Android 的 Java 層應用框架

控制(力)：控制核心
和諧(美)：整合核心
代碼實踐：Java 與 C/C++整合開發
關鍵要素：進程(Process)與線程(Thread)

■ 最時髦的尚方寶劍：雙層框架，力與美的組合!

傳統無框架

傳統上，在沒有框架的環境裡，應用程序(Application,簡稱 AP)由地頭蛇負責開發，它會調用大強龍所開發的操作平台(如 Linux, Windows Mobile 等)，如下圖所示：

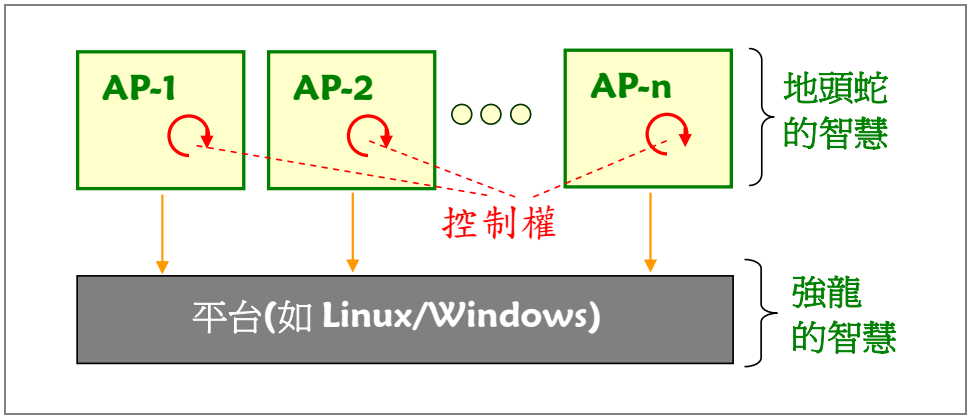

圖 1　傳統軟件架構違背「強龍/地頭蛇」商業模式

地頭蛇的智慧撰寫於 AP 軟件裡，而強龍的智慧撰寫於平台軟件裡。由於控制權掌握在地頭蛇手中，所以在軟件架構上，強龍反而受制於地頭蛇，這與「強龍/地頭蛇」商業模式相違背。於是，框架型式的軟件架構就逐漸蔚爲潮流。

單層框架

爲了讓軟件架構與「強龍/地頭蛇」商業模式能吻合一致，就得更改軟件架構。框架型式的軟件架構，讓強龍擁有主控權，成爲名符其實的強龍了。在這種軟件架構裡，是由框架裡的基類(Base class)來調用 AP 裡的子

類，於是框架擁有軟件執行上的控制權，由框架來指揮 AP 的運行。於是，開發框架的強龍就掌握軟件執行的主控權。框架就成爲強龍手中的尚方寶劍了。如下圖所示：

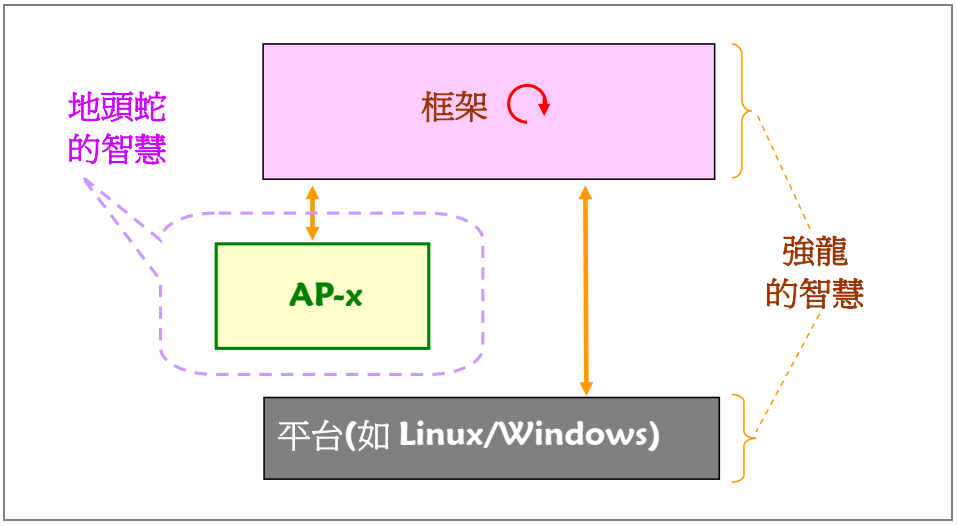

圖 2　框架型式軟件架構支持「強龍/地頭蛇」商業模式

基於這個框架型式的軟件架構，在各個應用領域裡的軟件開發者，皆能運用大強龍的母框架，來孵化出其獨特的 DSF 小框架。這種複合型的框架，就如下圖：

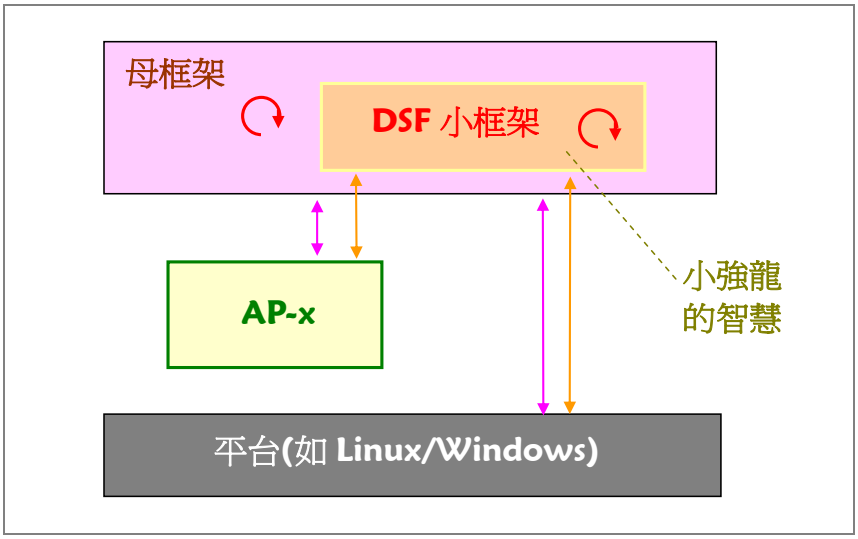

圖 3　人人做 DSF，人人都有機會成爲小強龍

雙層框架

由於框架與 AP 是依賴面向對象(Object-Oriented)電腦語言的繼承(Inheritance)機制來相互結合的。這是屬於源代碼(Source code)層級的結合，所以大強龍必須將框架的源代碼交給地頭蛇去合併入 AP 源代碼，才能一起

編譯完成可執行的軟件。爲了減輕地頭蛇開發 AP 的工作量，許多強龍都採取較爲簡潔，又很普及的 Java 語言來撰寫框架。大強龍、小強龍和地頭蛇都使用 Java 來表達其智慧，Java 語言的編譯器(Compiler)就能順利將三種智慧順利結合起來了。如下圖：

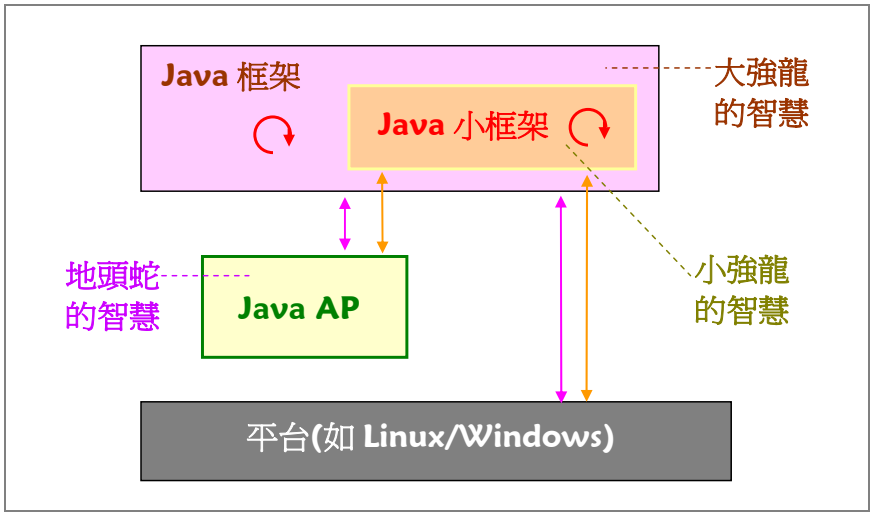

圖 4　Java 框架具有簡潔之美

雖然 Java 語言具有簡單容易之美，但是其執行效率比 C++低。爲了追求力與美的組合，在目前的產業裡，許多強龍都推出雙層框架，建立出 Java 和 C++並存的雙層框架。如下圖：

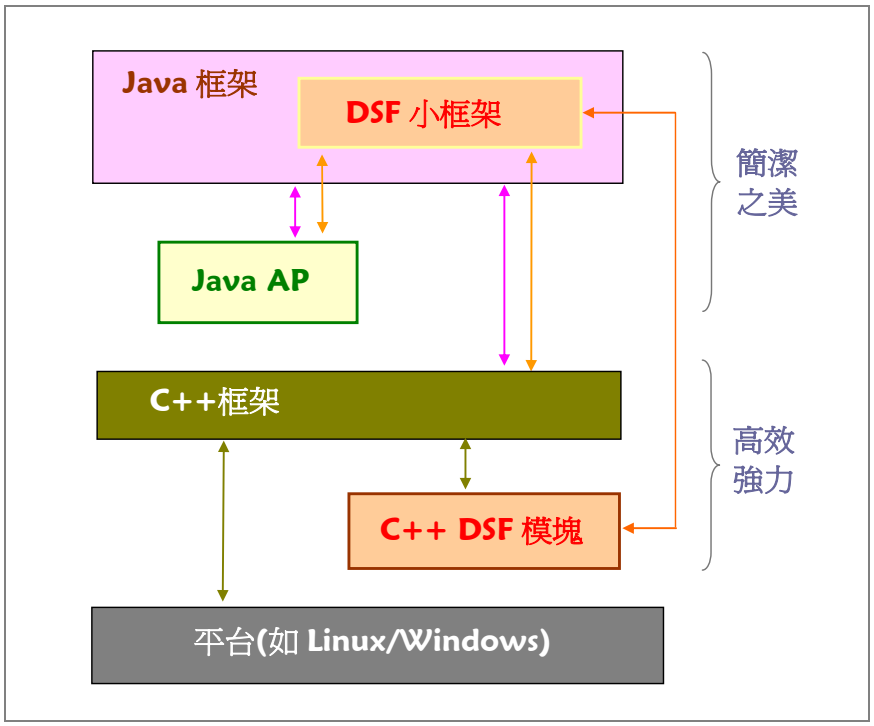

圖 5　Java/C++雙層框架：力與美的組合

雙層框架也提供給小強龍絕佳的生存和茁壯的空間。著名的 Eclipse 框架，以及現今當紅的 Android 都是 Java/C++雙層框架，例如：

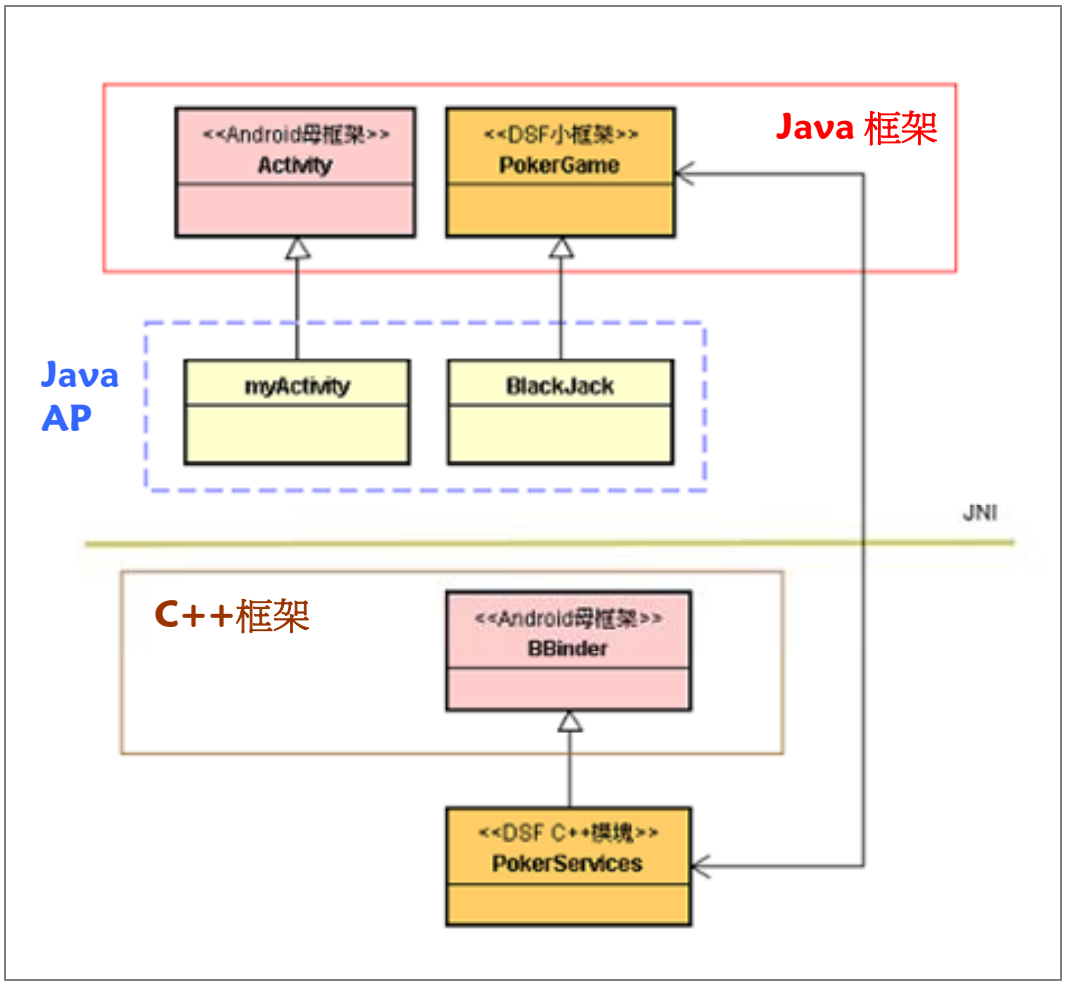

圖 6　Android 的雙層框架示例

除了雙層框架之外，有些大強龍更是推出三層框架，其非常有利於軟、硬件的整合。最上層框架，除了 Java 之外，也有些多層框架搭配其它動態語言，給予地頭蛇更簡單、更方便的 AP 開發環境。創造了大強龍、小強龍及地頭蛇多贏的局面。◆

以 Android 整合設計爲例：4 个嫡系基类：

Activity、Service、BroadcastReceiver 和 ContentProvider

Android 母框架裡提供了 4 個一等公民(或稱爲嫡系)的基類，包括：

- Activity: 處理 UI 互動的事情
- Service:幕後服務(如硬體及 Driver 的服務)

- BroadcastReceiver: 接收訊息及事件處理
- ContentProvider: 儲存共用資料

如下圖所示：

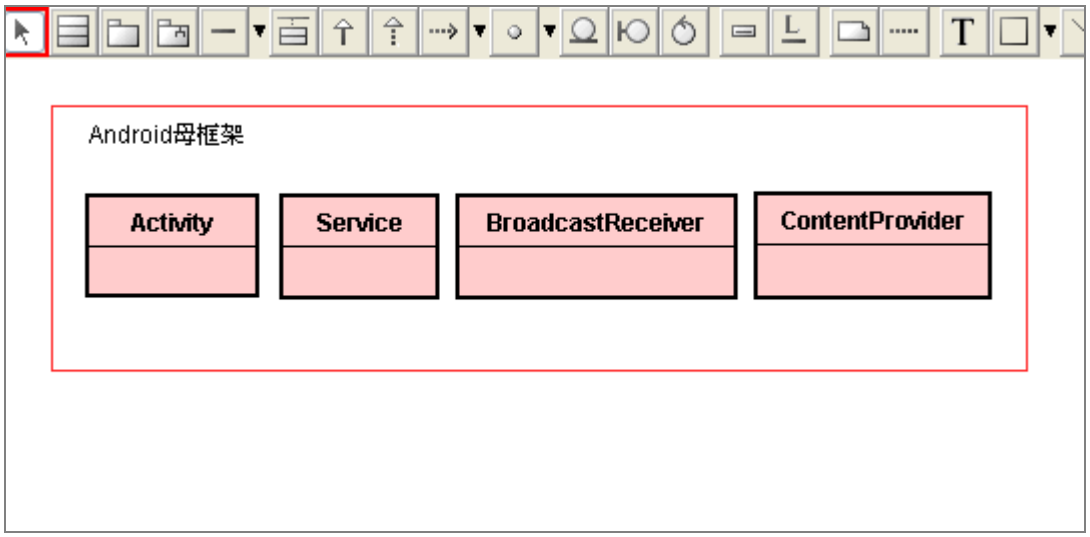

圖 7　Android 母框架裡的 4 個嫡系基類

基於這些基類，地頭蛇就可以撰寫 AP 子類，如下圖：

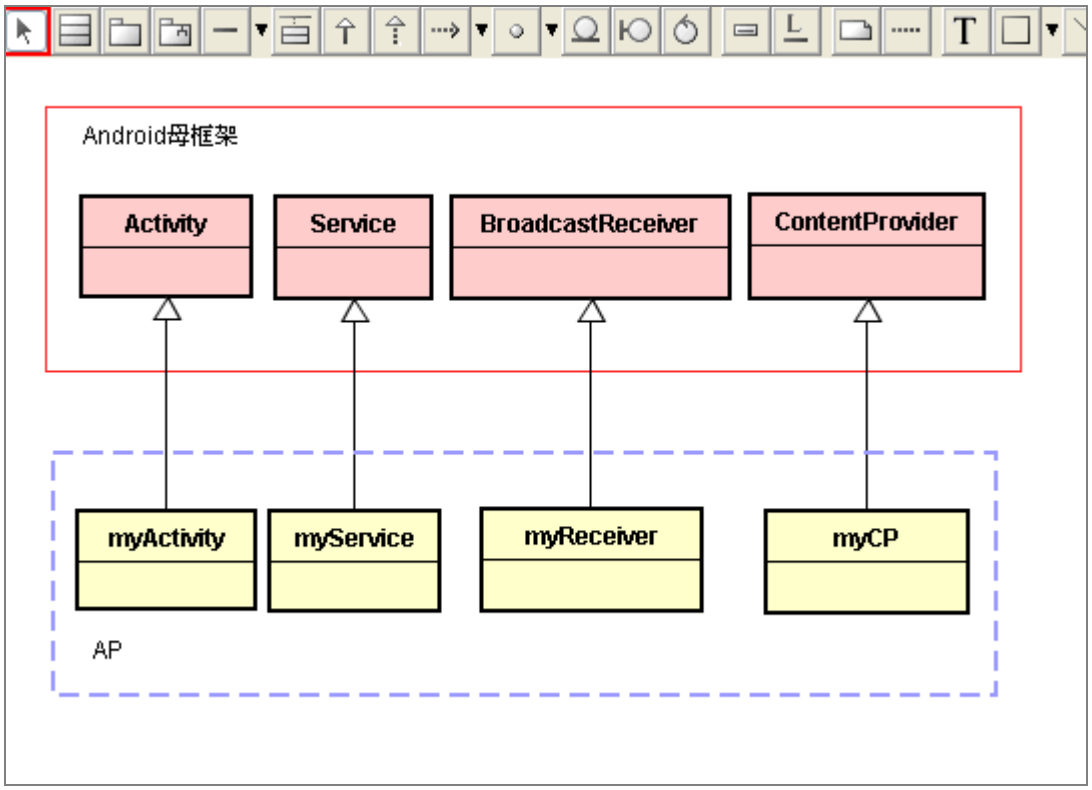

圖 8　強龍與地頭蛇雙方智慧結合了

這 4 種應用子類都是由 Android 母框架來負責創建(Create or New)其對象(Object)的。不過有趣的是：

- 強龍撰寫框架基類在先
- 地頭蛇撰寫應用子類在後

那麼框架事先又如何知道地頭蛇後來撰寫的應用子類的名稱呢? 如果不知道應用子類的名稱，又如何創建應用子類的對象呢? 答案是：依賴 AndroidManifest.xml 文檔。例如：

// AndroidManifest.xml

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

在執行階段(Run-time)，Android 母框架讀取這個由地頭蛇所寫的 XML 文檔。於是母框架得知地頭蛇撰寫了 3 個嫡系應用子類，如下圖：

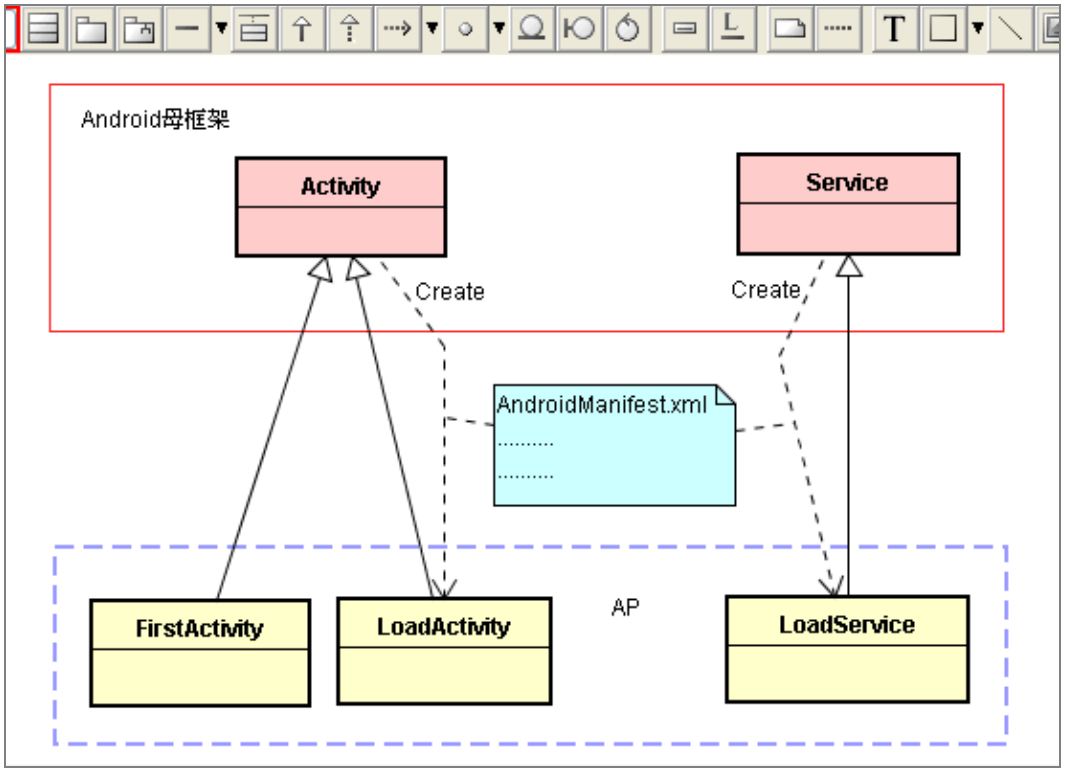

圖 9　母框架創建嫡系應用子類的對象

這些應用子類的對象可以全部在同一個進程(Process)裡執行，也可以在不同的進程裡執行。例如，母框架從上述 AndroidManifest.xml 裡讀到：

<service android:name=".LoadService" android:process=":remote">

母框架就會將 LoadService 應用子類的對象誕生於獨立的進程(名稱叫“remote”)裡。於是，FirstActivity 與 LoadService 之間就屬於跨進程的溝通了。這種跨進程的溝通，就是大家熟知的 IPC(Inter-Process Communication)機制了。待會兒，將會特別介紹 Android 母框架裡的 IPC 機制。

JNI：銜接上下層的 6 橋樑

■ JNI 本地程序開發技術

在雙層框架裡，上層是 Java 框架，而下層是 C/C++框架。這兩層框架之間會有密切的溝通。此時 JNI(Java Native Interface)就扮演雙方溝通的接口了。藉由 JNI 接口，可將 Java 層的基類或子類的函數實作部份挖空，而移到 JNI 層的 C 函數來實作之。例如，原來在 Java 層有個完整的 Java 類：

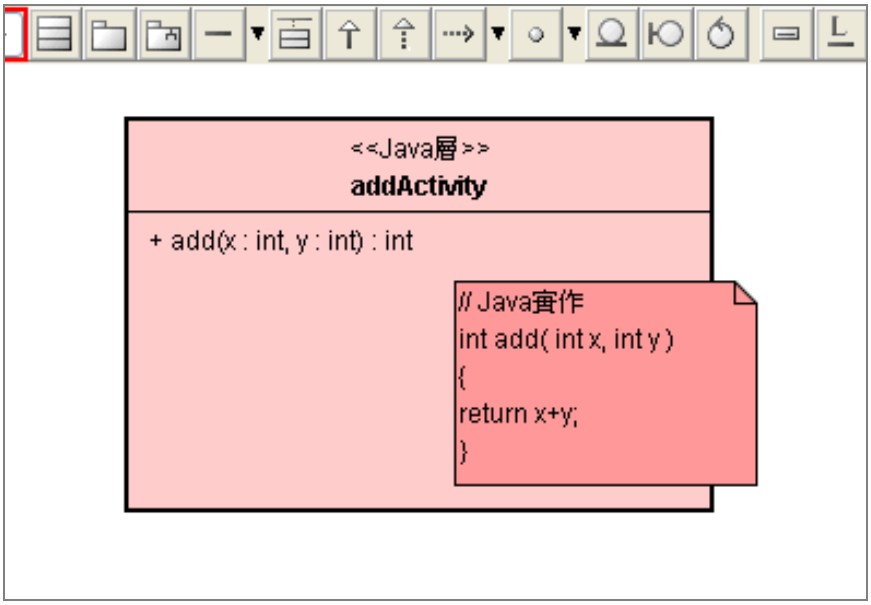

圖 10 一般的 Java 類

這是一個完整的 Java 類，其 add()函數裡有完整的實作(Implement)代碼。如果從這 Java 類裡移除掉 add()函數裡的實作代碼(就如同抽象類裡的抽象函數一般)，而成爲本地(Native)函數；然後依循 JNI 接口協定而以 C 語言來實作之。如下圖所示：

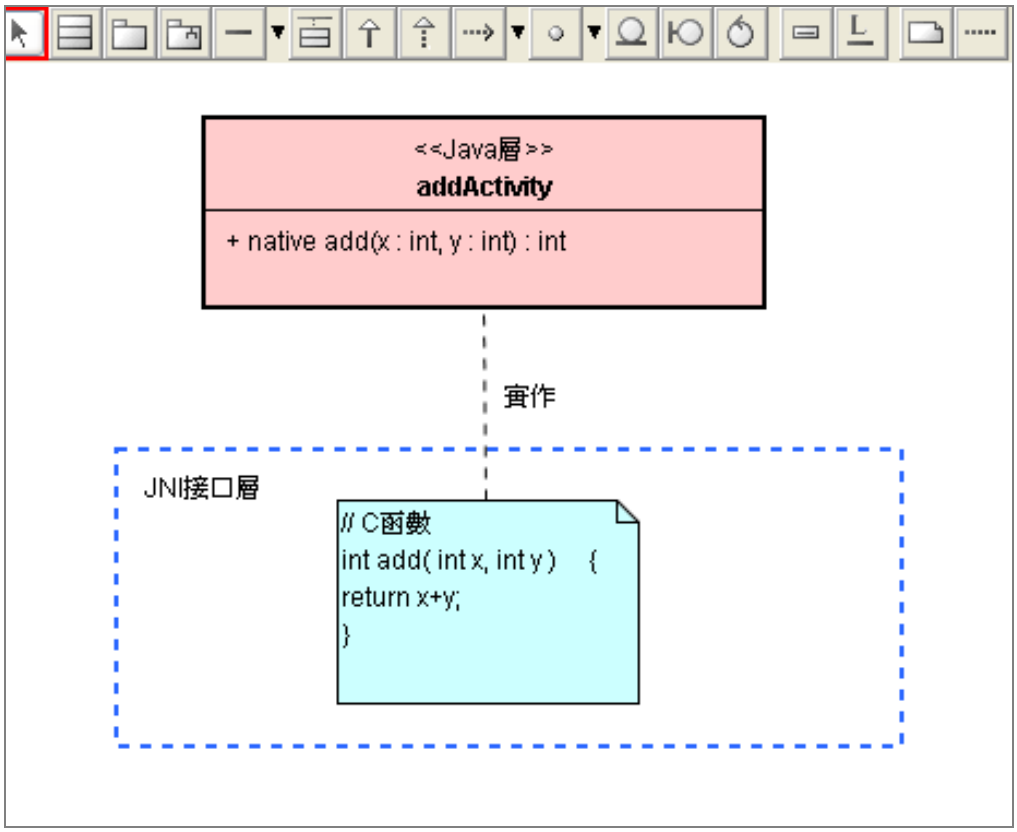

圖 11 以 C 語言來是做 Java 類的函數

這個 add()函數仍然是 Java 類的一部分，只是它是用 C 語言來實作而已。爲什麼要將 Java 類的 add()函數挖空呢? 其主要的理由是：Java 代碼執行速度較慢，而 C 代碼執行速度快。然而 Java 代碼可以跨平台，而 C 代碼與本地平台設備息息相關，所以稱之爲本地(Native)代碼。

在本地的C代碼裡，可以創建C++類的對象，並調用其函數。如下圖：

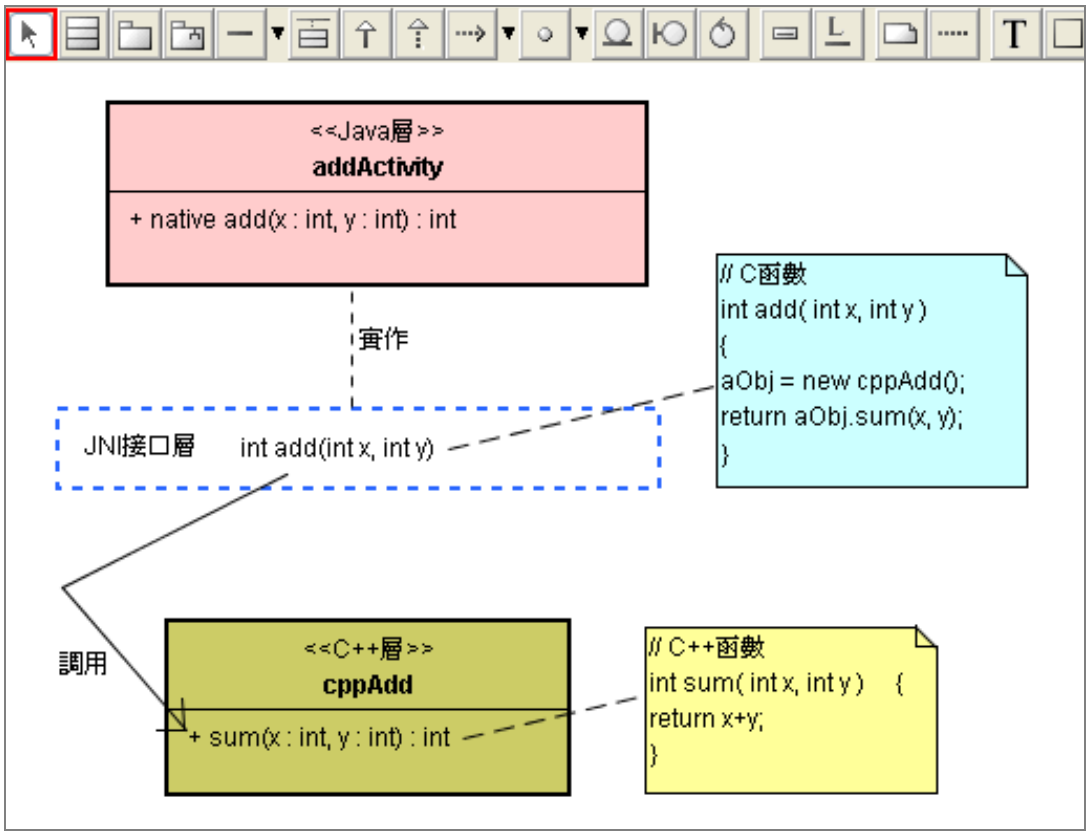

圖 12　Java 類透過本地程序來調用 C++函數

此圖可以簡潔地表示如下：

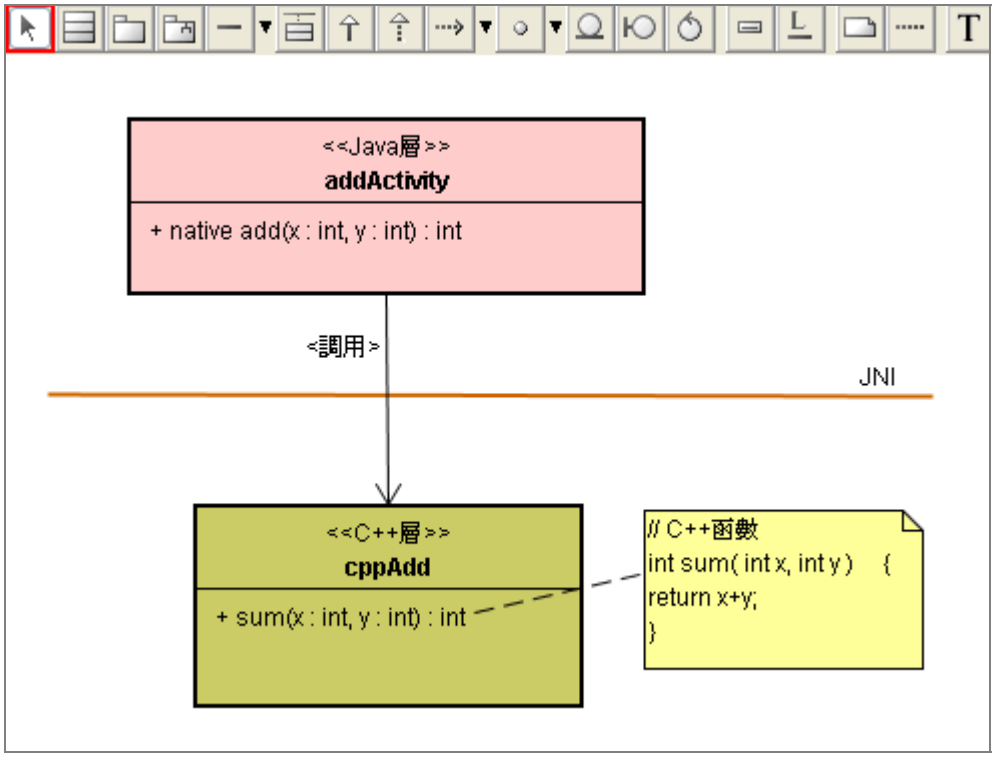

圖 13　JNI 是銜接 Java 層與 C++層的接口

藉由 JNI 接口，就能讓 Java 類與 C++類互相溝通起來了。這也是 Android 雙層框架的重要基礎機制。如下圖所示：

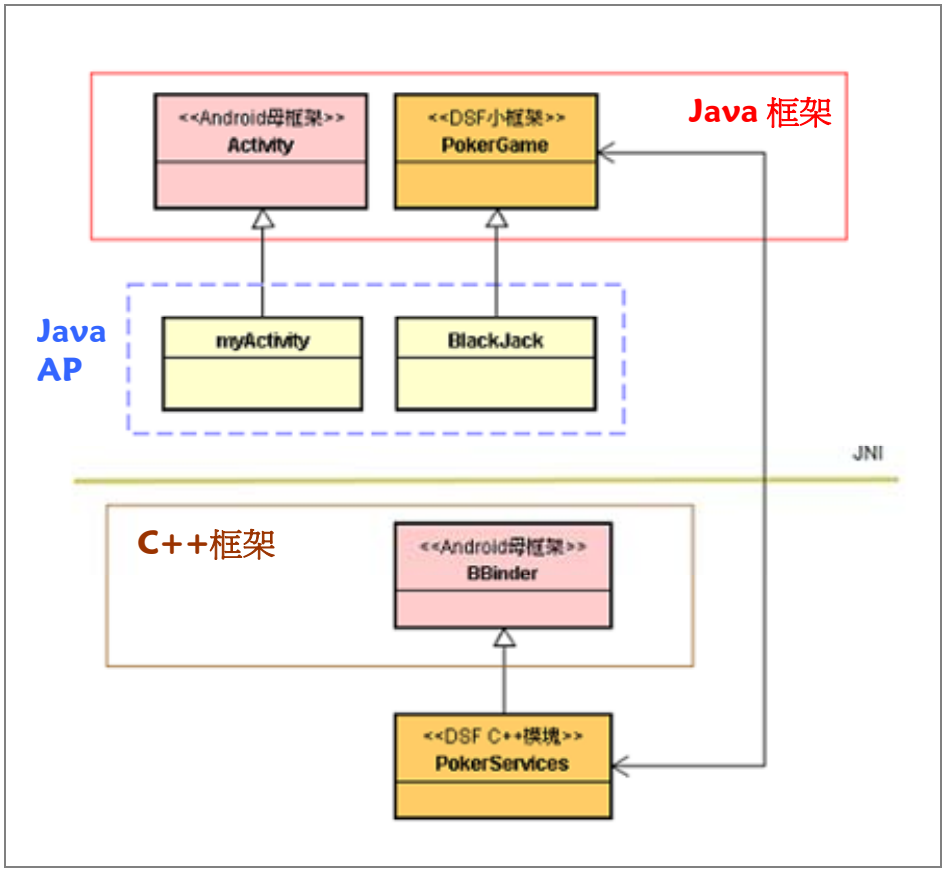

圖 14　JNI 接口是 Android 的雙層框架幕後的重要支柱

進程與線程模式

■ IPC 機制及多線程技術

在 Android 框架裡，一個應用套件(Application Package)通常含有多個 Java 類(Class)，這些類可以在同一個進程(Process)裡執行；也可以在不同的進程裡執行。基於 Linux 的安全限制，以及進程的基本特性(例如，不同進程的位址空間是獨立的)，如果兩個類(或其對象)在同一個進程裏執行時，兩者溝通方便也快速。但是，當它們分別在不同的進程裡執行時，兩者溝通就屬於 IPC 跨進程溝通了，不如前者方便，也慢些。

一個進程是一個獨立的執行空間，不會被正在其他進程裡的程序所侵犯。這種保護方法是 Android 的重要安全機制。於是，得先認識進程的內涵，才能進一步了解跨進程 IPC(Inter-Process Communication)機制。

在Android的進程裡，有一個虛擬機(Virtual Machine，簡稱VM)的對象，可執行Java代碼，也引導JNI本地程序的執行，實現Java與C/C++程序之間的溝通；如下圖：

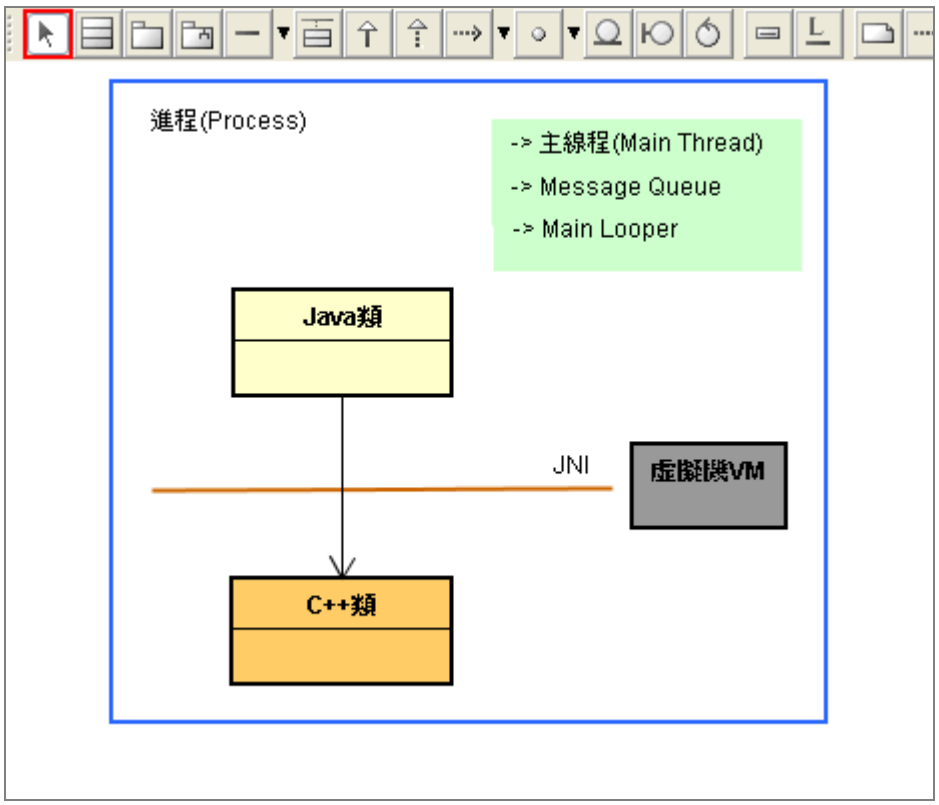

圖 15　Android 進程裡的基本元素

每一個進程在誕生時，都會誕生一個主線程(Main Thread)，以及誕生一個Looper類的對象和一個MQ(Message Queue)資料結構。每當主線程作完事情，就會去執行Looper類。此時，不斷地觀察MQ的動態。如下圖：

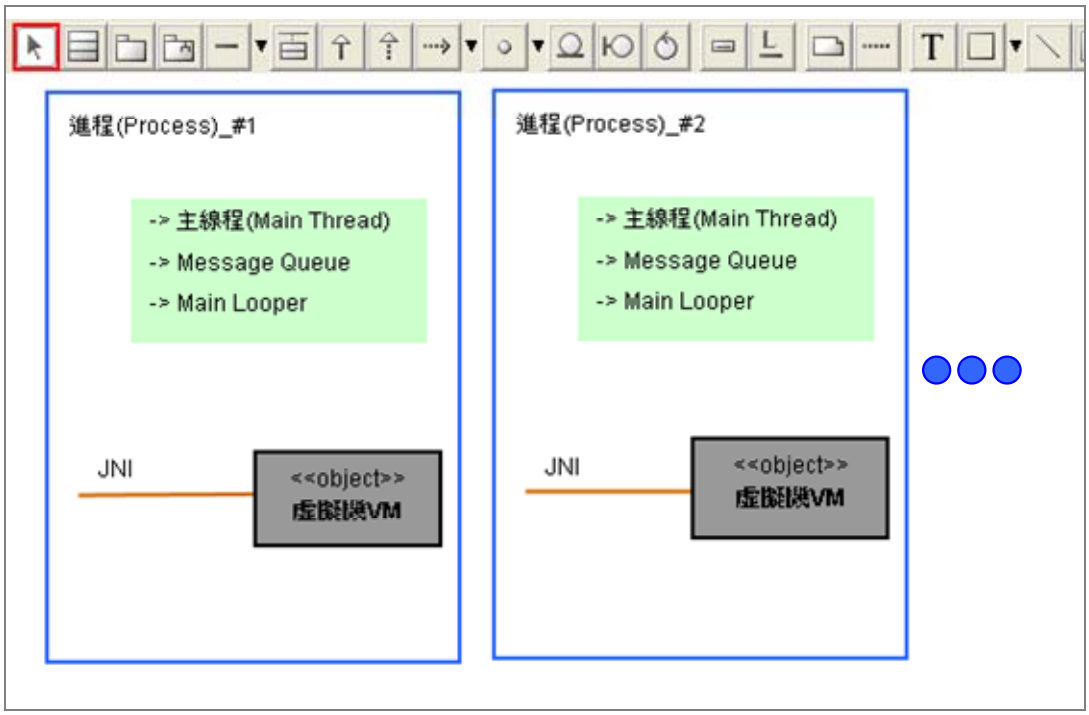

圖 16　Android 內部含有多個進程

主線程最主要的工作就是處理UI畫面的事件(Event)，每當UI事件發生時，Android框架會丟信息(Message)到MQ裡。主線程看到MQ有新的信息時，就取出信息，然後依據信息內容而去執行特定的函數。執行完畢，就再繼續執行Looper類，不斷地觀察MQ的動態。

大家都知道，當兩個類都在同一個進程裡執行時，兩者之間的溝通，只要採取一般的函數調用(Function Call)就行了，既快速又方便。一旦兩個類分別在不同的進程裡執行時，兩者之間的溝通，就不能採取一般的函數調用途徑了。只好採取 IPC 溝通途徑，如下圖：

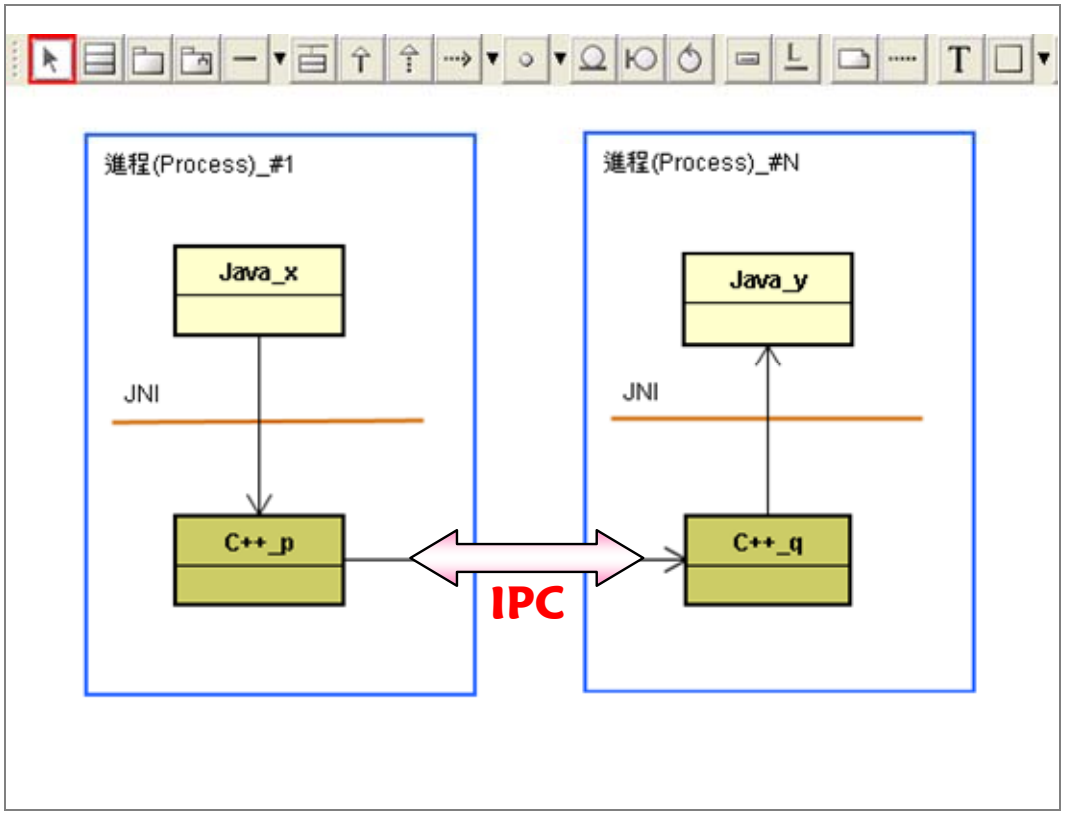

圖 17　Android 框架的 IPC 機制之例

Android 框架的 IPC 溝通仰賴單一的 IBinder 接口。此時 Client 端調用 IBinder 接口的 transact()函數，透過 IPC 機制而調用到遠方(Remote)的 onTransact()函數。例如下圖裡的 myActivity1、myActivity2 和 myService 分別在不同的進程裡執行，透過 C++層的 IBinder 接口來進行跨進程的 IPC 溝通。

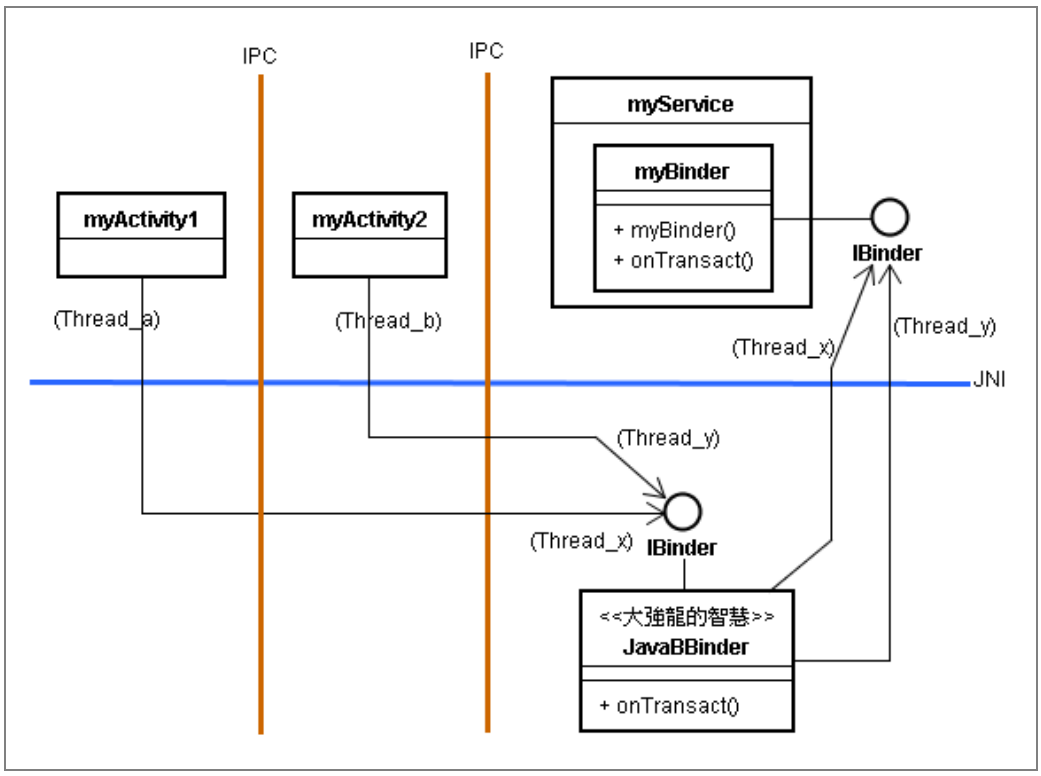

圖 18　Android IPC 機制的 IBinder 接口

在此圖的不同進程裡，各有其主線程(Thread)，這些線程可並行(Concurrent)執行，形成多線程(Multiple-Thread)的執行環境。例如上圖 18，myActivity1 和 myActivity2 並行執行，並透過 C++層的 JavaBBinder 類而共享(可能並行)了 Java 層的 myService 類之服務。◆

~~ END ~~