2010 Android 開發大會論文集

深入了解Android系統架構運作原理

深入研究Android應用存取雲端服務的最佳方案

於Android系統上的快速開機方法

Android 平台與系統晶片整合開發策略

Android 車載娛樂平台應用開發

Android編程之實現GPS定位

Using High-level View on Android Porting

Preparing for Connected TV, Google TV and Beyond

Android作業系統移植經驗分享

NTUSquare：結合雲端計算的Android 手機LBS服務平台

Jollen的HAL教學：深入淺出Android HAL 架構設計與實作

Android在i.MX5x平臺上的移植與優化

2010 Android平台社群開發大會

主辦單位：

協辦單位：

贊助單位：

活動網址：http://www.ctimes.com.tw/cf/2010-android.html

2010 Android 開發大會論文集

出版緣起：

智慧型手機在全球的銷售量飛快成長，而 Android 平台手機無疑是表現最亮麗的，為何 Android 在問世短短不到三年中，即能夠躍居智慧型手機的主流平台，原因在於它採行開放平台策略，提出易開發的框架平台，以及最重要地：獲得廣大開發社群的支持！

事實上，Android 的應用並不限於手機，新世代的平板電腦、Google TV/Conneted TV、車載資通娛樂設備等，無不積極往 Android 靠攏，讓 Android 的影響力不斷擴大。為凝聚產業與社群力量，並提供台灣開發者一個技術交流的機會，特別於 2010 年 10 月 21-22 日舉辦此次 Android 平台社群開發大會，共邀集近 20 位產、學、研各界專家出席發表專題，共襄盛舉，並有約二百位開發者熱情參與！

為讓此次大會的成果有更為深遠的影響力，除了專題演講外，同時也邀請講者提供相關的論文作品，並集結成為這本極具份量的論文集。未來此活動將會持續辦理，希望有更多開發者來共同參與，讓Android平台能在台灣這塊土地上開花結果!

CTimes編輯部

議程：

Day1 10月21日(四)

時間	主題	講師
09:30 - 10:10	Android Development on New ARM Architecture	ARM 應用工程師 Leon Chen
10:20 - 10:50	Android 平台與系統晶片整合開發策略	工研院資通所副所長謝明得博士
11:00 - 12:00	中國大陸Android市場與社群發展分析	北京易聯致遠 CTO 姚尚朗
13:30 - 14:10	中國高階 Android 智慧手機開發策略	中一無線架構師 Jollen Chen
14:10 - 14:50	Android 多媒體處理技術	Rockchip 總經理陳峰
15:10 - 15:50	Android 無線通訊模組技術	海華科技研發處黃忠誾處長
15:50 - 16:30	易於更新及維護的Android系統整合設計要點	台北科大資工系梁文耀助理教授
16:45 - 17:15	Android vs. iPhone開發環境與社群競賽	台大資工系陳彥仰助理教授

Day2 10月22日(五)

時間	主題	講師
10:00 -10:30	Preparing for Connected TV, Google TV and Beyond	MIPS策略行銷總監Kevin Kitagawa
09:30 -10:00	Android for eReaders and Smartbooks	Freescale亞太區資深經理蔣宏
10:45 -11:15	瞭解你的程式- Android平台之Java側寫工具	清大資工系系主任金仲達教授
11:15 -12:00	Android 與雲端運算之整合現況與趨勢	ysl的程式天堂主持人盧育聖
13:30 -14:10	Android 開放手機研發成果	中正大學資工羅習五助理教授
14:10 -14:50	Oxdroid 成果與發展藍圖	Oxlab Developer, Kanru
15:10 -15:50	Android 系統車載娛樂平台 技術開發	工研院南分院家網中心程永華
15:50 -16:30	Using high-level view on Android Porting	仕橙3G教室技術顧問鍾文昌
16:45 -17:25	Android應用架構的樣式導向設計	台大電機系王勝德教授

Contents

目錄

出版單位

CTimes

http://wwww.ctimes.com.tw

遠播資訊股份有限公司

台北市中山北路三段29號11樓

訂閱電話：

02-25855526分機225（發行部）

售價：新台幣 200元

元件化架構與元件間通訊機制

深入了解Android系統架構運作原理

■作者 梁文耀

本文中將介紹Android作業系統的整體架構，並透過Android元件化的架構與元件間的溝通與調用機制深入探討其設計理念與運作原理，包括Intent、Binder IPC、與JNI原生碼整合，並將提出一些值得讀者再深入探討的議題。

Android是一個開放式手持裝置軟體平台(Open Software Platform for Handheld Devices)，它是由Google及開放手機聯盟(Open Handset Alliance, OHA)所發展的一個基於Linux作業系統核心的行動裝置軟體平台。開放手機聯盟的初始成員包括Qualcomm, TI, Broadcom, Intel, Marvell, Nvidia, SiRF, WindRiver, T-Mobile, Sprint Nextel, China Mobile, KDDI, NTT DoCoMo, Motorola, Samsung, 及HTC等34家與硬體、軟體、及電信業者所組成。Android平台是由該機聯盟於2007年11月所公佈，同時提供軟體開發與模擬環境。Android的第一個正式版本於2008年10月發布。Android產品推出至今近兩年，這段期間，Android手機的銷售率或市佔率持續快速提升的消息不斷，足見此作業系統必定具有其獨特性並且存在其技術優勢。

從產品開發的角度來看，Android的特性包括開放源碼、免費、有自由軟體開發者社群的支持、與基本發展工具完備。Android是基於開放源碼的一個平台。然而，Google為了讓Android平台能被廣泛運用於手持式裝置的產品中，除了Linux Kernel的部份採用GPL (General Public License)的版權模式之外，作業系統核心以外的部份則捨一般Linux系統上以GNU計畫為基礎所發展的GPL開放源碼架構，而改採用以Apache與BSD為主的開放源碼版權模式，允許商業公司進行修改而不需將修改過的原始碼公開。如此可讓Android平台更容易商業化。

Android之所以受到矚目，除了較容易商業化之外，主要還因為它克服了過去Linux作業系統於手持式裝置較不易普及化的另一個重要因素 -- 分歧的使用者圖形介面與應用軟體開發方式；它使得許多Linux手持式裝置廠商與開發者，在面對各種「雖優良、但分歧」的應用程式開發環境時，無法肯定哪一種開發環境能確保其於該架構所投入的資金、人力、與時間成本皆可回收，因而變得躊躇不前，阻礙了Linux手持式裝置的發展。

由於Google在網路與軟體領域具有決定性的地位，因此，當Google提出了Android的手機軟體平台的時候，立刻吸引到各個軟硬體製造商與應用軟體開發者的注目。除了因為它是基於免費開放源碼的軟體平台卻適合商業化的因素之外，Android非常適合手持式裝置軟體運作的系統架構與跨平台的優勢，讓大家更對它抱以高度的期望。在這樣的“共識”之下，社群、產品製造商、軟體開發商、甚至個人工作室皆相繼投入資源研發相關軟體與產品，使得Android變成了基於

Linux的手持式裝置作業系統[1]中最閃亮的一顆星，甚至也有不少其它非手機的嵌入式系統產品製造商都開始考慮採用、甚至已經推出基於Android產品了。

Android作業系統架構

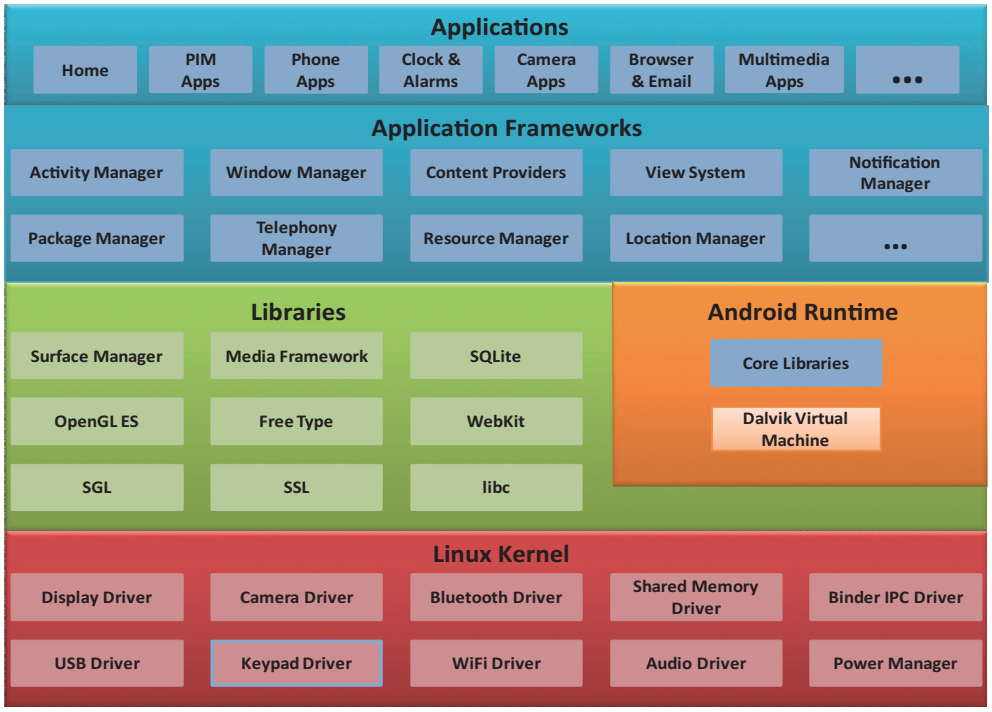
Android的架構如下圖一所示。

這個架構分成四個層次，由上至下分別為應用程式層(Application Layer)、應用程式框架層(Application Framework Layer)、函式庫層(Library Layer)、以及Linux核心層。Android的這個設計有下列六項優點：

- 應用程式高度跨平台可攜性
- 易於開發應用程式的元件化設計
- 使用者經驗一致性
- 適用於資源有限的手持式裝置
- 可兼顧效能
- 容易移植至不同硬體

Android的應用層與應用程式框架層是以Java程式來開發，主要原因即是考慮Java最大的優點：高度跨平台可攜性。由於Java是以虛擬機器為執行環境，因此可以不被實際硬體架構限制住，編譯好的程式碼可直接於不同平台上的Java虛擬機器執行。過去在嵌入式系統上由於效能的問題，因此採用Java為系統所使用的主要程式語言工具的情況並不多。但Android主要定位在中高階的手持式裝置，以目前系統晶片的發展來看，使用在這類應用的嵌入式微處理器效能與記憶體資源都足以支持Java的程式運行，因此Android採用Java是一可行的方案。

除了高移植性的優點之外，Android透過元件化的設計，讓程式開發變得非常容易。結合Java的跨平台特性，同時也讓



圖一 Android軟體架構圖[1]

應用程式在不同裝置上具備一致的使用者介面表現，達到了使用者經驗的一致性。關於Android系統的Java運行與其虛擬機器實現方式，我們將於下節進一步討論。

Android所提出的應用程式框架，除了讓程式開發者可以以堆積元件的方式快速建構應用程式之外，它的主要貢獻還包括為資源有限[2]的手持式裝置量身打造了合適的程式運行架構。不同於傳統Linux程式下標準C/C++程式運行方式，在Android上，程式的生命週期是由Android系統依照資源使用狀況與由程式執行狀態為基準所排定的重要性來決定。

也就是說，Android的應用程式的終止，是由Android系統來決定，而非由程式本身來決定。Android應用程式必須遵循應用程式框架所定義的執行流程來運行。在此運行過程中，程式可在適當的時期由應用程式框架得知程式本身的狀態，而能做出適當處置，讓程式不因突發的資源不足而突然當掉，避免重要資料突然遺失等嚴重問題的發生。

Android的軟體架構還包含函式庫層，讓有效能考量的程式(例如較密集的數學運算)能透過函式庫的原生碼(Native Code)來執行，克服Java使用虛擬機器以解譯(interpretation)的方式執行的先天限制，以兼顧其效能[3]。此外，透過函式庫與Linux核心這兩個層次，也可讓應用程式可間接操作硬體的資源；同時，也可讓應用程式和與硬體相關的程式區隔開來，使Android應用程式可很容易地移植到不同的平台之上。而在Android作業系統本身的移植過程中，在Linux核心與函式庫層移植完成後，應用程式層與應用程式框架層通常也可完整保留，直接於新平台上運行。

Java與Dalvik虛擬機器運作原理

Android的應用程式主要是以Java程式語言來撰寫，編譯成byte code，透過虛擬機器來執行，藉此達到讓應用程式具備高度可移植性的目的。Android支援標準的Java程式語言，以及Java 2 Standard Edition (J2SE)的標準核心類別函式庫(core class library)。但與傳統Java有一個很大的差異，即Android提供了自行開發的虛擬機器及執行檔格式；該虛擬機器稱為Dalvik VM，其所執行的檔案格式為Dalvik Executable (.dex)格式。Android使用Java的開發工具JDK來處理Java的程式編譯，產生的byte code (.class檔)再經由Android本身所提供一套工具“dx”將其轉為Dalvik VM所接受的.dex格式。

Dalvik執行檔針對記憶體的使用進行了最佳化，使其更適合運行於記憶體資源有限的嵌入式系統之中。而Dalvik VM則提供一個register-based的架構，使其與多數嵌入式系統所採用的精簡指令集架構(RISC)的微處理器更為貼近。相對於原本的JVM的堆疊式(Stack)架構，在Java程式執行解譯的過程中，Dalvik VM可讓.dex執行檔中不少的指令能直接對應到硬體架構，因此可以有較佳的執行效能。

Android的每一個應用程式均執行於不同的虛擬機器環境中，每一個虛擬機器則由獨立的Linux行程(process)所執行。這樣做的目的在於透過Linux以權限為基礎的安全模型(permission-based security model)來確保應用程式執行時的安全性。Dalvik的執行緒與低階記憶體管理亦是

透過Linux核心來達成。Android所採用的Linux核心版本[4]為2.6。稍後我們將再討論此執行架構與行程間通訊的關係。

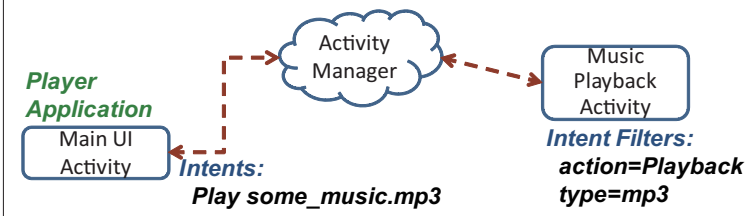
Android元件化程式開發與Intent

Android應用程式的基本運作單元稱為Activity，主要是負責實現使用者介面及處理使用者的要求。應用程式可以透過Android所提供的應用程式框架來使用系統所提供的元件或功能。例如，透過圖一中Application Framework的View System可使用圖形介面元件，如Lists、Grids、Text Boxes、及Buttons；而透過Telephony Manager可存取電話通訊的功能、透過Location Manager可取得GPS地理位置資訊等等。

一個應用程式可以包含一個或多個Activity。每一個Activity都可以視為一個應用程式中的一個獨立元件。Android允許應用程式透過Intent與Intent-filter機制來調用其他的Activity元件(或其它的Android元件)。Intent及Intent-Filter是Android作業系統所提供的一個獨特的元件調用機制，它是透過「意圖描述」的概念來達成元件之間的相互調用。

舉例來說，當一個應用程式想提供播放MP3音樂程式撥放的功能，它可以透過Intent表達『撥放一個MP3音樂檔』的意圖描述給Android作業系統。Android系統就可幫忙尋找系統中能夠滿足該意圖的其它元件，例如多媒體播放程式元件，然後啟動該元件完成音樂播放的工作。

在這個例子中，相對於應用程式透過Intent描述想達成的意圖，多媒體播放程式則是透過Intent-Filter這樣的「意圖篩選器」來告訴Android系統它所提供的元件可提供「播放MP3音樂檔」的功能。因此它可以滿足前例中「播放一個MP3音樂檔」的意圖，如下圖二所示。從另一個角度來看，Android系統提供了一個意圖要求者(提出Intent的程式)與功能提供者(提供Intent-Filter的元件)的媒合機制。



圖二 Activity間透過Intent及Intent-Filter來調用的方式

Activity可提供同一個應用程式的其他單元、甚至是其他應用程式來使用。事實上，Android允許程式開發者所開發的程式元件透過Intent/Intent-Filter機制被其他程式調用。為了讓作業系統能取得系統中所有程式元件的資訊，每個應用程式都必須定義其所實作的元件的相關資訊(例如名稱、屬性、權限、與Intent-Filter等等)於一個名為AndroidManifest.xml的檔案，於應用程式編譯時期，這個檔案將與程式碼一同包裹於Android的應用程式檔(Apk檔)中，在這些程式被安裝到裝置中時，Android系統即可從這些應用程式取得相關的資訊，作為執行時期查詢與媒合之用。

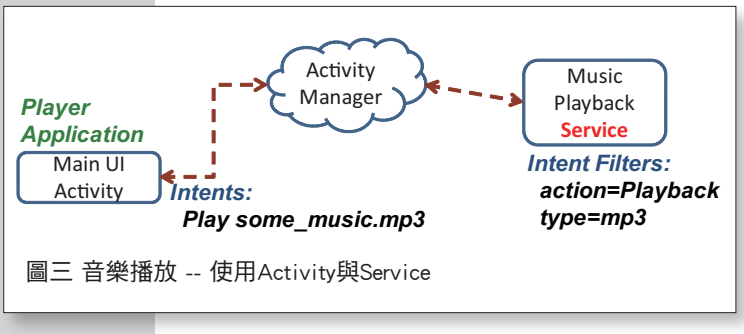
從另一個角度來看，Android程式元件的開發，讓應用程式開發者在撰寫程式的同時，也在無形中為所安裝的系統提供了新的元件。這樣的方式，進一步將傳統上透過函式庫使用物件導向設計及元件化概念來開發單一程式的過程，進一步延伸為「已開發完成的應用程式間也可以輕易

地調用彼此所實現的元件」，讓重複利用(Re-use)的概念更自然地被實現在一般程式開發者所撰寫的各種應用程式中。程式開發者可容易地引用其它程式的元件來實現所需的功能，而不須親自撰寫該元件或將該元件包含到程式當中。

應用程式框架也包含許多用來支援應用程式的服務。例如，圖一中的Location Manager可提供與位置相關的資訊與服務，Telephony Manager則提供與通訊相關的服務。Service與Activity的主要差別在於前者通常不提供使用者介面，而後者則提供使用者介面並負責處理使用者的要求。圖三的例子即表示一個應用程式的Activity可透過另一個以Service形式所實作的背景音樂播放程式來播放MP3音樂，允許使用者在切換到其它應用程式時，音樂可持續地於背景播放。

原生碼整合

在Android所提供的軟體架構中，由於Java的限制，與硬體相關或有效能考量的部份通常是透過原生函式庫(Native Library)或原生服務(Native Service)的方式來實現。函式庫的建置方式通常是以動態連結函式庫(dynamic link libraries)的方式來實作，方便程式碼於執行時期透過共享記憶體的方式讓使用該函式庫的各個程式共享該函式庫於記憶體中所佔據的空間，以減少系統中記憶體的浪費。



圖三 音樂播放 -- 使用Activity與Service

圖一中的綠色部分(由圖下方算起第二層)即代表Android架構中的函式庫層，其中包含了函式庫以及部分的原生服務行程。函式庫與原生服務主要是以C/C++的程式語言來撰寫。Java應用程式(及應用程式框架)與函式庫之間的呼叫則是透過Java Native Interface (JNI)介面來達成。以下圖四為例，與地理位置相關的應用程式(如Google Map)可以向Location Manager Service提出查詢位置資訊的請求，而Location Manager Service則可透過GpsLocationProvider以JNI間接呼叫方式與GPS硬體相關的原生函式庫“libpgs.so”，藉以取得地理位置資訊。

從圖四中我們也可以發現到從應用程式呼叫Service的服務是透過Binder IPC。Binder IPC是Android所提供的一個非常重要的行程間通訊機制(Inter-Process Communication)，主要是用來讓行程之間交換資料或是做為傳遞請求訊息及回應之用。在這個例子中，應用程式和Service是不同的行程，因此可以透過此行程間通訊機制來進行溝通。底下我們將對Android行程間通訊機制作一詳細說明。

程式安全與行程間通訊

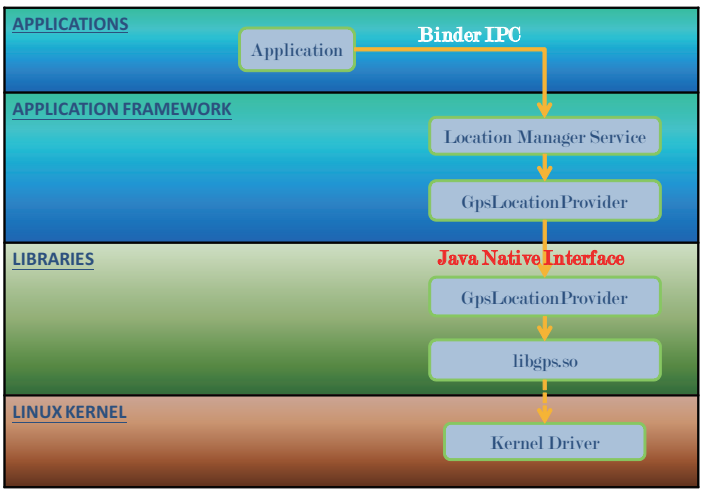
如同前面所提，為了安全的考量，「Android的每一個應用程式均執行於不同的虛擬機器環境中，而每一個虛擬機器則由獨立的Linux行程(process)所執行」，不同的Android應用程式在預設的情況下，實際上是執行於獨立的行程中。從作業系統的觀點來看，行程與行程之間

除了特殊情況所使用的共享記憶體分頁之外，邏輯上定址空間是完全區隔開來的，也就是各程式之間原則上無法直接存取彼此私有的資料。

除此之外，在Android中，每個應用程式在被安裝於系統中時，所給定的使用者編號(User ID)在預設情況下也是不同的，也就是說，應用程式無法藉由特定的系統呼叫或檔案擷取的方式，取得其他應用程式中的資料。透過如此的設計，Android讓每個應用程式無論是從定址空間或是存取權限來看，都是完完全全的區隔開來的，藉以達到保護的目的。

如同前面所提，Android應用程式的開發方式，讓程式設計者以元件化的方式來設計程式，不但能使用系統中其它程式所提供的元件，而且更進一步地，它也讓程式開發者所設計出來的程式元件可以提供給其他的應用程式來使用，充分運用重複利用(Re-use)的概念，讓程式的開發更加快速、容易。然而，如前段所述，由於每個應用程式被執行時，是運行於一個完全獨立的定址空間中，程式間無法直接存取對方的記憶體空間中的內容。因此，唯有透過行程間通訊(IPC)的方式能讓不同應用程式中的元件能夠互相溝通與調用。

Binder IPC是Android於Linux核心所新支援的一套行程間通訊機制。之所以在Linux本身所提供的行程間通訊機制之外另提出一套方法，主要的原因是在於為Android量身打造一套高效率且適合於Android應用程式間操作的行程間通訊子系統及介面。Binder IPC基本上是類似於分散式系統上訊息傳遞(Message Passing)的方法與遠端程序呼叫(Remote Procedure Call, RPC)的機制。透過一個名為Binder Interface的介面，它讓想要溝通的兩個行程(通常其中之一是以Service形式運行的元件)，由客戶(Client)端將訊息包裹，並且透過類似單純程序呼叫的方式，經由Binder介面傳送訊息至服務(Service)端，在服務端處理完該訊息所指定的工作後，再將結果打包，透過同一個Binder介面傳回客戶端。

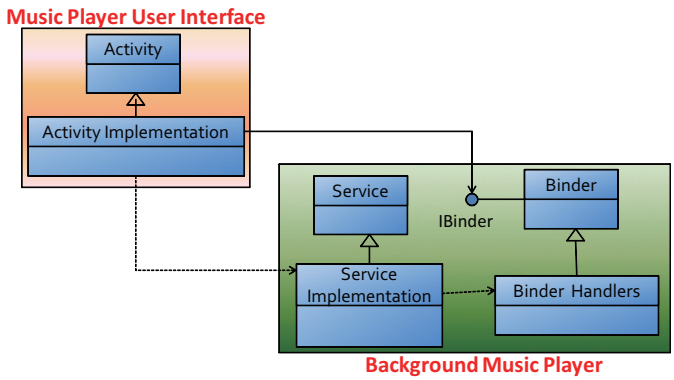


圖四 以JNI呼叫Library的範例[1]

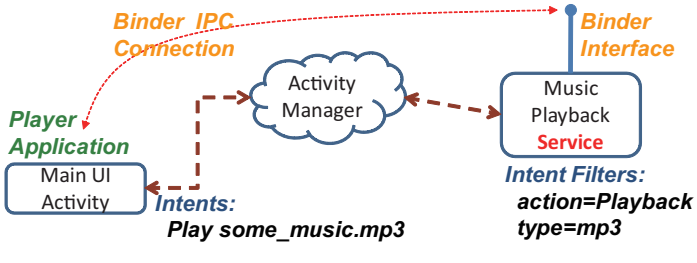
Binder IPC在Android系統中用得很廣，例如前面提到的Intent的傳遞、以及系統服務元件(如圖四的例子)的取得與調用。一般應用程式開發者也可能經常需要使用到Binder IPC[6]。例如前面所提的音樂播放程式(圖三)即是最典型的例子之一。在使用者透過以背景執行方式執行的音樂播放程式(Service元件)之後，該使用者可能接著執行其他的工作，例如處理電子郵件或瀏覽網頁。當使用者稍後想控制或調整音樂的播放內容時(例如暫停或選擇下一首歌曲)，就需要有一個方法可以讓使用者與於背景執行中的音樂播放Service元件聯繫溝通。Binder IPC便成為達到這

個目地的最佳方法。我們只要在Service中定義一個Binder介面，其他程式元件在與該Service建立連線之後即可透過該介面傳送指令或資料給於背景執行的Service，藉以控制該Service。此音樂播放程式範例的使用者介面元件與音樂播放服務元件及其Binder介面的物件關係如下圖五所示。

該圖左上方為音樂播放程式的使用者介面Activity元件。圖五右下方則為代表音樂播放程式的Service元件，該元件透過Binder類別實作了一個Binder介面(IBinder)。在Activity元件與Service元件建立連線時，該Binder介面資訊將透過Android框架傳遞給Activity元件。之後Activity將可透過該介面以Binder IPC的機制進行資料傳輸或控制。使用Binder介面溝通的音樂播放程式示意圖則如圖六所示。在Service端定義一個Binder介面，允許Activity透過該介面進行後續的通訊。



圖五 採用Binder介面的行程間通訊機制關係圖



圖六 使用Binder介面溝通的音樂播放程式示意圖

分層結構的優勢

由圖一的軟體架構圖與圖四的範例中我們可以看到Android的設計理念，也就是分層設計的方式。透過這個方式，可以將負責使用者介面的應用程式、應用服務(Service)、函式庫、以及硬體完全分開來，讓其中的每個組成元件都可以依照需求或對應的硬體來實作，增加應用程式的可移植性(Portability)與可重複利用性(Reusability)。例如應用程式可透過Intent的方式讓獨立建構的程式與其他程式元件相互調用，避免相同功能的元件重複製作。另一方面，它也允許程式開發者設計更適合的元件取代原有的元件。

以圖四為例，若所使用的GPS接收器不同，則只需要將驅動程式或是函式庫換成適用的驅動程式或函式庫即可。再以多媒體平台為例，若是所使用的平台運用了較佳的數位訊號處理器(DSP)，則只需要將DSP存取函式庫換成該處理器的函式庫即可，不需要改變多媒體框架(Media Framework)及應用程式，即可在不需變動應用程式的情況下，增加多媒體程式運作的效能。

為了讓程式開發者能較方便地開發驅動程式，同時解決核心驅動程式的GPL授權條款的問題，Android同時設計了一個在存在於使用者空間(user space)的驅動程式硬體抽象層(Hardware Abstraction Layer, HAL)。透過了這個方式，讓原本需要在核心空間(kernel space)運作的驅動程式有部分得以運行於使用者空間(user space)中，