介紹 Android 的核心服務

- **Core Service 是系統層的「前裝型」服務**
  - 大多以 C++類實現，有些以 Java 類實現。
  - 可透過 ServiceManager 來建立和綁定(Bind)核心服務。
  - 綁定後，可透過 IBinder 接口去執行其函數。

- **SDK-Service 是應用層的「後裝型」服務**
  - 以 Java 層的 Service 的應用子類實現。所有的 SDK-Service 都是由 ActivityManagerService 核心服務所掌管。
  - 在應用層(如 Activity)可調用 Android 框架裡 Context 類的 startService() 和 bindService()函數去綁定 SDK-Service。
  - 綁定後，可透過 IBinder 接口去執行其函數。

1. 核心服務的特性

- 核心服務通常在獨立進程(Process)裡執行。
- 必須提供IBinder接口，讓AP進行跨進程的綁定(Binding)和調用。
- 因爲共用，所以必須確保多線裎安全(Thread-safe)
- 以C++類定義，誕生其對象，請ServiceManager(簡稱SM)將該對象參考值加入到Binder Driver裡。
- AP可請SM協助而遠距綁定某核心服務，此時SM會傳IBinder接口給AP。
- AP可透過IBinder::transact()函數來與核心服務互傳訊息。
- 在開機階段執行INIT.RC時，就會啓動各主要的核心服務，例如CameraService、MediaPlayerService等等。
- 核心服務啓動在先，應用程序啓動在後。
- 核心服務大多以C++撰寫；也可以用Java撰寫(如AudioService)。
- 核心服務與SKD-Service不同，SDK-Service不會加入到Binder Driver裡。

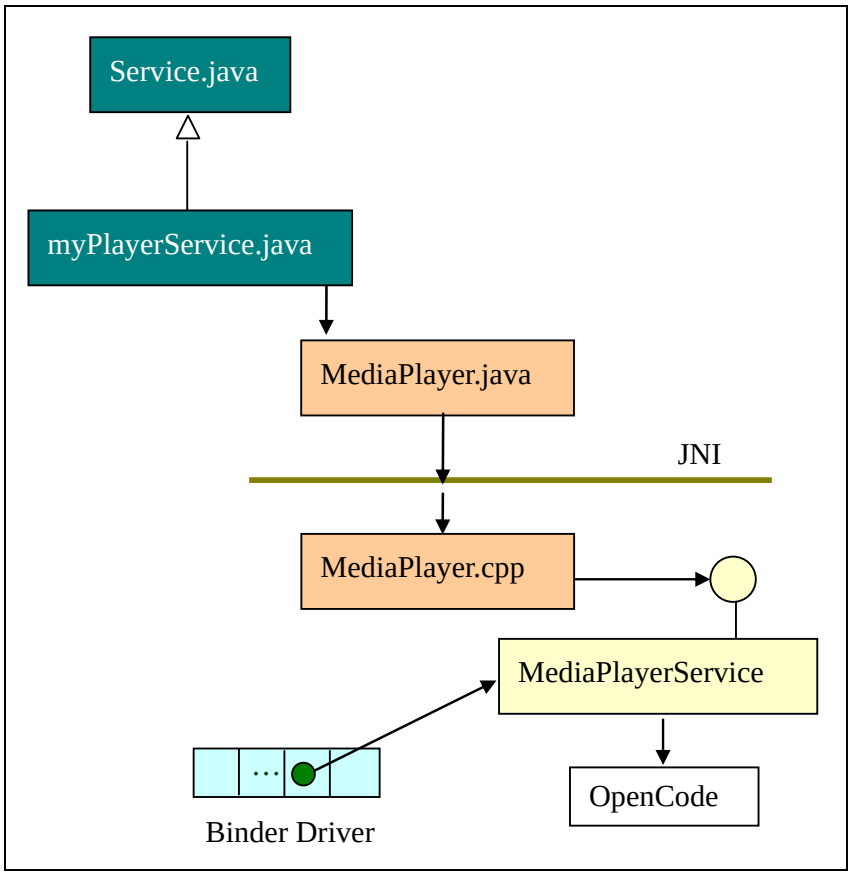
2. 認識 Android 的兩種服務

-- 以 MediaPlayer 服務爲例

2.1 兩種服務的區別

在 Android 平台，上層應用程序的 SDK-Service 與底層的核心 Service 有很大的

區別，只是許多人都將它們混在一起了。例如，MediaPlayer 範例，許多人都知道其結構爲：

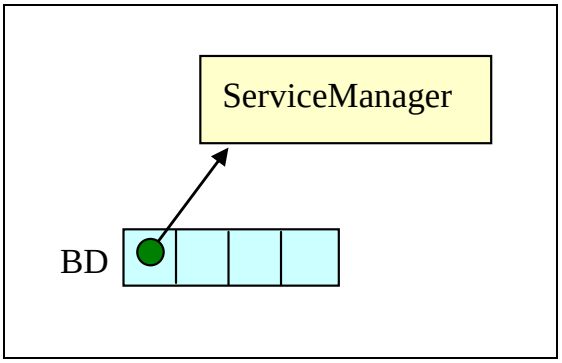


其中有兩種 Service：

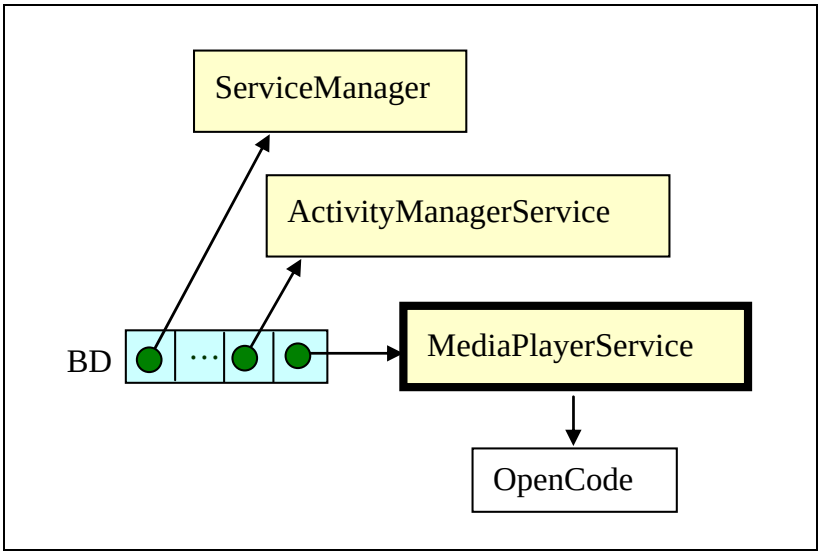
- 應用程序開發者所寫的 myPlayerService.java 類。這是 SDK service。
- Android 已經提供的 MediaPlayerService.cpp 類。這是核心 Service。

主要的核心 Service 都是在 Android 系統啓動時，就會先逐一登記到 Binder Driver(簡稱 BD)裡，隨時準備爲 SDK 應用程序進行服務。

天字第一號的核心 Service 就是 ServiceManager(簡稱 SM)。當 Android 系統啓動時，就會優先將之登記到 Binder Driver 裡，如下圖：



讓其他模塊(AP 或其他服務)能透過 IBinder 接口(可轉型爲 IServiceManager 接口)而遠距調用到它(即 SM)。於是陸續會有更多的核心 Service 調用 SM 的 addService()函數去登錄到 BD 裡。例如，ActivityManagerService 也是透過 SM 而登錄爲另一個核心服務。再如，MediaPlayerService 也繼續透過 SM 去登錄爲核心 Service。



當 Android 系統啓動完成(主要核心服務也啓動完成)之後，就可以啓動及執行應用程序了。執行應用程序時，各 SDK-Service 都由 ActivityManagerService 來掌管。於是應用程序裏的 Activity 等類可以使用 bindService()來綁定(Bind)到 myPlayerService，然後透過 IBinder 接口而遠端調用到 myPlayerService。之後，myPlayerService 再透過 JNI 而調用到 MediaPlayerService，進而調用到 OpenCode 組件了。

2.2 兩層服務的對應關係

Java 層的 Binder 類(即 Binder.java)是 Android 框架裡的重要元素，它提供了 IBinder 接口，讓 Activity 等元件能透過 IBinder 接口而遠端調用 SDK-Service。對於 Binder 類，我們可以從不同角度來看它所扮演的角色。

從 SDK-Service 來看，Binder 就扮演 Service 的接口對象角色。當 Activity 類想調用遠端的 SDK-Service 時，可要求在 SDK-Service 啓動時就去誕生一個 Binder 類的對象，由它提供 IBinder 接口給 Activity 元件，就搭起 Activity 與 SDK-Service 元件之間的溝通橋樑了。

從 IPC(Inter-Process Communication)角度來看，上述的溝通橋樑是跨進程的 IPC 機制。Binder 會透過 JNI 接口與本地(Native)的底層 Binder System 溝通合作。在 Activity 和 SDK-Service 所在的進程之間建立 Proxy/Stub 機制，以完成 Activity 與 SDK-Service 之間的連結(Connection)。一旦完成連結之後，Activity 就能透過 IBinder

接口而調用 SDK-Service 了，Proxy/Stub 機制還會進行跨進程的 Marshalling 動作，協助 Activity 與 SDK-Service 之間跨進程的資料交換。

從框架(Framework)的角度來看，Binder.java 是 C++層 BBinder.cpp 類在 Java 層的一個分身而已，它以父類(Superclass)形式出現，提供一個主動型 API 來調用 Java 層應用子類(如 myBinder.java)，協助底層的 Binder System(設計在先)能取得應用子類(設計在後)的客制化部分的資訊。例如有了 Binder.java 的協助，Binder System 得以從調用 Service 的 onBind()函數而得知如何調用 myBinder.java 所客制化的 onTransact()函數，如此讓 Android 框架幕後的 Binder System 能與目前的 myBinder.java 應用類結合起來，而達到框架的目的了。

2.3 細說 SDK-Service

表面上，Java 層的 myActivity 類調用 myBinder 的 IBinder 接口的 transact()函數及 onTransact()函數。在實踐機制裏，則是 myActivity 透過底層 Binder System 而返向調用 myBinder 的 execTransact()及 onTransact()函數。由於 myActivity 與 myService 之調用是透過底層 Binder System 來達成的，它們之間進行資料交換時，Binder System 就有機會進行 Marshalling 動作，而達成 IPC 溝通了。請看下述程序碼：

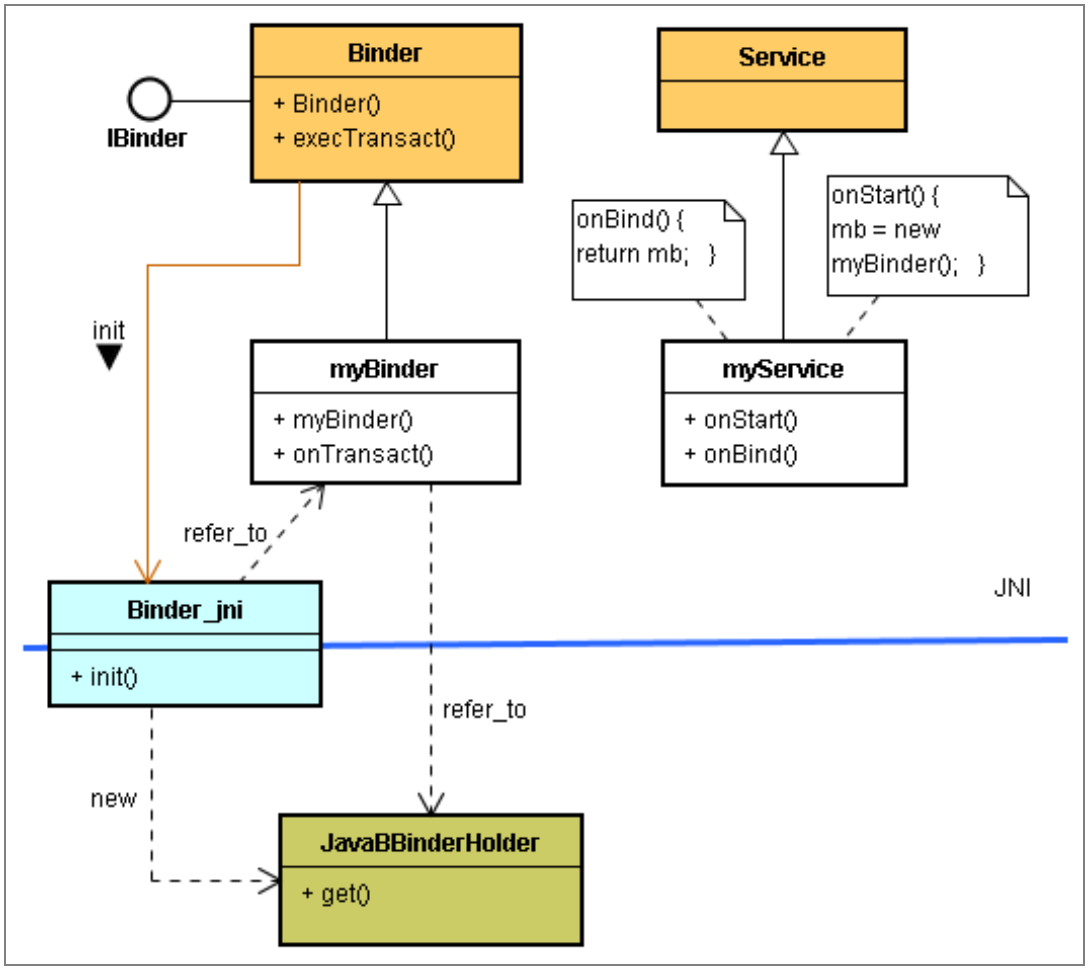
```
// myActivity.java
public class myActivity extends Activity implements OnClickListener {
        ……..
        public void onCreate(Bundle icicle) {
                ……..
                startService(new Intent("com.misoo.pk01.REMOTE_SERVICE"));
                ……..
        }
}}
```

```
// myService.java
public class myService extends Service {
        private IBinder mb = null;
        ……..
        @Override public void onStart() {
                mb = new myBinder();
        }
}
```

當 myActivity 調用 startService()時，就調用 Service.onStart()函數，執行到指令：

mb = new myBinder()

接著，調用 myBinder()建構式(Constructor)；進而調用父類 Binder()建構式，轉而調用 Native 的 init()函數。如下圖：



繼續看下述的程序碼：

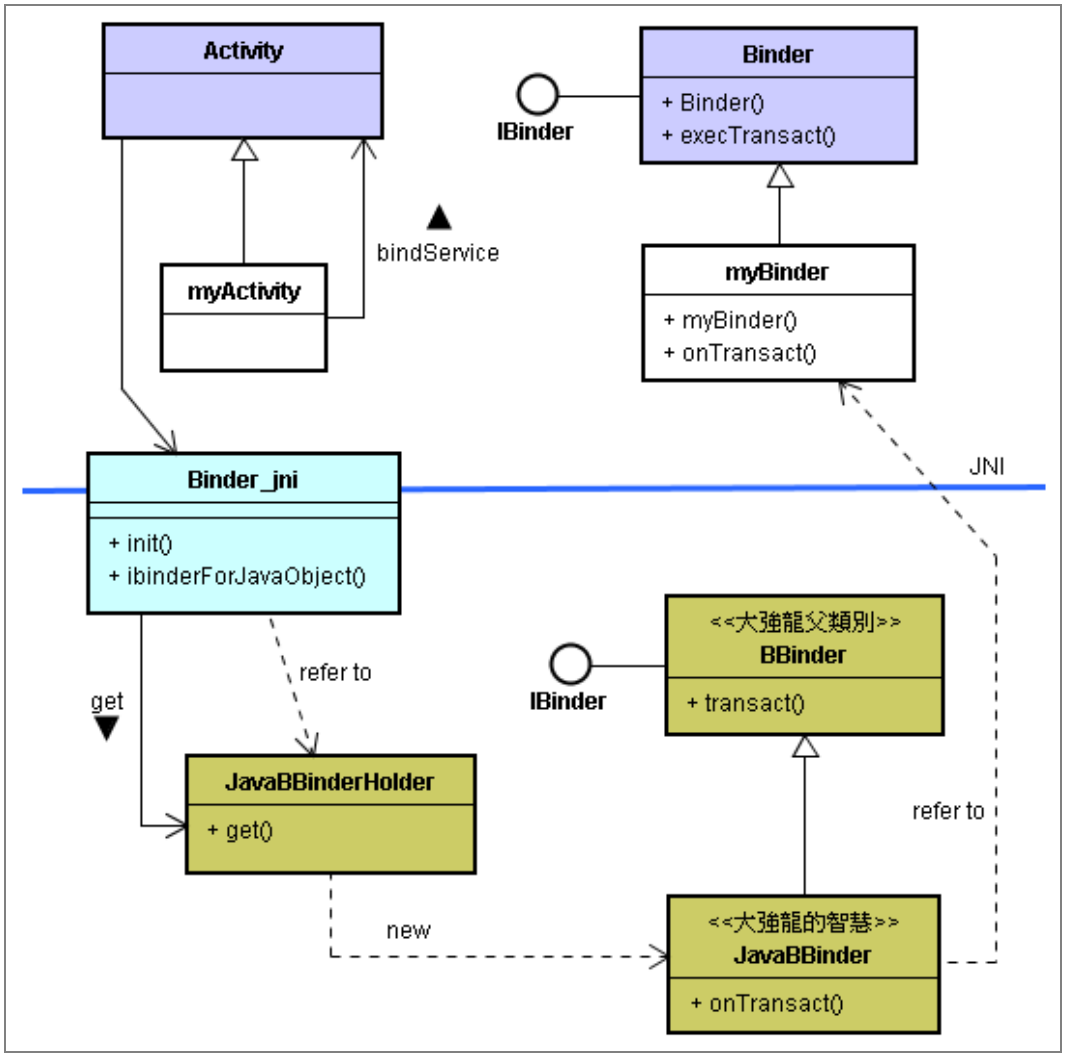
```
// myActivity.java
public class myActivity extends Activity implements OnClickListener {
        IBinder mb = null;
        ……..
        public void onCreate(Bundle icicle) {
                ……..
                startService(new Intent("com.misoo.pk01.REMOTE_SERVICE"));
                bindService(new Intent("com.misoo.pk01.REMOTE_SERVICE"),
                                mConnection, Context.BIND_AUTO_CREATE);
                ……..
        }
        private ServiceConnection mConnection = new ServiceConnection() {
                public void onServiceConnected(ComponentName className, IBinder
                                ibinder) {
                        mb = ibinder;
                 }
        };
        …….
        public void onClick(View v) {
                ……
                mb.transact(…);
                ……
```

```
        }
}}
```

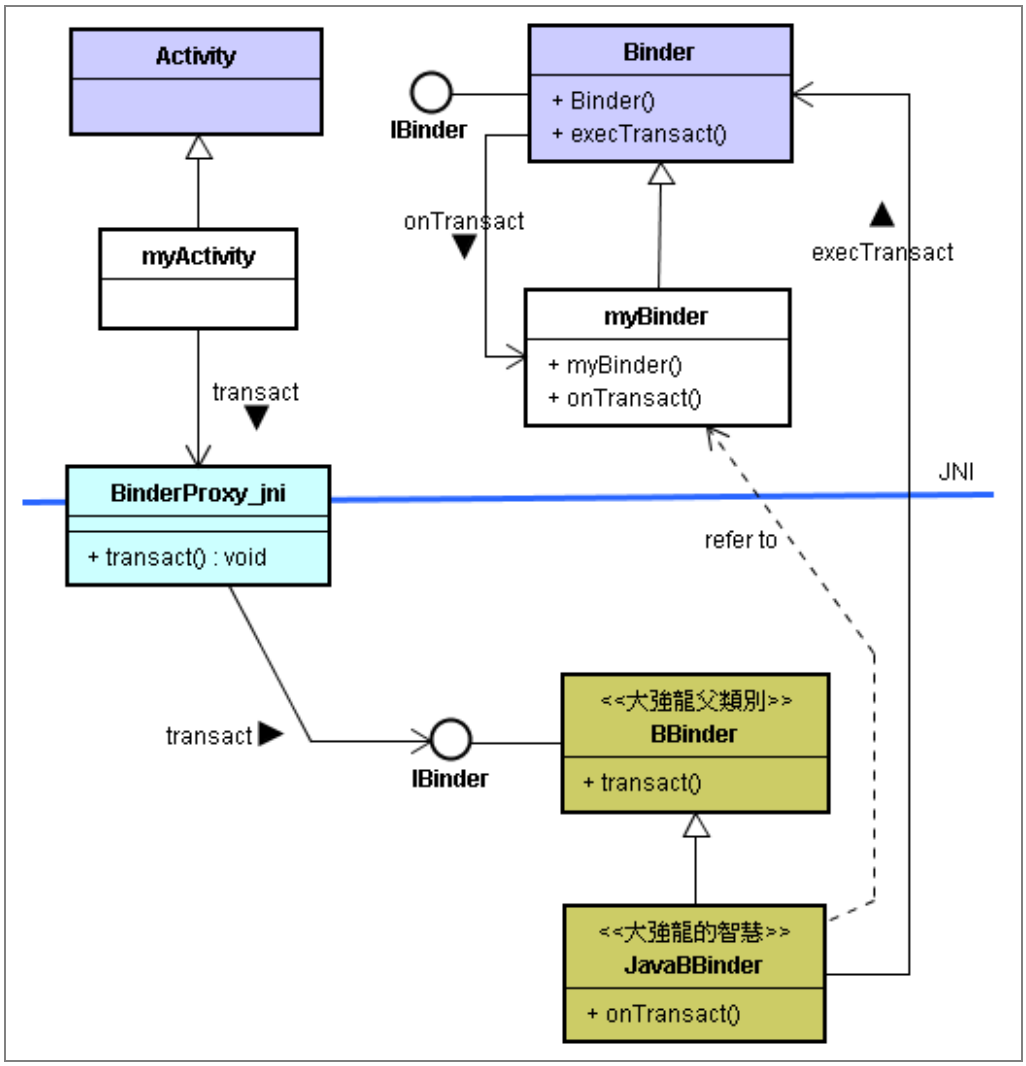
// myService.java

```
public class myService extends Service {
        private IBinder mb = null;
        ……..
        @Override public void onStart() {
                mb = new myBinder();
        }
        @Override public IBinder onBind(Intent intent) {
                return mb;
         }
}
```

當 myActivity 繼續調用 bindService()時，如果找到該服務，且它尚未被任何 Client 所綁定的話，就會調用 myService 的 onBind()函數。此時，執行到指令：return mb; 也就建立出一個連結(Connection)，如下圖：



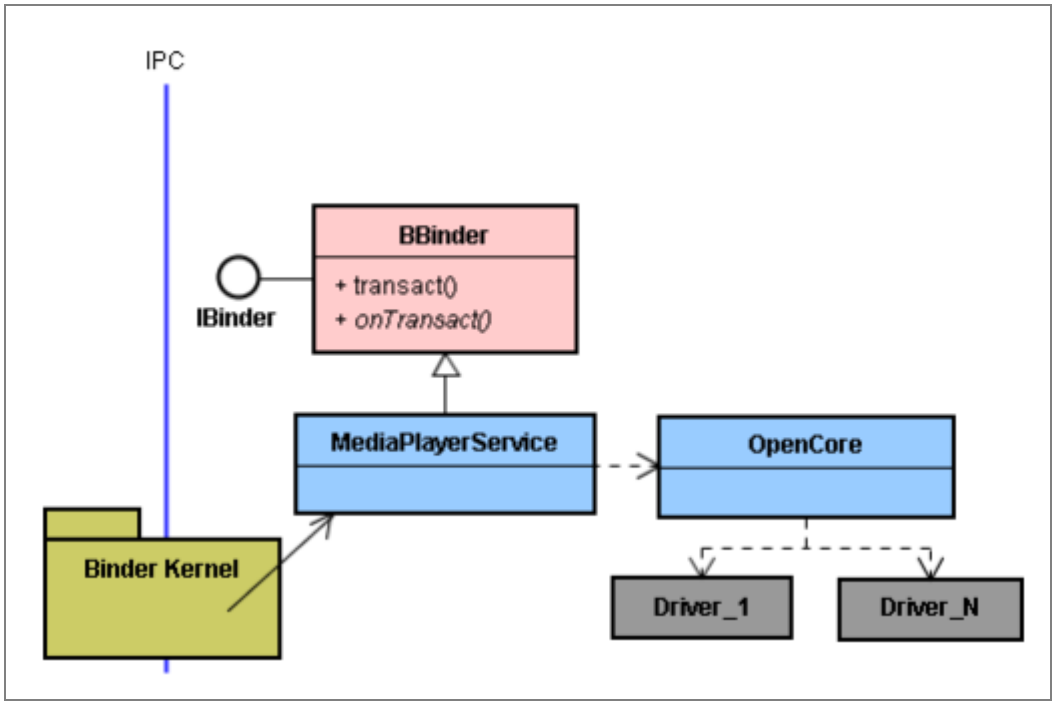
Activity 透過 Connection 而調用到 myBinder 裏的 onTransact()函數。



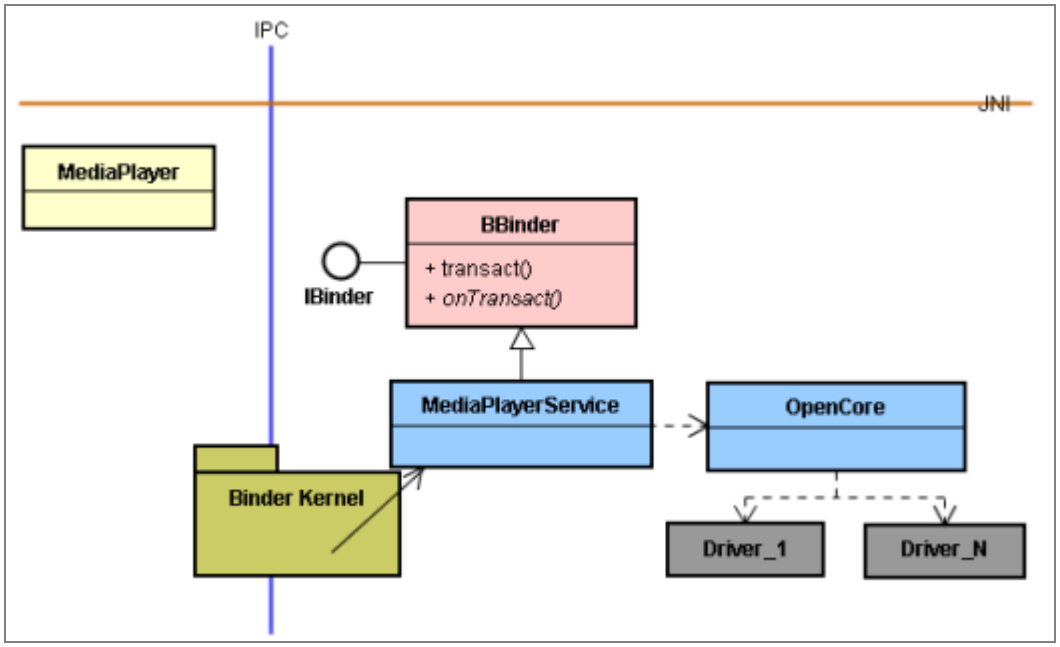
我們可以將 C++的 BBinder 和 JavaBBinder 類視爲引擎；將 Java 層的 Binder 父類視爲輪盤；並且將 myBinder 子類視爲輪胎(或者將 Binder/myBinder 視爲輪盤，並且將 myService 子類視爲輪胎)。從上述汽車之比喻，可更清楚理解 Android 架構之重要意義：實現強龍與地頭蛇分工合作的美好商業模式。在此商業模式裏，地頭蛇的獲利空間很大，願意盡力服務客戶，大力維護強龍的利益。

2.4 細說核心 Service：以 MediaPlayer 爲列

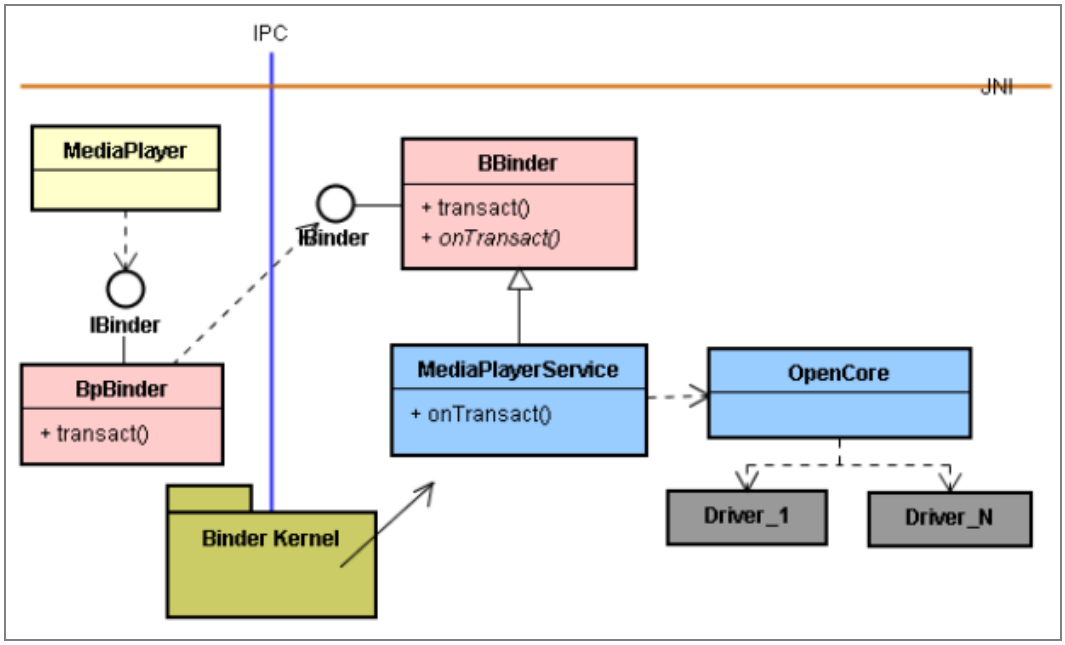
在 C++層實現核心服務時，可繼承 BBinder 父類而擁有 IBinder 接口；然後透過 ServiceManager 而登錄到 Binder Kernel 裡，如下圖：



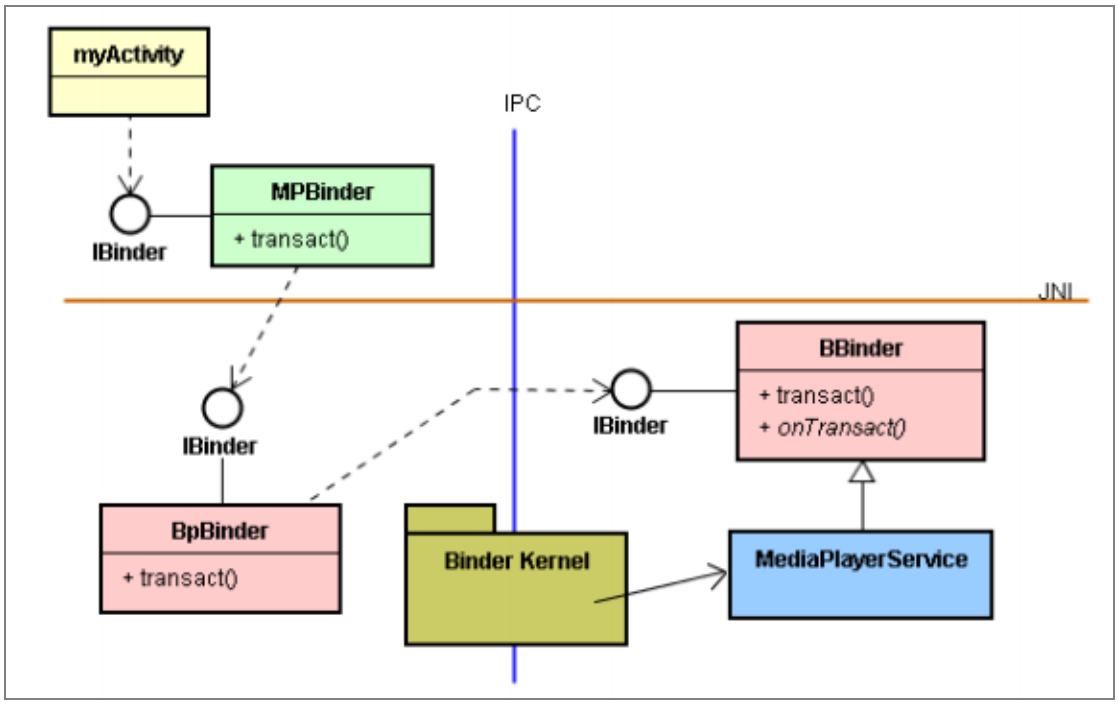
由於核心服務通常執行於獨立的進程裡，所以它的 Client 模塊(如下圖的 MediaPlayer.cpp 類)通常在另一個進程裡，與 MediaPlayerService 之間是跨進程的，如下圖：



C++層的 Client 模塊(如下圖的 MediaPlayer.cpp 類)就透過 ServiceManager 來綁定核心服務，如下圖：

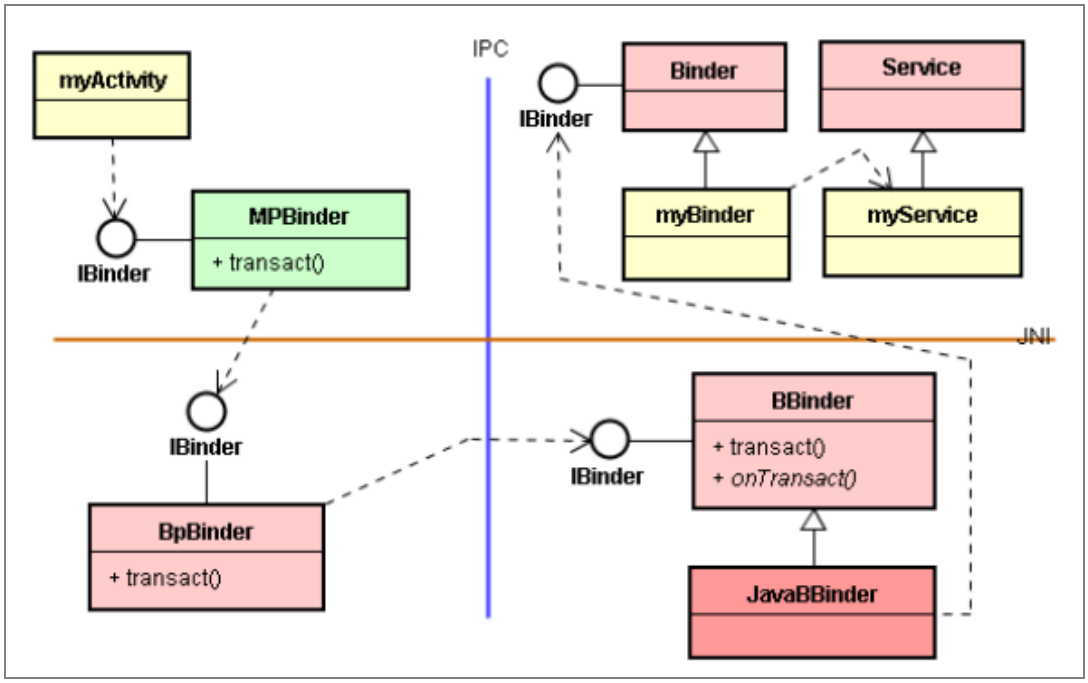


Java 層的 Client 模塊(如下圖的 myActivity.java 類)也能透過 ServiceManager 來綁定核心服務，如下圖：

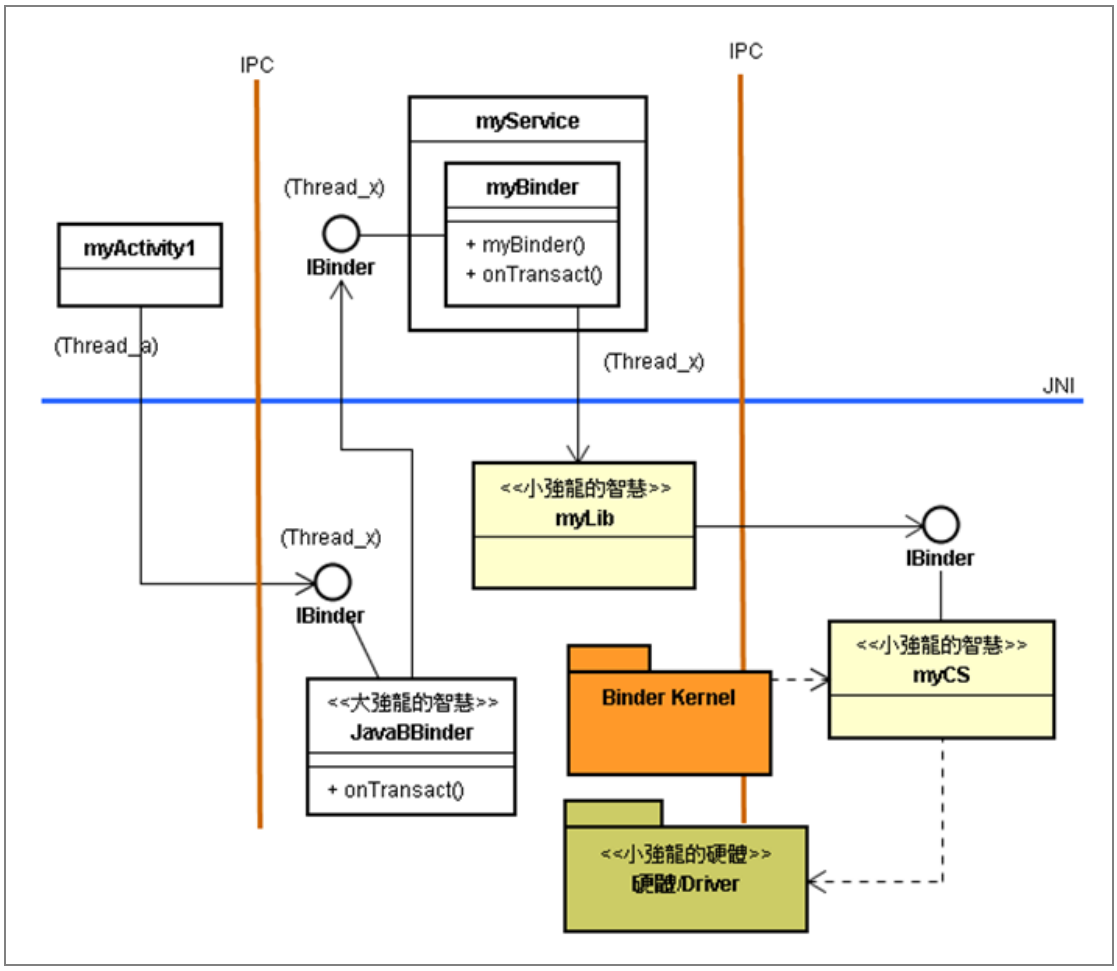


2.5 兩種服務幕後的 BBinder 類角色

同樣的 C++層機制，也能綁定 Java 層的 SDK-Service。如下圖：



此外，兩種服務又能進行更多樣化的組合，如下圖：



核心服務非常接近驅動程序和硬件，能有效凸顯硬體設備的差異化功能和性能。

~~ END ~~