當然，在談及海峽兩岸產業優勢之際，也必須談談其產業的劣勢，就是平台管理軟件的能力不足。我們擅長研發硬件伺服器(Server Hardware)，但並不擅長開發大型或分散式伺服器的平台管理軟件系統。因此，值此雲服務產業方興未艾之際，思考海峽兩岸未來雲與端產業的發展之路，是有其建設性的。

鑒於過去 30 年來，海峽兩岸 IT 產業長期受制於外人所掌控的封閉型平台管理軟件(如操作系統、數據庫管理系統等)，因而有被外人掐著脖子走的苦楚。這樣的情境似乎仍然會延續到新興的雲產業裡。因之，此刻來探討如何走出劣勢、邁向優勢，是有其價值的。那麼，如何克服上述的劣勢，邁向贏家之路呢？答案之一是：開發各種應用領域的 API(Application Programming Interface)。由於雲朵紛飛，在平台管理軟件層級的介面(Platform-layer Interface，簡稱 PI)將呈現百花齊放、千鳥爭鳴的現象。這必然讓全球數百萬應用軟件開發者，更加依賴於上層的應用領域層級的介面(Application Domain-layer Interface，簡稱 API)。例如，

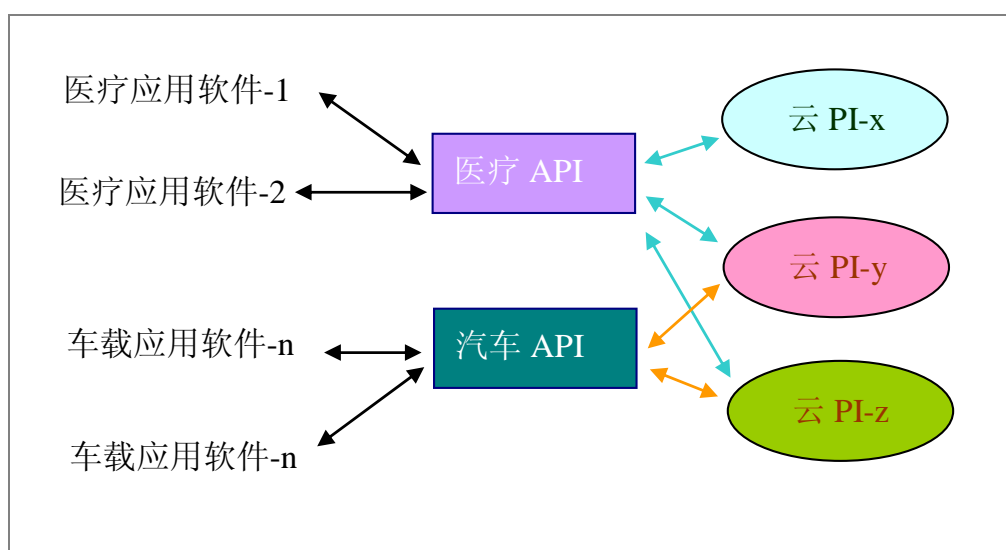


圖 1-1、API 在雲產業裡的角色

在應用軟件市集(如 Google AppStore 或 Apple AppStore)盛行的今天，誰能更號召全球數百萬應用軟件開發者(簡稱 AP 開發者)，將是雲產業中的強龍。由此而觀之，兩岸 IT 產業如果在 API 上先馳得點，加上大陸龐大市場的優勢，即使未能掌控雲層平台的 PI，也一樣能成為未來雲產業的強龍盟主。

1.2 雲端服務 API 開發之基本概念

API 的角色

在上一小節裡，提到海峽兩岸在雲與端產業裡的競爭優勢，以及能四兩撥千金的重要槓桿點：API。很多讀者問道：有沒有 API 開發的 How-to 呢？答案是：當然有。首先來看看 API 的角色，如下圖：

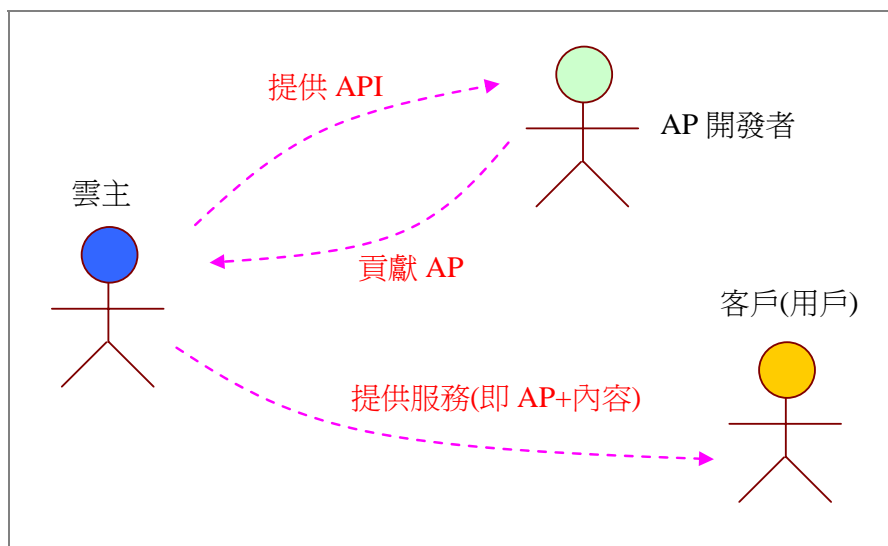


圖 1-2、雲 API 的角色

雲主就是雲服務的盟主(強龍)，全球第三方應用程式開發者(AP 開發者)是自願來協助雲主的地頭蛇。API 就成為雲主與眾多地頭蛇之間的分工介面，也是雙方系統的介面。例如 Facebook、Google 等都主動提供 API 給 AP 開發者，並且提供內容(Content)來源，則 AP 開發者就能據之而開發形形色色的應用程式，回饋給雲主。

隨著 API 愈來愈完整，愈多地頭蛇投靠該雲主，自然而然雲主擁有了更多 AP 和內容，於是將 AP 和內容整合為服務(Service)，提供給數以億計的用戶了。

- 誰掌握 API，誰就是老大
- 誰打造自己的尚方寶劍(即應用框架)，誰就是強龍
- 成功的雲主(提供雲服務者)是英雄
- 雲服務的客戶(即用戶)介面是 UI
- 雲主英雄莫不需要極多樣化的 UI
- 莊子有言：愈想射箭中靶心，愈要專注於箭身
- 大家知道：英雄愈想生小孩，愈要多疼愛美人
- 雲主知道：愈想要極多樣化 UI，愈需要屬於自己的 API
- 掌握 API 等於擁有控制權
- 手握尚方寶劍等於掌握 API
- 誰掌握 API，誰就是老大

雲服務 API 的分類

API(Application Interface)的型式有許多種，但可以分為兩類：「被動型 API」和「主動型 API」。被動型 API 通常是以程式庫(Software Library)形式呈現。因為它提供一群函數(Function)來讓應用程序調用之。它被應用程式調用，所以稱為被動型 API。

主動型 API 通常是以框架(Framework)形式呈現。因為它提供一群父類(Super class)來讓應用程序的子類(Sub class)繼承之。通常，它都主動調用應用程式，所以稱為主動型 API。

Framework-based 的 API

目前，無論是在雲層服務方面，或在行動端應用方面，大多採用 Framework-based 的主動型 API。例如，在雲層的 Web Server 上有如 Spring 等各式各樣的平台框架，以及其它應用領域框架。同時，在行動端應用方面，Android 也採用 Framework-based 的主動型 API。例如，Android 硬件抽象層(Hardware Abstraction Layer，簡稱 HAL)就是一個框架，如下圖：

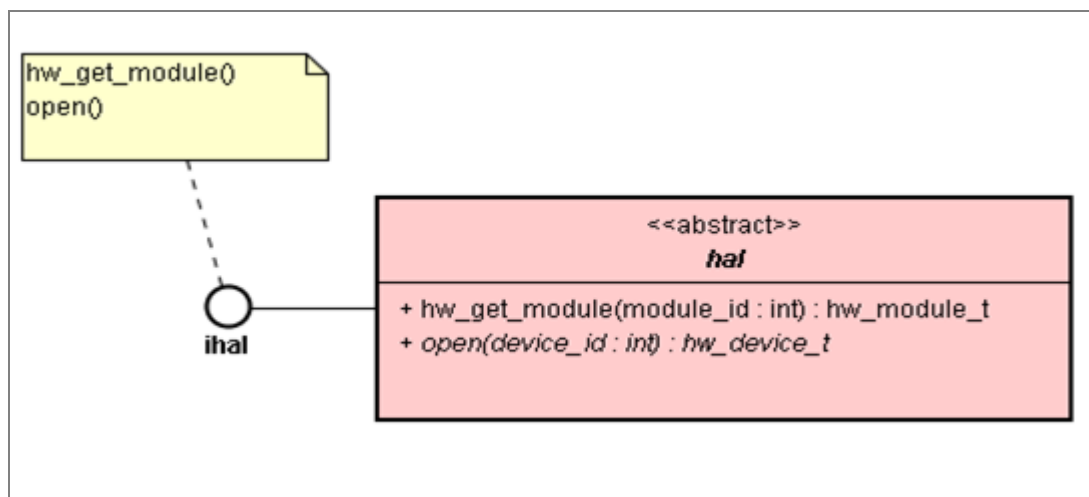


圖 1-3、Android 底層框架之例

表面上看來框架只有一個介面(`ihal`)，其實有兩個介面。如果將一個框架比喻為一張桌子，就很容易理解了，如下圖：

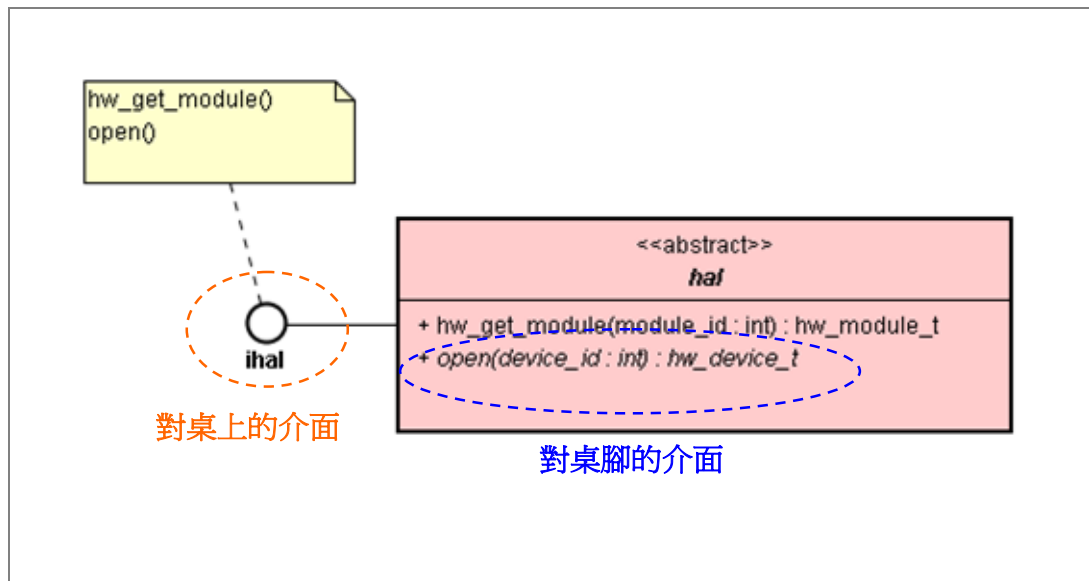


圖 1-4、將框架比喻為桌子

框架就如同桌面。AP 開發者會撰寫應用子類，成為桌腳，然後以類繼承機制，將桌腳搭配上，就成為一張空的桌子了。接著，AP 開發者，撰寫 UI 部分(又稱為 Client 部分)，成為桌上的東西(如水果等)，然後以依般介面機制，將之擺到桌面上，就成為一張完整的桌子了。如下圖：

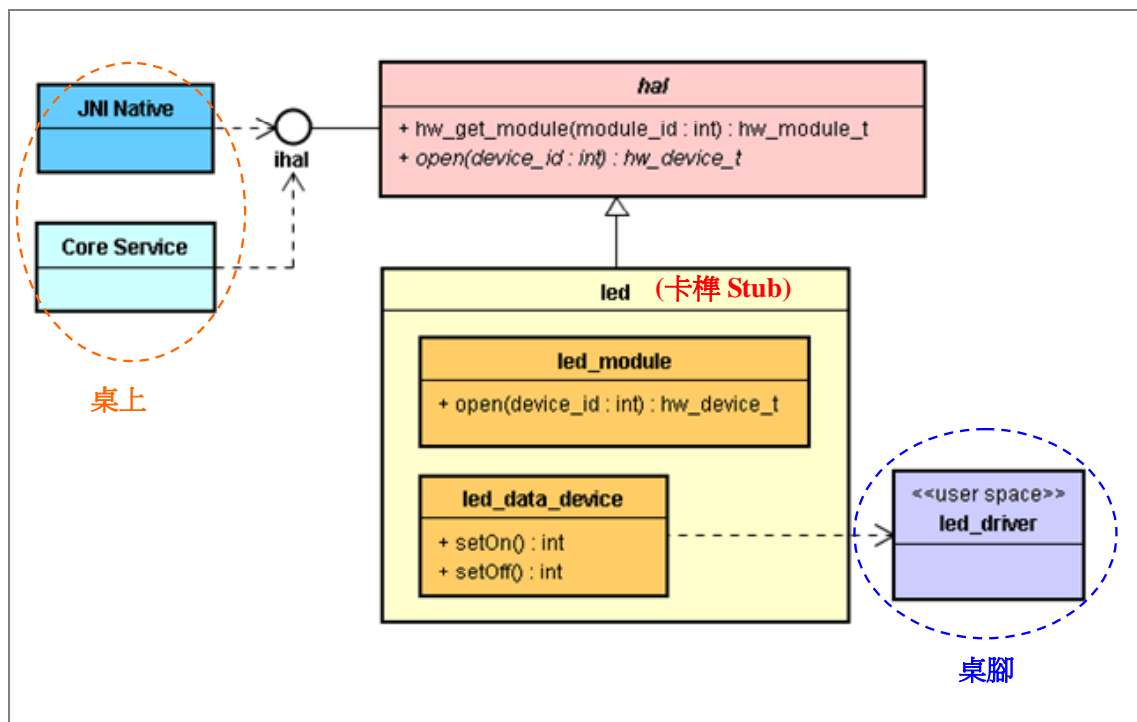


圖 1-5、完整的桌子(即可用的應用程式)

在 Android 行動端裡，除了上述的底層 HAL 框架之外，其上面的 Java 層

框架也是提供主動型的 API。

以上只是概念上說明雲服務 API 的幕後機制，尤其是目前流行的 Framework-based 的主動型 API。在後續的文章裡，將繼續說明詳細的 API 開發方法和技術。

1.3 雲計算：從 Twitter 的 API 談起

客(用)戶從兩個途徑去使用 Twitter 的雲服務：

- 使用瀏覽器(Browser)取得 Twitter 網頁的服務。
- 使用第 3 方應用開發者的應用程式(Application，簡稱 AP)去調用 Twitter 的 API，取得 Twitter 的服務。

從 2010 年 4 月份的統計分析，發現上述的途徑(1)佔 25%；而途徑(2)則佔 75%。可見提供體貼好用的 API，對雲主的重要性正日益增加中。還有，Twitter 每日接受外部應用程式超過 30 億項的 API 請求。這些數據顯示出，以 API 來號召天下八方豪傑(即全球各地的 AP 開發者)是雲主們欲稱霸武林的不二法門。

那麼，雲主的 API 又如何支援 AP 開發者呢？首先，看看 API 一般目的是：

- API 必須有效支援 AP 開發者去客制化出自己的雲層服務。
- API 必須有效支援 AP 開發者去開發出多樣化的行動端(如 Android 手機)的應用程式。

這了達成上述的目標，幾乎提供基於框架(Framework-based)型式的 API，包括雲層的領用框架(Domain Framework)，以及行動端的應用框架(Application Framework)。

於是，以 API 來號召天下八方豪傑(即 AP 開發者)是雲主們稱霸武林的不二法門。那麼，雲主們積極開發自己的框架，成為自己手中的尚方寶劍，將是決定勝敗的關鍵了。

框架的內涵決定於它的應用領域(Application Domain)。各個領域都可以開發出其專用的框架。也就說，各朵雲彩都有其特定領域，也有其專用框架。善於掌握框架，就能提供體貼的 API 來廣召各方英雄好漢來拔刀相助。在眾星拱月下，善於掌握框架的雲主就逐漸壯大、並迅速稱霸武林了。

1.4 為什麼流行框架型式(Framework-based)的 API 呢？

當今，無論是雲層或行動端，都流行 Framework-based 的 API。大家都知道，掌握介面標準(如 JDBC, SQL, W3C 等)就是強權，而強權之間必然爭奪介面標準(即 Interface or Protocol)的掌控權。例如，當年微軟和寶藍(Borland)爭奪 ODBC API 的掌控權，失去 API 的掌控權的一方，在競爭上，常常處於非常不利形勢。

所以，API 是強權爭奪的標的。那麼，什麼型式的 API 可以強化自己的競爭優勢呢？這要先釐清什麼是競爭優勢呢？

由於 Apple 公司的 iPhone 率先提供 API，開放給全球的第三方 AP 開發者使用，並設立 iPhone 的全球性第三方 AP 市集(Application Store)。因而匯集了數十萬 AP 開發者的加盟，讓 Apple 處於極佳的競爭優勢。Apple 是強龍，而追隨 Apple 的眾多第三方 AP 開發者是地頭蛇，因而形成極具獲利能力的「強龍/地頭蛇」商業樣式。

因之，當今競爭優勢的評估點，就在於地頭蛇數量的多少。獲得愈多地頭蛇的擁戴，就愈是強權。跟隨的地頭蛇數量少，就非強權(龍)。爲了鞏固強龍地位，強權競相提供更好的 API 來吸引全球的地頭蛇(第三方 AP 開發者)。其作爲包括：

- 增加 API 的服務內容，協助地頭蛇，讓地頭蛇降低 AP 開發成本。
- 提供更好的 API 的使用環境(如 AP 開發、測試工具等)，讓地頭蛇的開發工作更加流暢。
- 提供 AP 市集，擴大 AP 的行銷管道。
- 提供 AP 買賣的收費機制(包括廣告營收)，協助地頭蛇快速獲利。
- 等等。

以上所述都是強龍送給地頭蛇的貼心禮物，換取廣大地頭蛇的擁戴，以鞏固強龍的地盤。這些是成爲強龍的必備作爲，也就是必備條件。但是，並非充分條件。那麼，充分條件是什麼呢？答案很簡單，就是：掌控地頭蛇，讓他們不能光拿禮物，然後掉頭走人。所以，強龍的「充分」作爲是：透過 API 幕後的軟件機制來主導(即掌控)地頭蛇 AP 軟件架構，讓數十萬支 AP 軟件的架構都依賴於 API 幕後的機制，則廣大地頭蛇群，一方面能獲利，一方面又受約制，於是成爲穩固金湯的「強龍/地頭蛇」產業聯合架構。

這個神祕的幕後機制就是『應用框架(Application Framework)』。爲何應用框架具有這項神秘的力量呢？就請繼續看下回的解釋了。

1.5 雲計算：主動的框架型 API

在上一小節裡，簡介了目前 IT 產業爲何(Why)流行框架型 API。本文將進一步說明框架型式 API 的特色，以及其幕後框架對整體系統控制權的巨大影響。

被動型 API

平台開發者本應是強龍，雖然強龍制定 API，然而卻是被地頭蛇(即 AP 開發者)所調用，強龍成爲被調用者。於是，強龍失去控制權，導致龍困淺灘。

雖然大家都知道，掌控 API(介面或協定)就是主導者，就能主導金錢的流向，所以會是較大的獲利者。因此，API 的制定者不一定是強龍。然而，「API 制定者而且是調用者」就是強龍。至於，地頭蛇則是 API 的服從者、實踐者、被調用者。例如，在傳統的 Linux 環境裡，其 API 的主導形勢如下圖 1 所示：

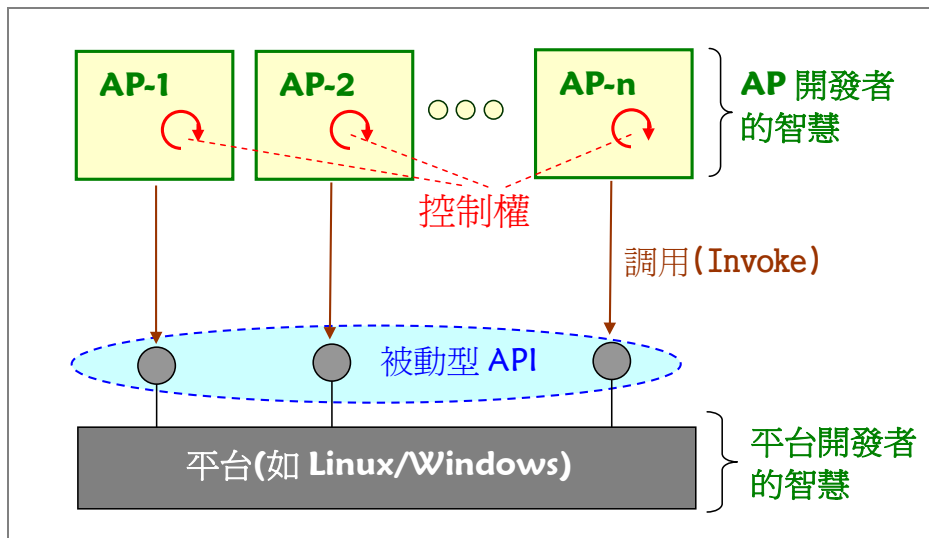


圖 1-6、被動型 API

- 平台提供者(即平台開發者)制定 API。
- 應用開發者主動調用 API。
- 導致平台開發者龍困淺灘(或虎落平陽被犬欺)，失去主導權。

由於控制權掌握在 AP 開發者手中，所以在軟件架構上，平台開發者反而受制於 AP 開發者，這常常嚴重傷害平台軟件變動的自由度，進而導致整體系統失去彈性和活力，陷入困境。因而，最近 20 年來，IT 產業逐漸放棄上述的被動型 API，改為新式的主動型 API，也就是框架型(Framework-based)的 API。例如，自從 1990 年代微軟的 COM/DCOM、2001 推出的微軟.NET Framework、2007 推出的 Google Android 等，皆是屬於框架型的 API。

主動的框架型 API

框架的角色

如果平台像地板，那麼框架就像天花板，而 AP 就像吊燈。如下圖 2 所示：

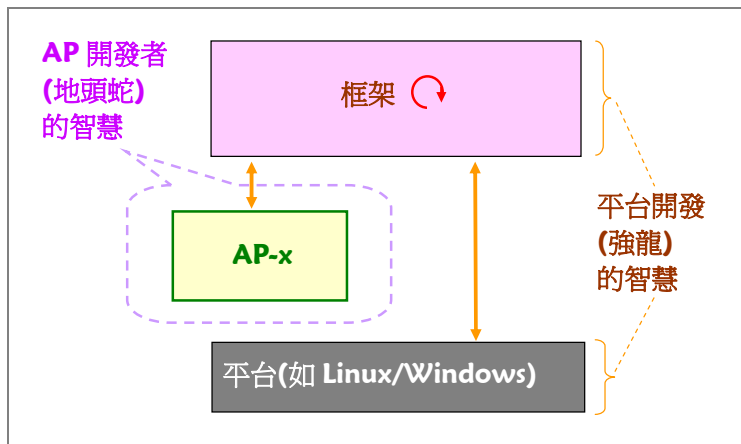


圖 1-7、框架像天花板

從日常生活的經驗裡，可知道由牆壁(或地板或天花板)來控制吊燈的燈光是很合理的安排。

框架的尙方寶劍

平台開發者應該是強龍，它就應該像好萊塢的大明星一樣，調用別人，而不是被別人隨意調用。

Hollywood(好萊塢)大明星的名言：

“Don’t call me, I’ll call you back.”

在這種 API 幕後是由一個框架來支撐。其中，由框架裡的父類(Super Class)來主動調用 AP 裡的子類(Subclass)。此時，框架擁有軟件系統執行上的控制權，由框架來指揮 AP 的執行。而父類裡的抽象函數(Abstract Function)就自然成為平台與 AP 之間的 API 了。因此，強龍的特權是：

- API 的制定者(即框架裡父類設計者)。
- API 的主動調用者。

如下圖所示：

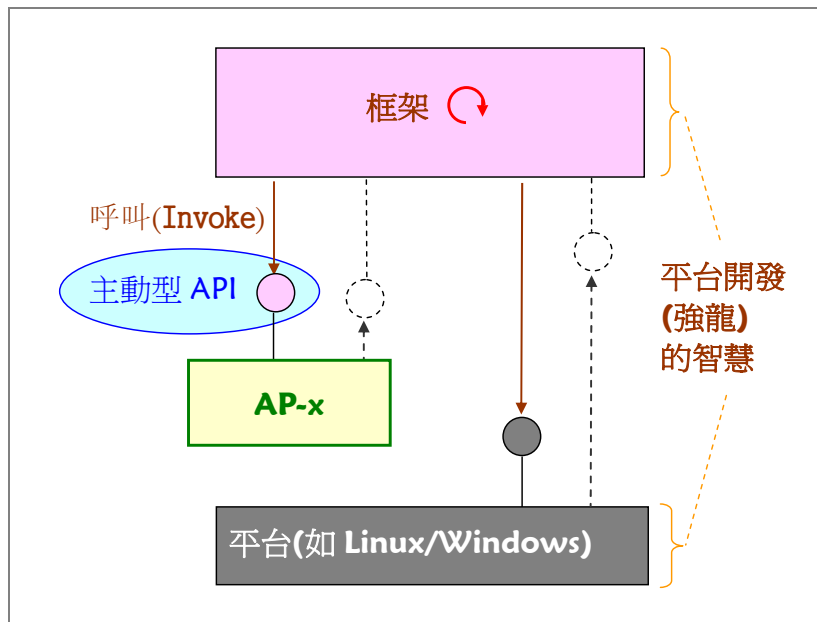


圖 1-8、主動的框架型 API

於是，平台框架開發者成為名符其實的強龍，掌握了軟件系統執行的主控權。框架就成為強龍手中的尚方寶劍了。

兩種 API 的微妙差異，影響主導權的互換

請細看上圖 1-8 裡的 API 介面(即灰色小圓圈)：

- 是誰制定的呢？答案是：平台開發者。(所以是灰色的)
- 是誰實踐(Implement)的呢？ 答案是：平台開發者。
- 是誰主動調用的呢？ 答案是：AP 軟件。

再請細看上圖 1-8 裡的 API 介面(即粉紅色小圓圈)：

- 是誰制定的呢？ 答案是：框架開發者(即平台開發者)。(所以是粉紅色的)
- 是誰實踐(Implement)的呢？ 答案是：AP 開發者。
- 是誰主動調用的呢？ 答案是：框架軟件。

至於，平台框架開發的基本途徑和技藝，就請繼續閱讀下一節囉。

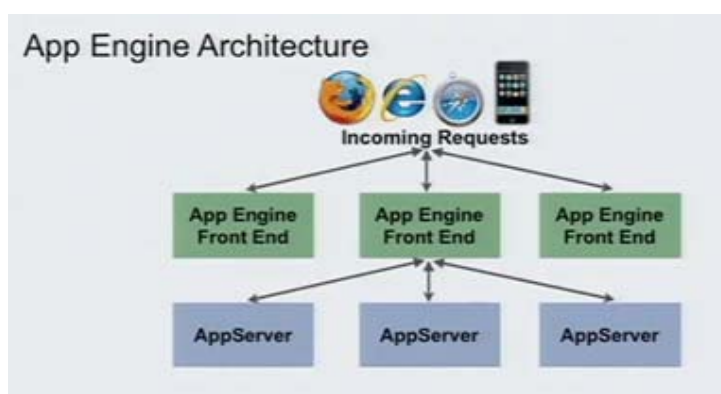
2. 雲層 API：以 Google AppEngine 為例

2.1 簡介 GAE(Google AppEngine)

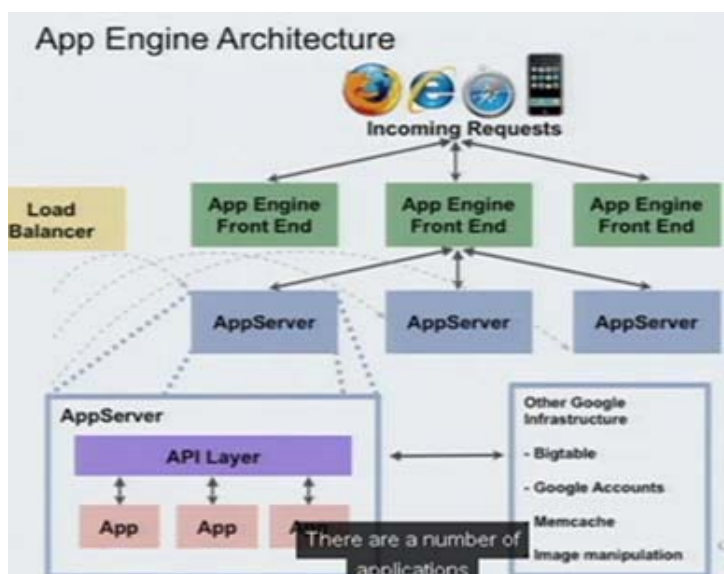
內涵：



架構：



架構(續)：



AppEngine 使用 Java 6 虛擬機器(JVM)執行 Java 網絡應用程式(Web Application)。其支持 Java Servlet 標準,可以在 WAR 目錄裡提供應用程式的 servlet 類、JSP、web.xml 等設定檔,讓 AppEngine 根據設定檔調用 servlet 來提供各項服務。

- AppEngine 的 Java SDK 提供多種工具,可讓您測試應用程式、上傳應用程式檔案,以及下載記錄資料。
- 其他電腦(包括端)只能透過標準連接埠的 HTTP (或 HTTPS) 要求,與應用程式連線。

2.2 到 GAE 上建置自己的雲層服務

AppEngine 是 Google 的雲服務引擎,讓地頭蛇能開發應用程式(AP),然後放在 Google 伺服器上執行,地頭蛇不需擔心頻寬、系統負載、安全維護等問題,一切由 Google 代管,地頭蛇可以專心地撰寫各式各樣的 AP。基於 AppEngine 的 AP 若每月超過 500 萬網頁面的流量就需要費用。若要開始使用,可到 <https://appengine.google.com/> 使用您的 Google 帳戶登入。



Google 帳戶

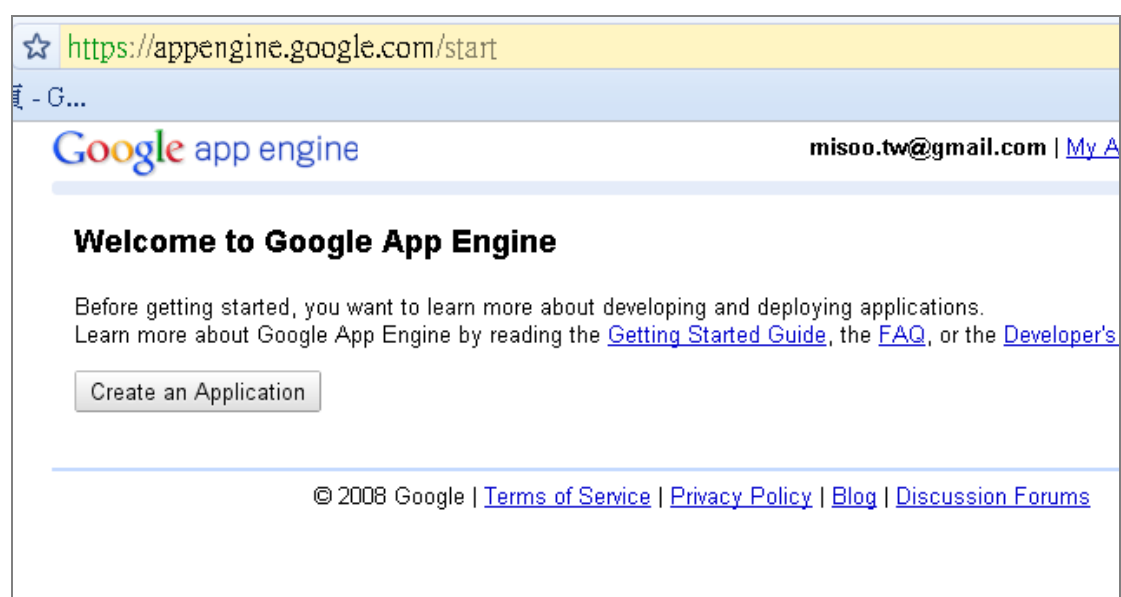
使用以下帳戶登入:

電子郵件: misoo.tw@gmail.com

密碼:

[無法使用您的帳戶?](#)
[以不同的使用者身分登入](#)

然後,開始開發你的雲層程式:



☆ <https://appengine.google.com/start>

頁 - G...

Google app engine misoo.tw@gmail.com | [My A](#)

Welcome to Google App Engine

Before getting started, you want to learn more about developing and deploying applications. Learn more about Google App Engine by reading the [Getting Started Guide](#), the [FAQ](#), or the [Developer's](#)

© 2008 Google | [Terms of Service](#) | [Privacy Policy](#) | [Blog](#) | [Discussion Forums](#)

2.3 GAE 的框架式 API

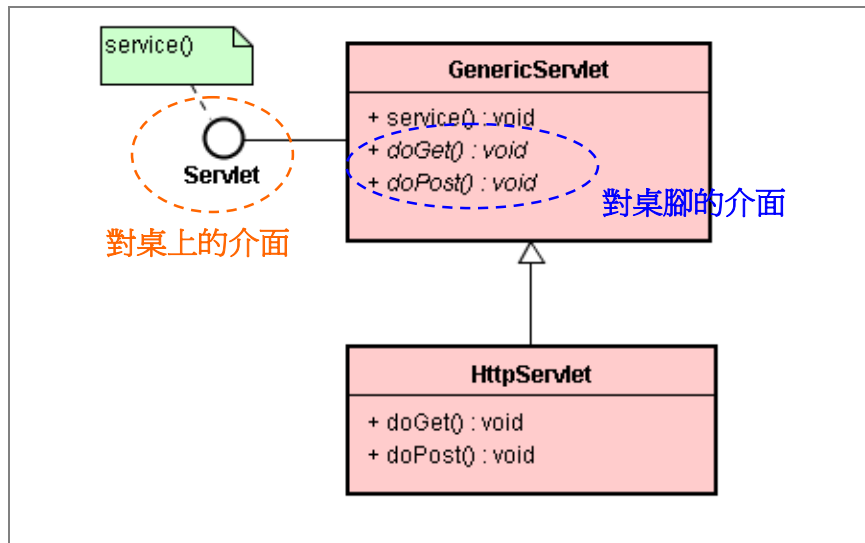
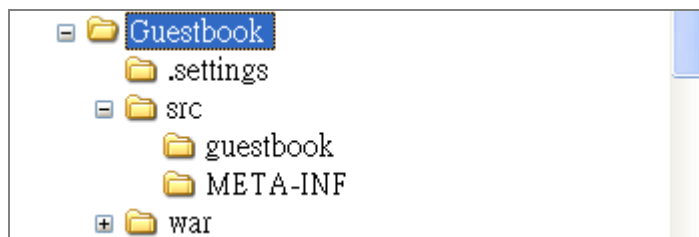


圖 2-1、框架式 API

- `get`: 可以傳遞 URL 參數；而 `Post`: 則不可以。
- 一般寫法：先用 `doGet()`，然後在 `doPost()` 裡調用 `doGet()` 函數。
- `get`: 透過 http header 來傳遞資料，有數量之限制；而 `post`: 則是透過 http body 來傳輸資料，沒有限制數量。

2.4 最簡單的 Servlet 範例：Hello World



```
// GuestbookServlet.java
package guestbook;
import java.io.IOException;
import javax.servlet.http.*;

public class GuestbookServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {
        resp.setContentType("text/plain");
    }
}
```

```

        resp.getWriter().println("Hello, world");
    }
}

```

茲以 UML 圖表示如下：

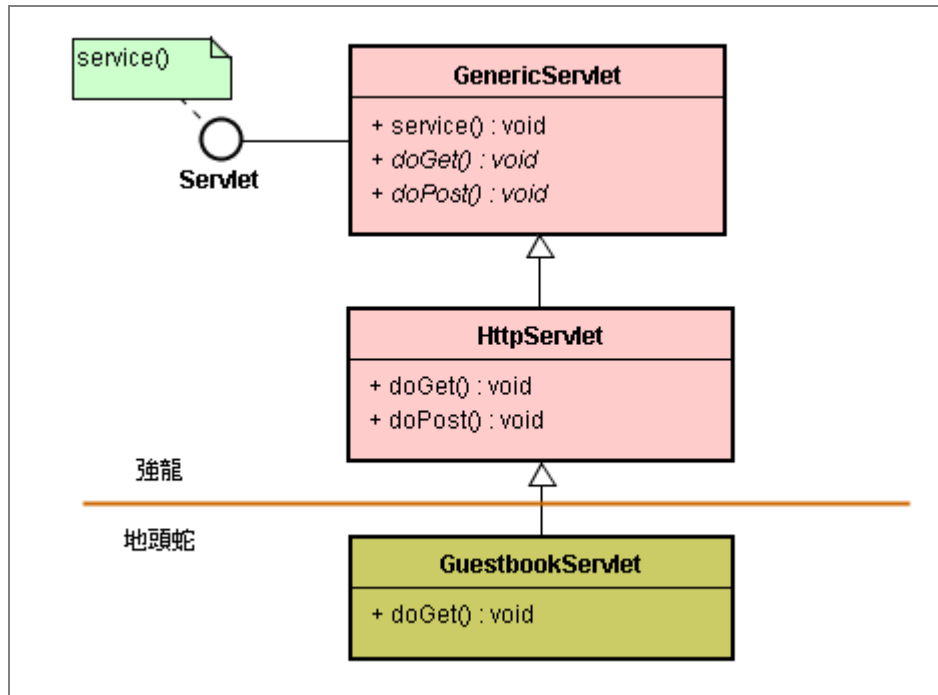


圖 2-2、GAE 框架支持「強龍/地頭蛇」商業樣式

2.5 另一個 Servlet 範例：圖片館

除了定義子類之外，還可以定義各式各樣的領域類，例如下述的 `MediaObject`、`PMF` 等類：

```

// UploadPost.java
package com.patrick.ccpmediastore;
import java.io.IOException;
import java.net.URLEncoder;
import java.util.Date;
import java.util.Iterator;
import java.util.Map;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.google.appengine.api.blobstore.BlobInfo;

```



```

import com.google.appengine.api.blobstore.BlobInfoFactory;
import com.google.appengine.api.blobstore.BlobKey;
import com.google.appengine.api.blobstore.BlobstoreService;
import com.google.appengine.api.blobstore.BlobstoreServiceFactory;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

@SuppressWarnings("serial")
public class UploadPost extends HttpServlet {
    private BlobstoreService blobstoreService = BlobstoreServiceFactory.getBlobstoreService();
    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();
        Map<String, BlobKey> blobs = blobstoreService.getUploadedBlobs(req);
        if (blobs.keySet().isEmpty()) {
            resp.sendRedirect("/?error=" + URLEncoder.encode("No file uploaded", "UTF-8"));
            return;
        }
        Iterator<String> names = blobs.keySet().iterator();
        String blobName = names.next();
        BlobKey blobKey = blobs.get(blobName);
        BlobInfoFactory blobInfoFactory = new BlobInfoFactory();
        BlobInfo blobInfo = blobInfoFactory.loadBlobInfo(blobKey);

        String contentType = blobInfo.getContentType();
        long size = blobInfo.getSize();
        Date creation = blobInfo.getCreation();
        String fileName = blobInfo.getFilename();
        String title = req.getParameter("title");
        String description = req.getParameter("description");
        boolean isShared = "public".equalsIgnoreCase(req.getParameter("share"));
        try {
            MediaObject mediaObj = new MediaObject(user, blobKey, creation,
                contentType, fileName, size, title, description, isShared);
            PMF.get().getPersistenceManager().makePersistent(mediaObj);
            resp.sendRedirect("/");
        } catch (Exception e) {
            blobstoreService.delete(blobKey);
        }
    }
}

```

```

resp.sendRedirect("/?error=" +
    URLEncoder.encode("Object save failed: " + e.getMessage(), "UTF-8"));
}}}

```

3. Android 框架與雲層框架的整合

3.1 Android 框架之例

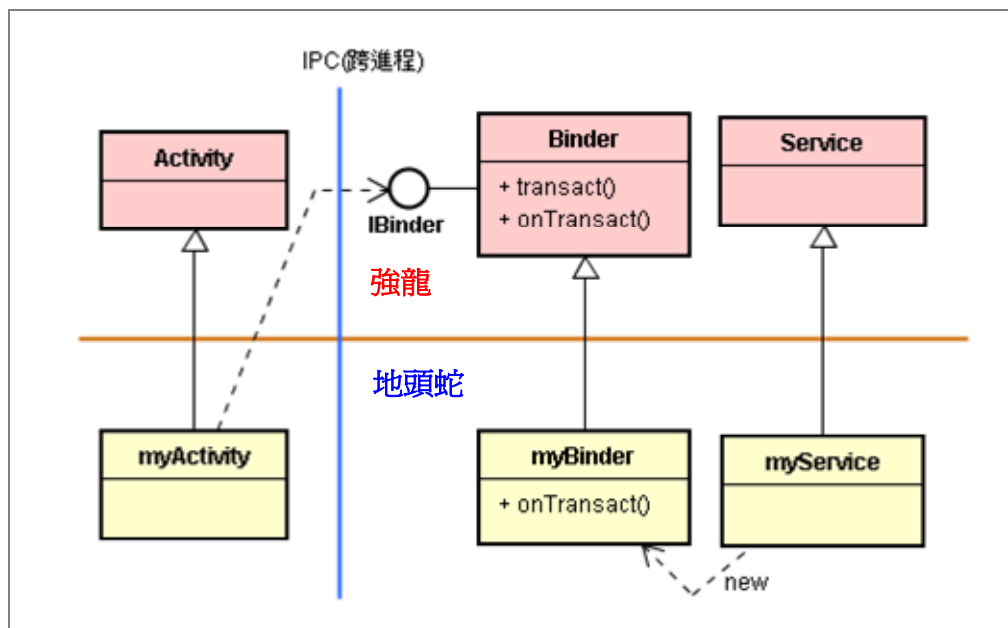


圖 3-1、Android 框架支持「強龍/地頭蛇」商業樣式

歷經 20 多年來的發展，應用及平台框架(Framework)已經成為軟件相關產業各領域的霸主手中的尚方寶劍了。君不見，長久以來位居軟件業霸主的微軟公司，其手中的尚方寶劍：.NET 就是框架。此外，在行動(Mobile)平台方面，即將成為新興霸主的 Google 公司，其手中的尚方寶劍：Android 也是框架。由此觀之，在軟件相關產業裡，手中沒有尚方寶劍的業者，或不努力研鑄尚方寶劍的國度，將注定成為軟件武林擂台邊的啦啦隊而已。

Android本身是個可執行的完整框架平台。它具有如下特性：

- 開放原始程式碼
- 力與美均衡：在如手機的有限資源、電信的大量資料壓力下，好用與效率必須兼具。Android使用Java與C++雙語言，藉由JNI來銜接，是個優越的設計樣本。
- 善用設計樣式(Design Patterns)。
- IPC Binder機制：這是高效率的遠距溝通機制。

- 執行緒(Thread)模型。
- 安全(Security)機制。
- UI架構。
- 網路服務機制。
- Intent-based 溝通機制。

等等。人人皆可基於這個開發母板，而開發自己擅長領域的框架，並融入到 Android 框架裡，一起執行，立即能看到框架的實際運作情形。此外，還能與 Google GAE 等雲層框架整合，創造雲與端的完美組合。

3.2 細說 Android 框架裡遠距通訊介面

-- 以播放音樂為例

茲以跨進程啟動 myService 為例，並誕生小執行緒(及其 MQ)：

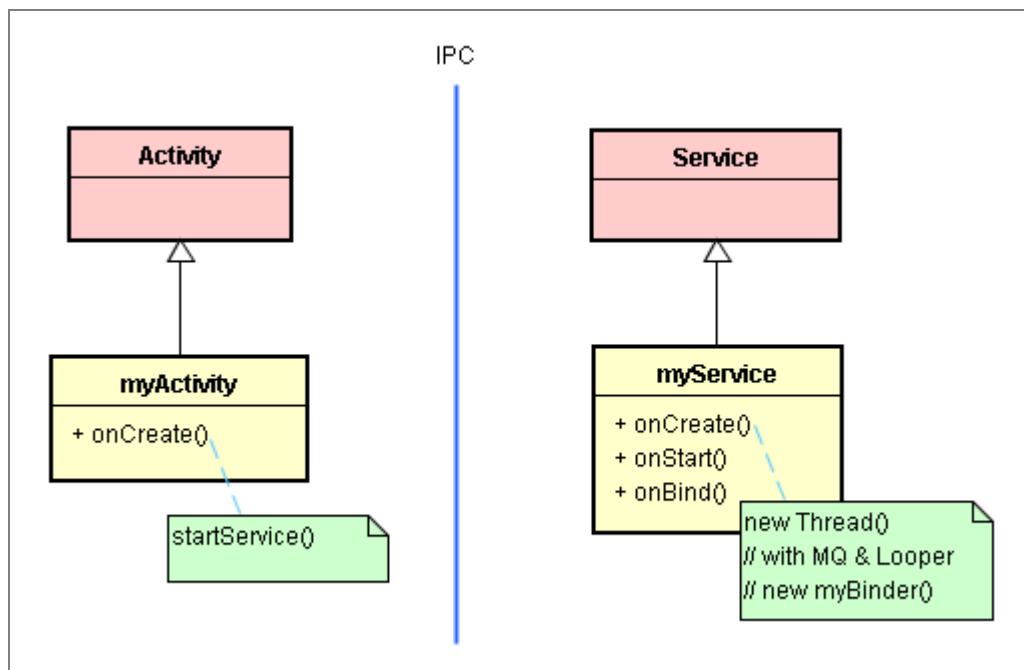


圖 3-2、由 Android 框架啟動遠距的 myService

此外，`myService.onCreate()`也誕生 `myBinder` 物件：

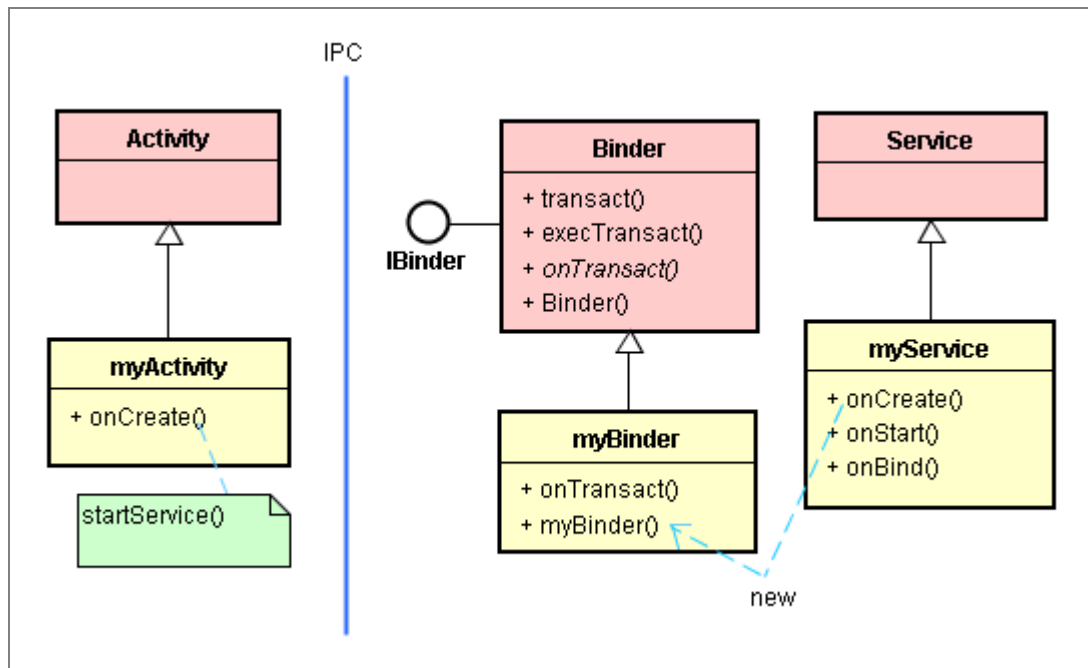


圖 4-3、由 myService 誕生 myBinder 物件

將 myBinder 物件資料傳入 C/C++層：

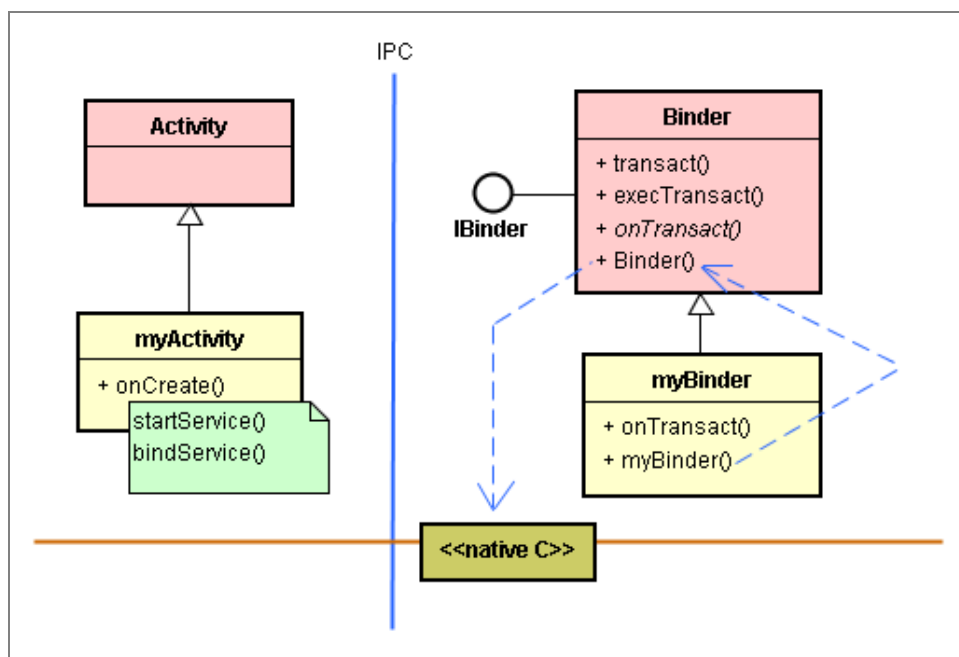


圖 3-4、由 Binder()建構式通知 Android 框架核心

Binder Kernel 調用 onBind(), 找到 myBinder 物件，建立 C/C++層的分身：

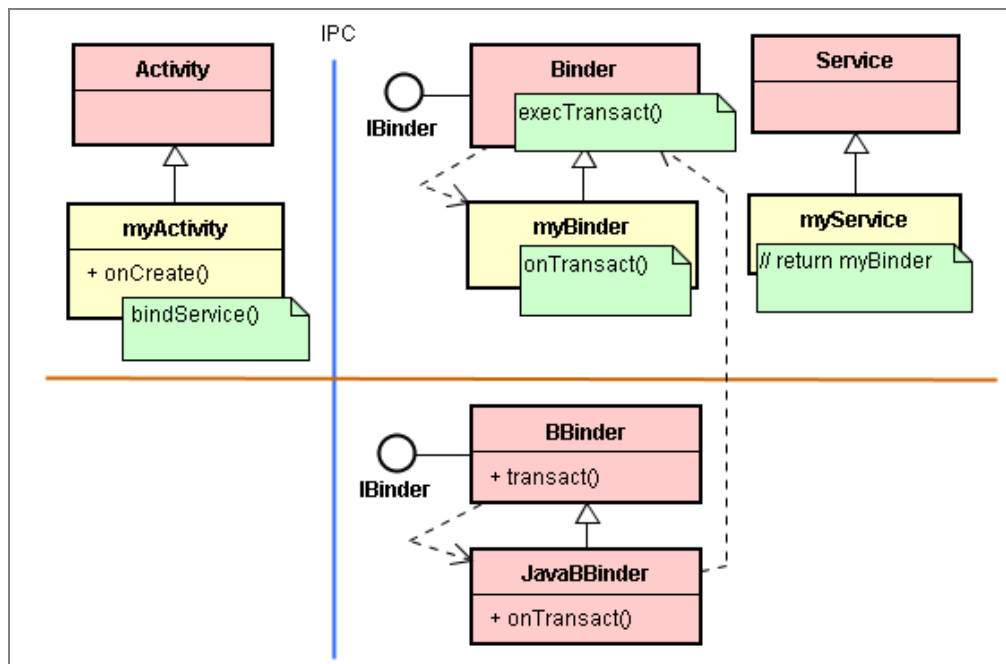


圖 3-5、由 Android 框架核心準備建立跨進成通訊管道

建立了跨進成通訊管道，並且傳回 IBinder 接口給 myActivity：

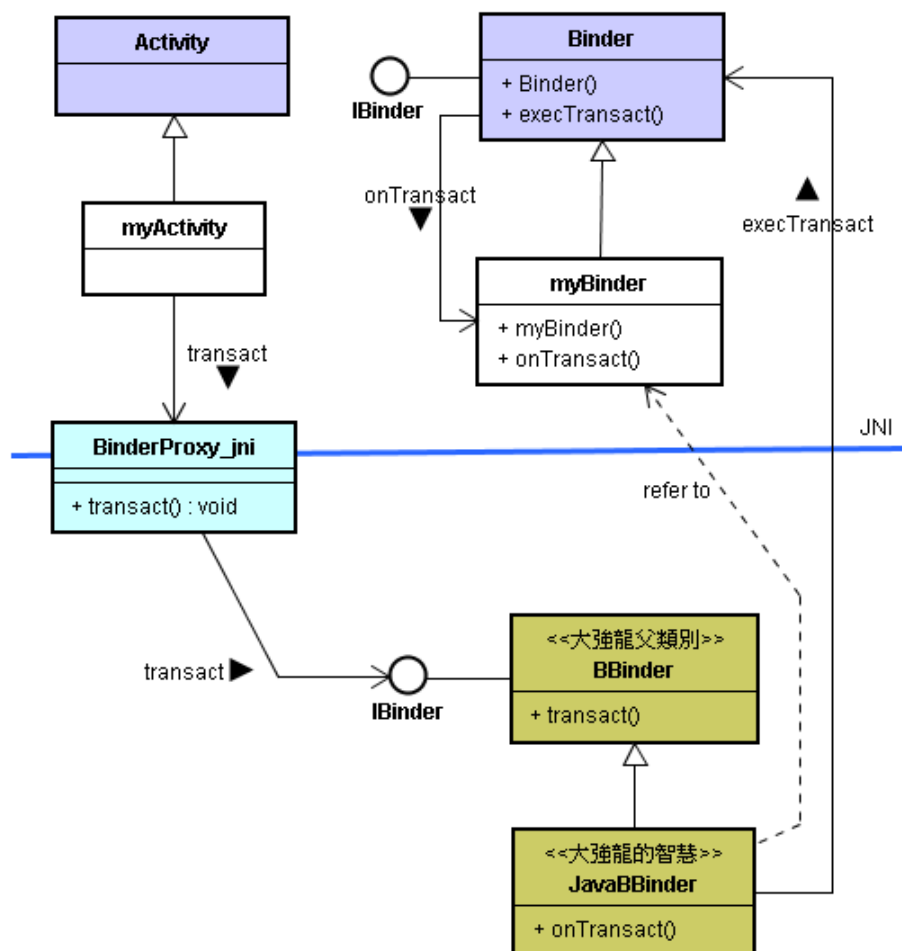


圖 3-6、建立了跨進成通訊管道

在此範例裡，myActivity 調用 IBinder 接口，執行 myBinder 的 BindThread 送信息給小執行緒，由小執行緒去播放 mp3 音樂。

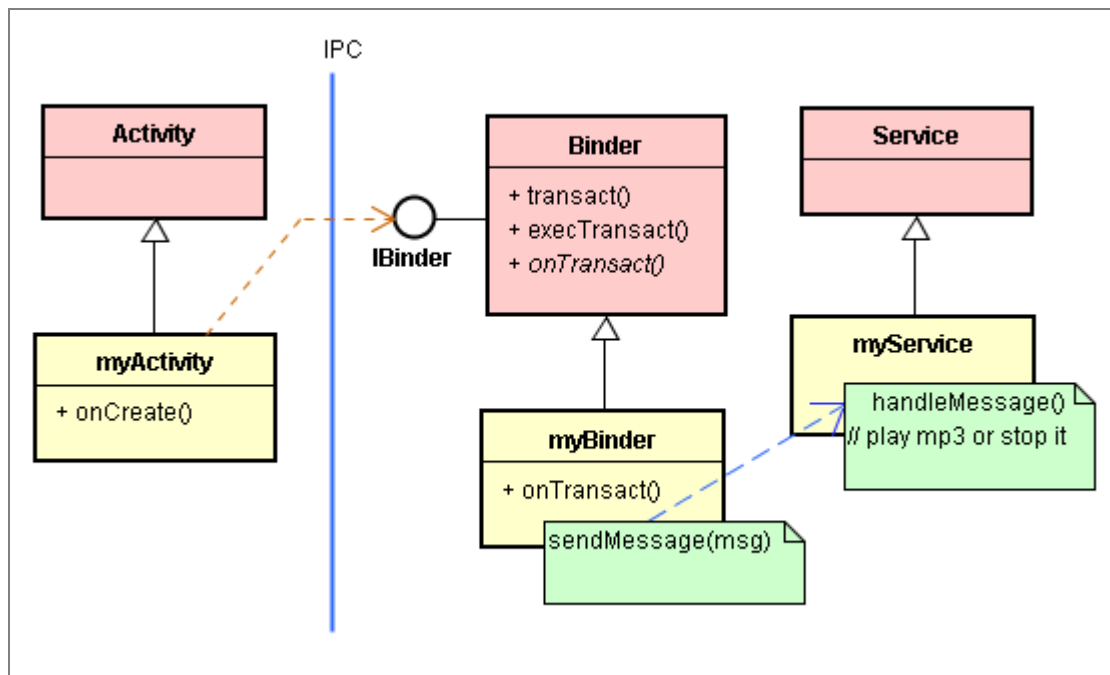


圖 3-7、透過 IBinder 介面來與 Service 溝通

小執行緒也送 Broadcast 信息給 myActivity，將字串顯示於畫面上：

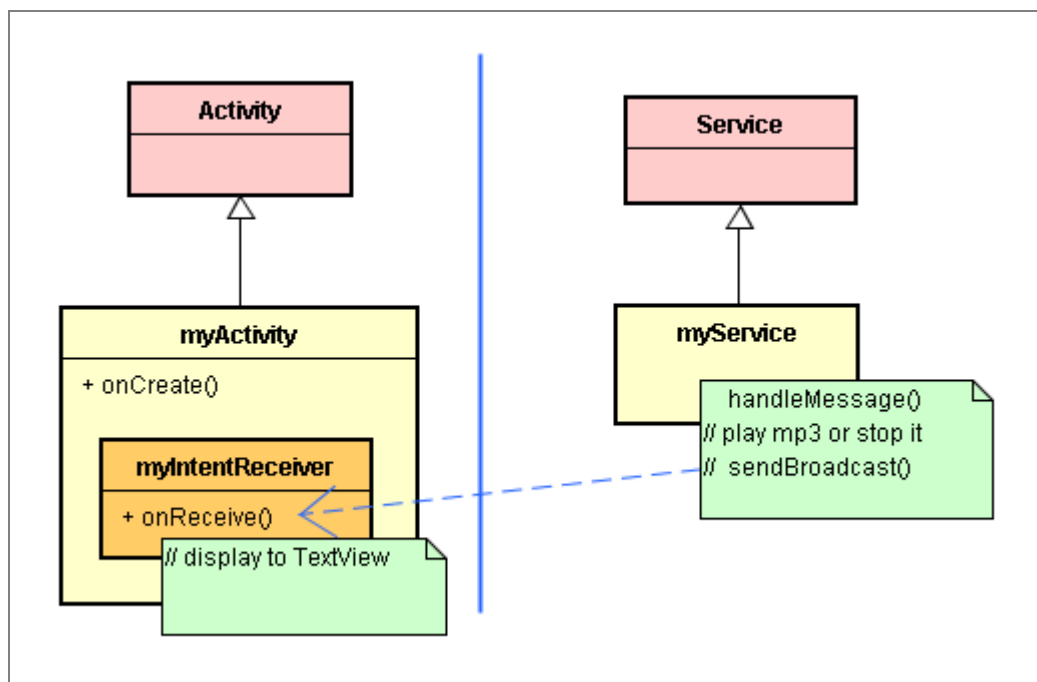
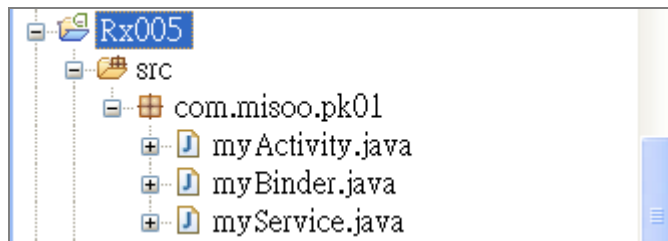
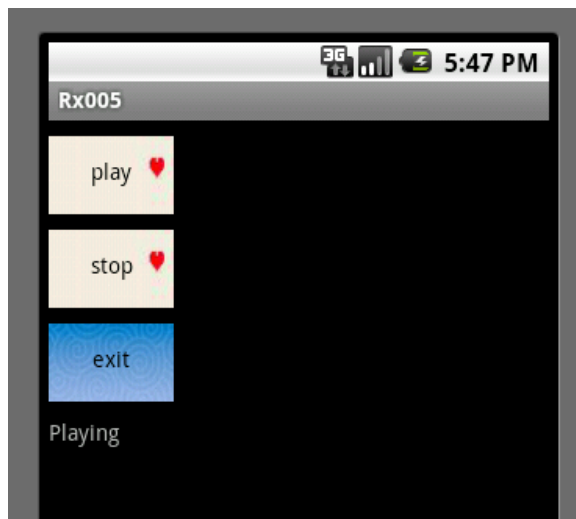


圖 3-8、Service 透過 Broadcast 訊息來通知 Activity

茲列出原始程式碼：



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.misoo.pk01">
    <application android:icon="@drawable/icon">
        <activity android:name=".myActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".myService" android:process=":remote">
            <intent-filter>
                <action android:name="com.misoo.pk01.REMOTE_SERVICE" />
            </intent-filter>
        </service>
    </application>
</manifest>
```



```
// myActivity.java
package com.misoo.pk01;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
```

```

import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.Parcel;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class myActivity extends Activity implements OnClickListener {
    private final int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
    private final int FP = LinearLayout.LayoutParams.FILL_PARENT;
    private Button btn, btn2, btn3;
    public TextView tv;
    private IBinder ib = null;
    private final String MY_S_EVENT =
        new String("com.misoo.pk01.myService.MY_S_EVENT");
    protected final IntentFilter filter=new IntentFilter(MY_S_EVENT);
    private BroadcastReceiver receiver=new myIntentReceiver();

    public void onCreate(Bundle icle) {
        super.onCreate(icle);

        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        btn = new Button(this);
        btn.setId(101);    btn.setText("play");
        btn.setBackgroundResource(R.drawable.heart);
        btn.setOnClickListener(this);
        LinearLayout.LayoutParams param =
            new LinearLayout.LayoutParams(80, 50);
        param.topMargin = 10;
        layout.addView(btn, param);

        btn2 = new Button(this);
        btn2.setId(102);    btn2.setText("stop");
        btn2.setBackgroundResource(R.drawable.heart);
        btn2.setOnClickListener(this);
        layout.addView(btn2, param);

        btn3 = new Button(this);
        btn3.setId(103);    btn3.setText("exit");
        btn3.setBackgroundResource(R.drawable.cloud);
        btn3.setOnClickListener(this);
        layout.addView(btn3, param);

        tv = new TextView(this);
        tv.setText("Ready");
        LinearLayout.LayoutParams param2 = new
            LinearLayout.LayoutParams(FP, WC);
        param2.topMargin = 10;
        layout.addView(tv, param2);
        setContentView(layout);
        //-----

```



```

registerReceiver(receiver, filter);
//-----
bindService(new Intent("com.misoo.pk01.REMOTE_SERVICE"),
            mConnection, Context.BIND_AUTO_CREATE);
}
private ServiceConnection mConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName className, IBinder
        ibinder) {
        ib = ibinder;
    }
    @Override
    public void onServiceDisconnected(ComponentName name) {}
};
public void onClick(View v) {
    switch (v.getId()) {
        case 101:
            Parcel data = Parcel.obtain();
            Parcel reply = Parcel.obtain();
            try {
                ib.transact(1, data, reply, 0);
            } catch (Exception e) {
                e.printStackTrace();
            }
            break;
        case 102:
            data = Parcel.obtain();
            reply = Parcel.obtain();
            try {
                ib.transact(2, data, reply, 0);
            } catch (Exception e) {
                e.printStackTrace();
            }
            break;
        case 103:
            finish();
            break;
    }
}
//-----
class myIntentReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        int bn = intent.getIntExtra("key",-1);
        if(bn == 0)
            tv.setText("Playing");
        else
            tv.setText("Stop.");
    }
}

//-----
// myService.java
package com.misoo.pk01;

```

```

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.Handler;
import android.os.IBinder;
import android.os.Looper;
import android.os.Message;
import android.util.Log;

public class myService extends Service implements Runnable {
    private IBinder mBinder = null;
    private Thread th1;
    public static Handler h;
    private MediaPlayer mPlayer = null;
    public static Context ctx;
    private final String MY_S_EVENT =
        new String("com.misoo.pk01.myService.MY_S_EVENT");

    @Override
    public void onCreate() {
        super.onCreate();
        ctx = this;
        mBinder = new myBinder();
        //-----
        // 誕生一個子執行緒及其MQ
        // 等待Message
        //-----
        th1 = new Thread(this);
        th1.start();
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }

    public void run() {
        Looper.prepare();
        h = new EventHandler(Looper.myLooper());
        Looper.loop();
    }
    //-----
    class EventHandler extends Handler {
        public EventHandler(Looper looper) {
            super(looper);
        }
        public void handleMessage(Message msg) {
            String obj = (String)msg.obj;
            if(obj.contains("play"))
            {
                if(mPlayer != null) return;
                //-----
                Intent in = new Intent(MY_S_EVENT);
                in.putExtra("key", 0);
                ctx.sendBroadcast(in);
            }
        }
    }
}

```

```

//-----
mPlayer = MediaPlayer.create(ctx, R.raw.dreamed);
try {
    mPlayer.start();
} catch (Exception e) {
    Log.e("Play", "error: " + e.getMessage(), e);
}

}
else if(obj.contains("stop")) {
    if (mPlayer != null) {
        //-----
        Intent in = new Intent(MY_S_EVENT);
        in.putExtra("key", 1);
        ctx.sendBroadcast(in);
        //-----
        mPlayer.stop();
        mPlayer.release();
        mPlayer = null;
    }
}
}
}

}

//-----
// myBinder.java
package com.misoo.pk01;
import android.os.Binder;
import android.os.Message;
import android.os.Parcel;

public class myBinder extends Binder{
    @Override
    public boolean onTransact(int code, Parcel data, Parcel reply, int flags)
        throws android.os.RemoteException {
        switch(code){
            case 1:
                // 將Message丟到子執行緒的MQ to draw Graphic
                String obj = "play";
                Message msg = myService.h.obtainMessage(1,1,1,obj);
                myService.h.sendMessage(msg);
                break;
            case 2:
                // 將Message丟到子執行緒的MQ to stop drawing
                obj = "stop";
                msg = myService.h.obtainMessage(1,1,1,obj);
                myService.h.sendMessage(msg);
                break;
        }
        return true;
    }
}

```

4.3 Android 框架 + GAE 框架

Android 的跨進程介面 IBinder，其角色相當於雲層裡的 Servlet 介面：

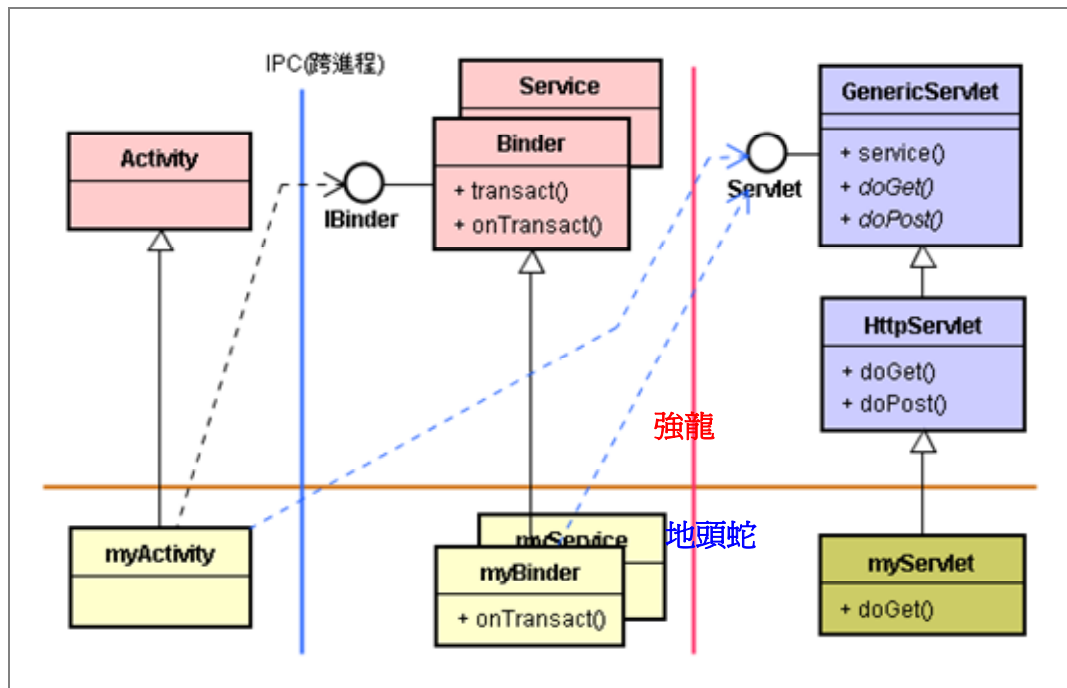
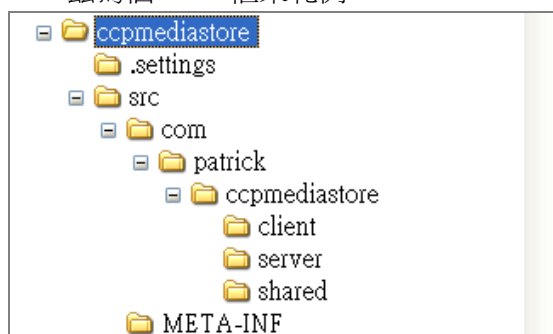


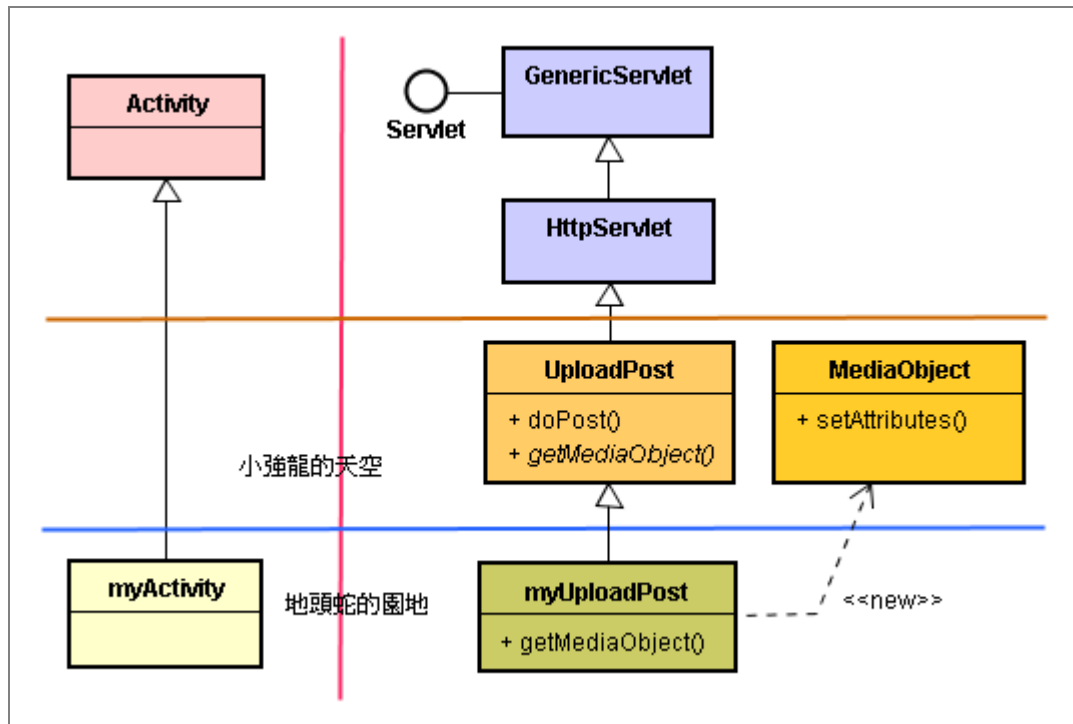
圖 4-9、Android 端框架與 GAE 雲框架的完美整合

無論 Android 還是 GAE 雲層都是以框架來支撐「強龍/地頭蛇」商業樣式。

3.4 Android 框架 + GAE 框架之範例

茲寫個 GAE 框架範例：





其中的 UploadPost 和 myUploadPost 兩個類的程式碼如下：

```

//UploadPost.java
package com.patrick.ccpmediastore;
import java.io.IOException;
import java.net.URLEncoder;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
public abstract class UploadPost extends HttpServlet {

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
        IOException {

        try {
            PMF.get().getPersistenceManager().makePersistent(getMediaObject(req));
            resp.sendRedirect("/");
        } catch (Exception e) {
            resp.sendRedirect("/?error=" +
                URLEncoder.encode("Object save failed: " + e.getMessage(), "UTF-8"));
        }
    }

    protected abstract MediaObject getMediaObject(HttpServletRequest req);
}
  
```

```
// myUploadPost.java
package com.patrick.ccpmediastore;
import java.io.IOException;
import java.util.Date;
import java.util.Iterator;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.google.appengine.api.blobstore.BlobInfo;
import com.google.appengine.api.blobstore.BlobInfoFactory;
import com.google.appengine.api.blobstore.BlobKey;
import com.google.appengine.api.blobstore.BlobstoreService;
import com.google.appengine.api.blobstore.BlobstoreServiceFactory;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

@SuppressWarnings("serial")
public class myUploadPost extends UploadPost {

    private BlobstoreService blobstoreService =
        BlobstoreServiceFactory.getBlobstoreService();
    protected MediaObject getMediaObject(HttpServletRequest req) {
        UserService userService = UserServiceFactory.getUserService();
        User user = userService.getCurrentUser();

        Map<String, BlobKey> blobs = blobstoreService.getUploadedBlobs(req);
        Iterator<String> names = blobs.keySet().iterator();
        String blobName = names.next();
        BlobKey blobKey = blobs.get(blobName);
        BlobInfoFactory blobInfoFactory = new BlobInfoFactory();
        BlobInfo blobInfo = blobInfoFactory.loadBlobInfo(blobKey);

        String contentType = blobInfo.getContentType();
        long size = blobInfo.getSize();

        Date creation = blobInfo.getCreation();
        String fileName = blobInfo.getFilename();

        String title = req.getParameter("title");
        String description = req.getParameter("description") + "from myNewUploadPost";
        boolean isShared = "public".equalsIgnoreCase(req.getParameter("share"));
        MediaObject mediaObj = new MediaObject(user, blobKey, creation,
            contentType, fileName, size, title, description, isShared);
        return mediaObj ;
    }
}
```

一樣的框架設計思維、方法和技術，應用於 Android 行動端上，同時應用於 GAE 雲層上。框架成為實現雲與整合 DAAS(Domain As A Service)的利器。

~~ END ~~