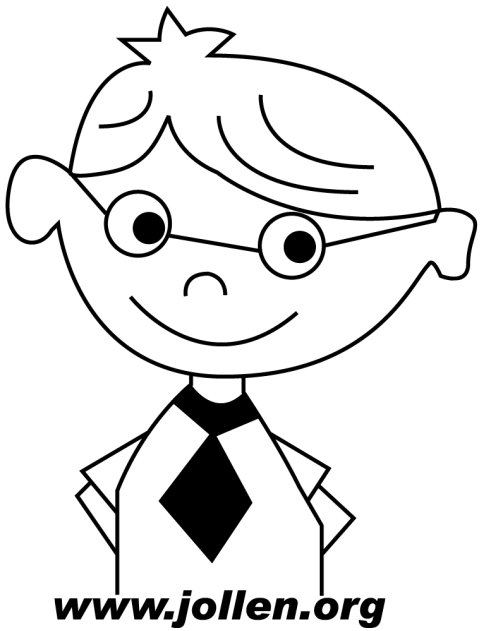


主講：Jollen Chen <jollen@jollen.org>
Blog：http://www.jollen.org/blog
筆記：http://www.jollen.org/wiki
課程：http://www.jollen.org/consulting

課程日期：2010/3/29 (一)
課程時間：09:00~16:30、共6小時

Jollen 的 Android Framework & HAL 軟硬整合培訓班 《2010.03 深圳班》

關於仕橙3G教室

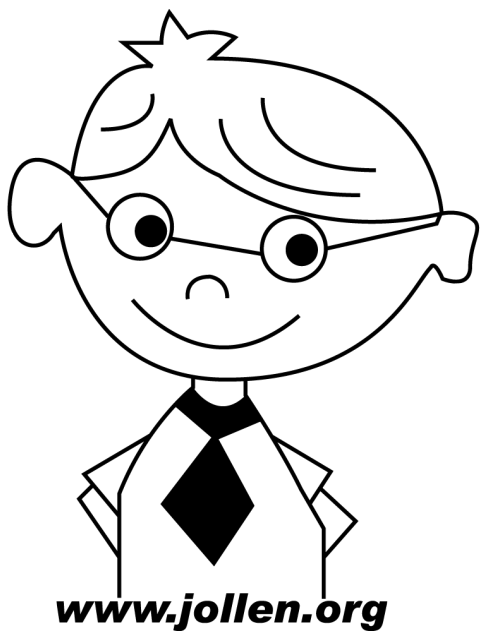
仕橙研策科技 (Moko365 Inc) 創立於2009年，開辦「仕橙3G教室」，初立初期以Android作業系統以及「行動通訊軟體」的培訓為主。服務對象為台灣、大陸與美國的企業，提供專業Android與嵌入式Linux的內訓課程，以及通訊方面的技術咨詢服務。

仕橙3G教室將致力於建立可信賴的「行動通訊技術服務品牌」。

關於Jollen's Consulting

由 Jollen Chen 所成立的個人工作室，主要進行高品質的課程研發，以及技術研究。由 Jollen 所精心規劃的 Embedded Linux 課程是台灣最資深的嵌入式 Linux 課程，2003-2009年與「財團法人自強工業科學基金會」合作，在新竹地區培訓大量的嵌入式 Linux 人才，為台灣嵌入式 Linux 技術與人才培養做出貢獻。多年來的課程與技術研究深獲肯定，服務客戶遍及台灣、美國與大陸企業。

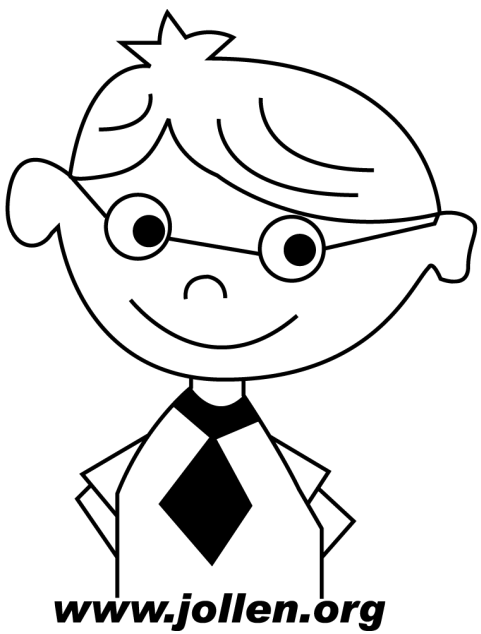
2009 年，Jollen's Consulting 與大陸地區業者合作，提供大陸地區嵌入式 Linux 與 Android 的培訓服務，巡迴上海、北京與深圳等地。Jollen 的嵌入式 Linux 與 Android 課程專業性受到業界肯定；由 2009 年初開始至今，提供台灣、大陸與美國公司內訓服務，總計超過 25 個場次。



關於本課程

「Jollen 的 Android Framework & HAL 軟硬整合培訓班」是台灣唯一的 Android 框架與底層課程。本課程由 Jollen's Consulting 精心規劃，至今已為大陸與台灣地區多家企業供培訓與咨詢服務。

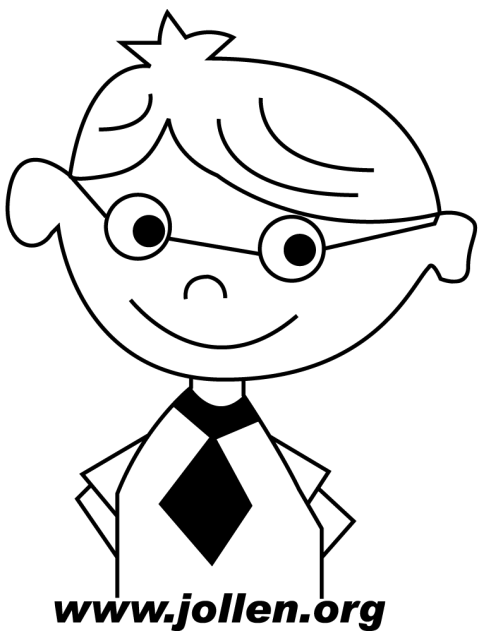
本課程理論與實務兼具，透過簡單的範例與程式碼討論，以至簡方式呈現，因此受到許多技術人員的歡迎。本課程目前是台灣業界唯一指名課程。



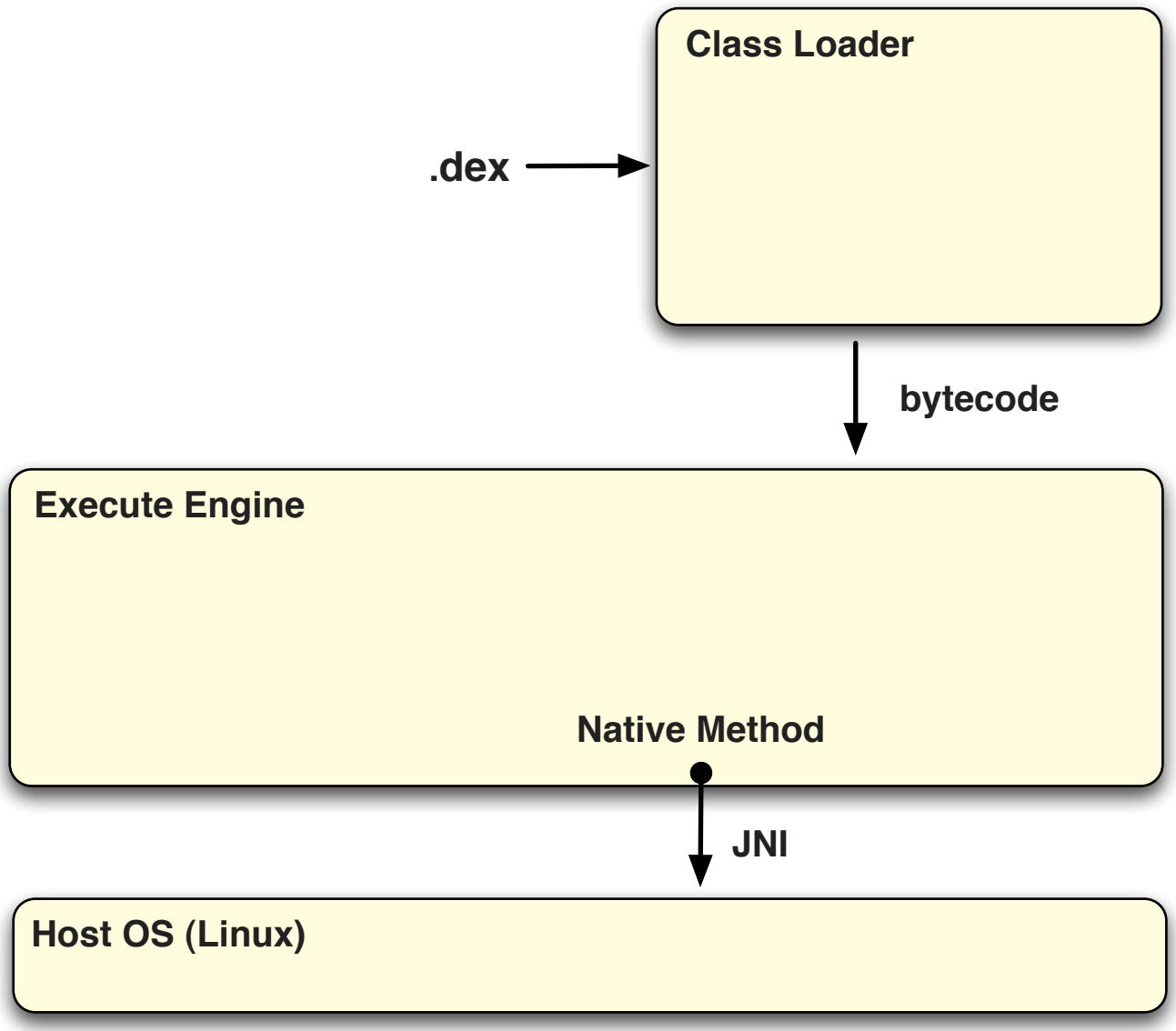
第06堂課： Android Service

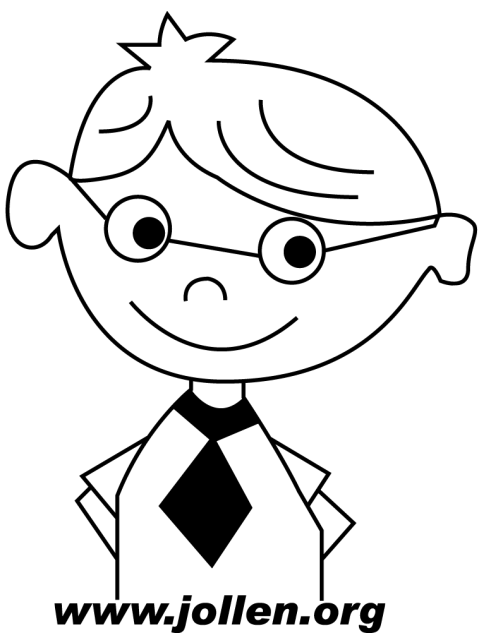
本堂主題

第六堂	Android Service
6.1	Android Process 模式
6.2	Component 與 Main Thread
6.3	SystemServer 介紹
6.4	ServiceManager介紹
6.5	專題討論：SensorManager與SensorService實例



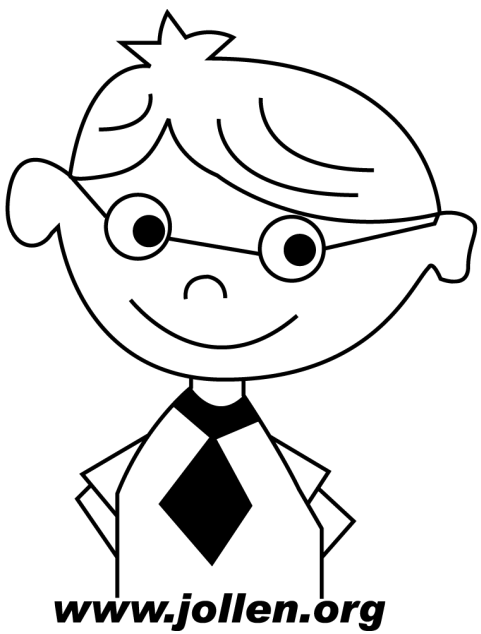
Dalvik VM Block Diagram





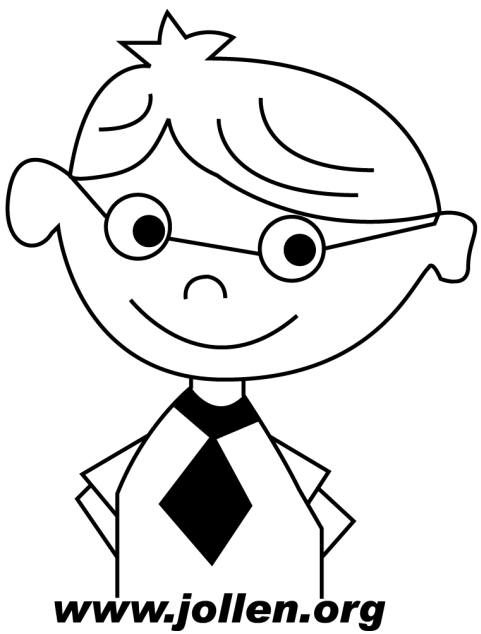
Android Service

- Android Service 與 android.app.Service 是不同的概念
  - Android Service 又稱為 System Service、或 Server
  - android.app.Service 又稱為 Application Service
- Android Service：與硬體溝通的橋樑



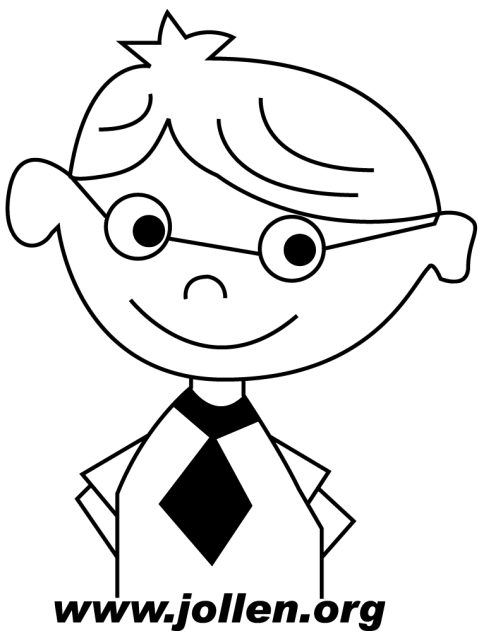
Android Process

- Process 在作業系統的定義上，指的是「執行中的程式」，在 Android 的應用程式模式中，代表的是低階的執行程式，也就是系統層（kernel）的部份
- Android SDK的定義
  - 同一個APK裡的所有Component在同一個process裡執行



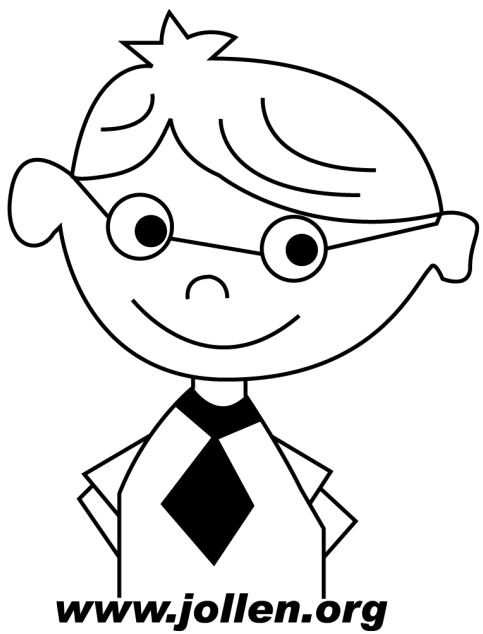
主要 Component

- 撰寫應用程式的四大 component
  - Activity
  - Service
  - BroadcastReceiver
  - Provider



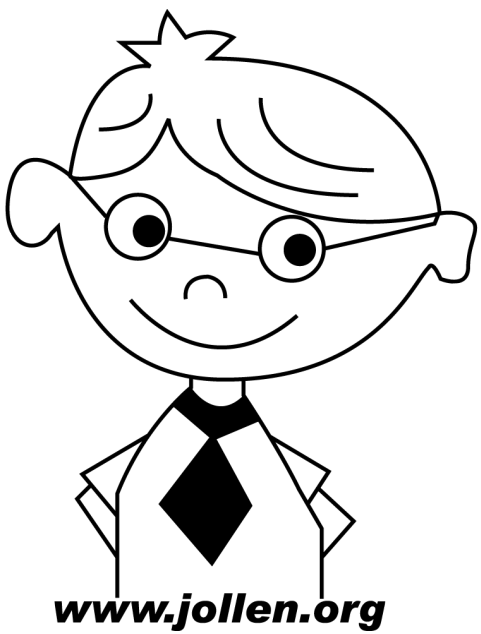
Dalvik VM

- Android 的 Java 虛擬機
- 一個 process 一台 VM
- Zygote 管理所有 VM process
  - Zygote 為所有 VM 的 parent process
  - 如同 Monitor process 的角色
- VM process 可平行執行
- Process 可共享 loaded class

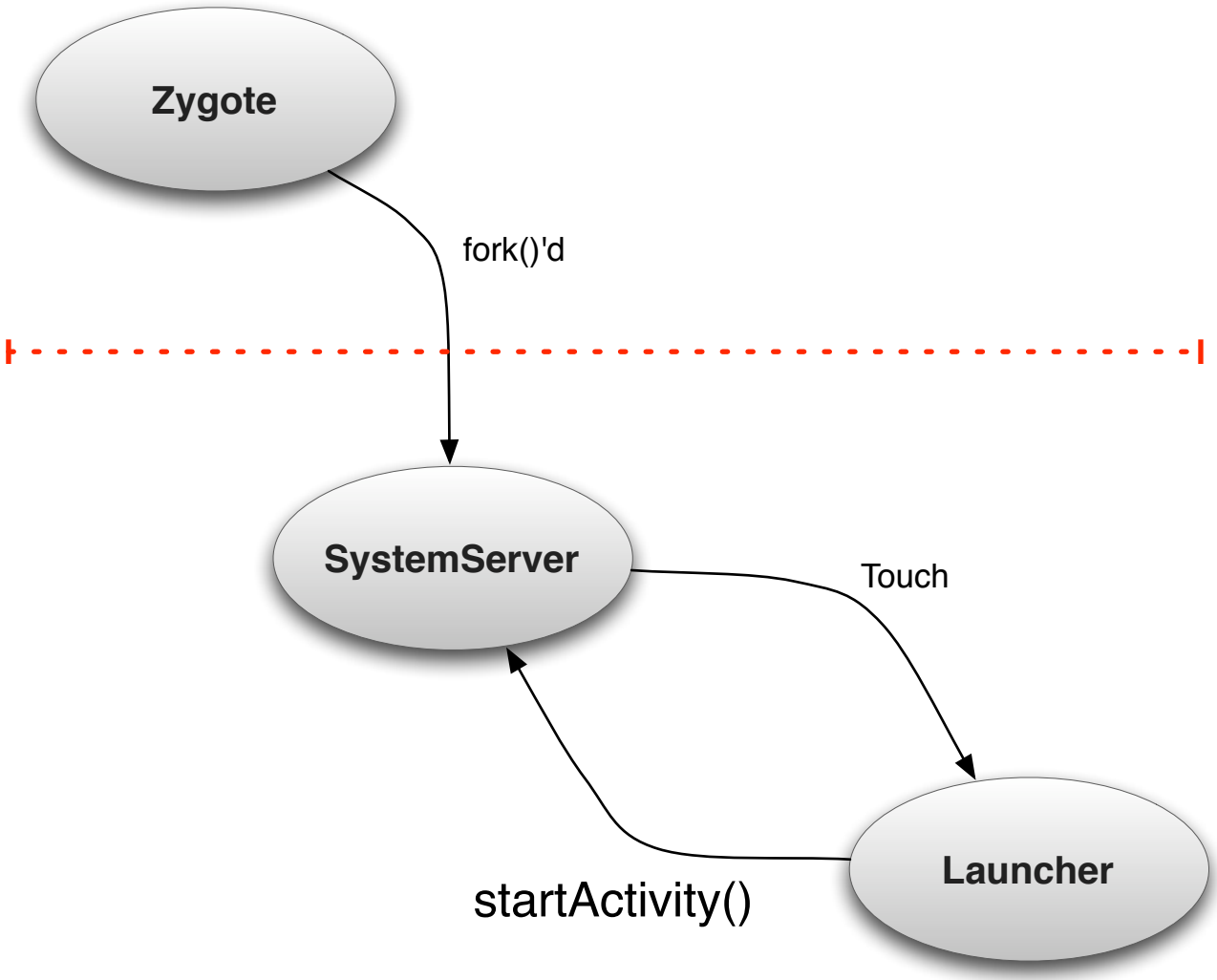


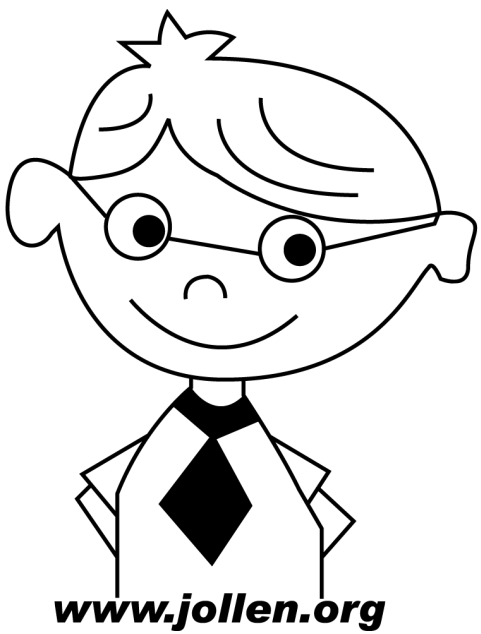
Zygote

- Android的主程式
- Android作業系統由Zygote做為起點
- zygote process會監聽socket並等待命令。依據命令的不同，來fork child process並執行

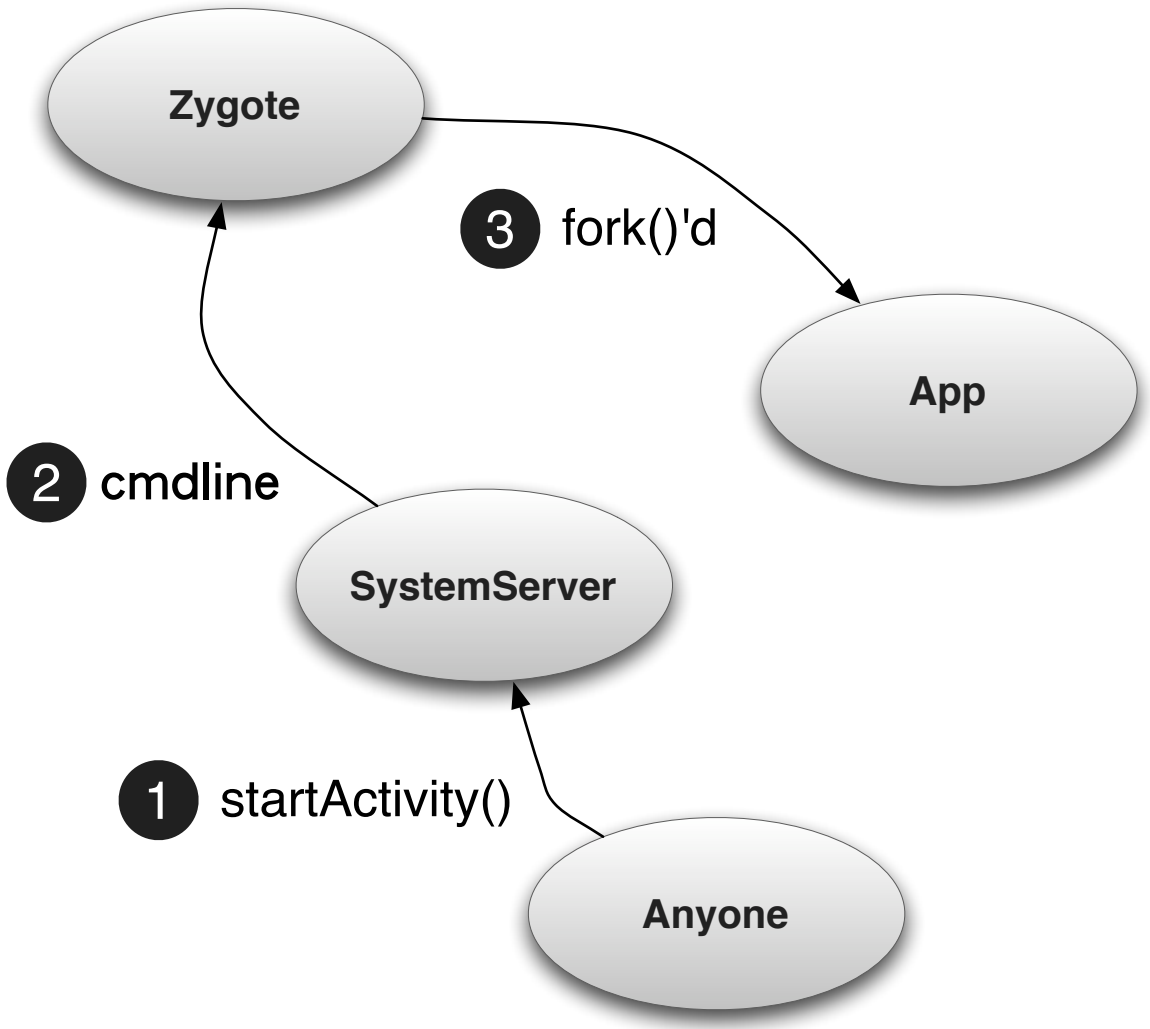


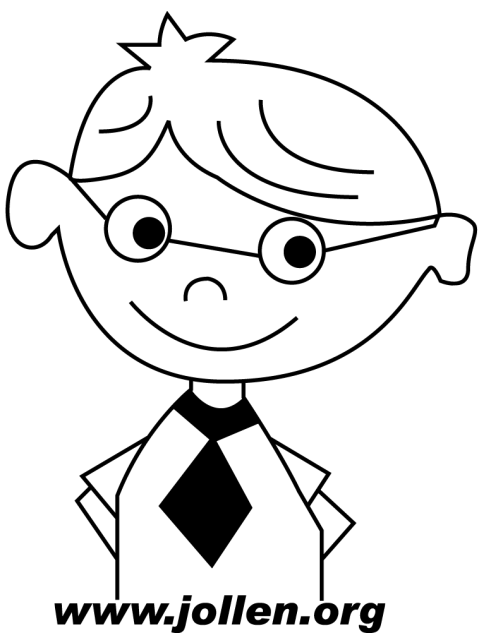
Zygote 啟動 SystemServer





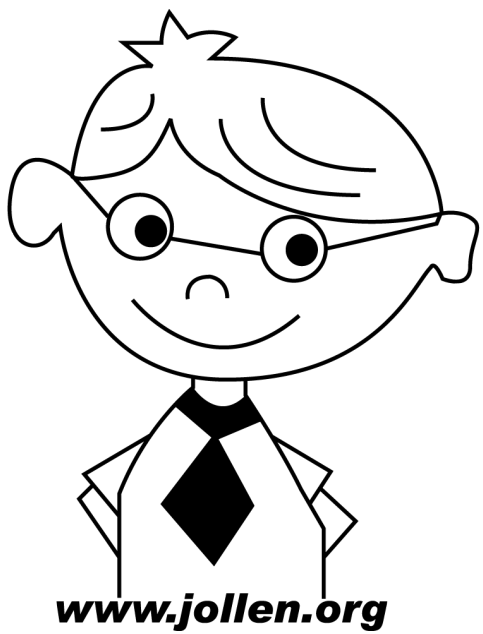
Zygote fork()s 其它應用程式





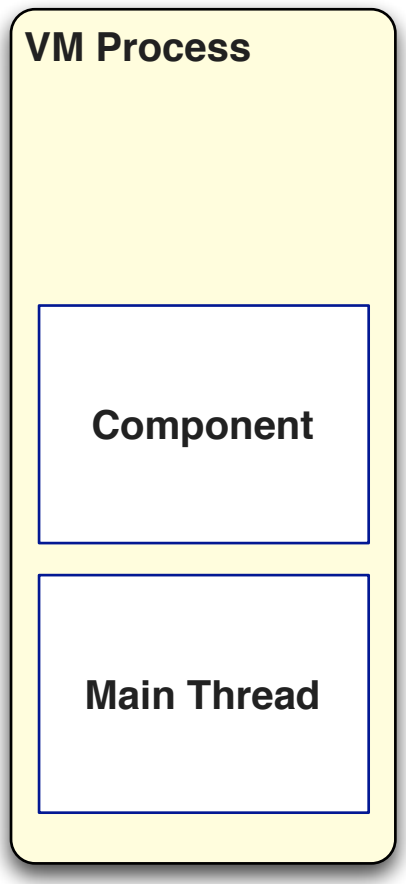
From the Beginning

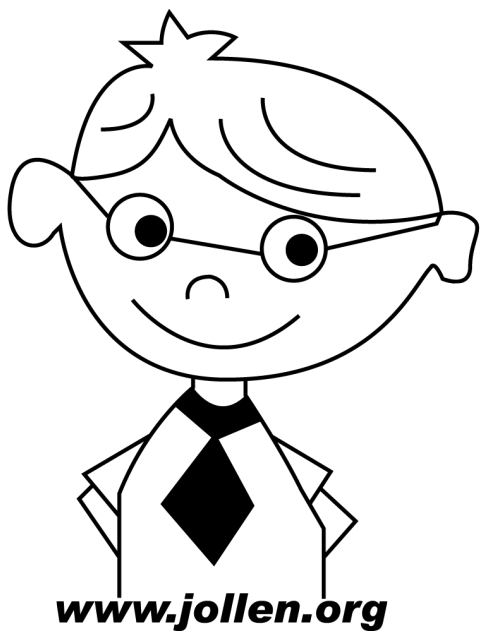
- javac Hello.java
- dx --dex --output=hello.jar Hello.class
- export BOOTCLASSPATH=$ANDROID/core.jar:
  $ANDROID/ext.jar:$ANDROID/framework.jar:
  $ANDROID/android.policy:$ANDROID/
  services.jar
- dalvikvm -cp hello.jar Hello



Start VM

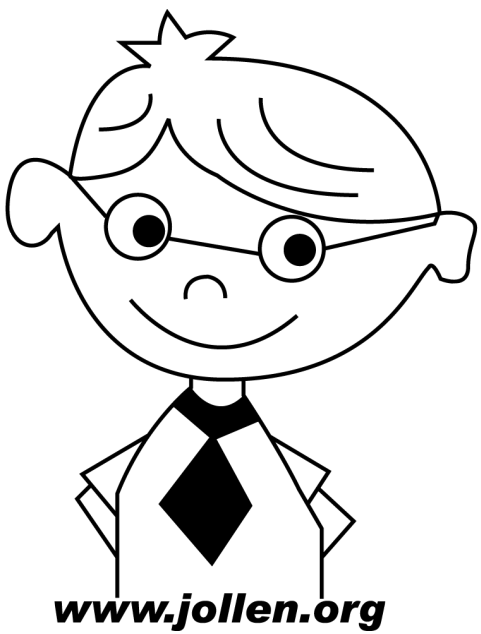
- ./dalvikvm
  - 即 VM Process
  - Start VM
  - Prepare Main Thread





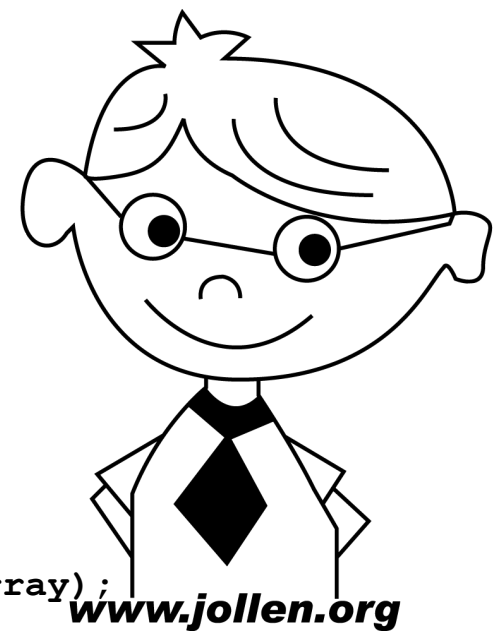
JNI_CreateJavaVM()

```
int main(int argc, char* const argv[])
{

    /*
      * Start VM.  The current thread becomes the main thread of the VM.
      */
     if (JNI_CreateJavaVM(&vm, &env, &initArgs) < 0) {
         fprintf(stderr, "Dalvik VM init failed (check log file)\n");
         goto bail;
     }

}
```



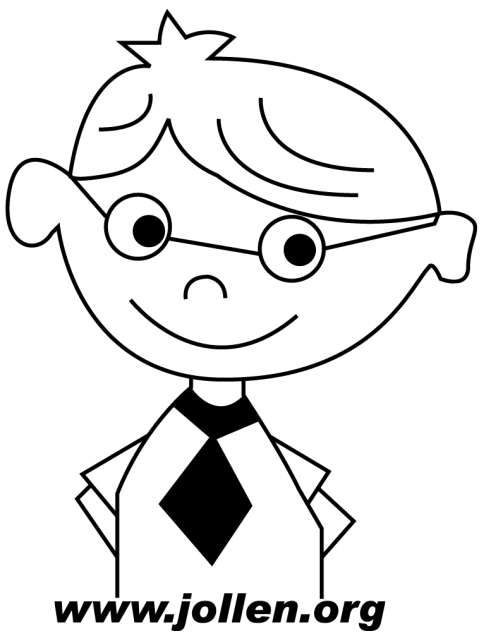
Command Line Mode

```
int main(int argc, char* const argv[])
{
    /*
     * Find [class].main(String[]).
     */
    jclass startClass;
    jmethodID startMeth;
    char* cp;

    /* convert "com.android.Blah" to "com/android/Blah" */
    slashClass = strdup(argv[argIdx]);
    for (cp = slashClass; *cp != '\0'; cp++)
        if (*cp == '.')
            *cp = '/';

    startClass = (*env)->FindClass(env, slashClass);

    startMeth = (*env)->GetStaticMethodID(env, startClass,
                    "main", "([Ljava/lang/String;)V");

    if (!methodIsPublic(env, startClass, startMeth))
        goto bail;

    /*
     * Invoke main().
     */
    (*env)->CallStaticVoidMethod(env, startClass, startMeth, strArray);
}
```



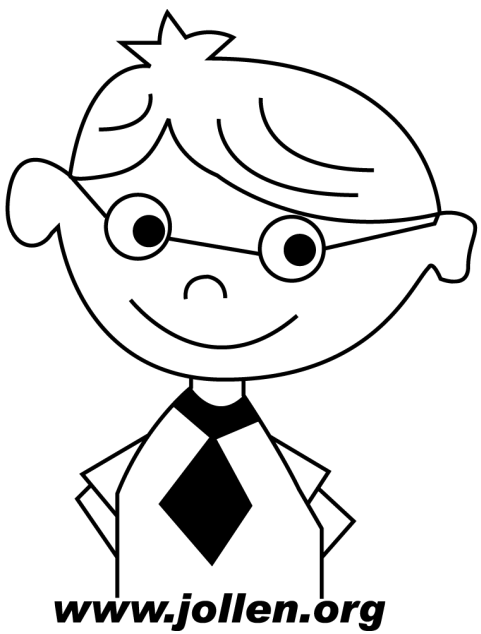
Invoke main() method

- Main() must be public , static, return void and accept String array as the only parameter
- Start point of a Java program
- Start main thread (initial thread)



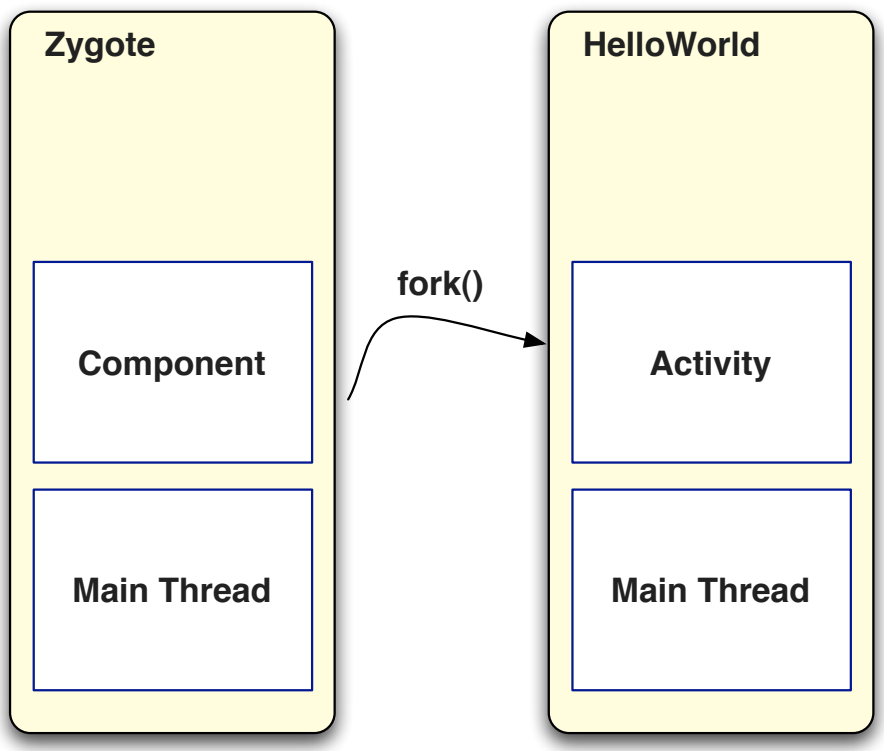
ZygoteInit.main()

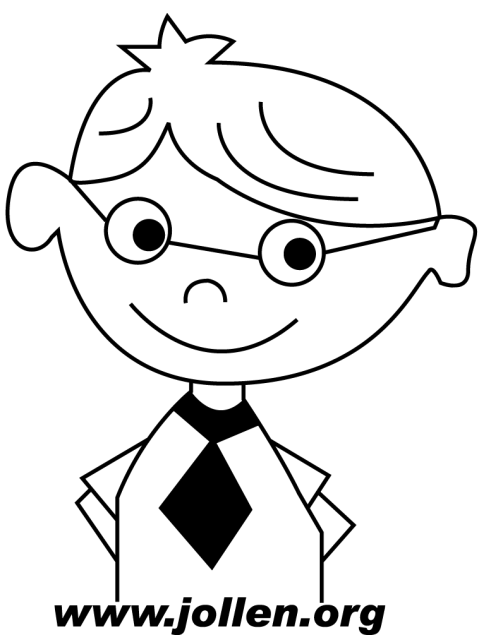
```
public class ZygoteInit {
    ...
    public static void main(String argv[]) {
            registerZygoteSocket();
            preloadClasses();

            // Do an initial gc to clean up after startup
            gc();

            if (argv[1].equals("true")) {
                startSystemServer();
            }

            Log.i(TAG, "Accepting command socket connections");

            if (ZYGOTE_FORK_MODE) {
                runForkMode();
            } else {
                runSelectLoopMode();
            }

            closeServerSocket();
    }
    ...
}
```



執行外部 APK

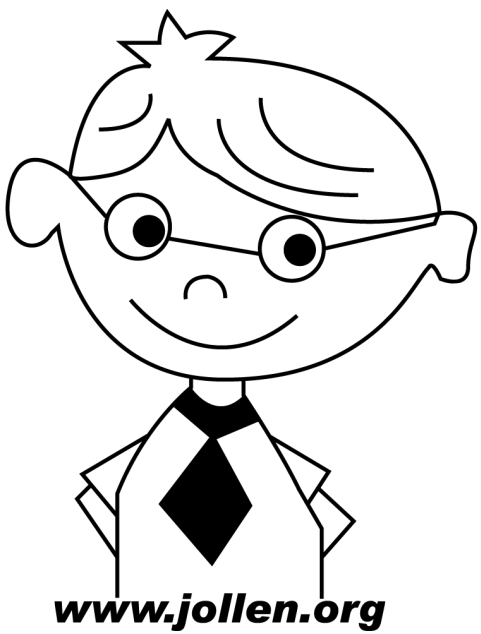
```
public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```





Zygote.fork() 的 Native 實作

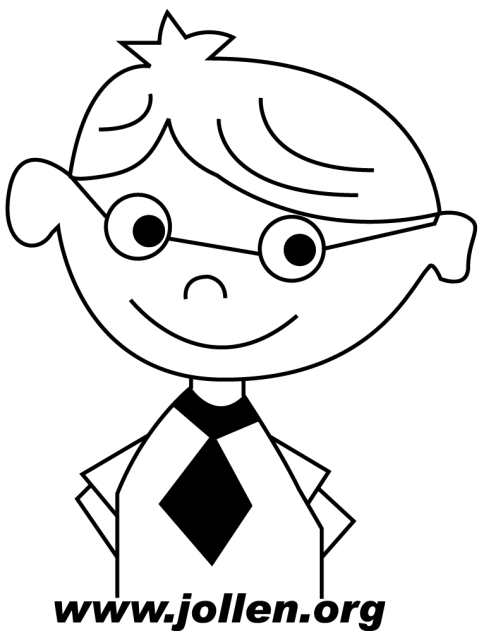
```
static void Dalvik_dalvik_system_Zygote_fork(const u4* args, JValue* pResult)
{
    pid_t pid;

    setSignalHandler();

    dvmDumpLoaderStats("zygote");
    pid = fork();

    RETURN_INT(pid);
}
```



See: http://www.jollen.org/wiki/Zygote_Native

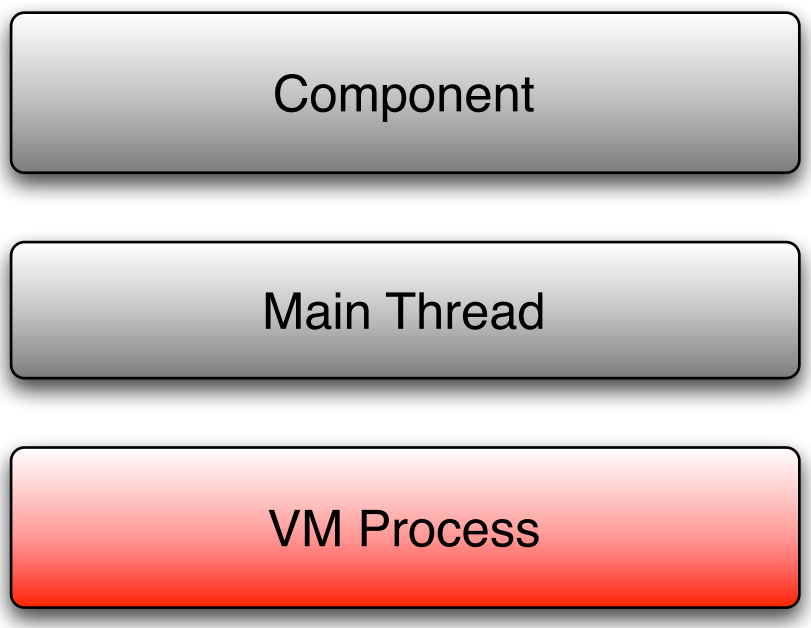
Framework 不主動執行

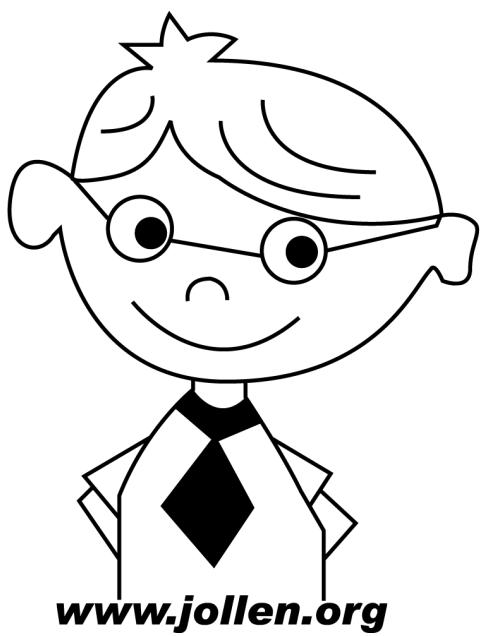
- Framework 只是 Java Library
  - Framework (呈現形式是 *.jar) 只是Android 應用程式的 Library
- 手機桌面 (Home Screen)
  - 桌面不是由 Framework 畫出
  - 由 Launcher 處理 Home Screen



Android Process 模式

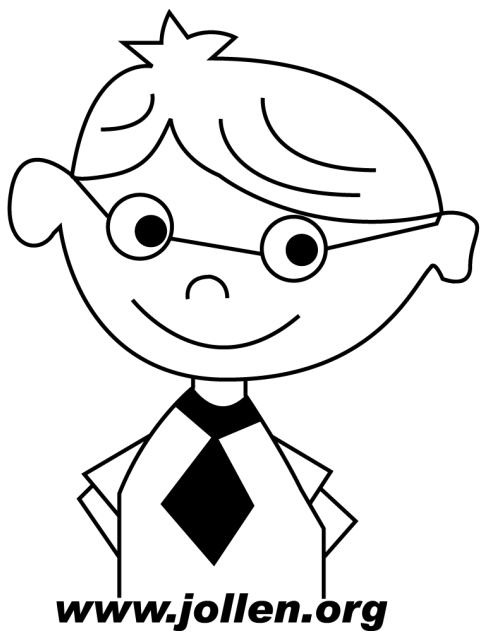
- 一個 process、一個 main thread
- Android Process 必有一個 Main Thread
- Main Thread 執行 Main Component





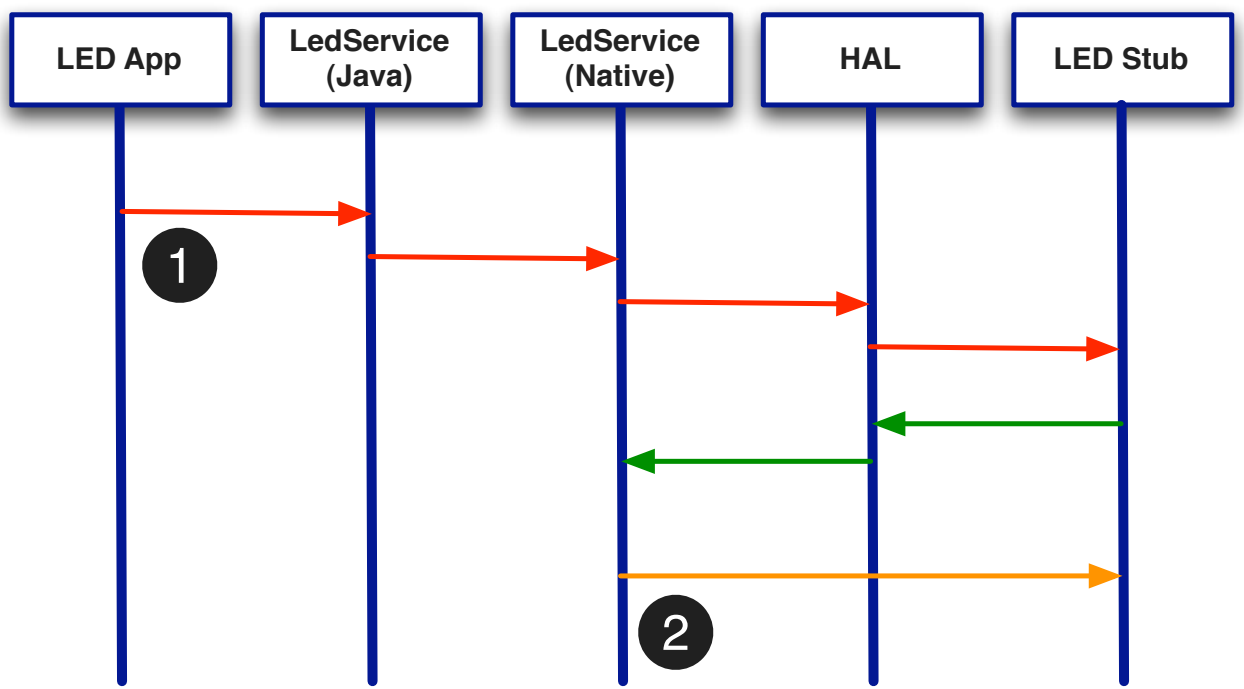
Main Thread & Secondary Thread

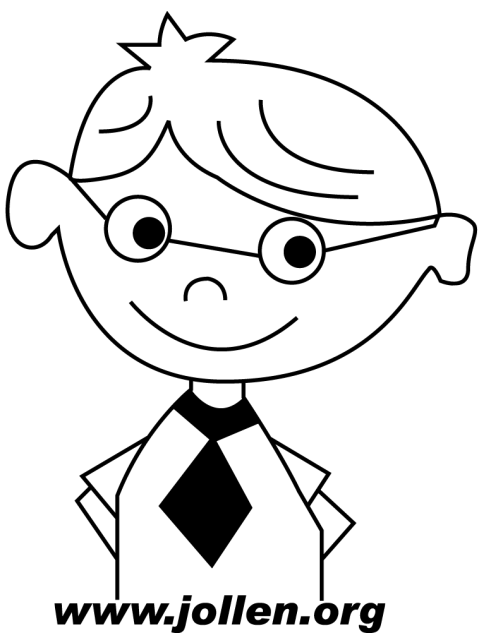
- 每一個 Process 都有一個 Main Thread
- Main Thread
  - 由 Dalkvik VM 產生
  - 執行 main() method
  - Native PThread
- Secondary Thread
  - 由應用程式產生
  - 執行 java.lang.Thread
  - Native PThread



使用 IPC 模式

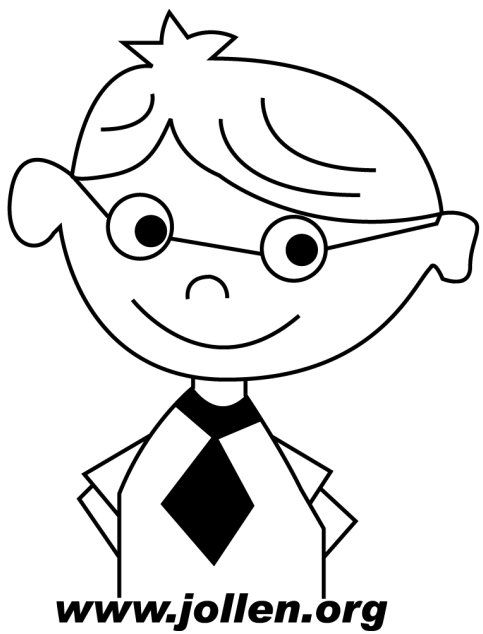
- Direct Method Call
  - LedClient
- Remote Method Call (IPC、Binder)
  - LedTest





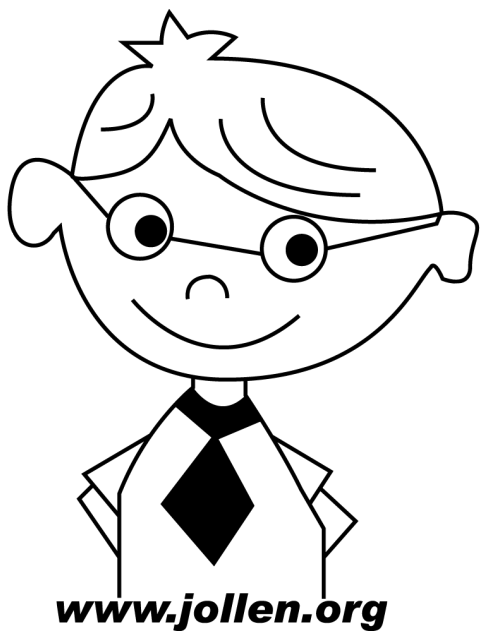
SystemServer & ServerThread

- SensorManager 取得 SensorService
  - 才能在應用程式裡控制硬體
- SystemServer
  - 提供 Android Service 物件的 Process
- ServerThread
  - 由 SystemServer 啟動的 Java Thread
  - 提供 SensorService 物件
  - 以及其它近 30 個 Android Service



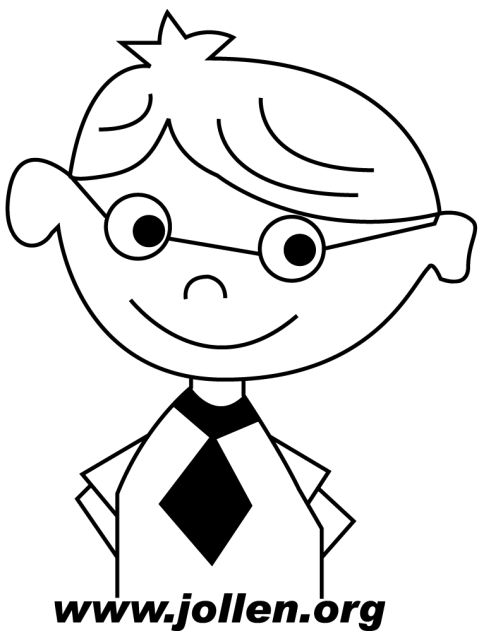
SystemServer

```
public class SystemServer
{
    private static final String TAG = "SystemServer";

    native public static void init1(String[] args);

    public static void main(String[] args) {
        System.loadLibrary("android_servers");
        init1(args);
    }

    public static final void init2() {
        Log.i(TAG, "Entered the Android system server!");
        Thread thr = new ServerThread();
        thr.setName("android.server.ServerThread");
        thr.start();
    }
}
```



Native Service

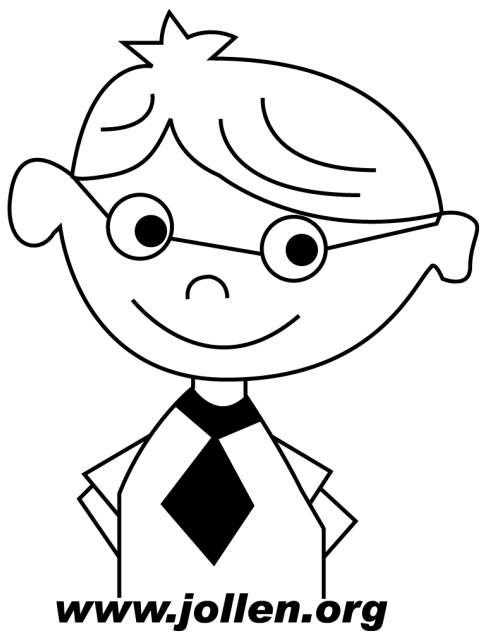
- 有哪些Native Service
  - SurfaceFlinger
  - AudioFlinger
- SystemServer
  - init1()：啟動Native Service
  - init2()：啟動Android Service



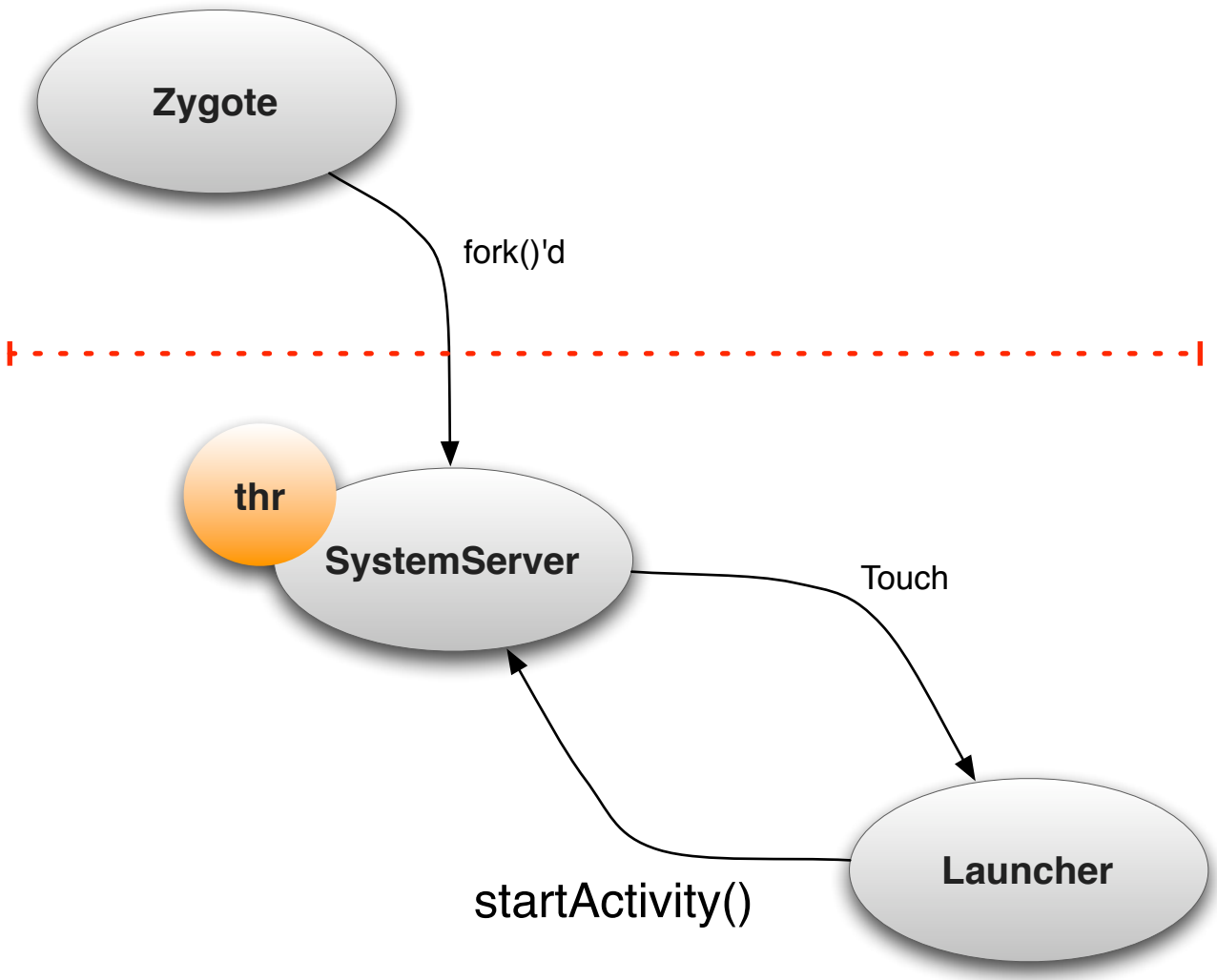
啟動 ServerThread

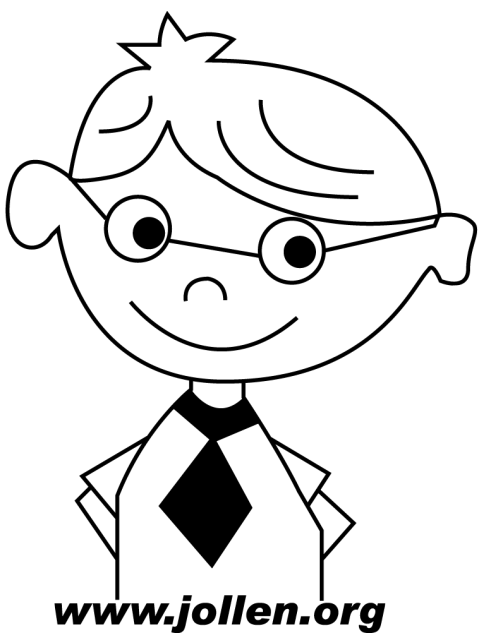
```
class ServerThread extends Thread {
    ...
    public void run() {
  ...
        HardwareService hardware = null;
  ...
        Log.i(TAG, "Starting Hardware Service.");
        hardware = new HardwareService(context);
        ServiceManager.addService("hardware", hardware);
  ...
    }
...
}
```

```
public class SystemServer
{
...
    public static final void init2() {
        Log.i(TAG, "Entered the Android system server!");
        Thread thr = new ServerThread();
        thr.setName("android.server.ServerThread");
        thr.start();
    }
...
}
```



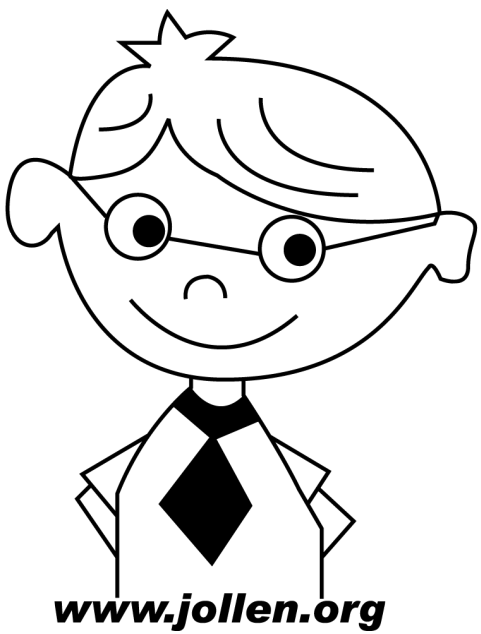
SystemServer 就緒





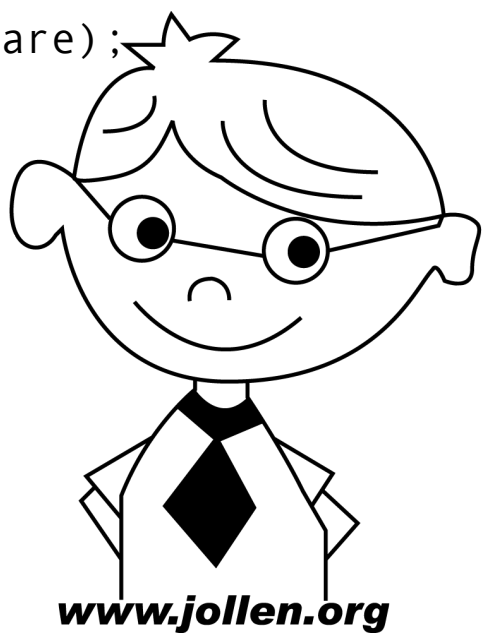
ServiceManager 與 getSystemService API

- ServiceManager 實作 IServiceManager
- ServiceManager.addService() 加入 Android Service 物件
  - 應用程式呼叫 getSystemService() 來取得



加入 Android Service 物件

```
class ServerThread extends Thread {
    ...
        public void run() {
            ...
            HardwareService hardware = null;
            ...
            Log.i(TAG, "Starting Hardware Service.");
            hardware = new HardwareService(context);
            ServiceManager.addService("hardware", hardware);
            ...
        }
    ...
}
```



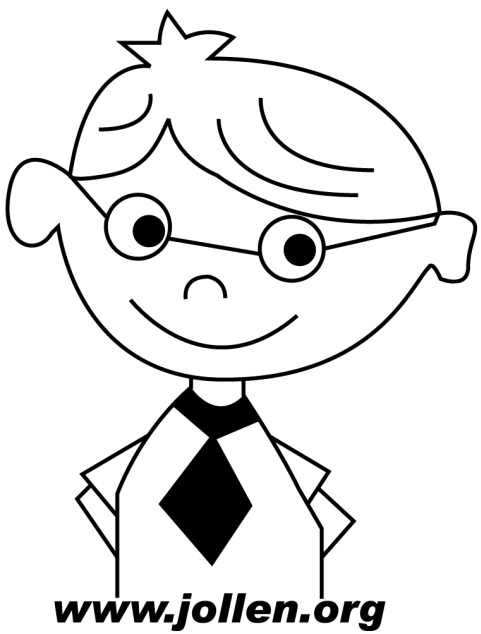
使用 getSystemService()

```
public class mokoidSensor extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SensorManager sensor = (SensorManager)getSystemService(SENSOR_SERVICE);
        sensor.getSensors();
    }
}
```

第07堂課： SystemService 與 HAL 整合

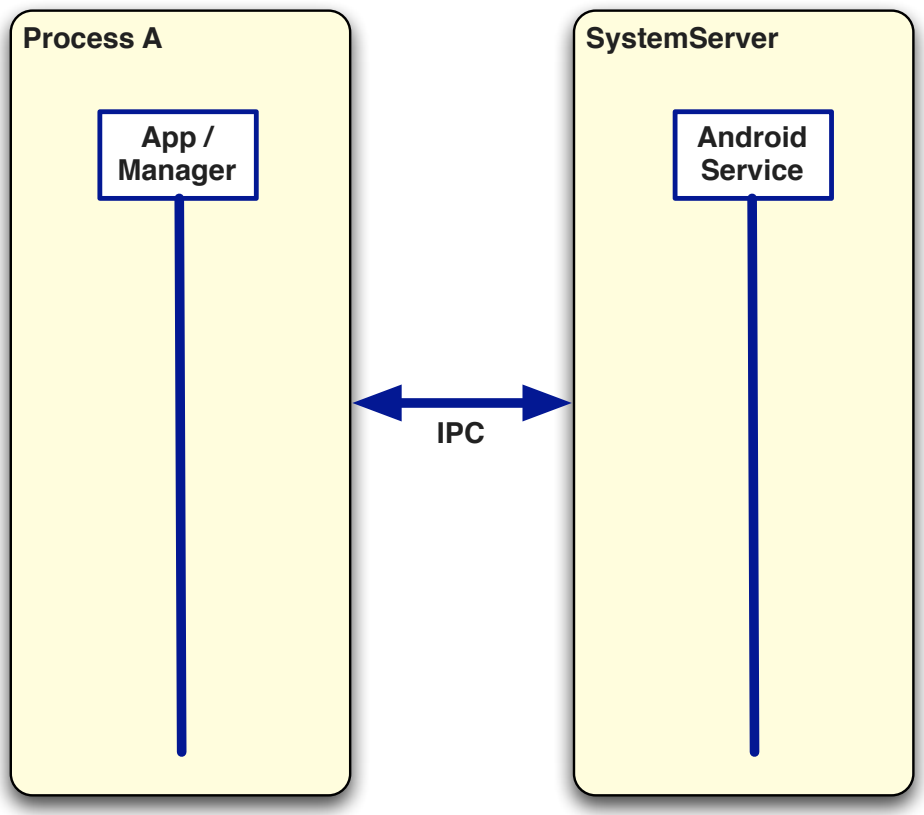
本堂主題

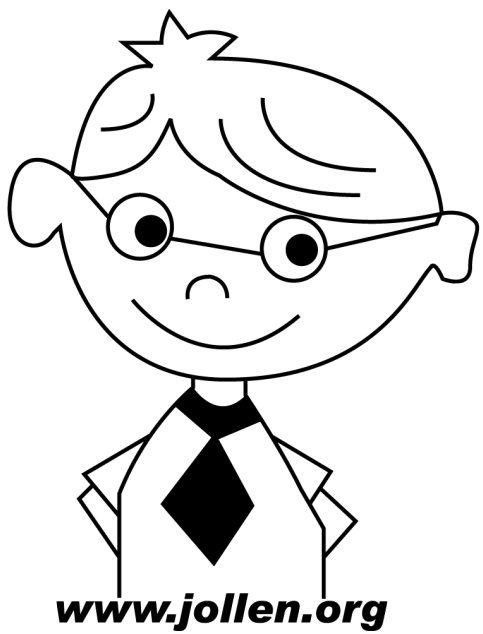
第七堂	SystemService 與 HAL 整合
7.1	IPC、 Remote method call與Binder觀念說明
7.2	AIDL 介紹與IInterface設計觀念解析
7.3	Activity & ApplicationContext
7.4	ServiceManager
7.5	專題討論：LedService設計與ILedService探討



了解 SystemServer 與 IPC

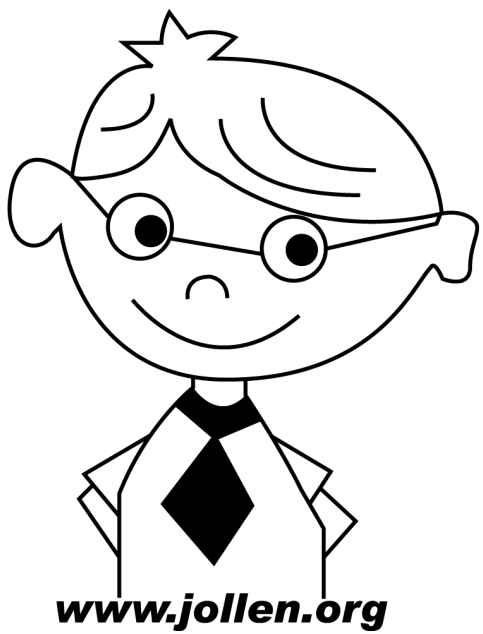
- 了解什麼是 SystemServer
- 了解 Android 的 IPC (Binder)



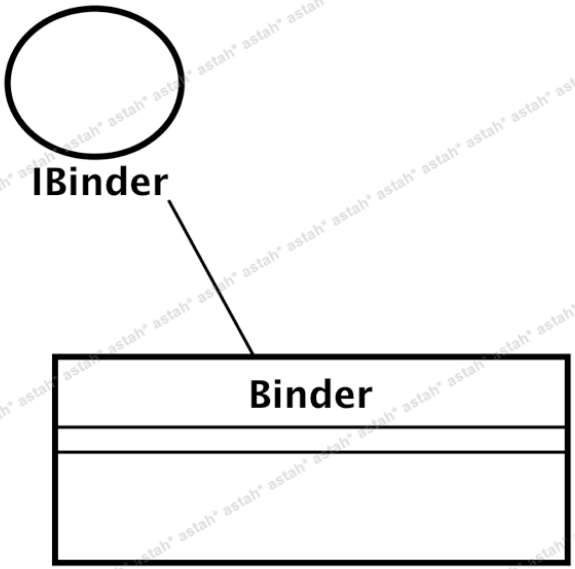


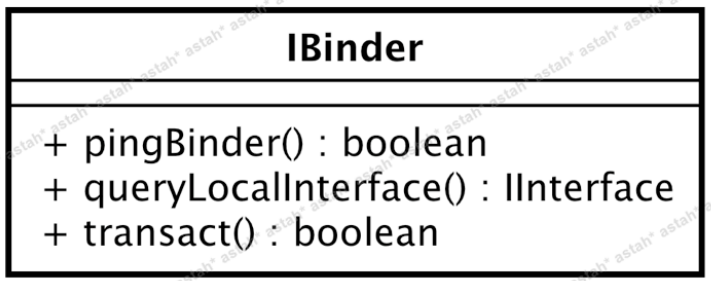
IPC 與 Remote Method Call

- Inter-Process Communication
  - System-Leve (低階部份)
  - Android kernel 的 Binder 驅動程式支援 Android Process 的 IPC
- Remote Method Call
  - Method-Level (Java、高階部份)
  - 使用 IBinder 做 Remote Method Call
- Proxy Object
  - Architect-Level (架構設計、抽象部份)
  - 執行 AIDL 產生的程式碼取得 Proxy Object

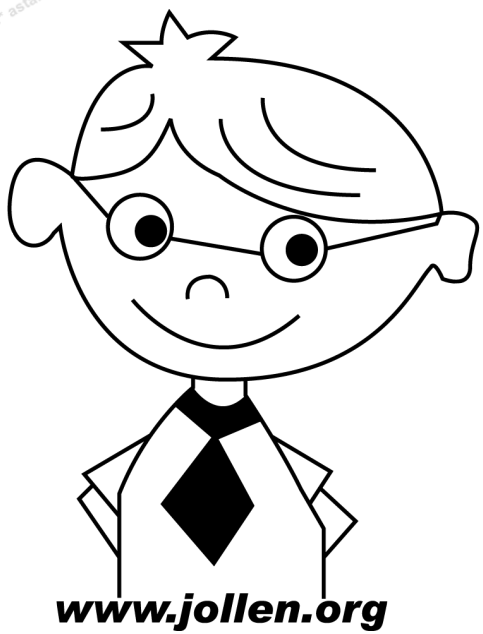


IBinder Interface



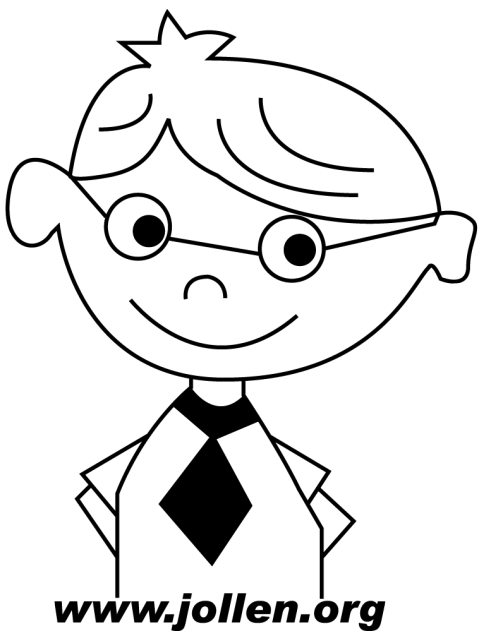



```
public class Binder implements IBinder {
}
```



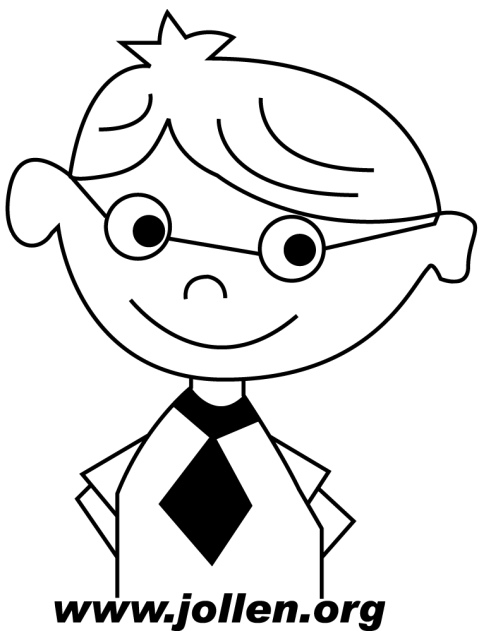
什麼是 Binder

- Binder Class 實作 IBinder Interface
- Binder 定義 Native Method
- Binder 是實作與 Linux Binder 驅動程式溝通的 Class



IBinder

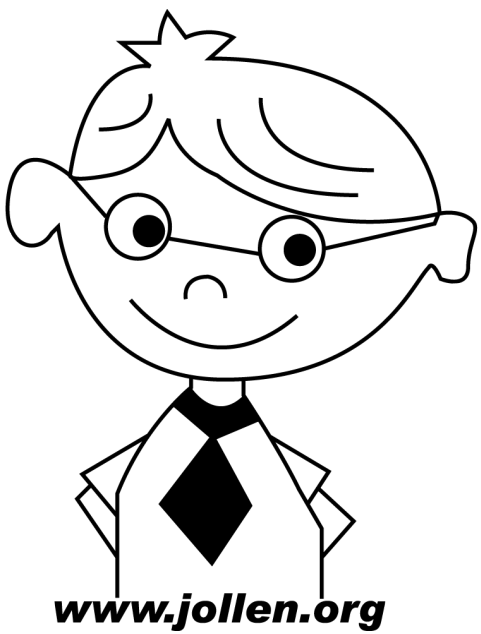
- IBinder是與Remote Object互動的介面
- 透過IBinder與Remote Object溝通
- 如何物件成為「Remotable Object」？



IBinder

- Remotable Object 負責實作 IBinder
- Remote Object 才能「和我溝通」
- 繼承 Binder 即可成為 Remotable Object
  - Android 框架的設計做法

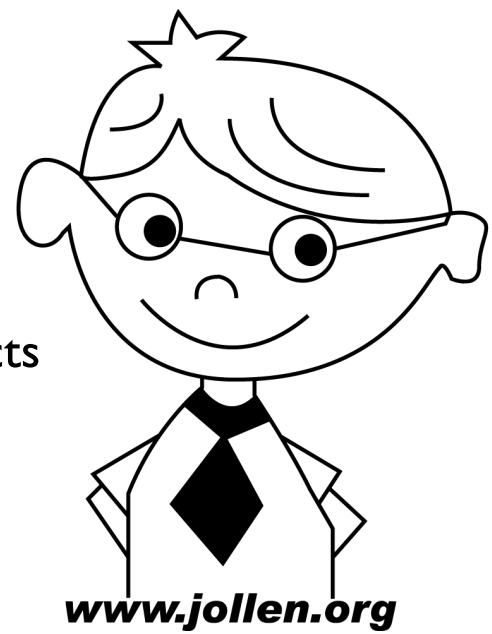
```
public class LedService extends Binder {
        ...
}
```



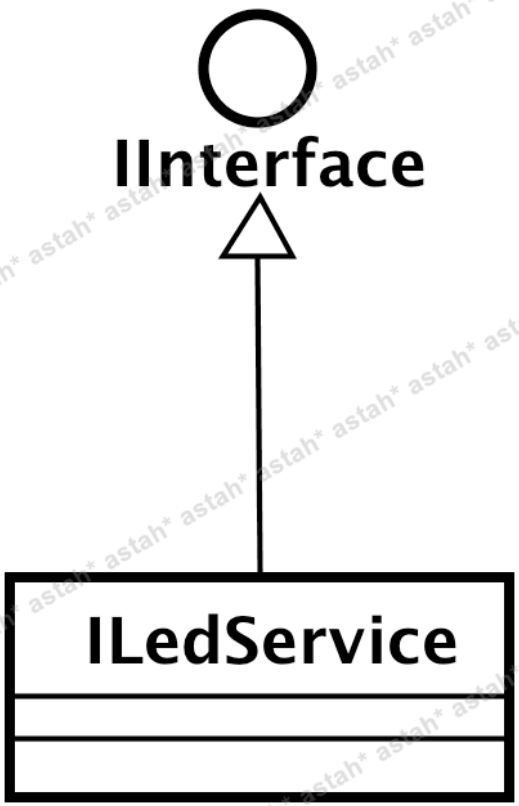
IInterface

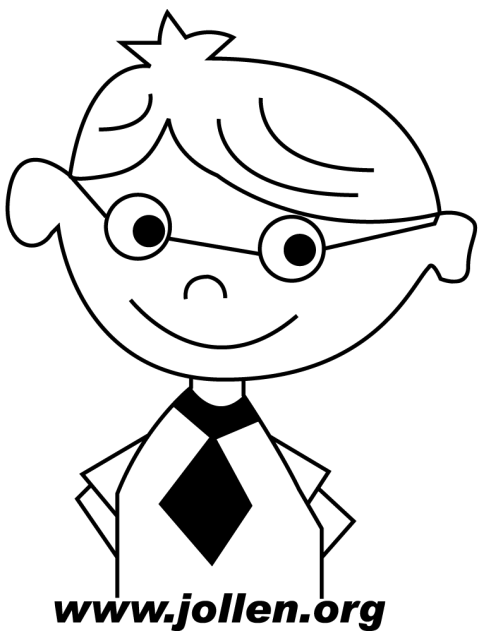
- 自已定義的 Interface 都必須繼承Interface
  - Android 框架的設計

```
package android.os;

/**
 * Base class for Binder interfaces. When defining a new interface,
 * you must derive it from IInterface.
 */
public interface IInterface
{
    /**
     * Retrieve the Binder object associated with this interface.
     * You must use this instead of a plain cast, so that proxy objects
     * can return the correct result.
     */
    public IBinder asBinder();
}
```

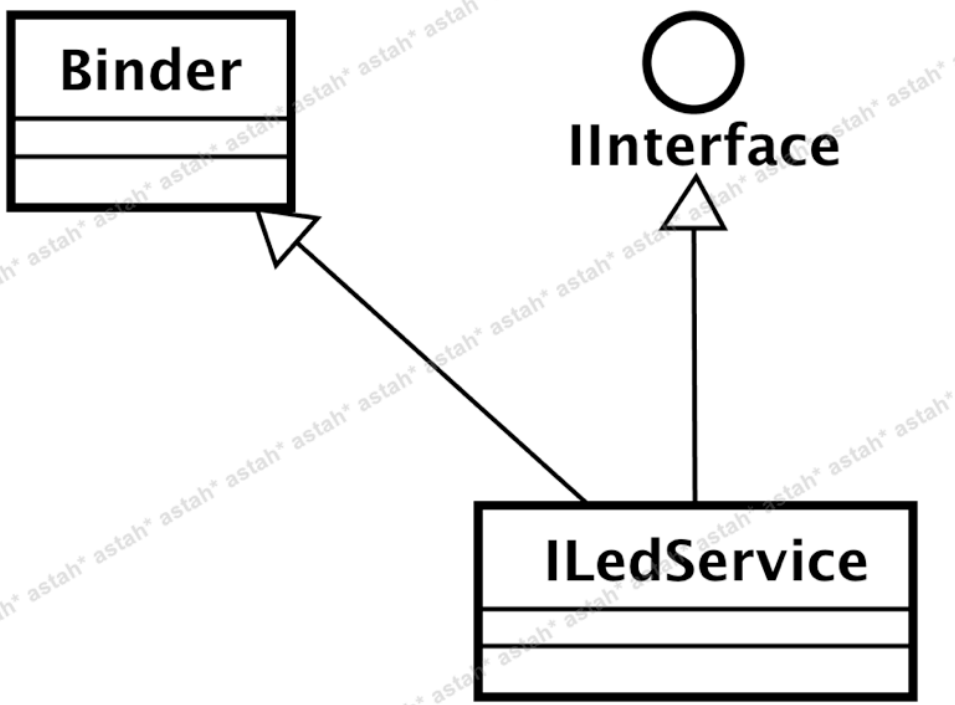


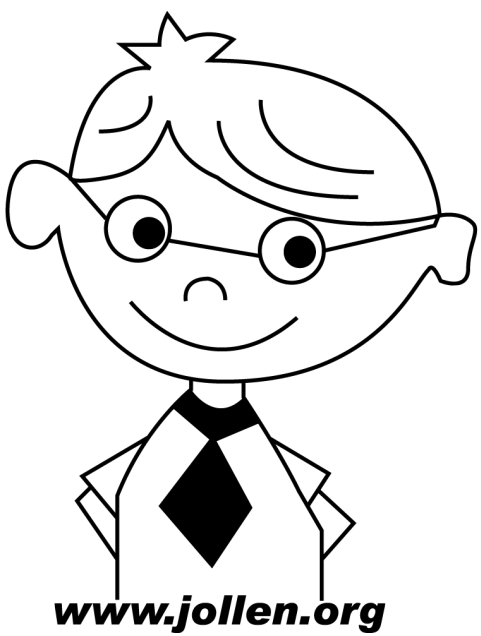
定義 ILedService





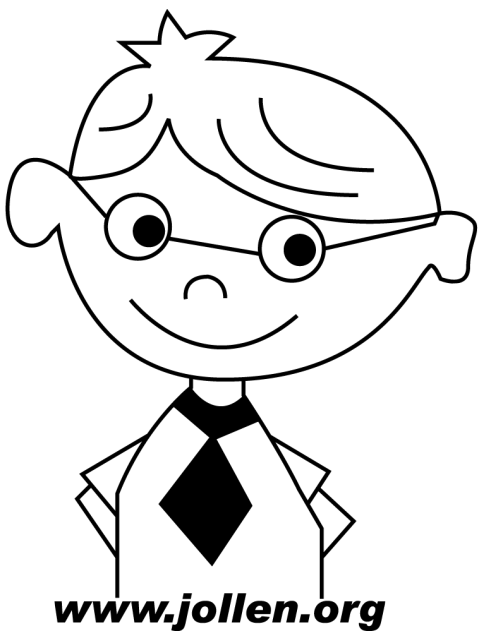
ILedService 成為 Remotable





AIDL

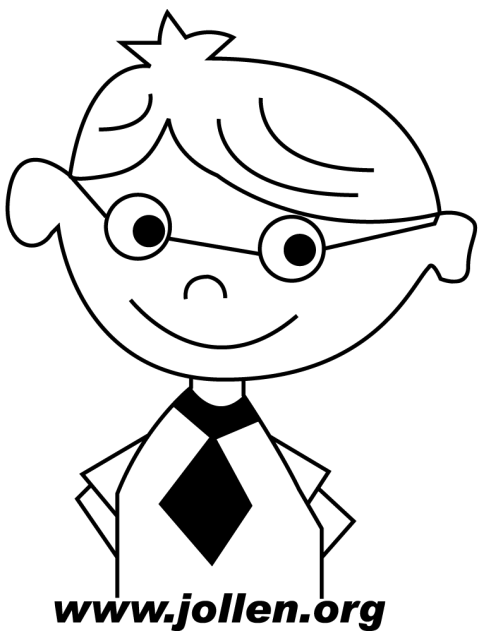
- Android Interface Description Language
- 自動產生處理 Binder (IPC) 的程式碼
- 使用 aidl.exe 工具
  - 定義 My Interface (*.aidl)
  - 自動產生 *.java



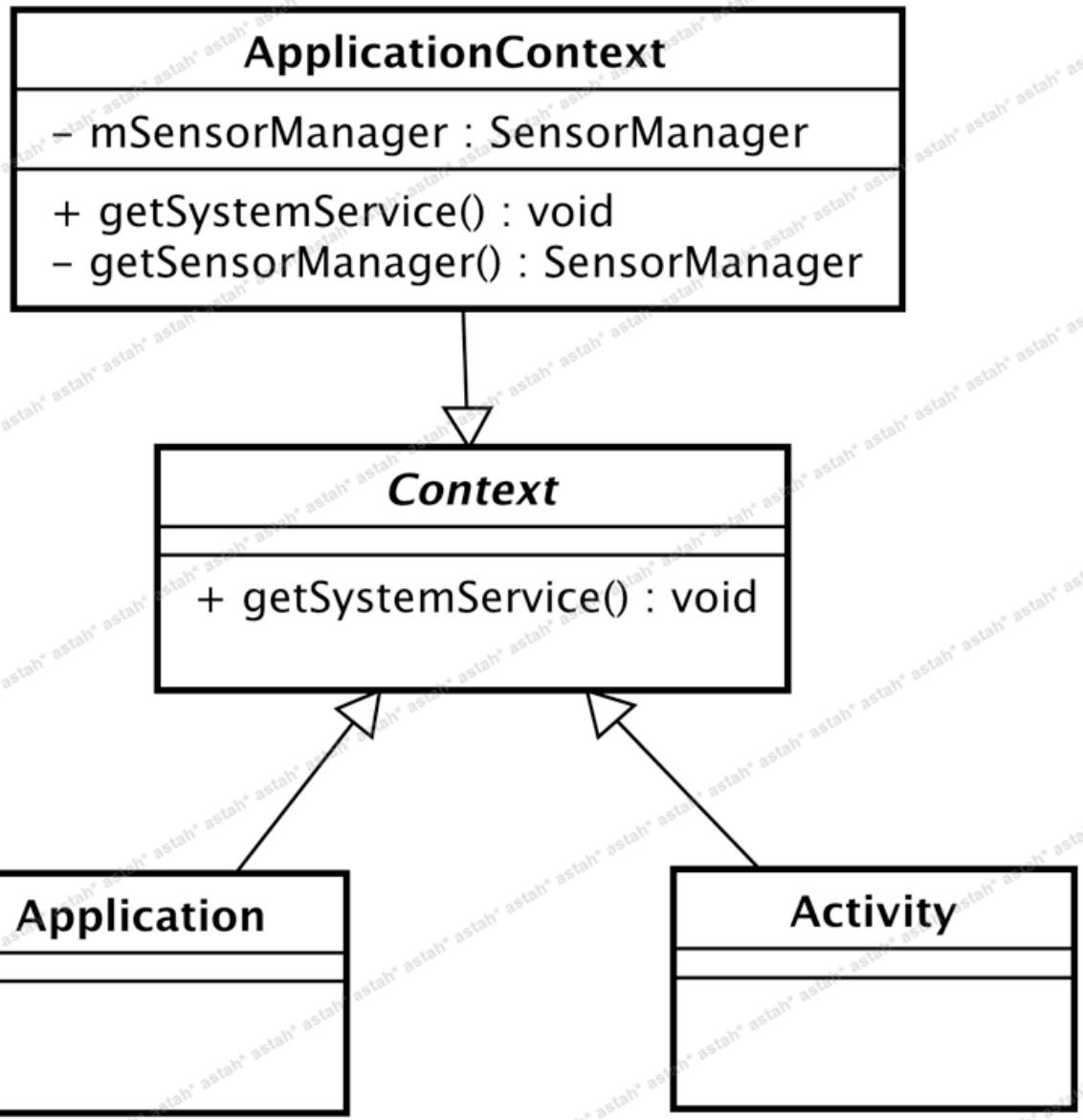
使用 AIDL

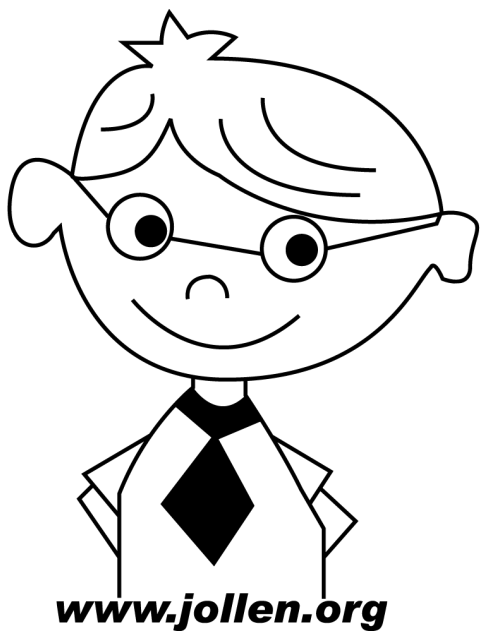
- ILedService.aidl
- 自動產生 ILedService.java
  - ➡ 免去自行實作 Binder 程式碼的麻煩

```
interface ILedService {
        boolean setOn(int led);
        boolean setOff(int led);
}
```



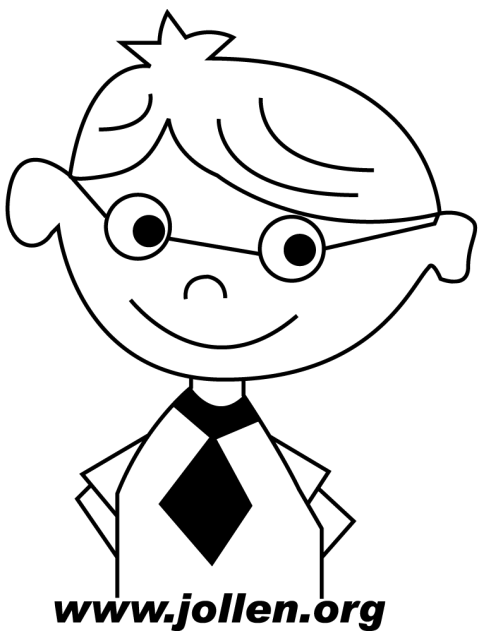
Activity & ApplicationContext





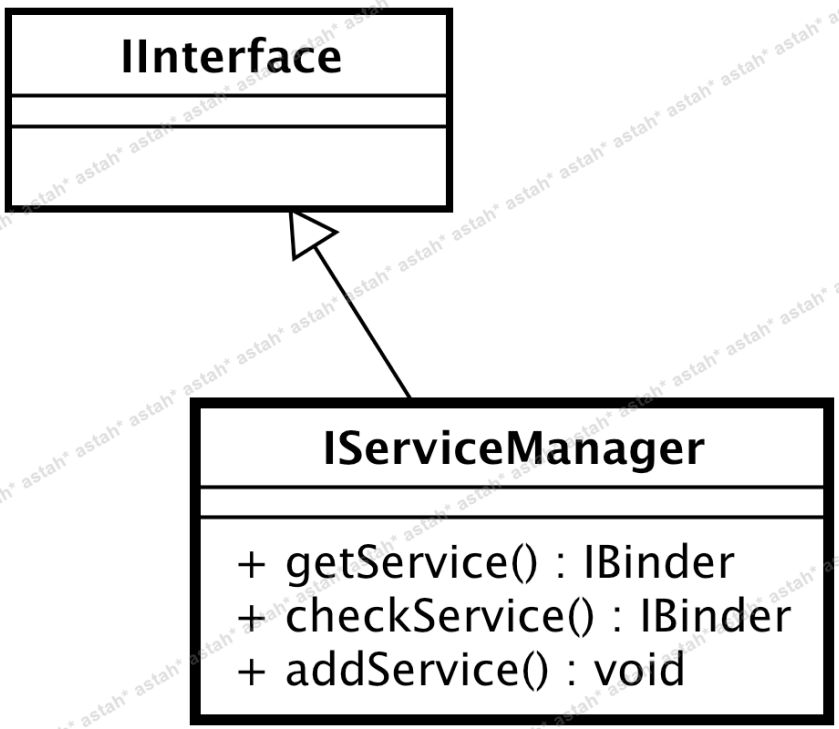
IServiceManager 的設計

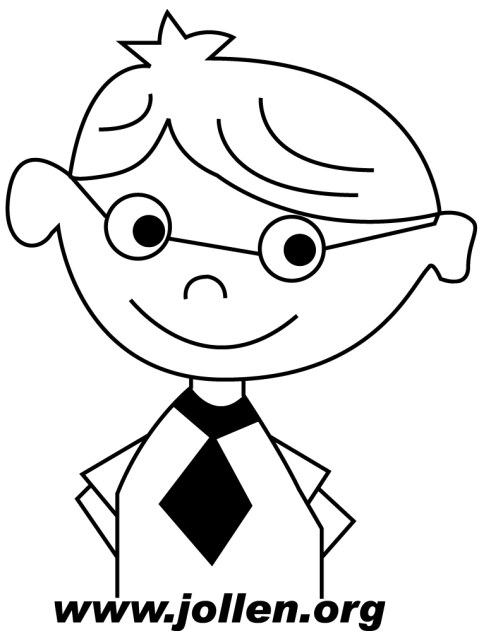
```
public interface IServiceManager extends IInterface
{
    public IBinder getService(String name) throws RemoteException;
    ...
}
```



IServiceManager

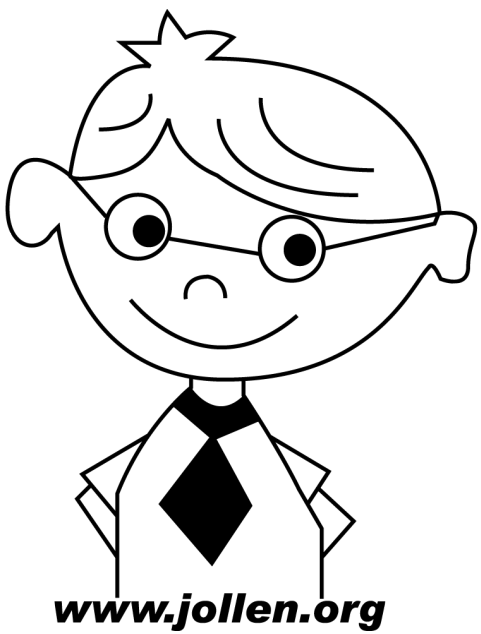
- IServiceManager.getService()傳回IBinder物件
  - IBinder是什麼？





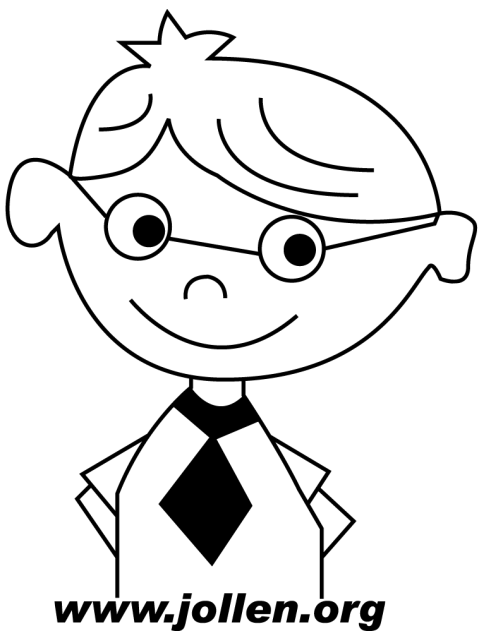
ServiceManager

```
public final class ServiceManager {
    private static final String TAG = "ServiceManager";

    private static IServiceManager sServiceManager;
    private static HashMap<String, IBinder> sCache = new HashMap<String, IBinder>();

    private static IServiceManager getIServiceManager() {
        if (sServiceManager != null) {
            return sServiceManager;
        }

        // Find the service manager
        sServiceManager =
ServiceManagerNative.asInterface(BinderInternal.getContextObject());
        return sServiceManager;
    }
    ...
}
```



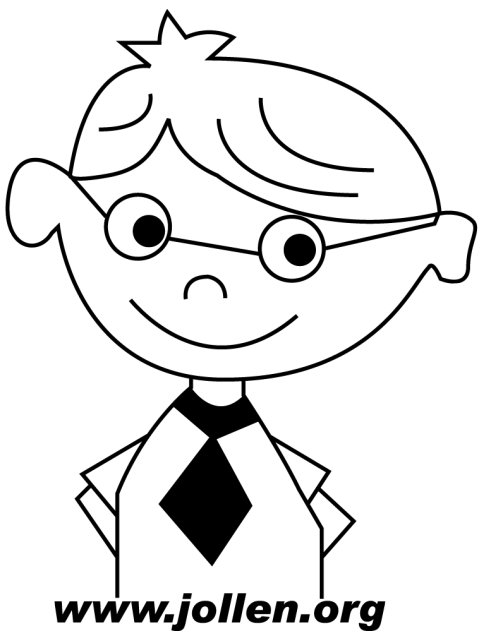
ServiceManager.getService()

```
public final class ServiceManager {
    ...
    public static IBinder getService(String name) {
        try {
            IBinder service = sCache.get(name);
            if (service != null) {
                return service;
            } else {
                return getIServiceManager().getService(name);
            }
        } catch (RemoteException e) {
            Log.e(TAG, "error in getService", e);
        }
        return null;
    }
    ...
}
```

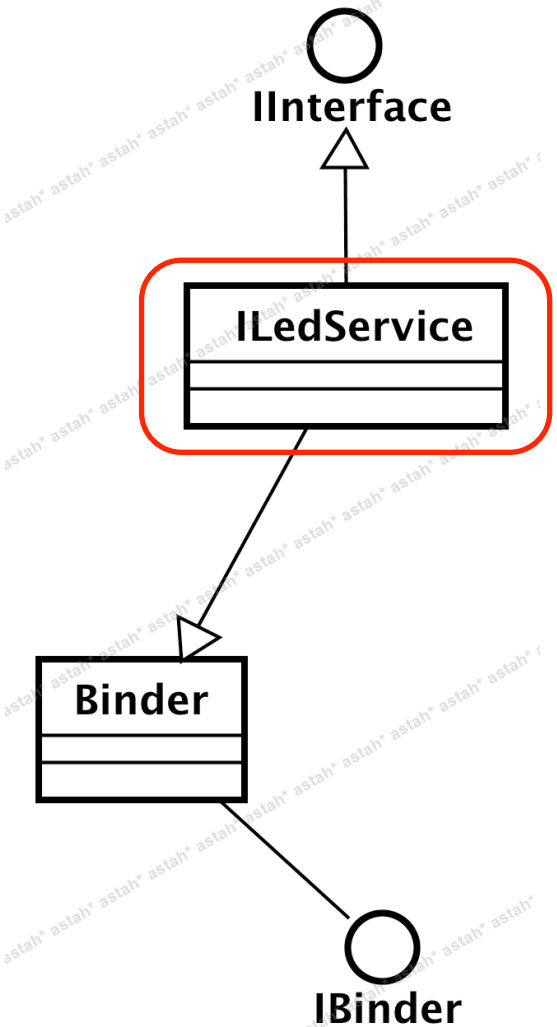


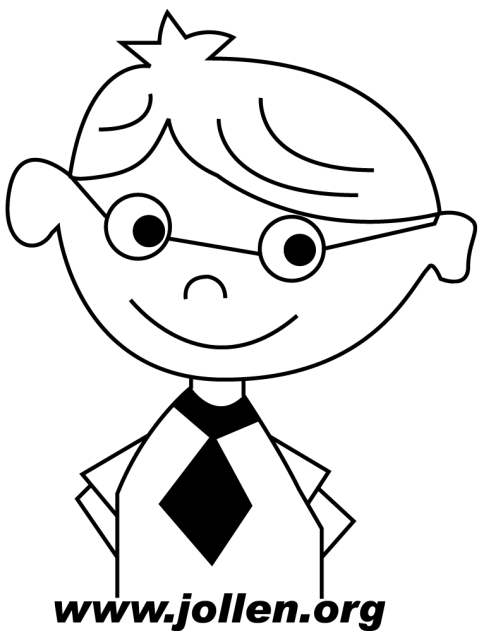
ServiceManager.addService()

```
public final class ServiceManager {
    ...
    public static void addService(String name, IBinder service) {
        try {
            getIServiceManager().addService(name, service);
        } catch (RemoteException e) {
            Log.e(TAG, "error in addService", e);
        }
    }
    ....
}
```



The Service Object

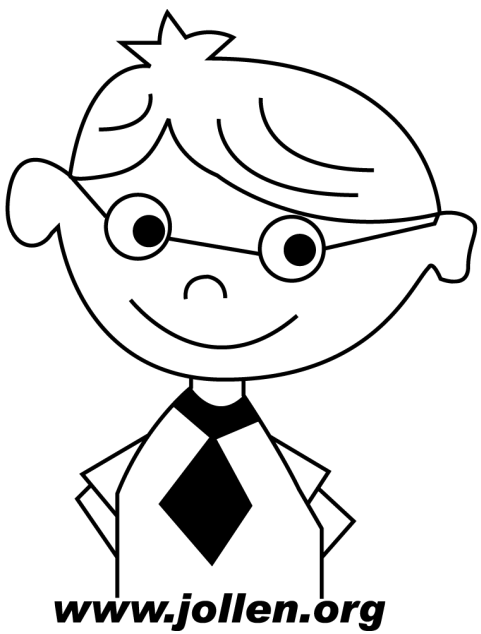




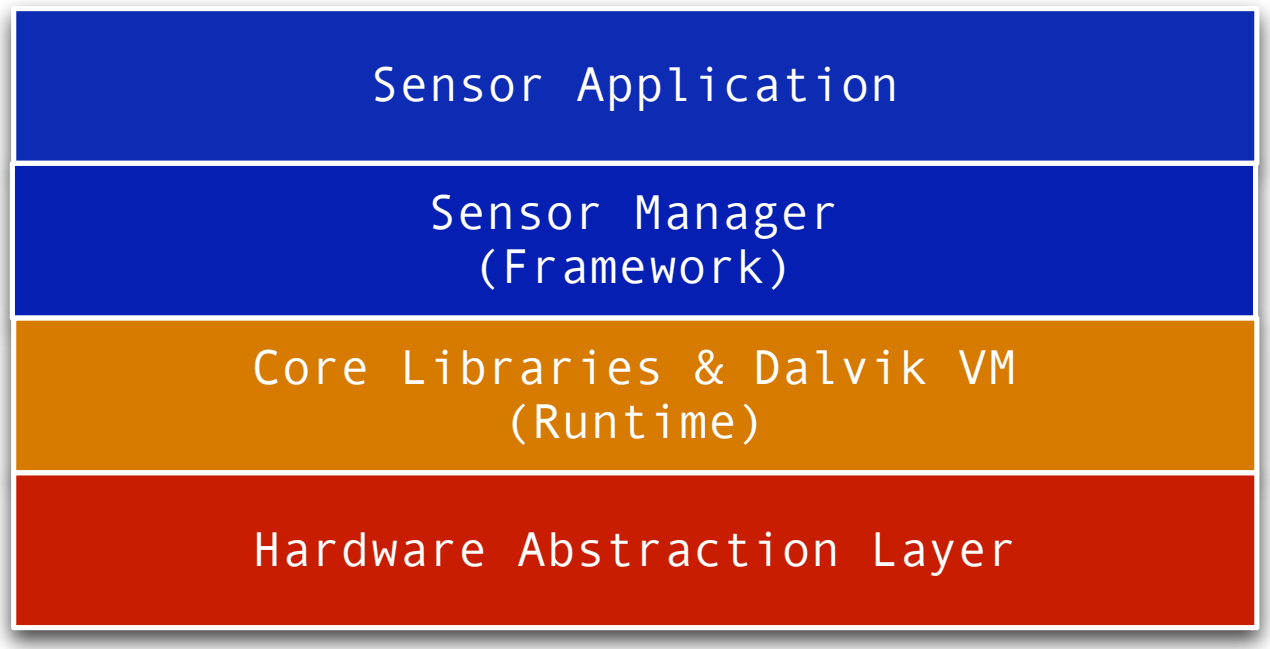
第08堂課： Manager API (Refinement-架構優化)

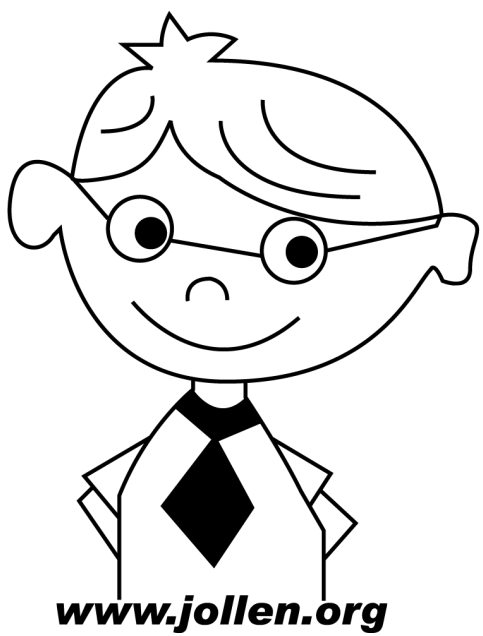
本堂主題

第八堂	Manager API (Refinement-架構優化)
8.1	SensorManager與 SensorService實例探討
8.2	Remote Object觀念與IBinder介紹
8.3	如何以Proxy Object整合Android Service
8.4	Long operations 的解析與實作細節
8.5	RemoteException 的解析與實作細節
8.6	Handler 與 Message 的解析與實作細節
8.7	Error Handling



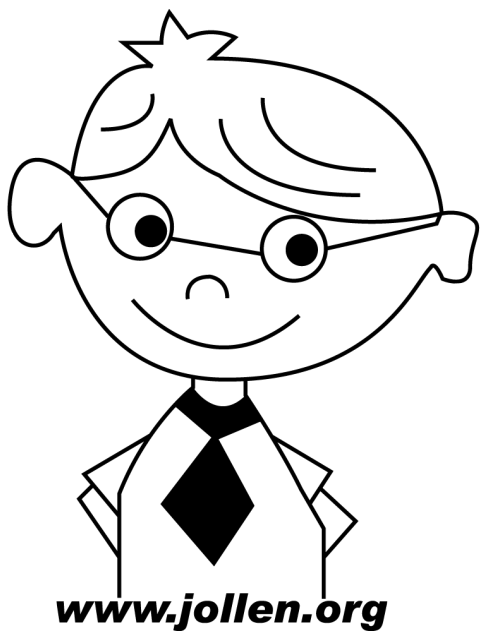
SensorManager





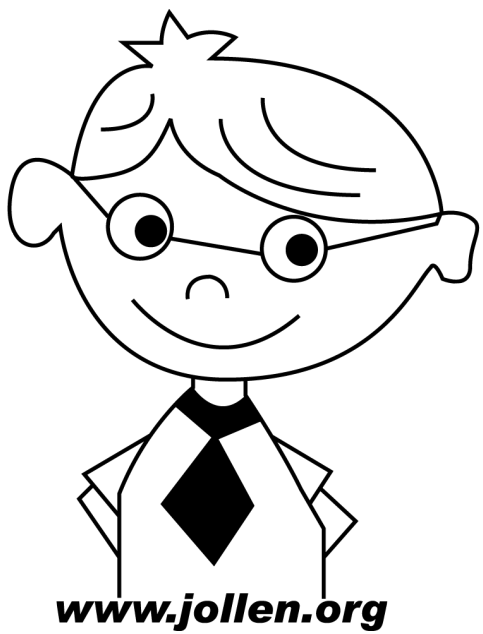
現有實作分析

- LedService 觀念的進化
  - 由原本 API 方式；轉換為
  - 實作 ILedService.Stub 的做法
- 1. 設計 LedService class
- 2. 實作 LedService API (method)
- 3. 完成 Native Service 與 JNI 實作
- 4. 完成 HAL Stub 實作
- 5. 進行架構優化



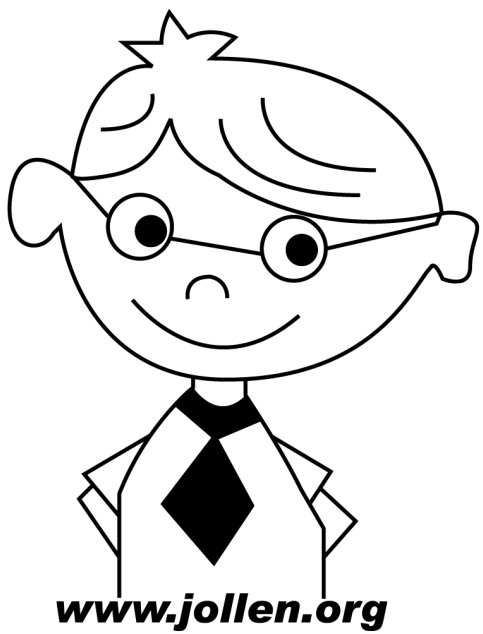
硬體API的架構優化

- 優化(Refine)的目的是為了
  - 符合 Architect
  - 以得到架構帶來的好處
- 1. 設計界面 (Interface)
  - ILedService
- 2. 架構設計變更
  - LedService 原本是 API 類別、變更為 ILedService.Stub 的實作類別
- 3. 啟動 Service



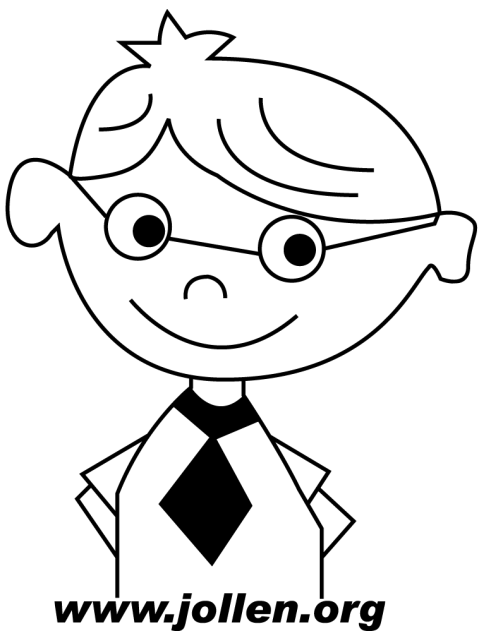
應用程式的設計變更

```
package com.mokoid.LedClient;
import com.mokoid.server.LedService;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class LedClient extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Call an API on the library.
          LedService ls = new LedService();
          ls.setOn(1);

        TextView tv = new TextView(this);
        tv.setText("LED 0 is on.");
        setContentView(tv);
    }
}
```



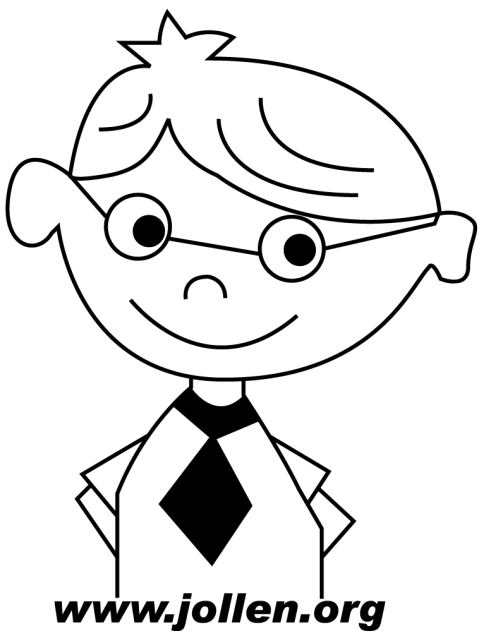
架構上的差異性

- 原始做法：
  - new object
  - use API
- 優化後：
  - get system service
  - use object
  - remote method call



設計 Remote Object

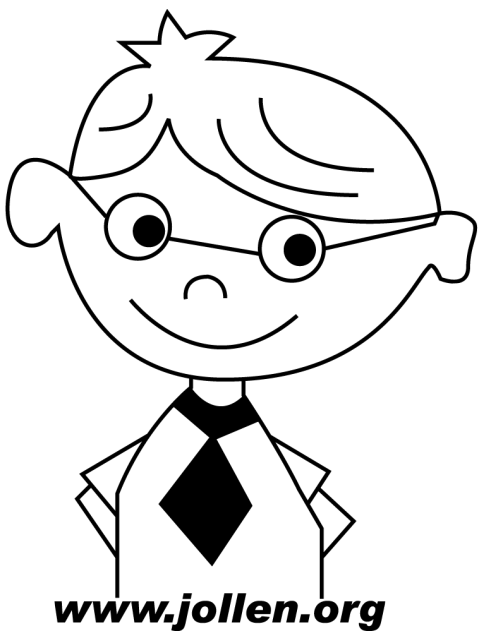
- LedService物件是一個遠端物件
- 這個物件並不存在於「自己的Process裡」
- Android框架的設計：
  - 凡是要定義新的IInterface都必須繼承IInterface



如何實作 Remotable Object

- 置於遠端的 Object 稱為 Remote Object
- 可置於遠端的 Object 稱為 Remotable Object
  - 可與其溝通 (Communication)

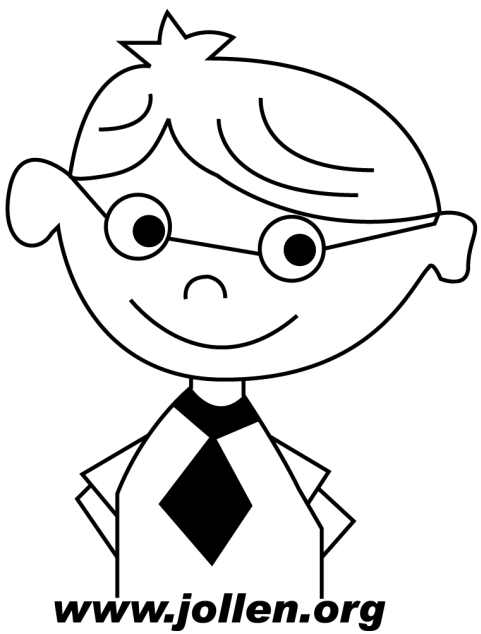
```
public final class LedService extends Binder {
  ...
}
```



使用 AIDL 實作 Remotable Object

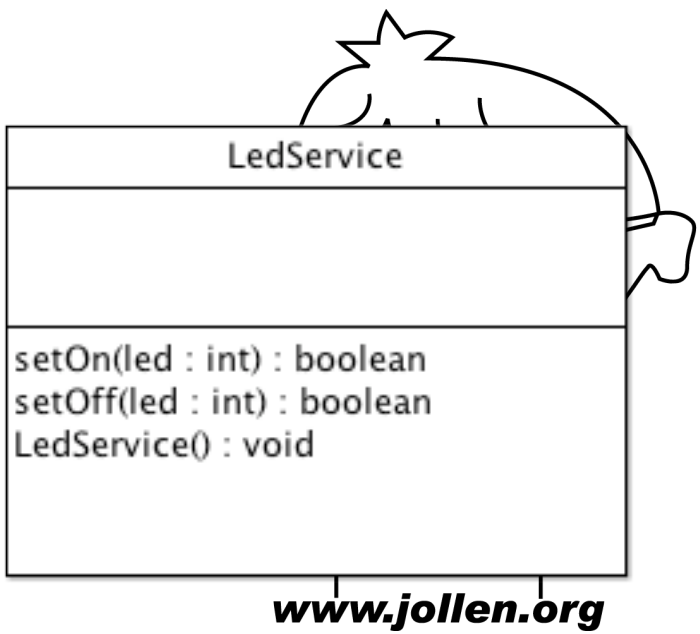
- 把LedService變成「Remotable Object」

```
public final class LedService
                extends ILedService.Stub ~~Binder~~ {
        ...
}
```

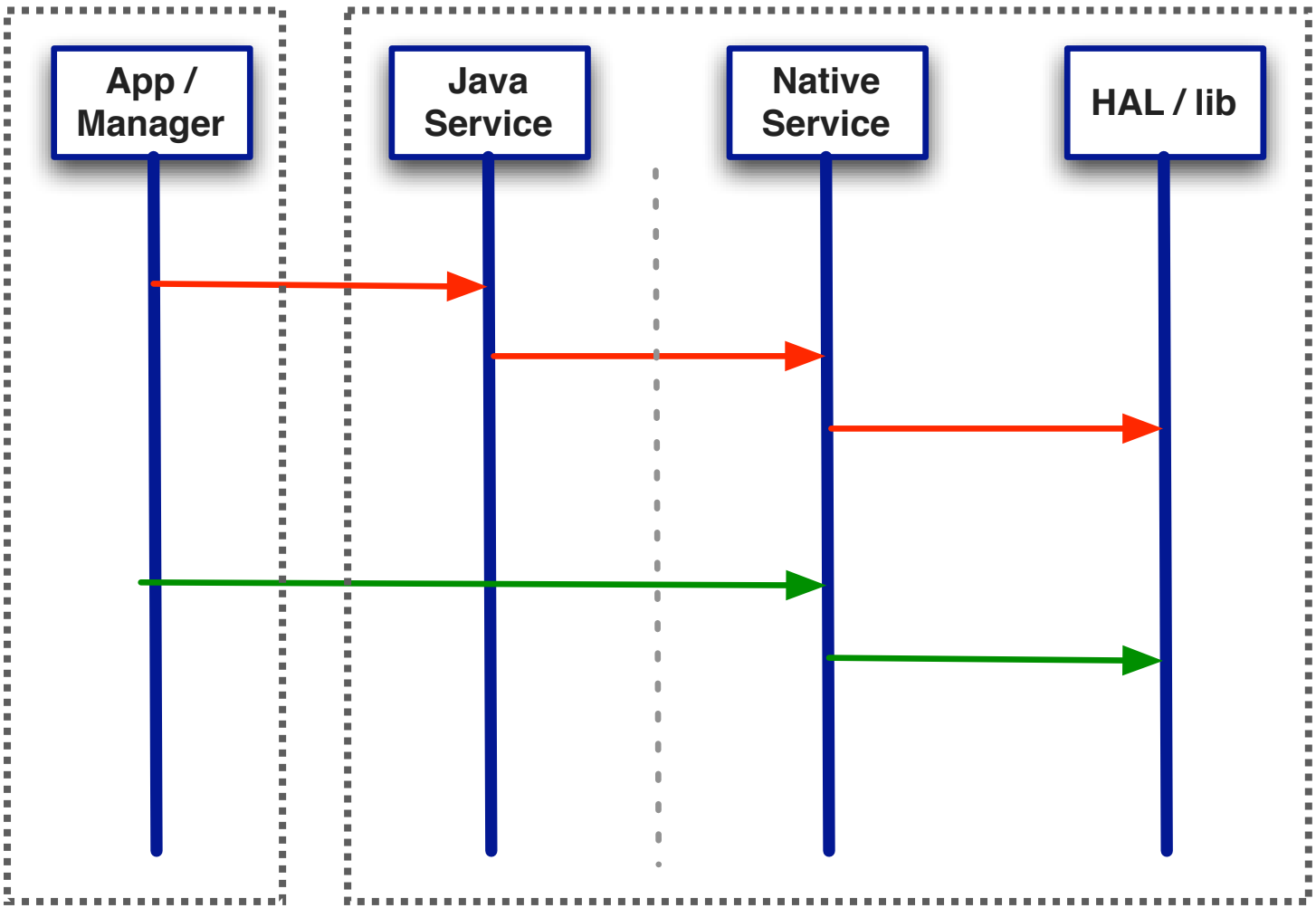


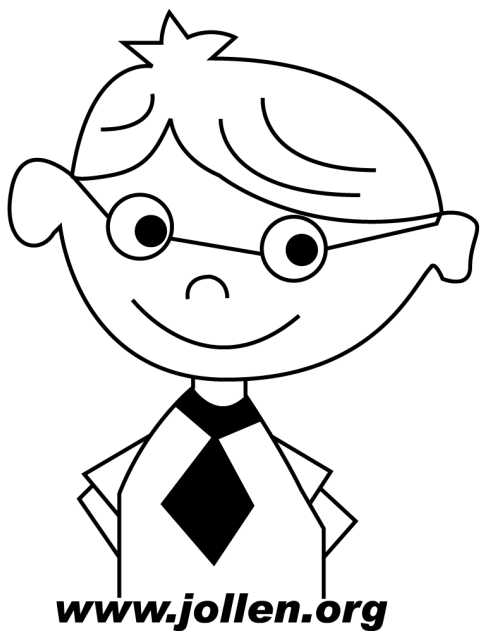
實作 ILedService.Stub

```
public class LedService extends ILedService.Stub {
    public LedService(Context context) {

    }

    public boolean setOn(int n) {

    }

    public boolean setOff(int n) {

    }
}
```



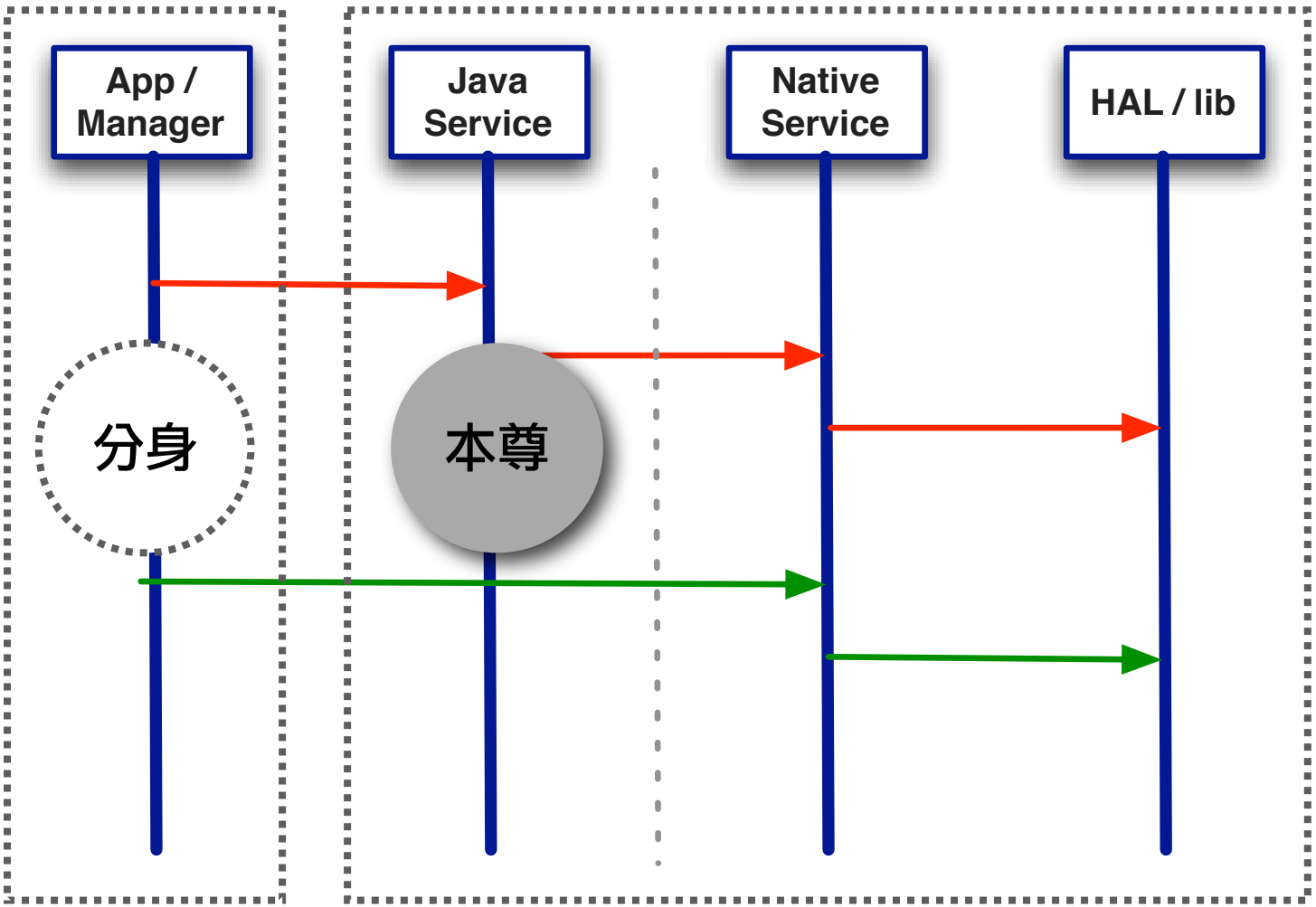
Remote Method Call

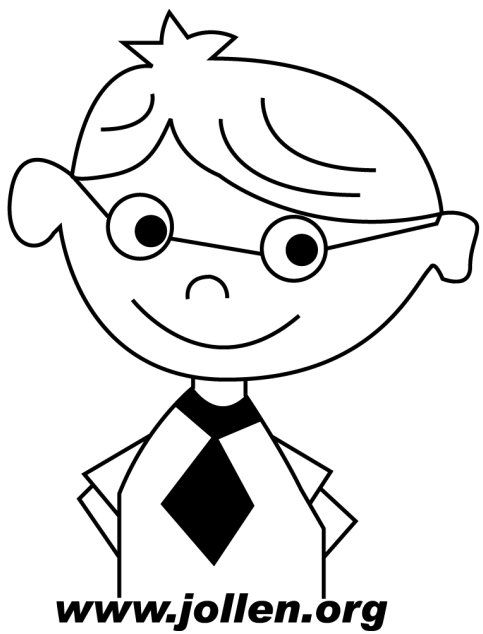




框架的 Proxy Object 與 Remote Object 觀念

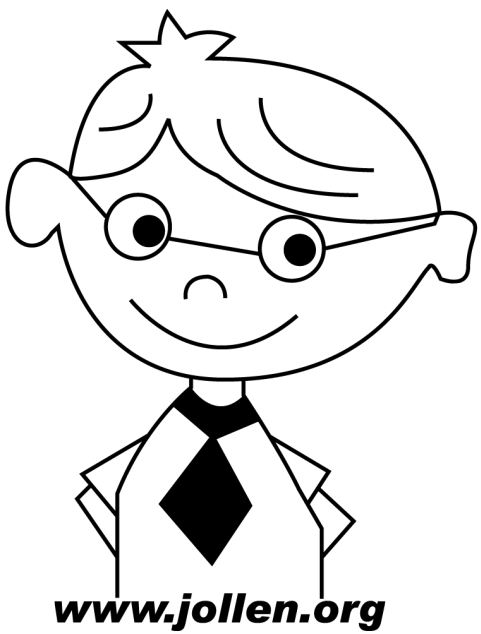
- 一種「代理人」的觀念





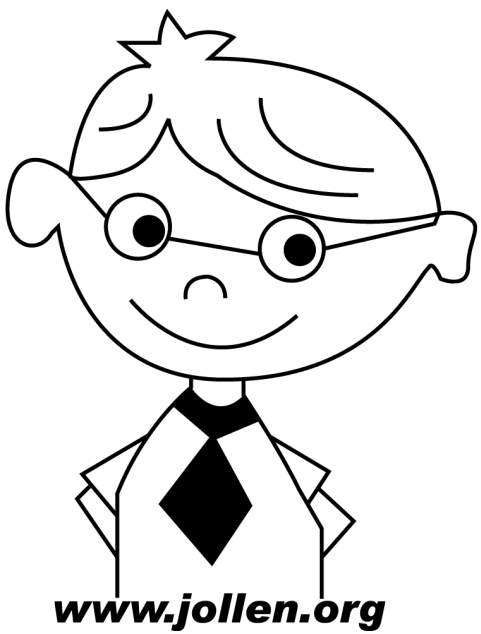
ILedService 是分身

```
private static class Proxy implements ILedService
{
private android.os.IBinder mRemote;
Proxy(android.os.IBinder remote)
{
mRemote = remote;
}
public boolean setOn(int led) throws android.os.RemoteException
{
...
}
public boolean setOff(int led) throws
android.os.RemoteException
{
...
}
static final int TRANSACTION_setOn =
(IBinder.FIRST_CALL_TRANSACTION + 0);
static final int TRANSACTION_setOff =
(IBinder.FIRST_CALL_TRANSACTION + 1);
}
```



本尊實作分身的 Interface

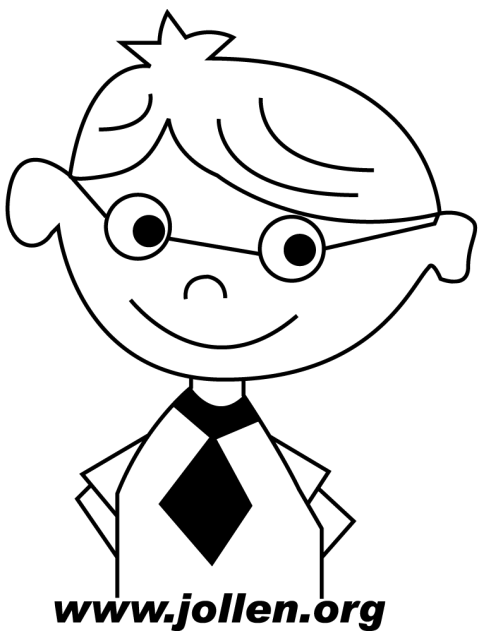
```
public final class LedService extends ILedService.Stub {
    public LedService(Context context) {
        _init();
    }

    public boolean setOn(int n) {
        return _set_on(n);
    }

    public boolean setOff(int n) {
        return _set_off(n);
    }

    private static native boolean _init();
    private static native boolean _set_on(int led);
    private static native boolean _set_off(int led);
}
```



Binder & IBinder 的設計與應用

- Binder 讓物件成為 Remotable Object
- IBinder 是與 Remotable Object 溝通的介面

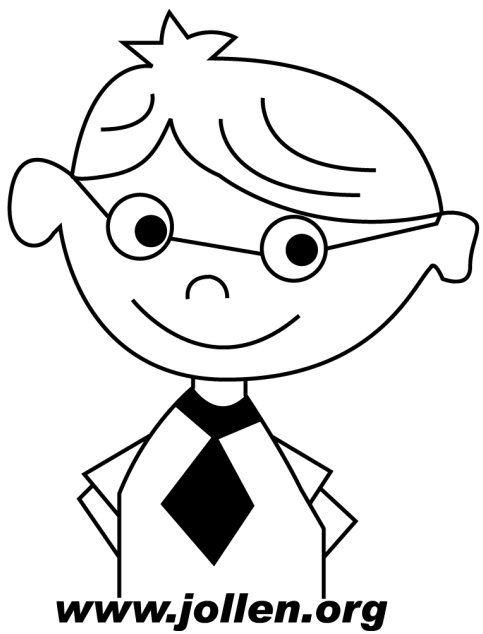
```
private IBinder mRemote;
mRemote = ServiceManager.getService(Context.LED_SERVICE);
```



Proxy Object 的應用

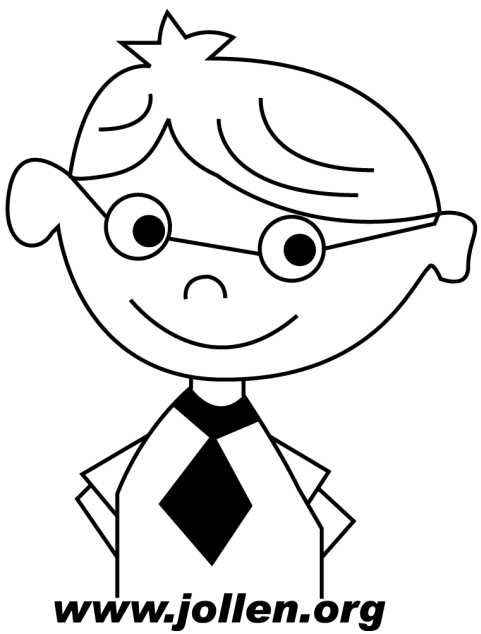
- ILedService.Stub.asInterface() 可取得 ILedService 的 Proxy Object
- 普遍使用在應用開發上、使用 AIDL

```
private IBinder mRemote;
mRemote = ServiceManager.getService(Context.LED_SERVICE);

private ILedService mLedService;
mLedService = ILedService.Stub.asInterface(mRemote);
```



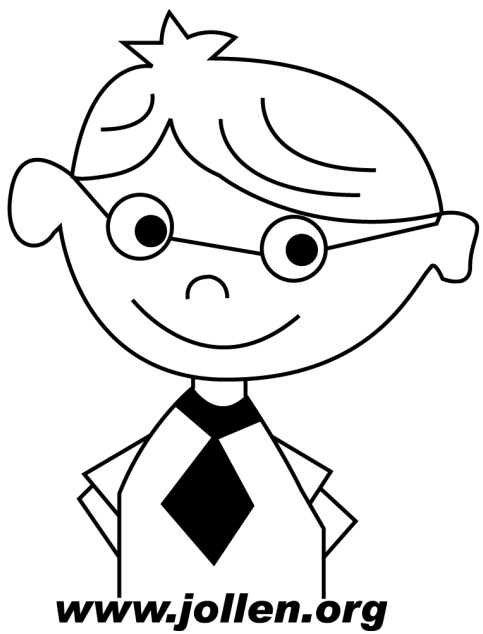
取得 Proxy Object

- ILedService.Stub.Proxy()可為我們取得ILedService的Proxy Object
- 透過分身來與遠端object溝通



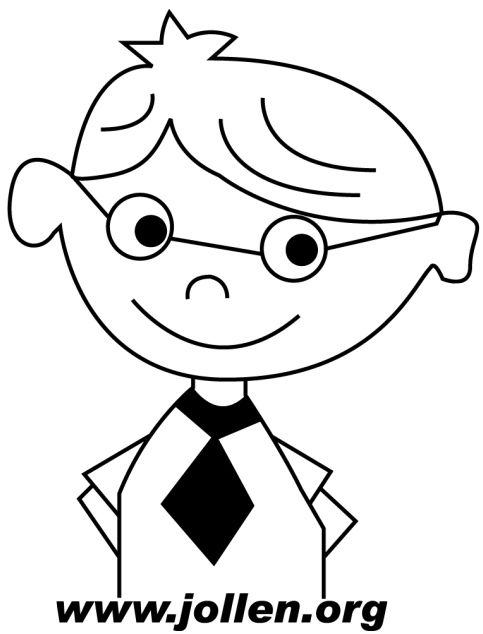
Proxy Object 應用範例

```
public class LedManager
{
   private ILedService mLedService;

    public LedManager() {
        mLedService = ILedService.Stub.asInterface(
                ServiceManager.getService("led"));
    }

    public LedOn(int n) {
     mLedService.setOn(n);
    }

    public LedOff(int n) {
     mLedService.setOff(n);
    }
}
```



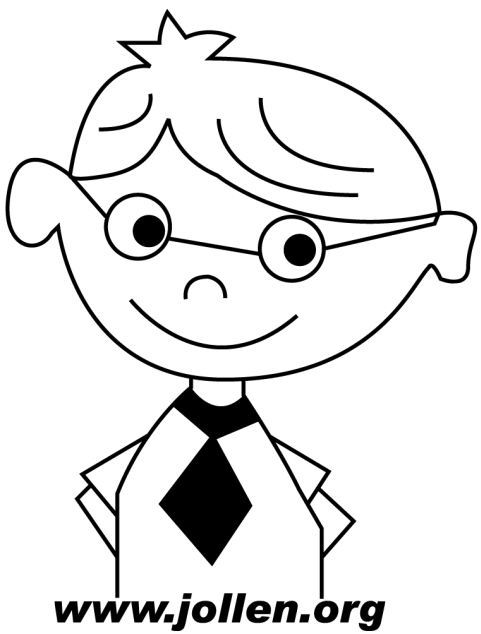
ILedService.asInterface()

```
public static ILedService asInterface(android.os.IBinder obj)
{
        if ((obj==null)) {
            return null;
        }

        android.os.IInterface iin =
(android.os.IInterface)obj.queryLocalInterface(DESCRIPTOR);

        if (((iin!=null)&&(iin instanceof ILedService))) {
            return ((ILedService)iin);
        }
        return new ILedService.Stub.Proxy(obj);
}
```



啟動自定的 Service

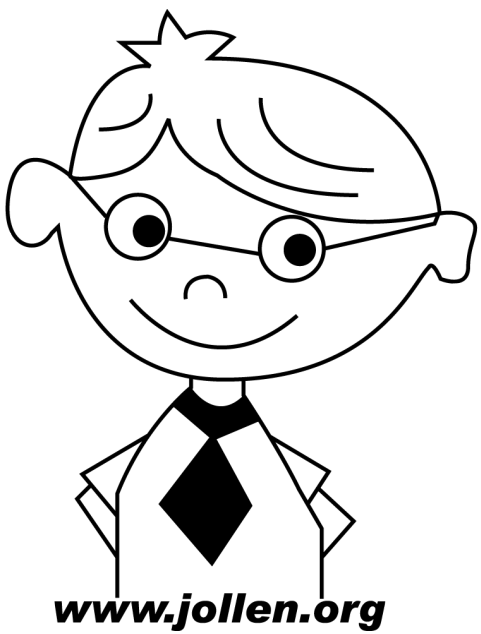
- 啟動 LedService 的時機有二個
  - SystemServer 啟動時
  - Load Class 時
- 透過 ServiceManager



加入 My Service

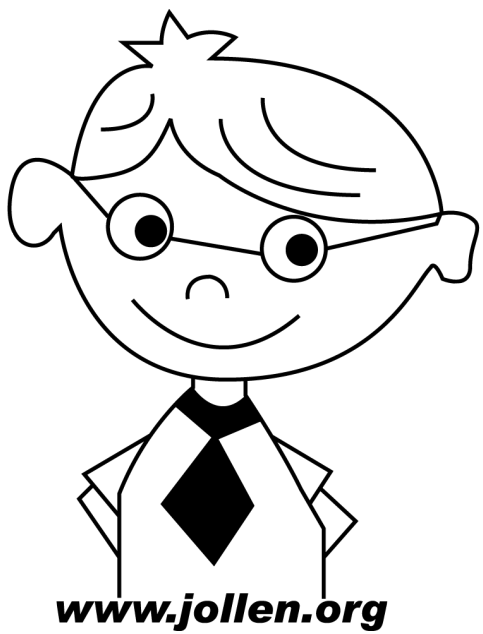
- 加入 Service
  - public static void addService(
    String name,
    IBinder service)
- ServiceManager

ServiceManager.addService(“led”, led);



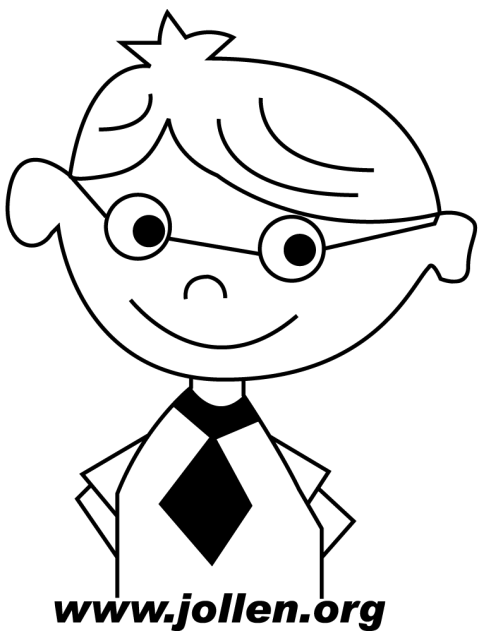
android.app.Activity

- 與使用者互動的物件
- Activity 類別（class）負責建立視窗（window），我們可以透過 View 類別將UI放置在視窗上
- 當 Activity 被啟動（active）或執行（running）時，就是在 foreground（前景）模式，在 foreground 模式的 Activity 會被顯示在螢幕上



android.app.Service

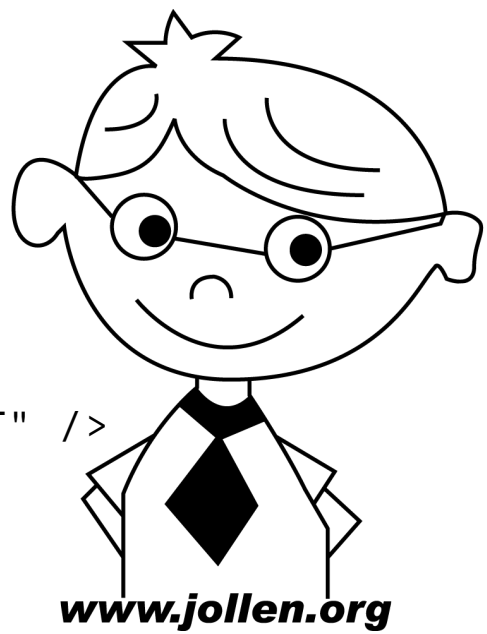
- 在 Android 應用程式裡，有一種沒有 UI 的類別（android.app.Service），稱之為 Service
- Service 是一個 background process（背景程序），透過背景程序，我們可以實作一些不需要 UI 的功能，例如：在背景撥放音樂



In Seperated Process

- Android 應用程式的主物件
  - Main Component
- startActivity() 與 startService()
  - 啟動 Activity/Service 物件
  - 皆可執行在獨立的 Process

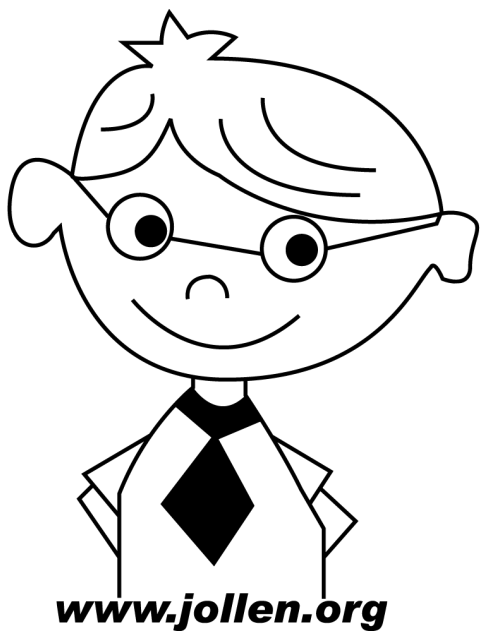
```
<activity android:name=".HelloMokoSettings"
          android:process=":mokoSettings"
          android:label="@string/app_name">
  <intent-filter>
    <action android:name="com.moko.hello.SETTINGS" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```



錯誤處理機制

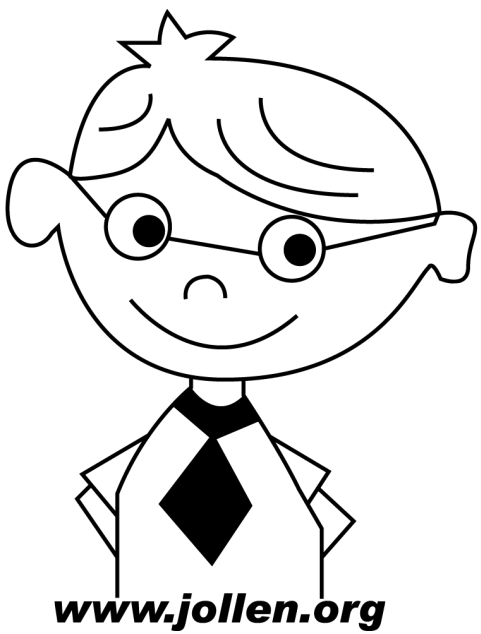
- 最簡單的錯誤處理方式
  - Message
  - android.os.Message
- 1. 先做一個 Handler
  - android.os.Handler

```
class MyHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        ...
    }
}
```



Handler 範例

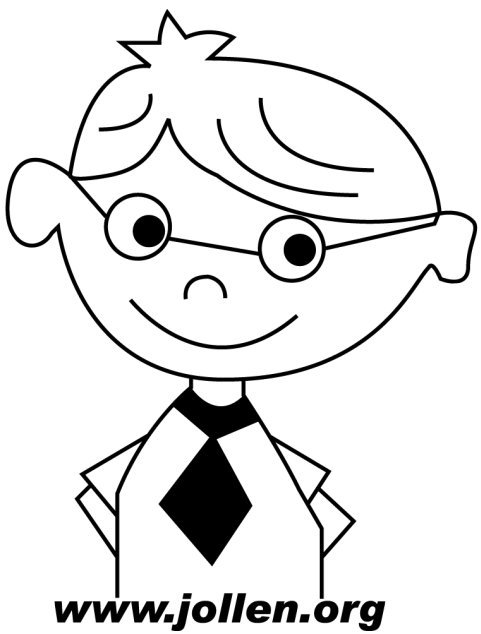
```
LedHandler myHandler = new LedHandler();

class LedHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case LED_HARDWARE_ERROR:
          /* do something */
           break;
        }
        super.handleMessage(msg);
    }
}
```



送出 Message

- 2. 送出 Message

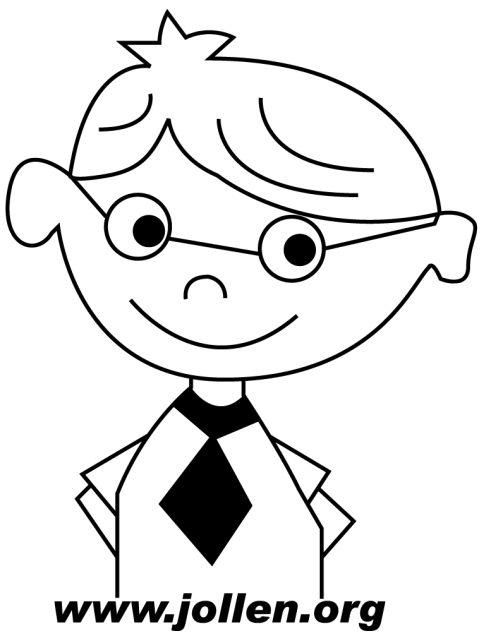
```
Message m = new Message();

m.what = LED_HARDWARE_ERROR;
myHandler.sendMessage(m);
```



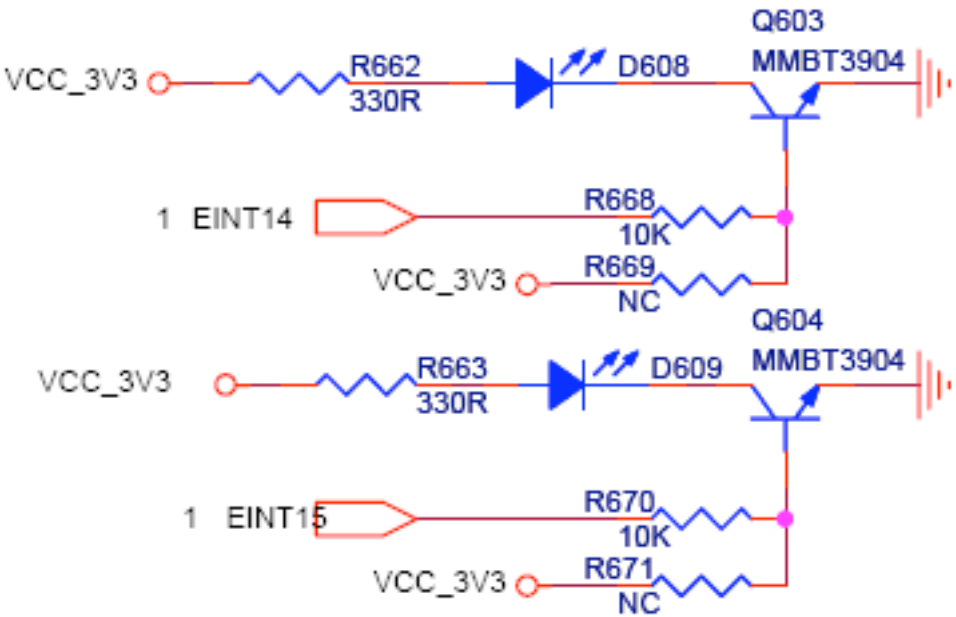
第09堂課： 分組專題探討

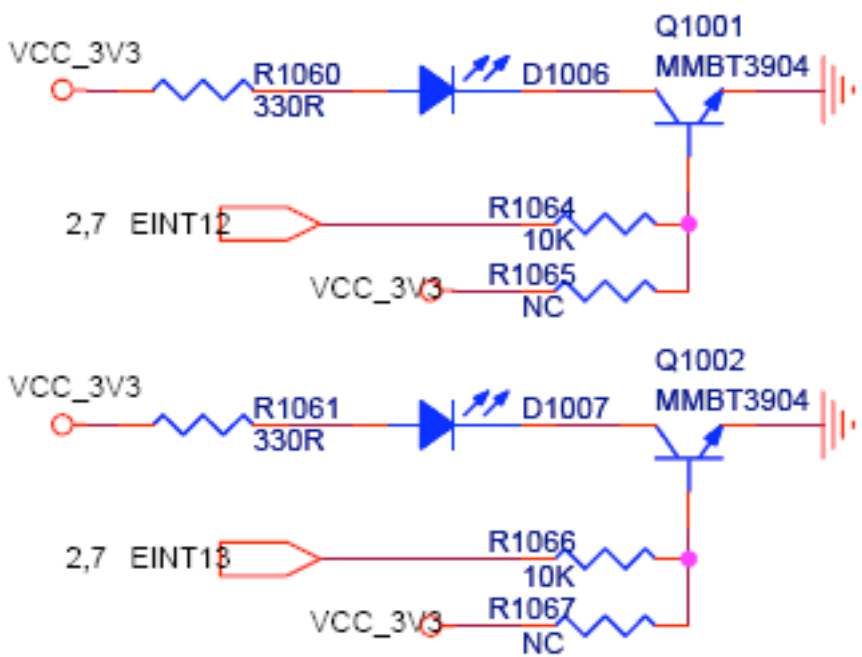
本堂主題

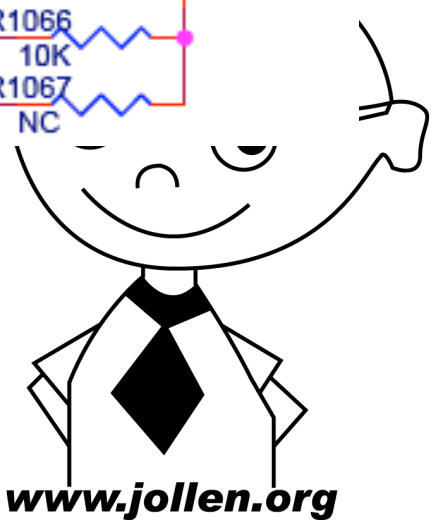
第九堂	分組專題探討
9.1	Example 1: 整合驅動程式至Android 框架
9.2	Example 2: 撰寫Anroid應用程式以控制LED
9.3	Example 3: 標準的Android HAL 架構
9.4	Example 4: Motor HAL Stub



MokoidBoard LED 接腳圖







閱讀 S3C6410 Datasheet

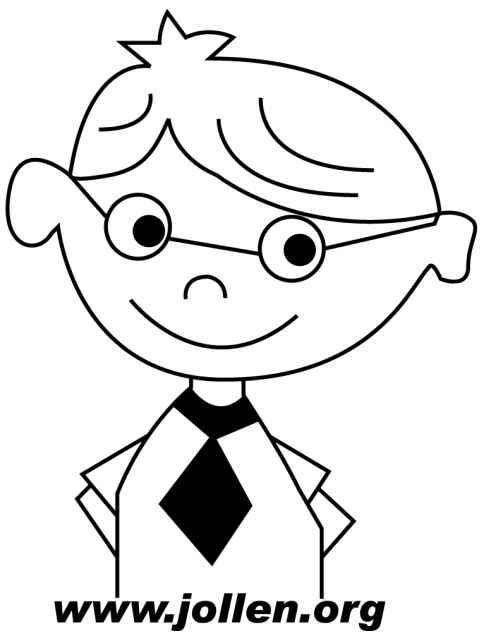
S3C6410 includes 187multi-functional input/output port pins. There are 17 ports as listed below:

PortName	Number of Pins.	Muxed pins	Power Inform.
GPA port	8	UART/EINT	1.8~3.3V
GPB port	7	UART/IrDA/I2C/CF/Ext.DMA/EINT	1.8~3.3V
GPC port	8	SPI/SDMMC/I2S_V40/EINT	1.8~3.3V
GPD port	5	PCM/I2S/AC97/EINT	1.8~3.3V
GPE port	5	PCM/I2S/AC97	1.8~3.3V
GPF port	16	CAMIF/PWM/EINT	1.8~3.3V
GPG port	7	SDMMC/EINT	1.8~3.3V
GPH port	10	SDMMC/KEYPAD/CF/I2S_V40/EINT	1.8~3.3V
GPI port	16	LCD	1.8~3.3V
GPJ port	12	LCD	1.8~3.3V
GPK port	16	HostIF/HIS/KEYPAD/CF	1.8~3.3V
GPL port	15	HostIF/KEYPAD/CF/OTG/EINT	1.8~3.3V
GPM port	6	HostIF/CF/EINT	1.8~3.3V
GPN port	16	EINT/KEYPAD	1.8~3.3V
GPO port	16	MemoryPort0/EINT	1.8~3.3V
GPP port	15	MemoryPort0/EINT	1.8~3.3V
GPQ port	9	MemoryPort0/EINT	1.8~3.3V

EINT GPN

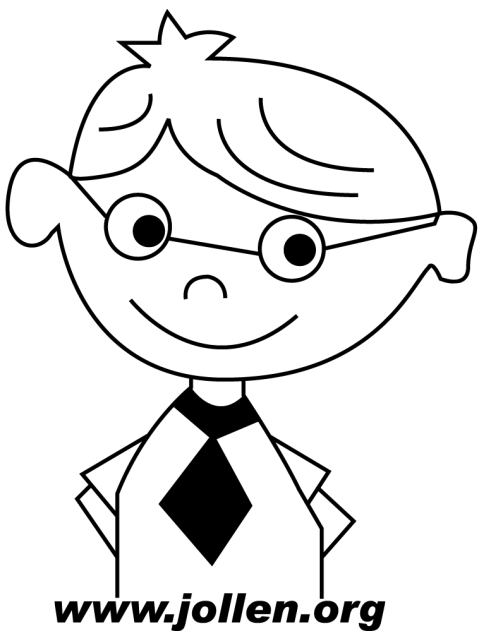
There are three control registers including GPNCON, GPNDAT and GPNPUD in the Port N Control Registers. GPNCON, GPNDAT and GPNPUD are alive part.

Register	Address	R/W	Description	Reset Value
GPNCON	0x7F008830	R/W	Port N Configuration Register	0x00
GPNDAT	0x7F008834	R/W	Port N Data Register	Undefined
GPNPUD	0x7F008838	R/W	Port N Pull-up/down Register	0x55555555



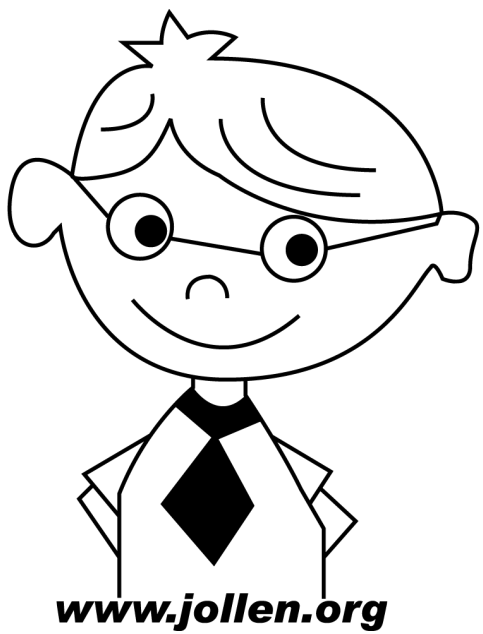
GPNCON

GPN12	[25:24]	00 = Input 10 = Ext. Interrupt[12]	01 = Output 11 = Reserved]	00
GPN13	[27:26]	00 = Input 10 = Ext. Interrupt[13]	01 = Output 11 = Reserved	00
GPN14	[29:28]	00 = Input 10 = Ext. Interrupt[14]	01 = Output 11 = Reserved	00
GPN15	[31:30]	00 = Input 10 = Ext. Interrupt[15]	01 = Output 11 = Reserved	00



GPNDAT

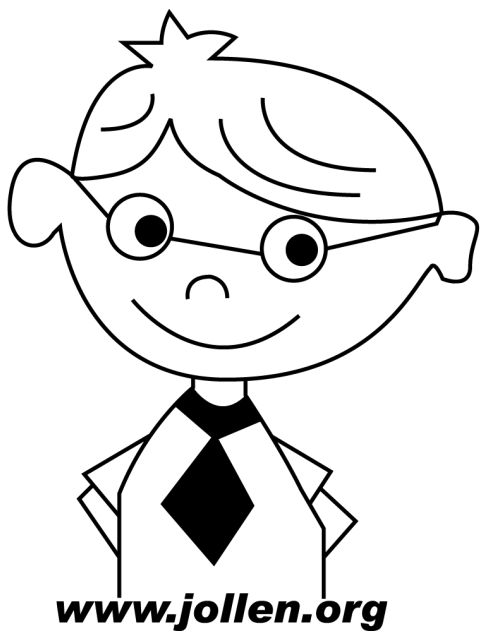
GPNDAT	Bit	Description
GPN[15:0]	[15:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.



DC Motor 驅動程式控制命令

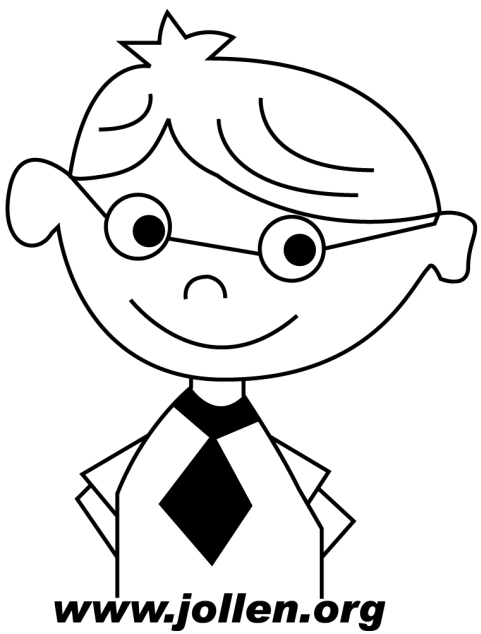
```
/* ioctl_dcm.h */
#define DC_IOCTL_START          1
#define DC_IOCTL_STOP           2
#define DC_IOCTL_SPEED_UP       3
#define DC_IOCTL_SPEED_DOWN     4
```

```
/* test_dcm.c */

/* 啟動 */
ioctl(fd,DC_IOCTL_START,NULL);
/* 停止 */
ioctl(fd,DC_IOCTL_STOP,NULL);
/* 加速 */
ioctl(fd,DC_IOCTL_SPEED_UP,NULL);
/* 減速 */
ioctl(fd,DC_IOCTL_SPEED_DOWN,NULL);
```



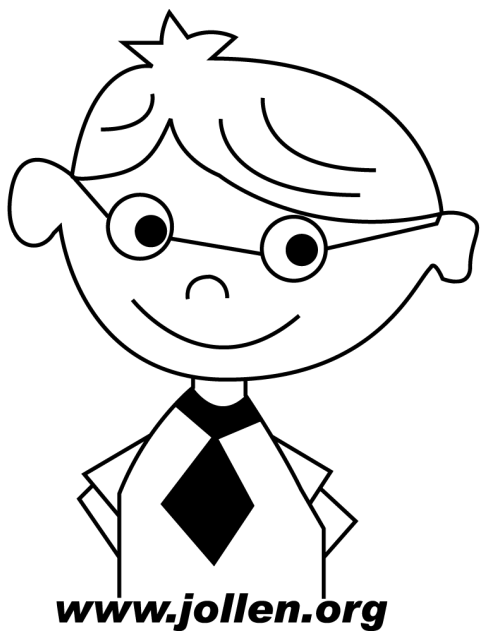
JNI 與 Motor Control Library

- JNI 的處理
  - 製作 libmokoid_runtime_dcm.so
- Motor Control Library
  - 根據 Linux 驅動程式提供的控制命令
  - 實作 libdcm.so

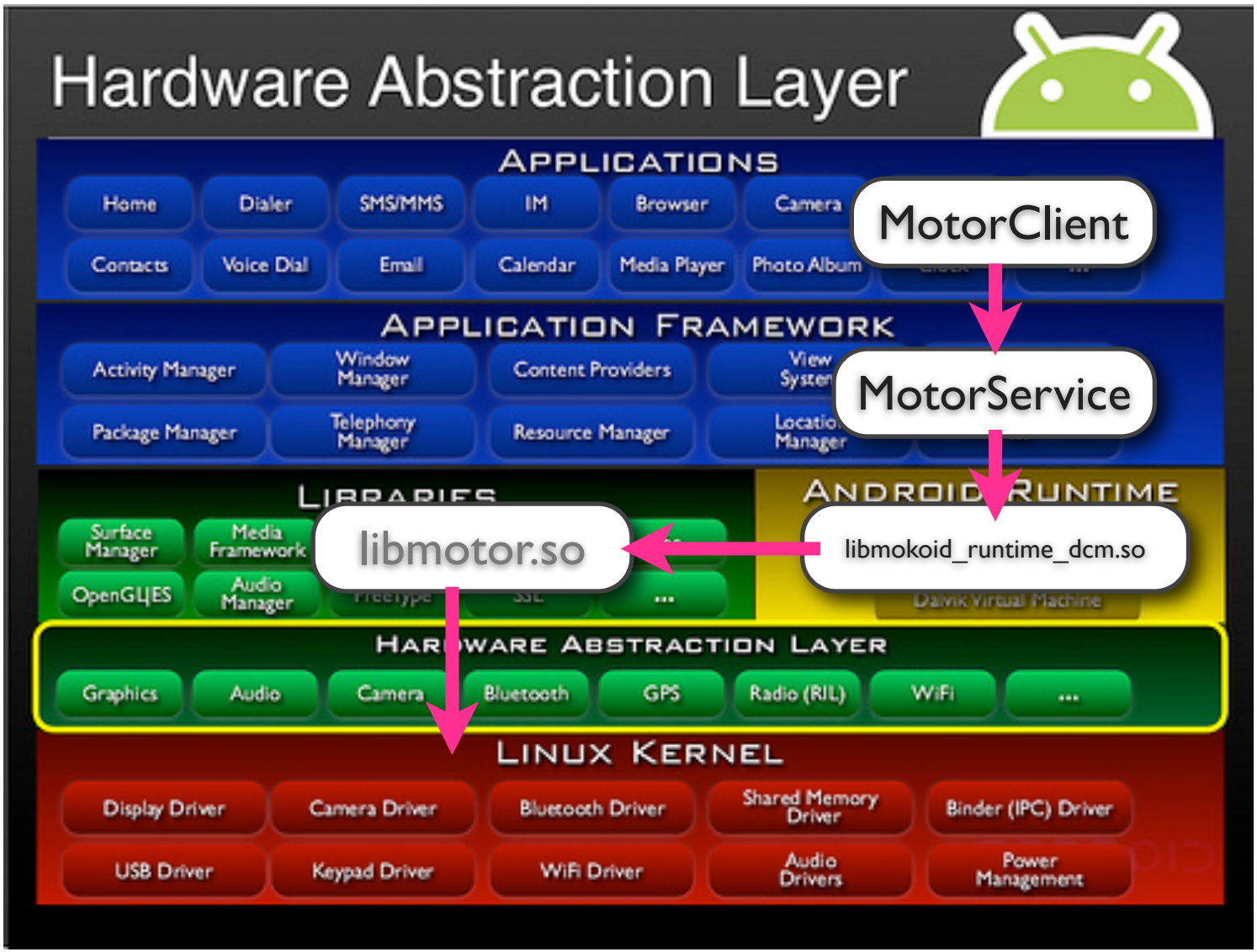


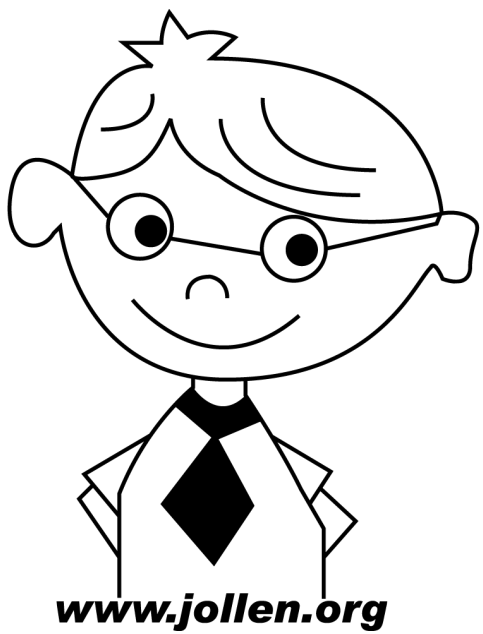
整合 DC Motor

- 將 Android 移植到 MokoidBoard
  - 移植週邊：例如 LED、DC Motor
  - 假設已完成開發、如何將程式碼整合至 Android 並製作 image
- 整合老師提供的 DC Motor 程式碼
  - MotorService.java
  - MotorClient.java
  - libmotor.c
  - com_mokoid_server_MotorService.cpp



DC Motor 實作


Hardware Abstraction Layer
APPLICATIONS
Home
Dialer
SMS/MMS
IM
Browser
Camera
Contacts
Voice Dial
Email
Calendar
Media Player
Photo Album
MotorClient
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
Package Manager
Telephony Manager
Resource Manager
MotorService
LIBRARIES
Surface Manager
Media Framework
OpenGL|ES
Audio Manager
libmotor.so
ANDROID RUNTIME
libmokoid_runtime_dcm.so
Dalvik Virtual Machine
HARDWARE ABSTRACTION LAYER
Graphics
Audio
Camera
Bluetooth
GPS
Radio (RIL)
WiFi
LINUX KERNEL
Display Driver
Camera Driver
Bluetooth Driver
Shared Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management


www.jollen.org


moko365
嵌入式產品設計 專業培訓 廠訓諮詢
Mobile Communication
professional 365 days a year
仕橙3G教室
WWW.MOKO365.COM
專注Android技術的培訓專家
仕橙3G教室的「Android應用開發」與「Android驅動程式」課程，是廣受台灣與大陸技術人員好評的專業課程。仕橙3G教室採行教練式訓練課程，業界唯一，訓練您由概念、理論到架構，以及協助建立基本實作能力。

感謝參與本次培訓課程