

Jollen 的 Android Framework in A Nutshell

| 演講 · 台北場 |

主辦單位：仕橙3G教室 (Moko365 Inc.)

活動宗旨

Android 框架設計與系統程式有很密切的關係，用「總而言之」的方式來說明框架設計與系統程式的設計，是「Jollen 的 Android Framework in A Nutshell」演講的理念。本演講從「Android框架設計」與「Android 系統程式」切入，期望以簡約的方式，說明Android作業系統的整體結構與設計理念，進而給予需要開發自有Android作業系統分支的技術人員一個良好的進入點。

活動精神

Android 框架並沒有想像中那麼複雜或難以理解，「 Jollen 的 Android Framework in A Nutshell」演講，希望與大家分享技術研究成果，並打破這一層看似複雜的技術議題。本活動邀請到在Android框架設計、系統程式以及Linux 驅動程式都有很深入研究的講師 Jollen Chen 先生，為大家講解 Android Framework的結構與觀念。

活動官網

<http://www.jollen.org/androidframework>

講題大綱

時程	內容	講師	時間
13:00~13:30	報到		
13:40~14:00	第一堂：Android HAL實作	Jollen Chen	20分鐘
14:00~14:50	第二堂：Android Service系統結構	Jollen Chen	50分鐘
14:50~15:40	第三堂：Android Framework架構	Jollen Chen	50分鐘
15:40~16:00	Tea Time		
16:00~16:40	第四堂：Android 系統程式與原理	Jollen Chen	40分鐘
16:40~17:00	第五堂：Android 技術發展策略	Jollen Chen	20分鐘
17:00~17:10	Q&A		

* 詳細講題內容請參見活動官網！

2010
四 3 / 11

下午1:40~5:10 (下午1:00開始報到)

地點：詳見官網

參加費用

報名費用：免費。

珍惜免費資源，本活動將酌收工本費 NT\$500元整（費用包含：講義費、場地、茶點等）。缺席者恕不退費。

報名流程

- 填寫線上報名表單
- 報名資料審核通過後、寄發繳費通知
- 繳費確認後、寄發報名成功確認函

注意事項

- 因場地座位有限，我們將依「完成繳費」順序，為您保留座位
- 未出席者，我們將不另行退費，當日講義將以快遞方式寄送到您指定地點
- 主辦單位保留本活動相關內容的變動權利



講師简介

Jollen Chen(陳俊宏先生)，台灣最資深的嵌入式Linux專家與講師。Jollen在嵌入式Linux與Linux驅動程式領域擁有豐富的顧問與授課經驗，並曾參與過多項已上市的嵌入式產品研發。Jollen熱衷於培訓領域，並成立了 Jollen's Consulting工作室；其工作室致力於專業的嵌入式Linux與Android 課程開發，產出的課程與技術研究品質深獲肯定，服務客戶遍及台灣、美國與大陸企業。Jollen過去負責知名的開放手機計畫 Openmoko 大中華區的行銷與推廣工作；現任職於深圳手機公司，擔任架構長職務，負責手持產品的系統架構設計與產品技術規劃工作。

Jollen 的 Android Framework in A Nutshell

| 演講 · 新竹場 |

主辦單位：財團法人自強工業科學基金會

活動宗旨

Android 框架設計與系統程式有很密切的關係，用「總而言之」的方式來說明框架設計與系統程式的設計，是「Jollen 的 Android Framework in A Nutshell」演講的理念。本演講從「Android 框架設計」與「Android 系統程式」切入，期望以簡約的方式，說明 Android 作業系統的整體結構與設計理念，進而給予需要開發自有 Android 作業系統分支的技術人員一個良好的進入點。

活動精神

Android 框架並沒有想像中那麼複雜或難以理解，「Jollen 的 Android Framework in A Nutshell」演講，希望與大家分享技術研究成果，並打破這一層看似複雜的技術議題。本活動邀請到在 Android 框架設計、系統程式以及 Linux 驅動程式都有很深入研究的講師 Jollen Chen 先生，為大家講解 Android Framework 的結構與觀念。

活動官網

<http://www.jollen.org/androidframework>

講題大綱

時程	內容	講師	時間
13:00~13:30	報到		
13:40~14:00	第一堂：Android HAL 實作	Jollen Chen	20分鐘
14:00~14:50	第二堂：Android Service 系統結構	Jollen Chen	50分鐘
14:50~15:40	第三堂：Android Framework 架構	Jollen Chen	50分鐘
15:40~16:00	Tea Time		
16:00~16:40	第四堂：Android 系統程式與原理	Jollen Chen	40分鐘
16:40~17:00	第五堂：Android 技術發展策略	Jollen Chen	20分鐘
17:00~17:10	Q&A		

* 詳細講題內容請參見活動官網！

2010
四 4/8

下午1:40~5:10 (下午1:00開始報到)
報名與演講地點：詳見官網

參加費用

報名費用：免費。

珍惜免費資源，本活動將酌收工本費 NT\$500元整（費用包含：講義費、場地、茶點等）。缺席者恕不退費。

報名流程

- 填寫線上報名表單
- 報名資料審核通過後、寄發繳費通知
- 繳費確認後、寄發報名成功確認函

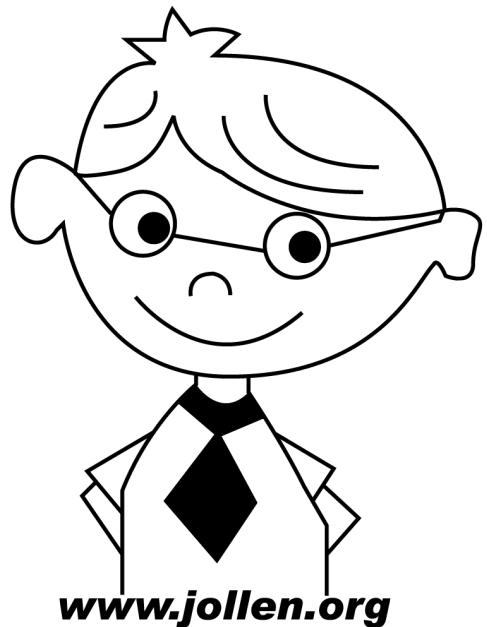
注意事項

- 因場地座位有限，我們將依「完成繳費」順序，為您保留座位
- 未出席者，我們將不另行退費，當日講義將以快遞方式寄送到您指定地點
- 主辦單位保留本活動相關內容的變動權利



講師簡介

Jollen Chen(陳俊宏先生)，台灣最資深的嵌入式 Linux 專家與講師。Jollen 在嵌入式 Linux 與 Linux 驅動程式領域擁有豐富的顧問與授課經驗，並曾參與過多項已上市的嵌入式產品研發。Jollen 热衷於培訓領域，並成立了 Jollen's Consulting 工作室；其工作室致力於專業的嵌入式 Linux 與 Android 課程開發，產出的課程與技術研究品質深獲肯定，服務客戶遍及台灣、美國與大陸企業。Jollen 過去負責知名的開放手機計畫 Openmoko 大中華區的行銷與推廣工作；現任職於深圳手機公司，擔任架構長職務，負責手持產品的系統架構設計與產品技術規劃工作。



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 : 2010/3/28 (六)
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諮詢服務。

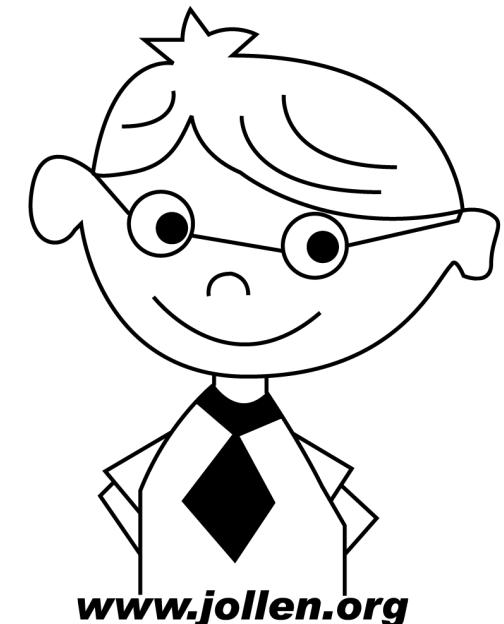
仕橙3G教室將致力於建立可信賴的「行動通訊技術服務品牌」。



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。

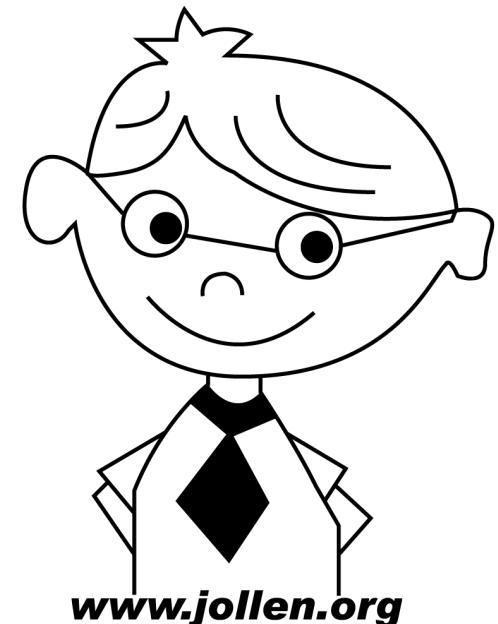
2009 年，Jollen's Consulting 與大陸地區業者合作，提供大陸地區嵌入式 Linux 與 Android 的培訓服務，巡迴上海、北京與深圳等地。Jollen 的嵌入式 Linux 與 Android 課程專業性受到業界肯定；由 2009 年初開始至今，提供台灣、大陸與美國公司內訓服務，總計超過 25 個場次。



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諮詢服務。

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。

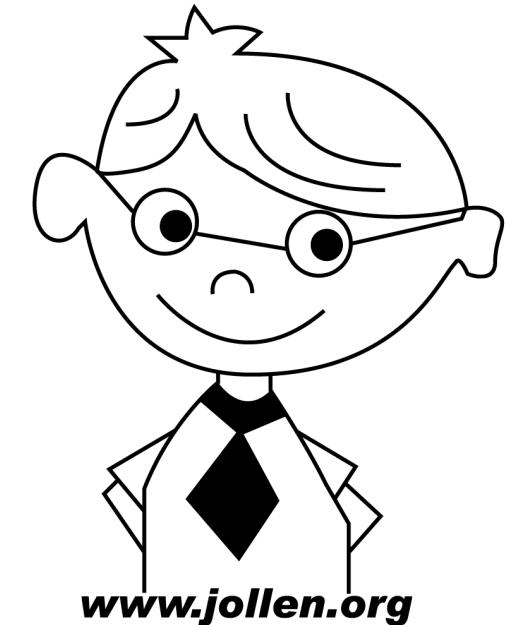




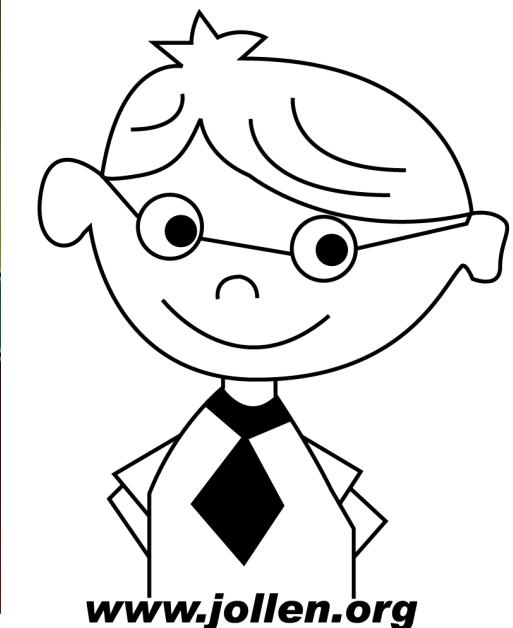
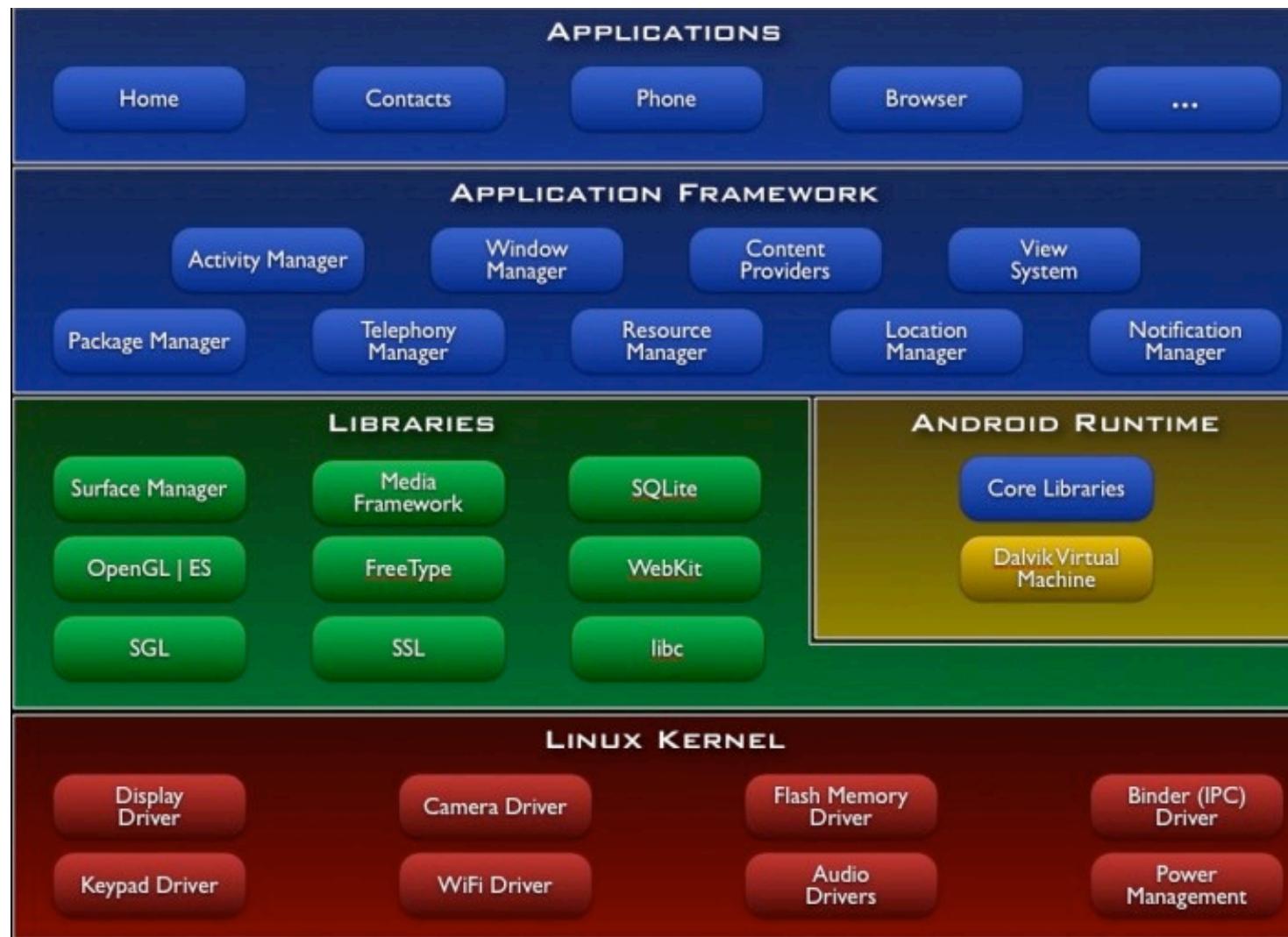
第01堂課：Android OS 開發環境建立與編譯

本堂主題

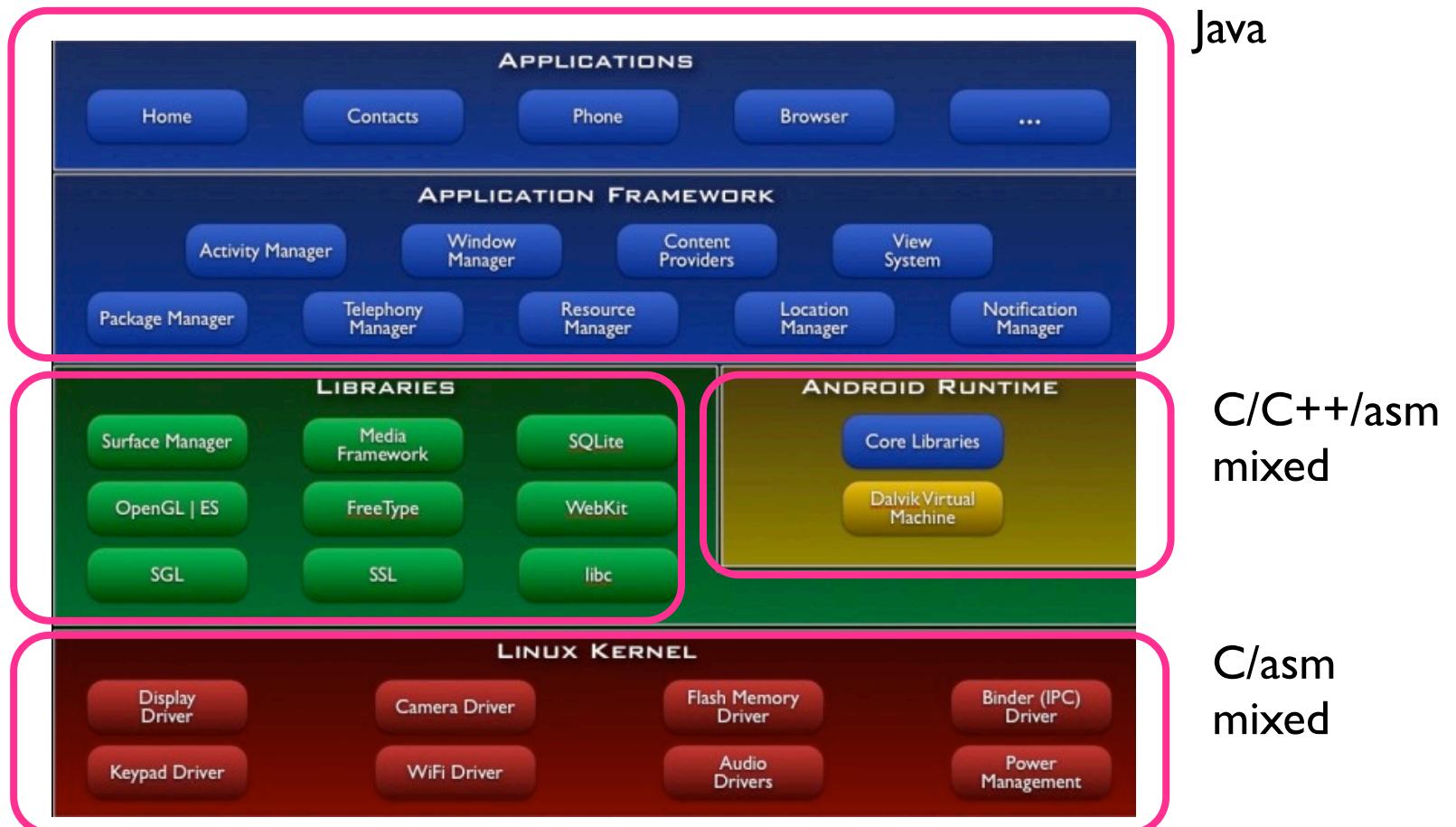
第一堂	Android OS 開發環境建立與編譯
1.1	Android 框架(framework)架構介紹
1.2	如何取得 Android 原始碼
1.3	Android product 分支建立
1.4	編譯 ARMv5+ Android 系統 (image製作)
1.5	使用 Android emulator
1.6	Vanilla Kernel & Android Kernel 編譯



Android 架構



Programming Language



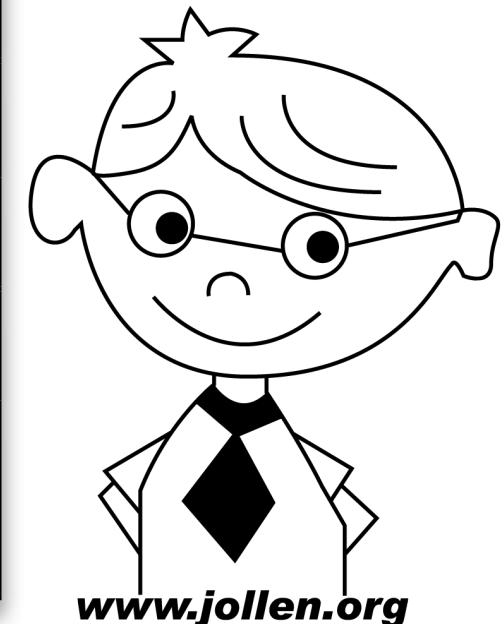
C/C++/asm
mixed

C/C++/asm
mixed

C/asm
mixed

各呈現形式

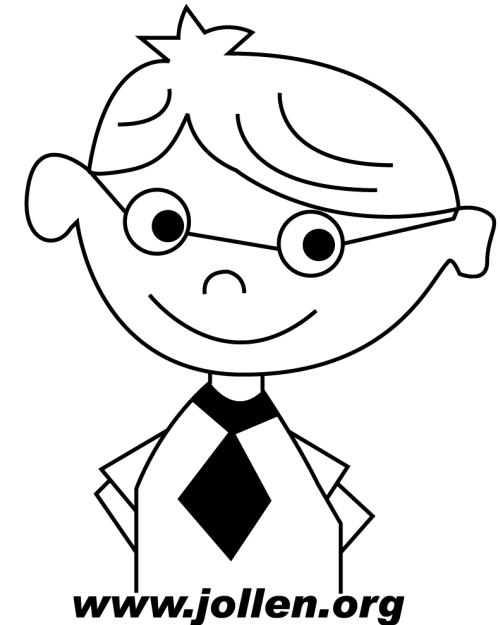
Layer	Form	Language	Ship
應用程式	*.apk	Java	*.apk/system.img
framework	*.jar	Java	system.img
core libraries	*.so	C/C++	system.img
HAL	*.so	C/C++	*.apk/system.img
Dalvik	由 zygote 管理、載入應用程式時產生 Dalvik process(Linux process)		



Application Framework

□ 提供 Android API

- SensorManager
- WifiManager
- TextView
- Button



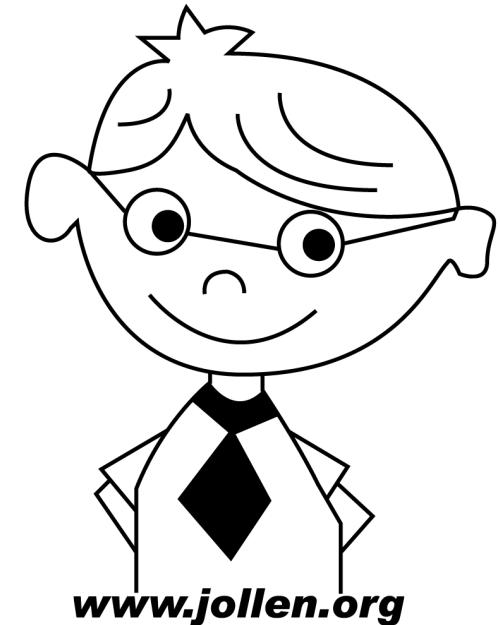
Android Runtime

□ 包含二大部份

- Dalvik VM
- Core Libraries

□ Dalvik VM 在 systems software 部份

- 提供 JNI
- 提供 threading



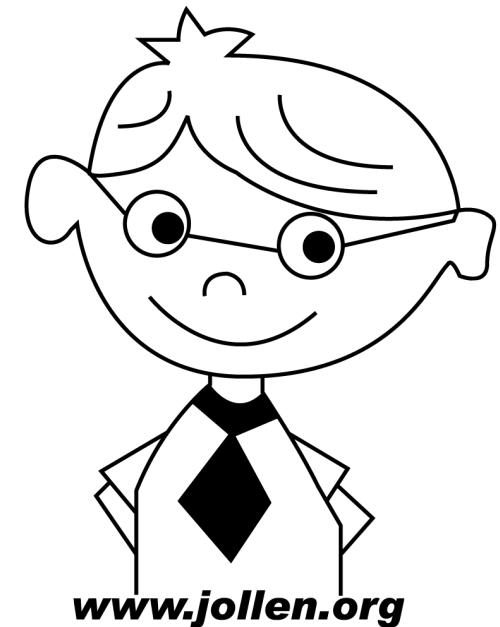
Runtime 的工作

□ 處理 JNI

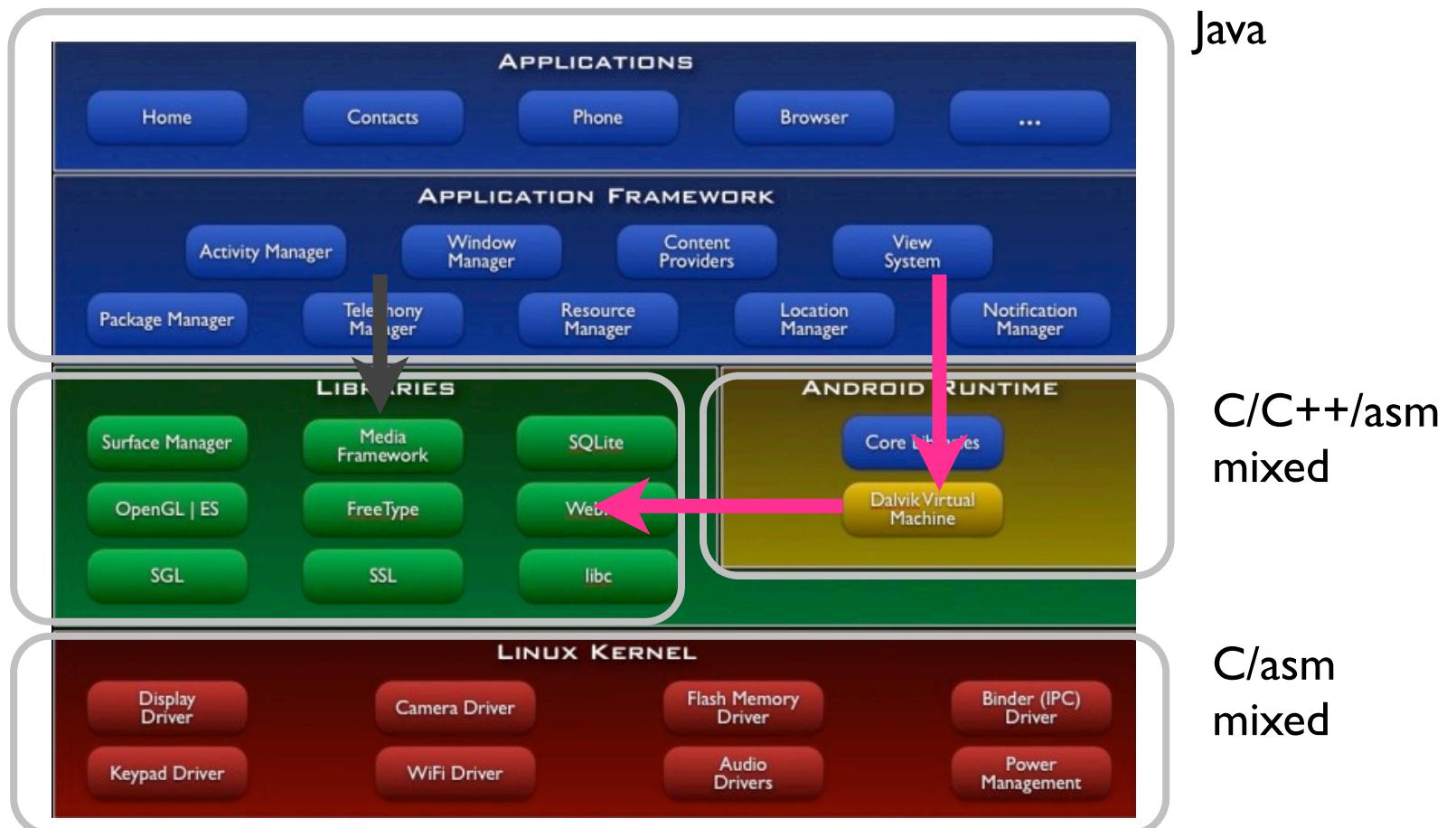
- 執行 native method 時、呼叫 native function

□ native function 以 shared library 形式存在

- native function 以 C/C++ 撰寫

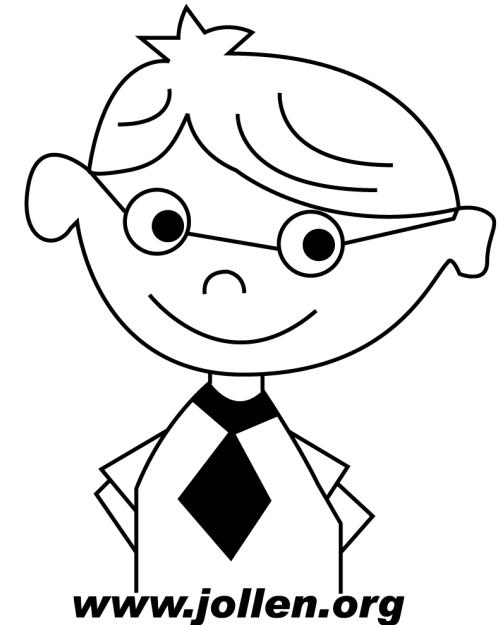


JNI & External Libraries



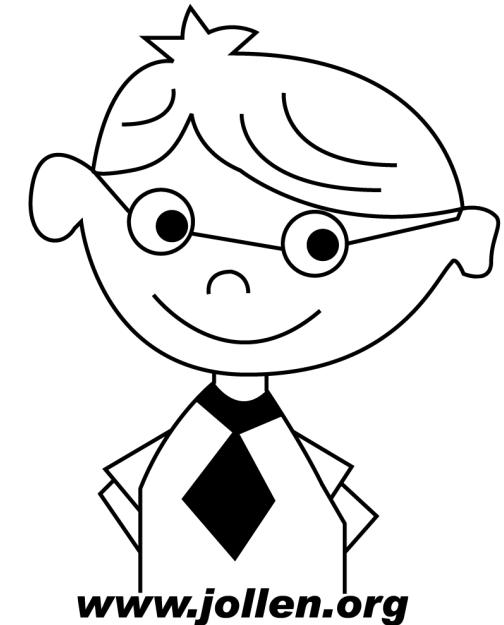
Product Tree

- 以GNU automake做為基礎的編譯架構
 - build/ 以及 Android.mk
 - 巨集語法參考GNU automake手冊
- 分支維護者只需要了解Android build system所提供的「功能」
- 研讀 mokoid-led 範例、了解如何建立自己的 Product Tree

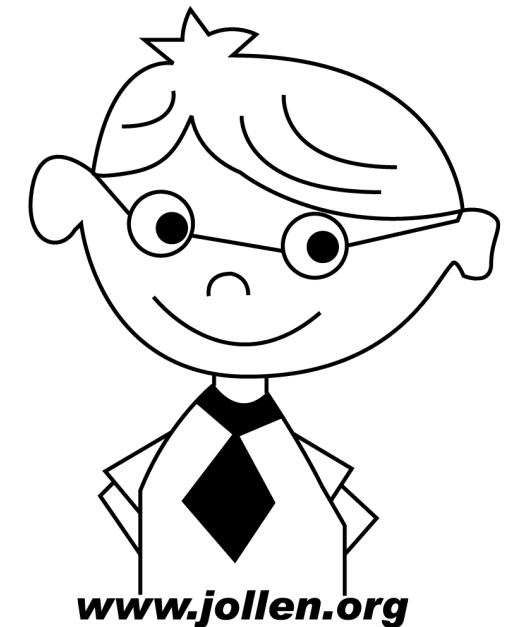
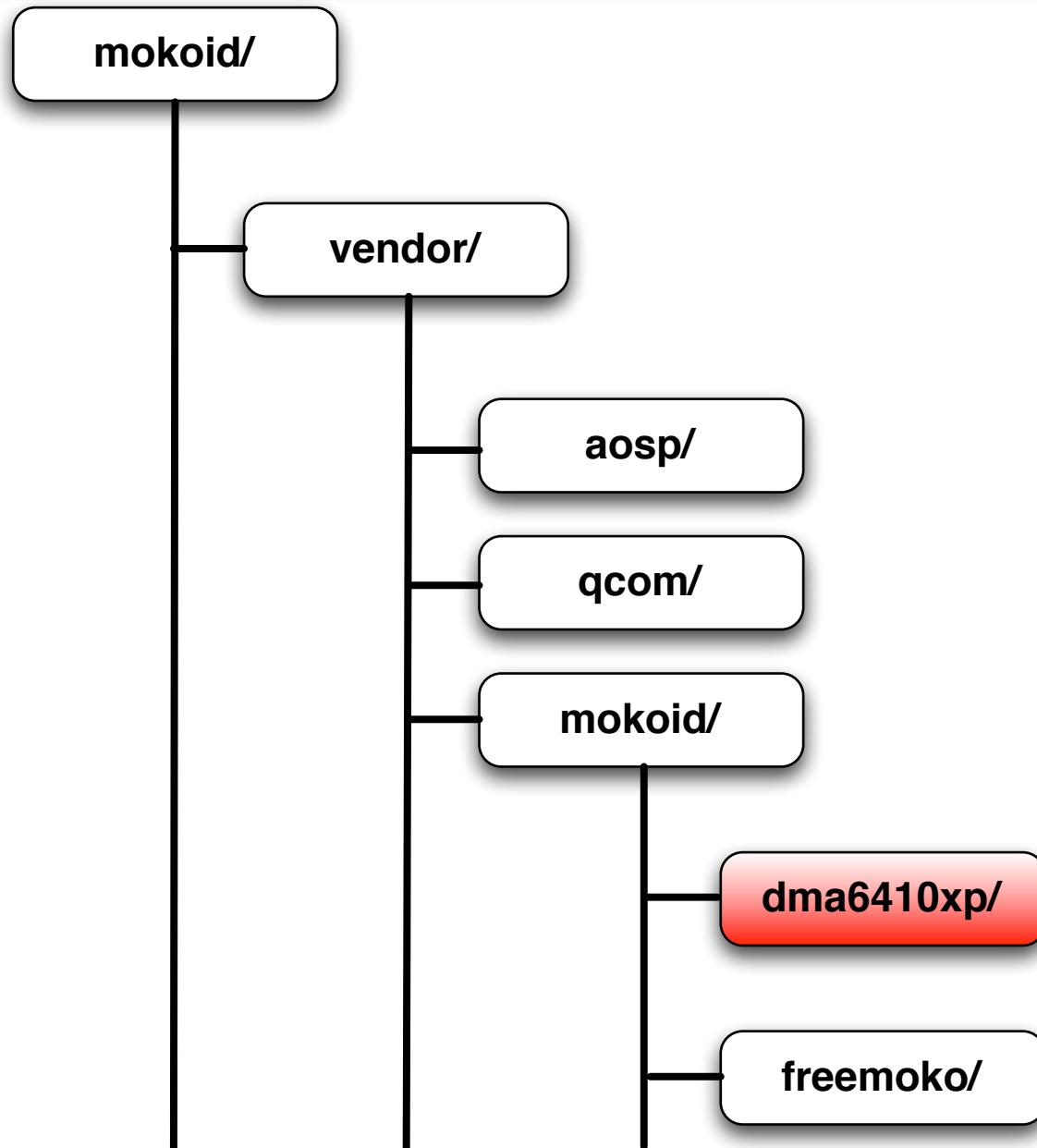


製作 Product Tree

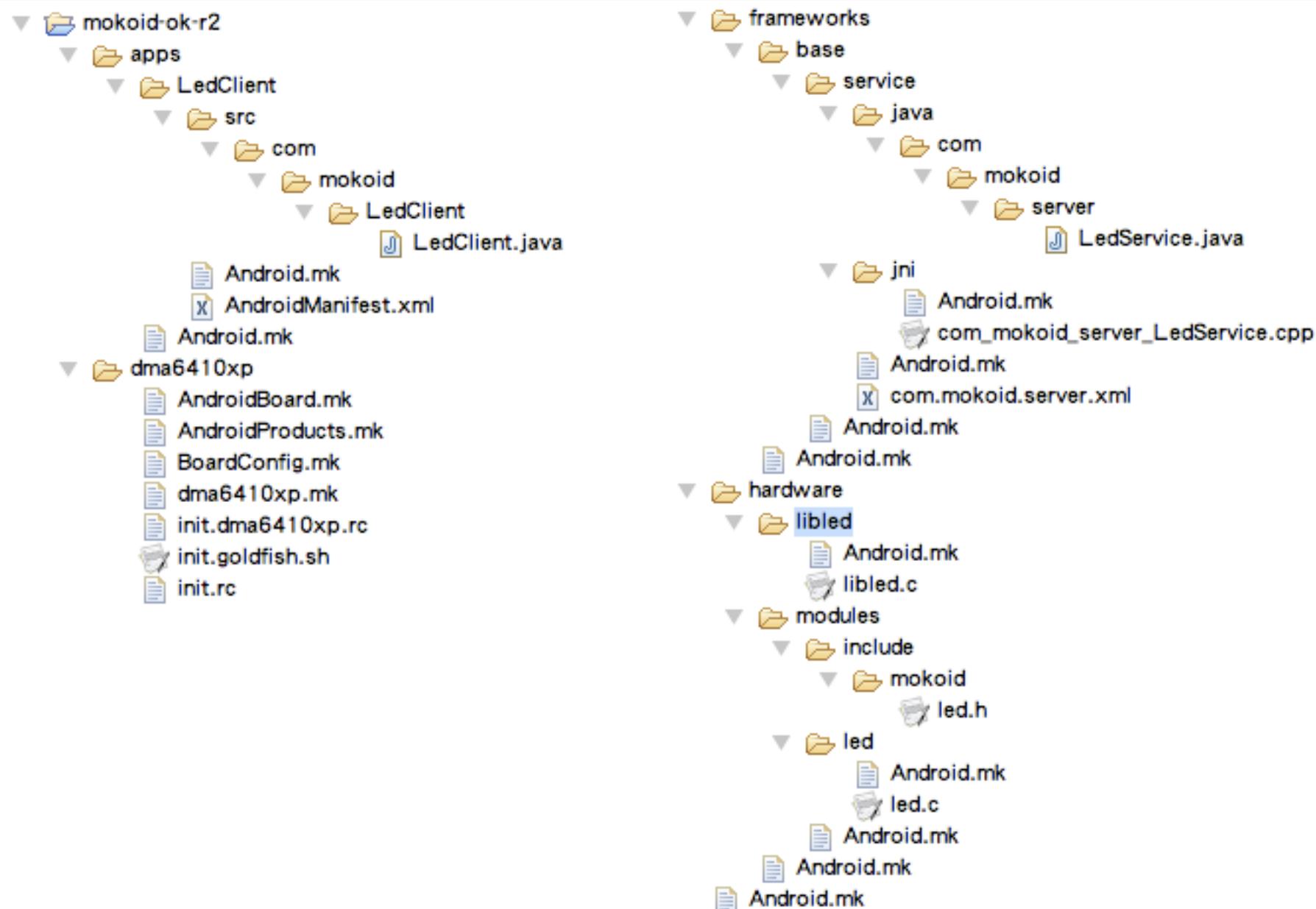
- 提供分支維護方便的編譯途徑
- 分為功能與預定義變數
- 功能
 - 以 include 語法引入「動作」
- 預定義變數
 - predefined makefile variables
 - 定義變數指定動作的「參數」



Product Tree 目錄結構



Mokoid Product 內容



Mokoid Product 目錄結構

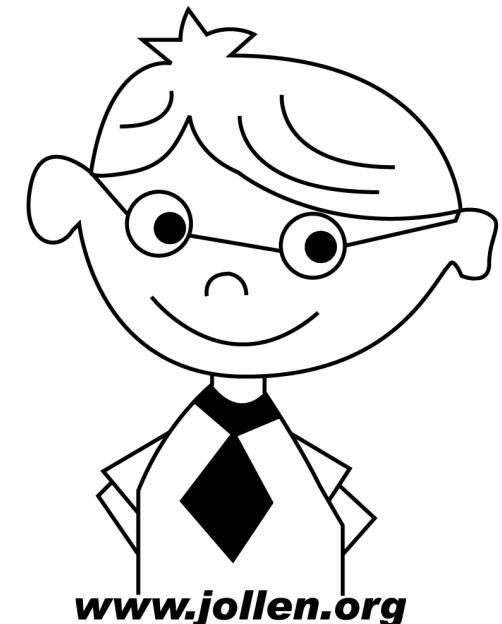
架構層面	目錄結構	Note
產品分支名稱	mokoid/dma6410xp/	
Framework 實作	mokoid/frameworks/	
HAL 實作	mokoid/hardware/	
Application 實作	mokoid/apps/	

實作細節註記

Framework 實作	目錄結構	Note
實作 Service	frameworks/base/service/java frameworks/base/service/jni	
Java Library 名稱	com.mokoid.server	
Java Library 檔名	mokoid.jar	

編譯與測試 Mokoid LED

- 以 mokoid 分支為例
 - 學習 Product Tree 建立
 - 編譯 Android 系統
- 為加速編譯，請將 out.tar.bz2 解壓縮至
Android 原始碼根目錄下
 - 可節省 1-2 小時編譯時間



分支維護策略-3M

□ HAL目前沒有標準

- it's an implementation detail

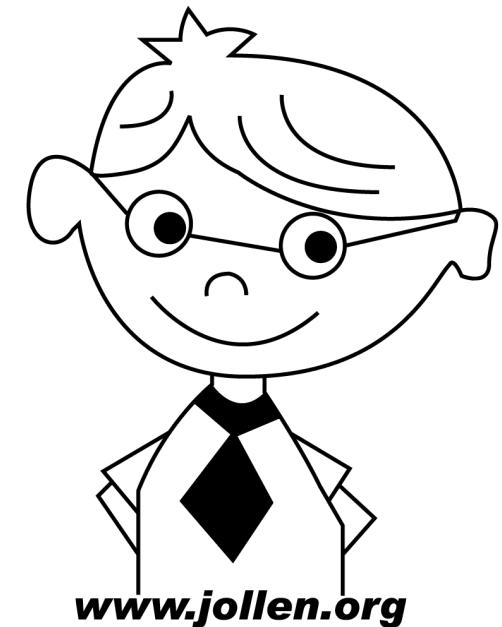
□ 做法取決於自己的實作細節

- 怎麼做都對

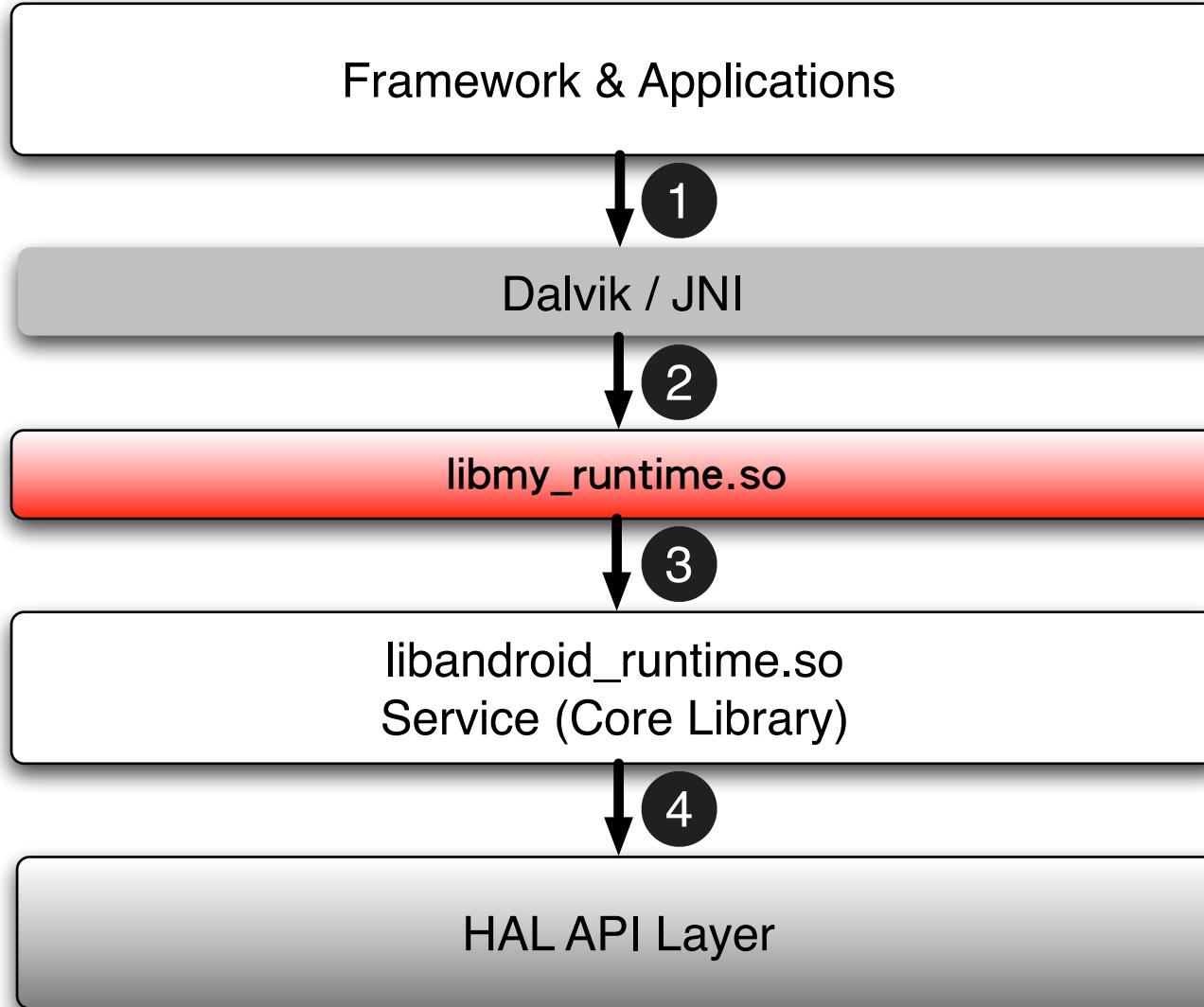
□ 因應之道：

- My Jar/My Runtime/My Stub

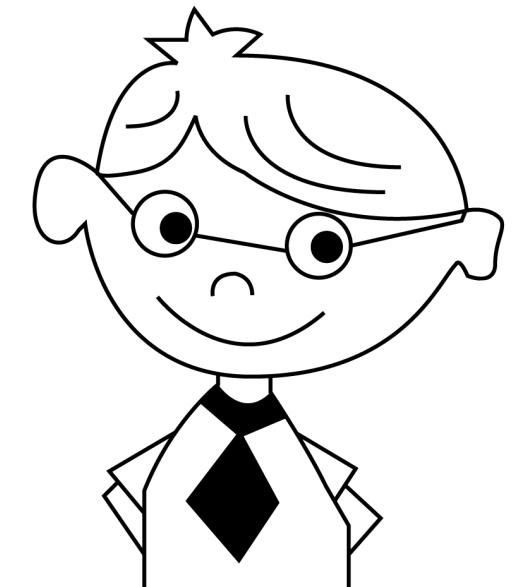
□ 易於管理與後續維護



3M 分支維護策略



製圖：<http://www.jollen.org>



建立編譯環境 (Ubuntu 8.04)

```
$ sudo apt-get install git-core gnupg sun-java5-jdk flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev zlib1g-dev
```

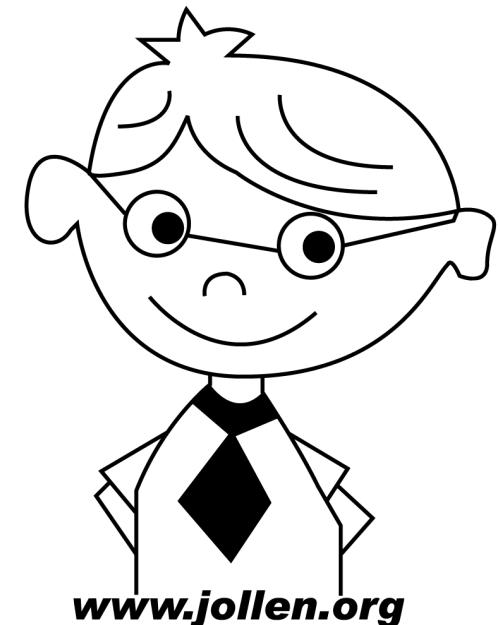
```
$ wget http://android.git.kernel.org/repo > repo
```

```
$ chmod a+x repo
```

```
$ sudo cp repo /sbin
```

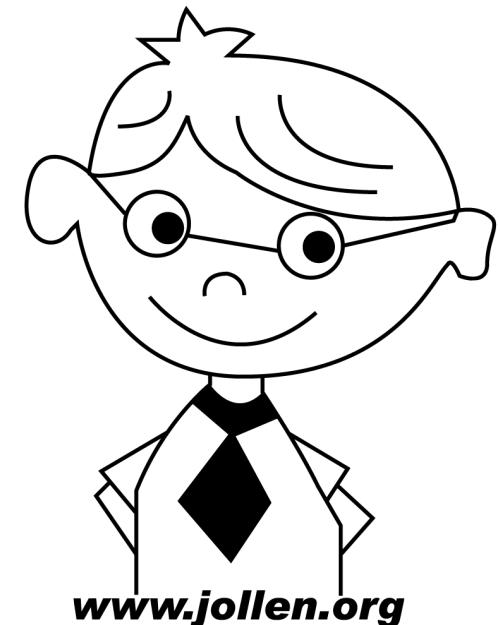
```
$ mkdir work
```

```
$ cd work
```



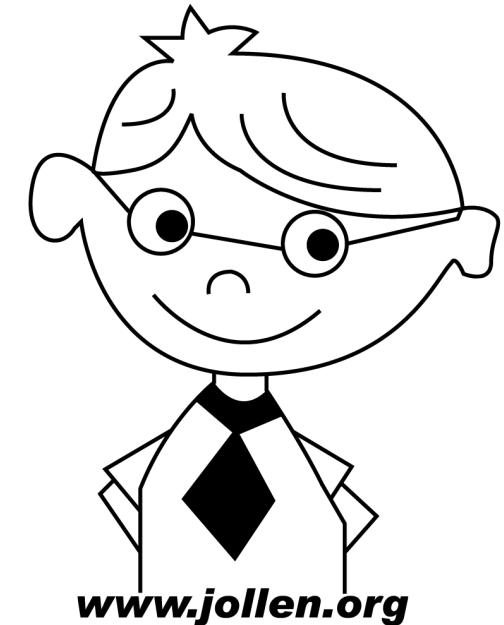
取得Android原始碼

- 將 android-cupcake.tar.bz2 解壓縮到 work/ 目錄下
 - 事先下載的 Android 原始碼
- <http://source.android.com>



取得 Mokoid 分支

- 將 mokoid-led.tar.bz2 解壓縮
- 必須放置於 Product Tree 下
 - vendor/mokoid/
- 編譯與維護 Android 需採取標準的 Product Tree 方式



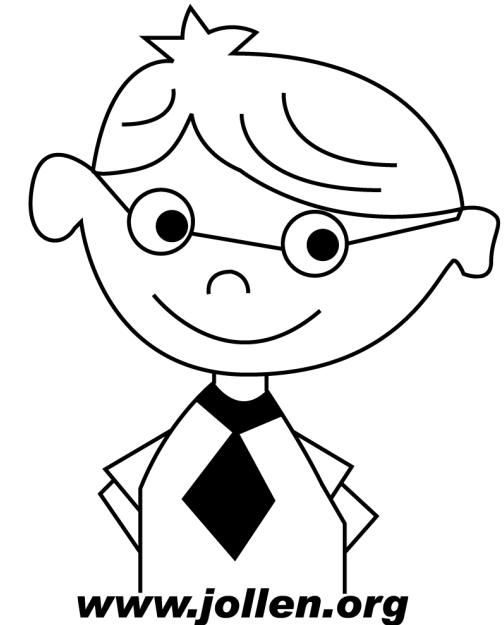


嵌入式產品設計 專業培訓 廠訓諮詢

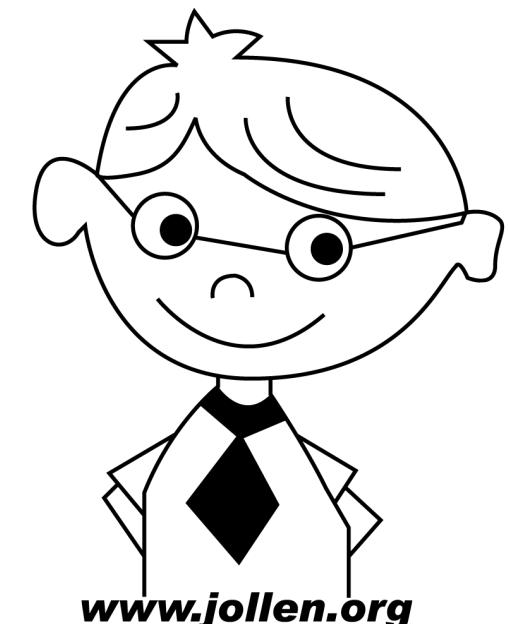
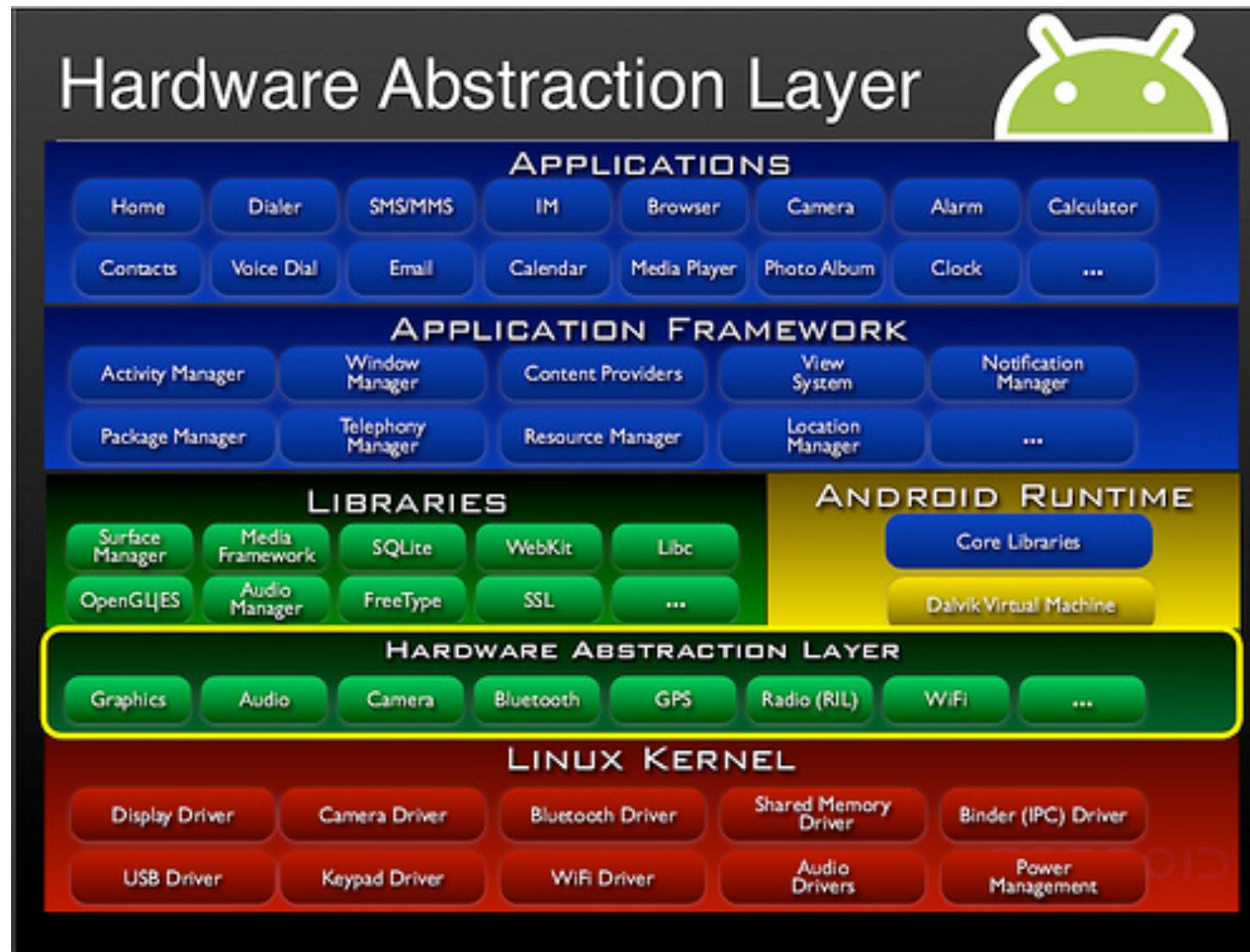
第02堂課：HAL 技術詳解

本堂主題

第二堂	HAL 技術詳解
2.1	HAL 的意義與二進位佈署(Binary File Deploy)
2.2	Service 與 Manager 的意義與用途
2.3	libhardware 與 HAL API
2.4	Stub & Module 的觀念
2.5	專題討論：定義並撰寫第一個 HAL Stub

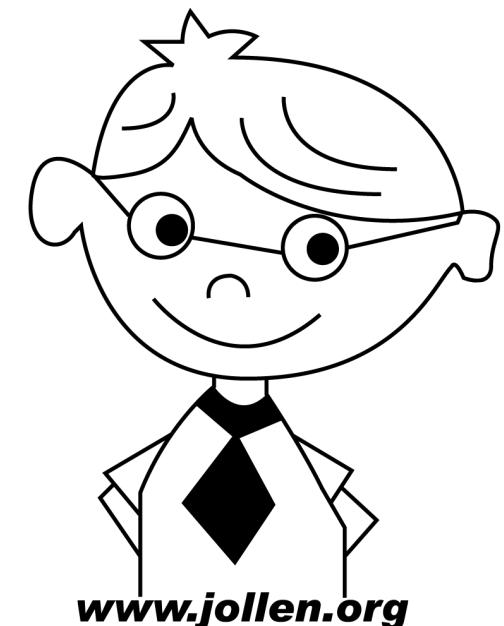


Android HAL 概念



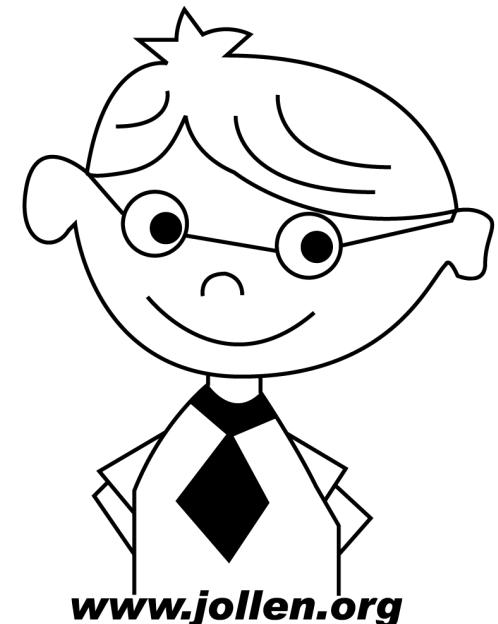
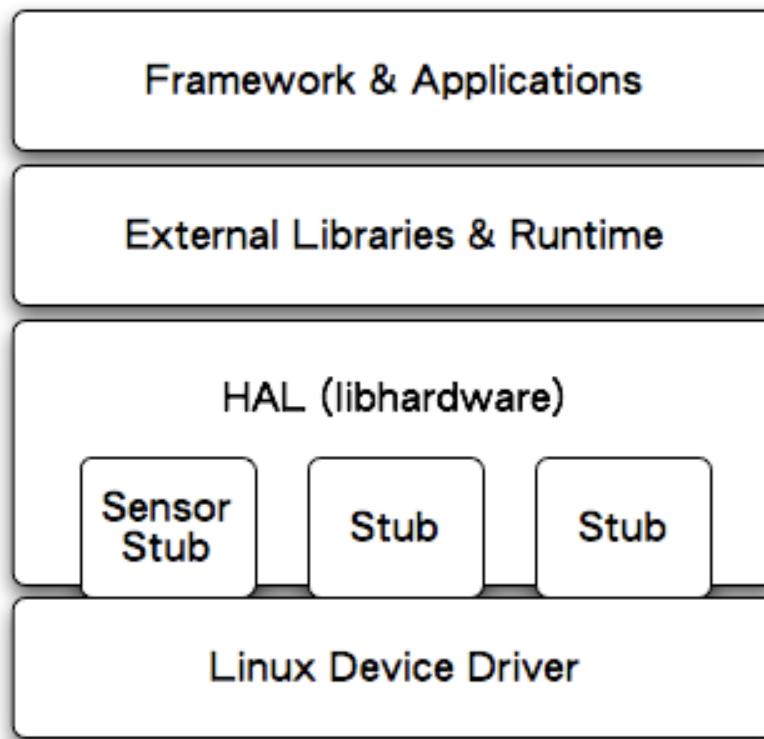
HAL 簡介與現況分析

- 現行 HAL 架構圖由 Patrick Brady 於 2008 Google I/O 演講「Anatomy & Physiology of an Android」中所提出
- 因應廠商「開發封閉源碼驅動程式模組」的要求下所規劃的架構與觀念
- 現行 HAL 的「抽象程度」還不足
- 需要變動框架以整合 HAL 模組

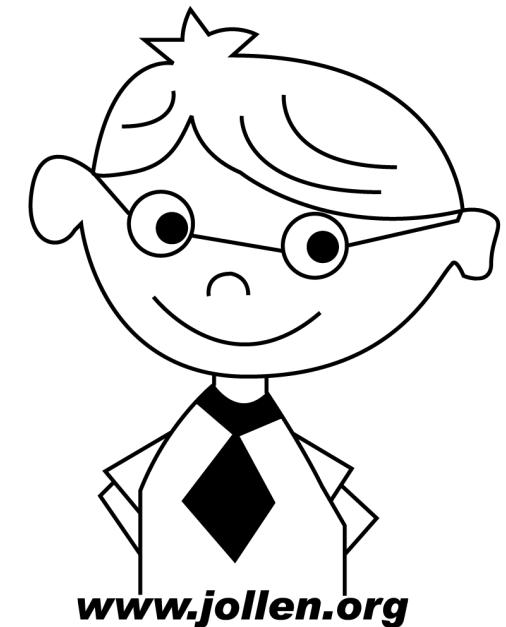
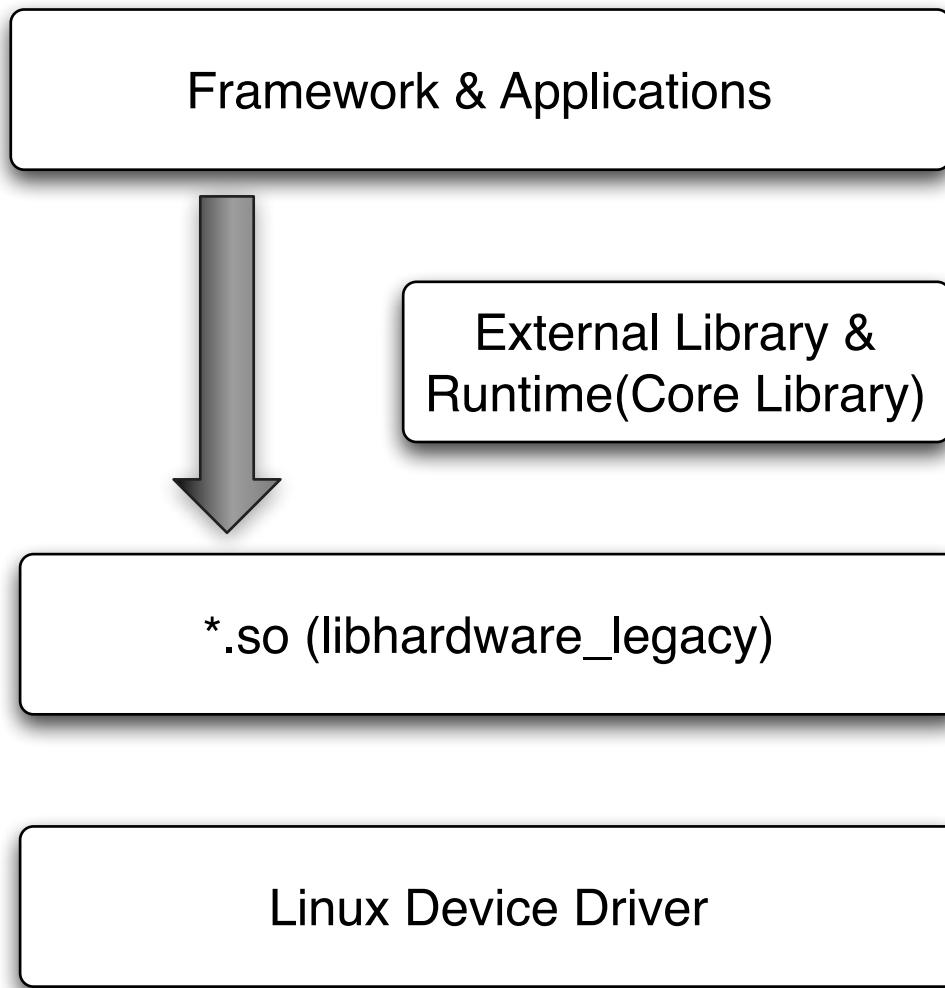


HAL 架構

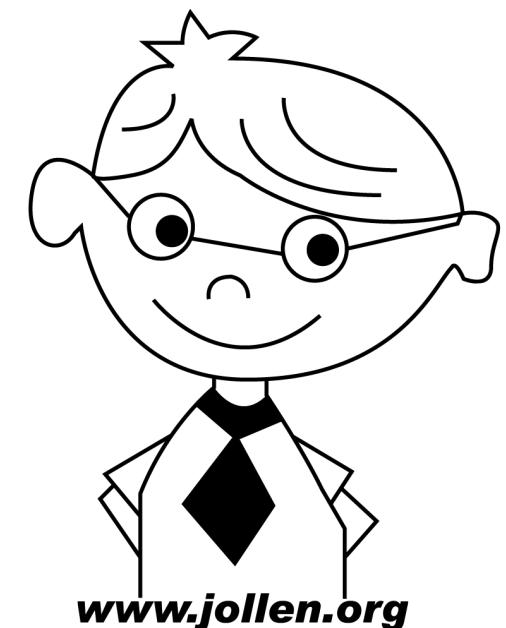
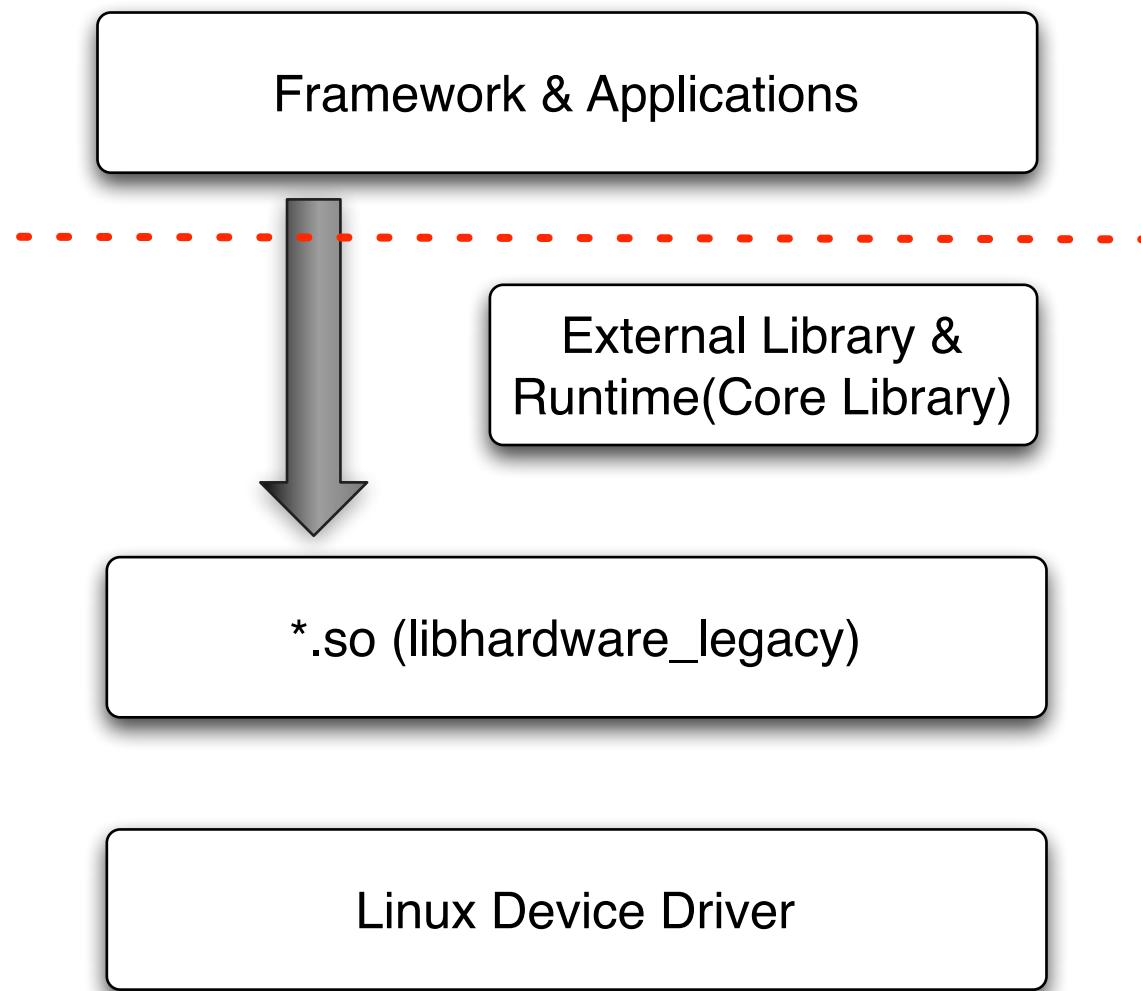
- HAL stub 是一種代理人（ proxy ）的概念
- Stub 向 HAL 「提供」操作函數（ operations ）



舊有的與新式的



JNI在什麼地方



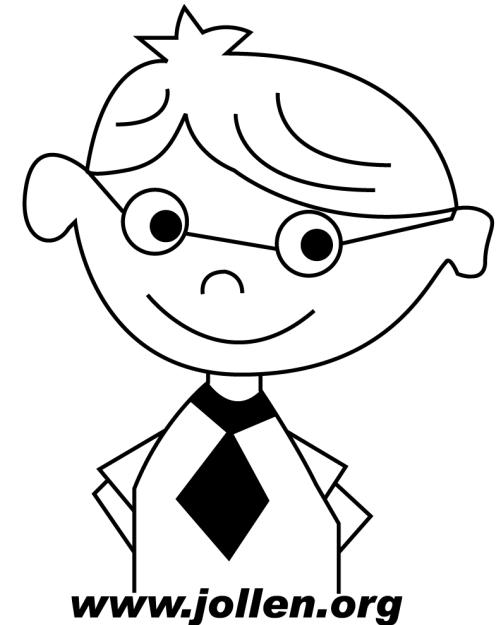
二種實作解決方案

□ 舊有的(legacy)HAL

- 在原始碼的libhardware_legacy下
- 稱之為HAL module

□ 新式的HAL

- 在原始碼的libhardware下
- 稱之為HAL stub



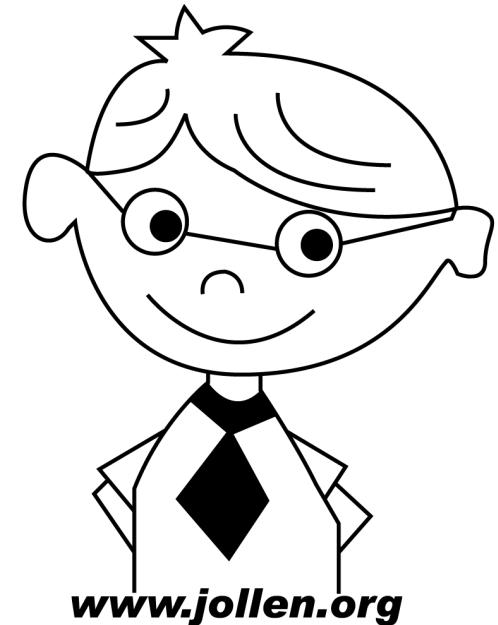
Module與Stub

□ Module

- 技術上扮演被呼叫的角色
- 在Android架構裡僅做為程式庫(library)的用途

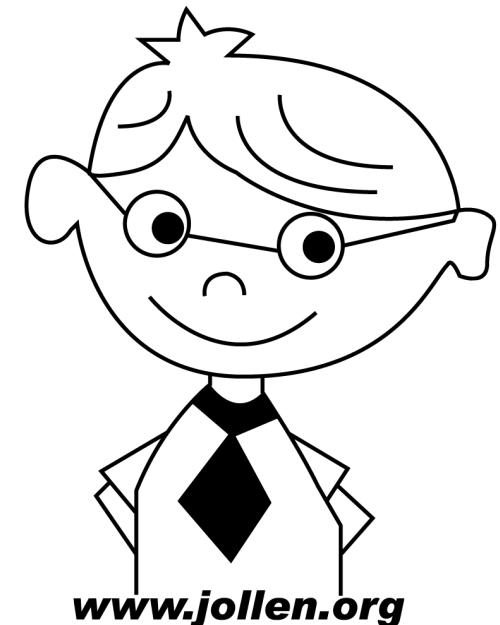
□ Stub

- Stub又稱為代理人(proxy)
- 技術上扮演操作供應者的角色(operations provider)



HAL Stub

- HAL Stub 目前沒有嚴格的標準
 - Interface 由廠商定義
 - Implementation detail 自己決定
- 以 Product Tree 方式進行維護
 - 易於管理
- 參考 Mokoid 範例



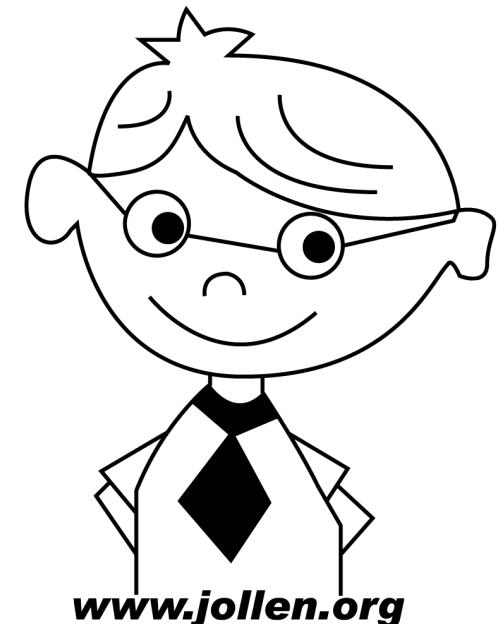
HAL Stub 背景知識

- HAL Stub 就是 Android 硬體控制模組

- /system/lib/hw/led.goldfish.so

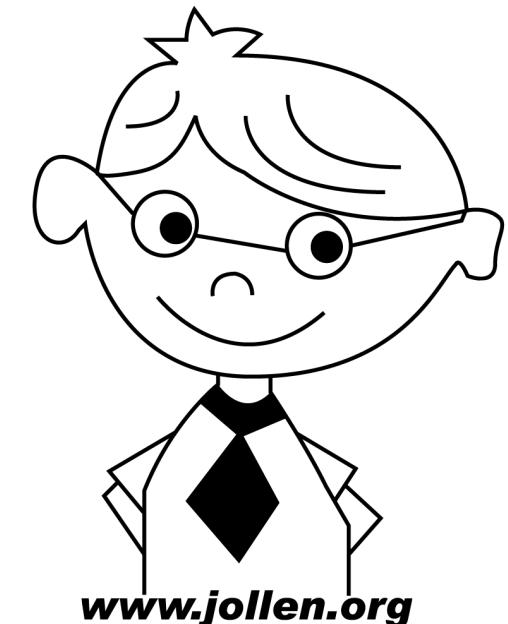
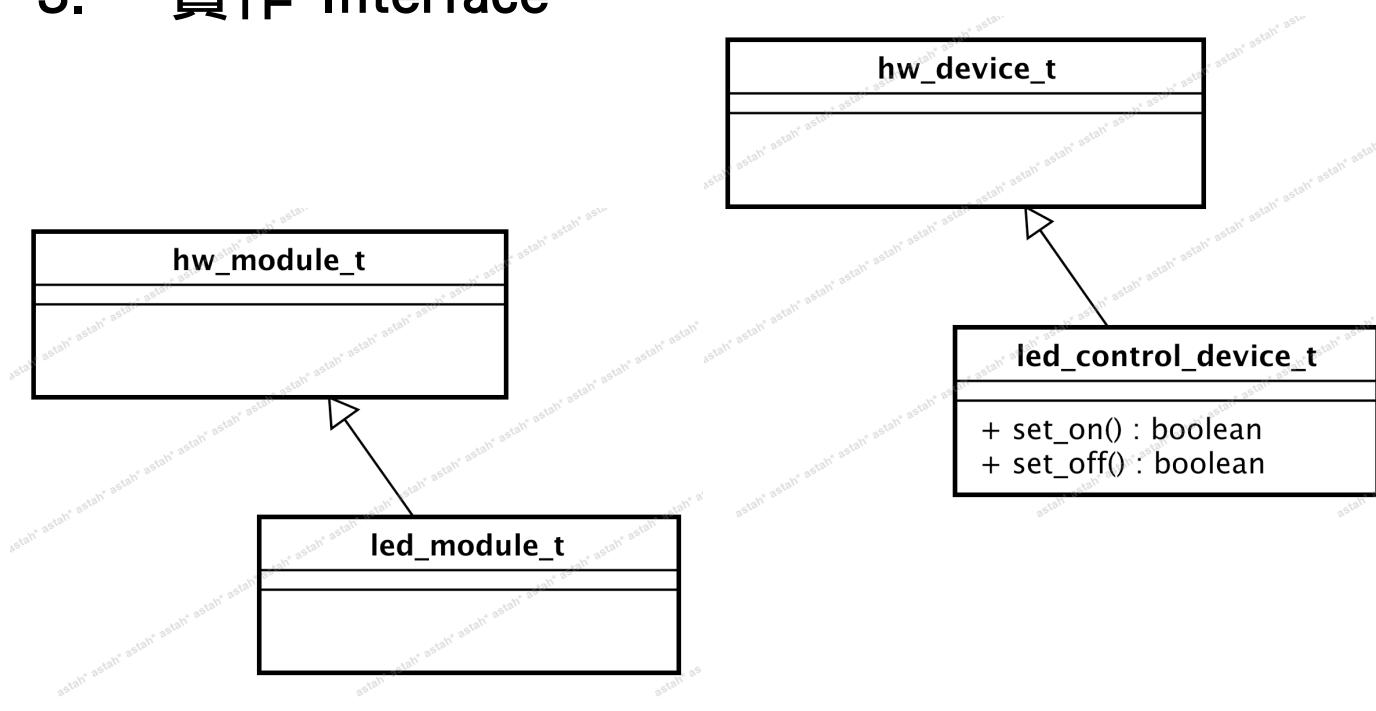
- 用 C 寫繼承

```
struct led_module_t {  
  
    struct hw_module_t common;  
  
    /* Place attributes here. */  
  
    /* Place methods here. */  
};
```



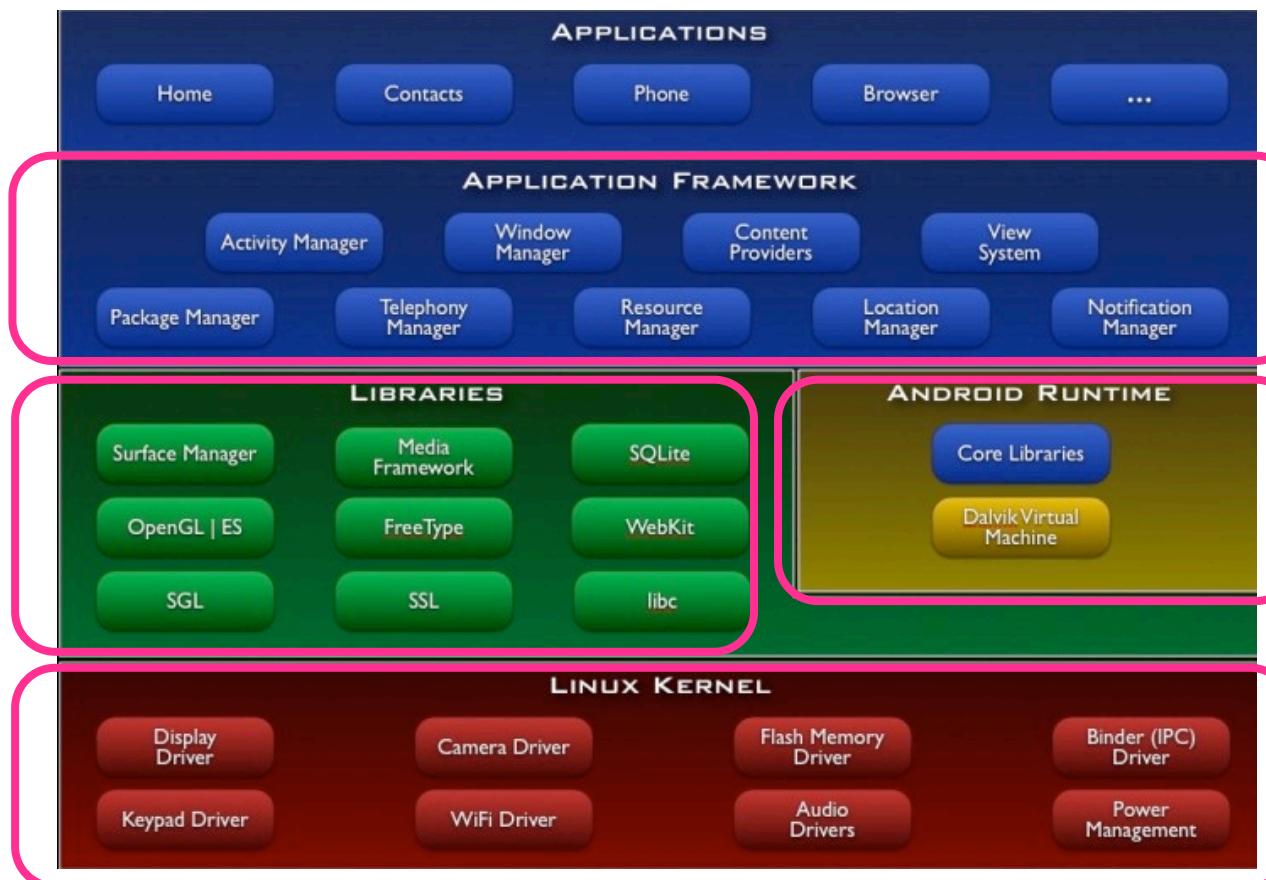
HAL Stub 實作步驟

1. 繼承 `hw_module_t` 與 `hw_device_t`
2. 定義 Interface (Stub Operations)
 - 理論上這些應該要有規範或標準
3. 實作 Interface



Android Framework & HAL

C/C++
開發 Library

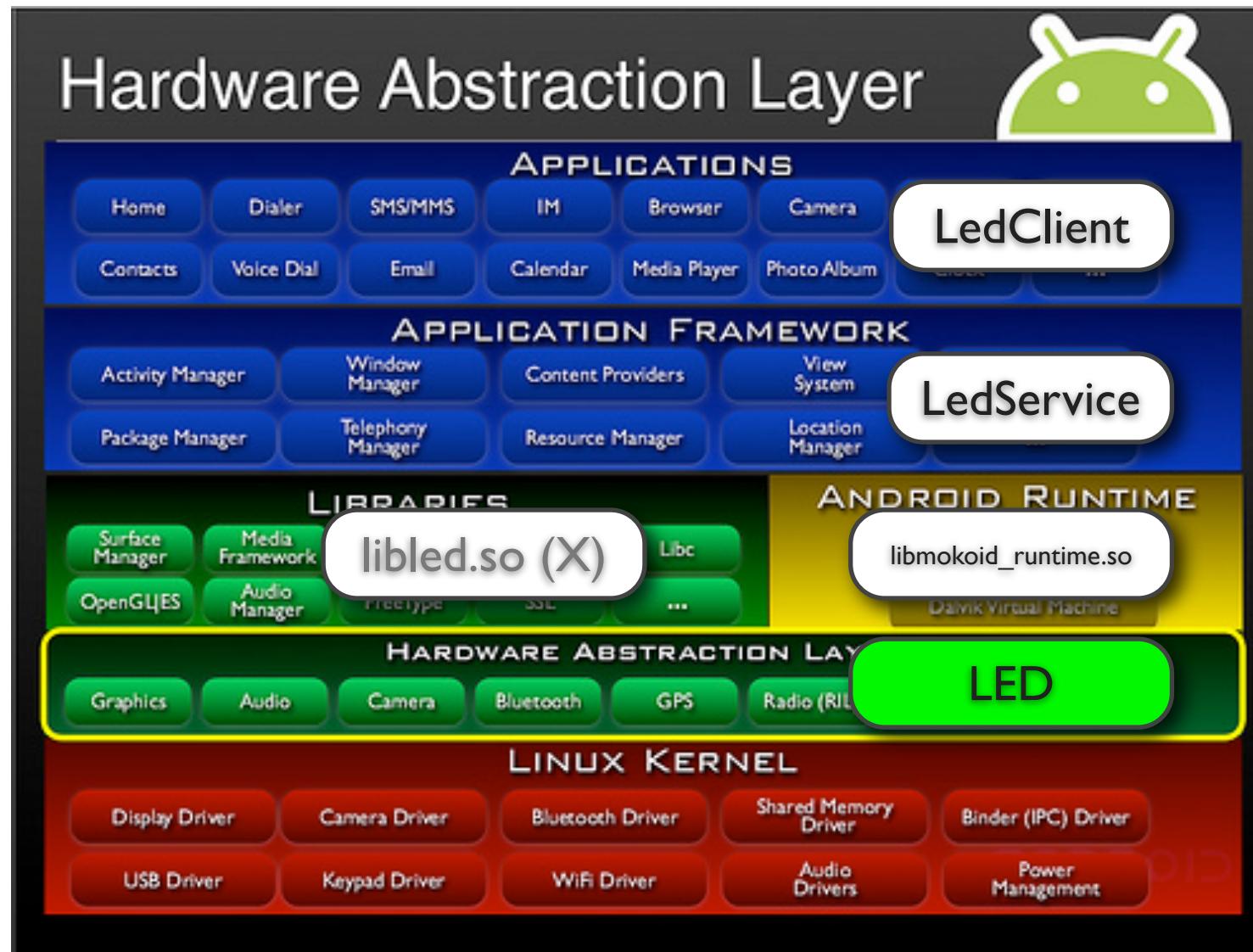


C/C++
加入 API

C/C++
加入 JNI

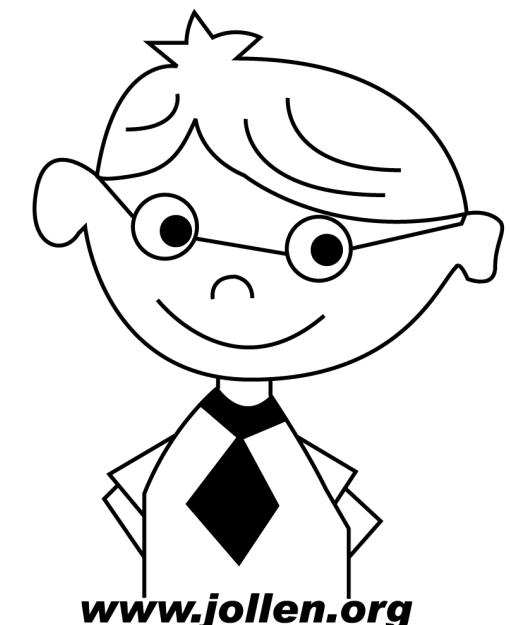
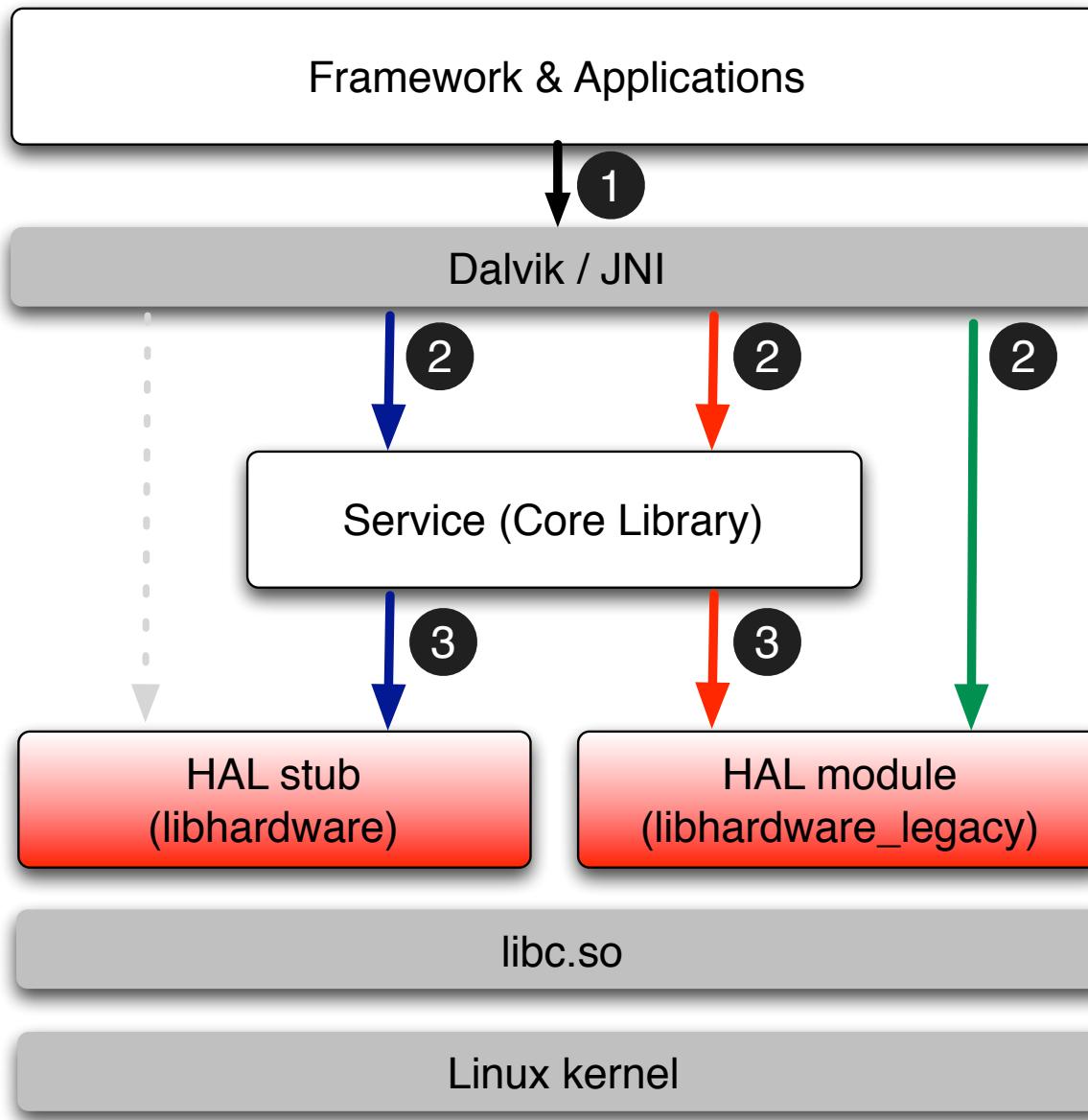
(略)

HAL Stub

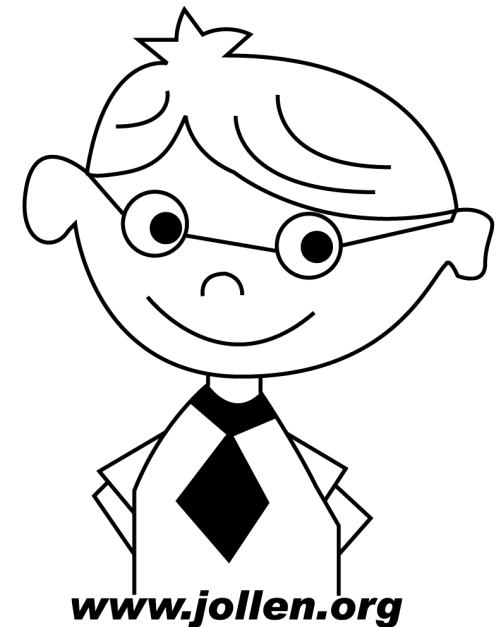
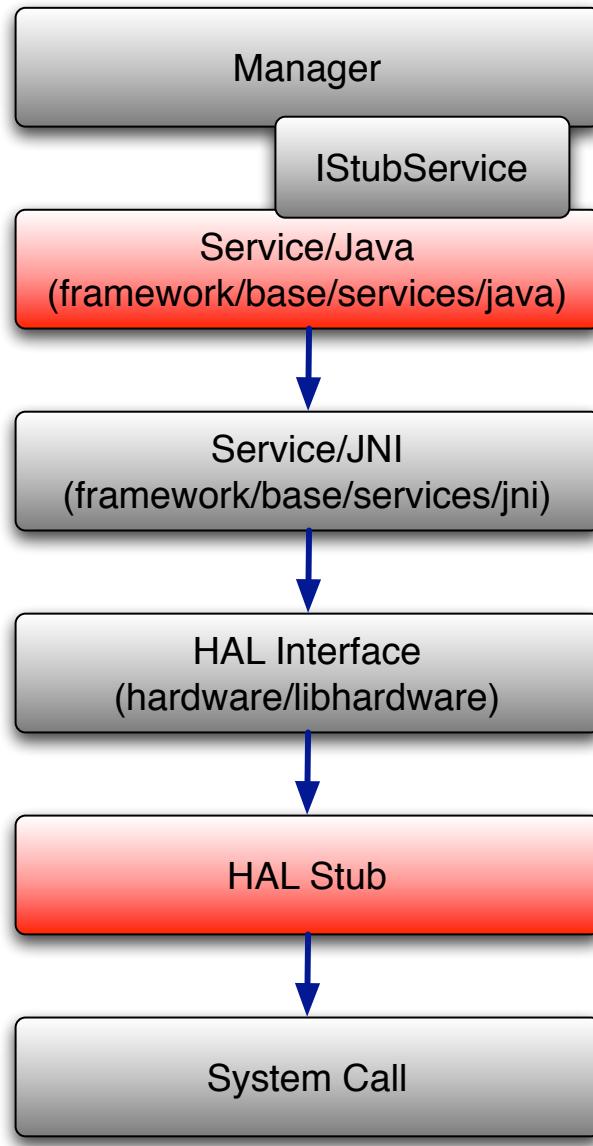


用 C/C++ 開發
HAL Stub

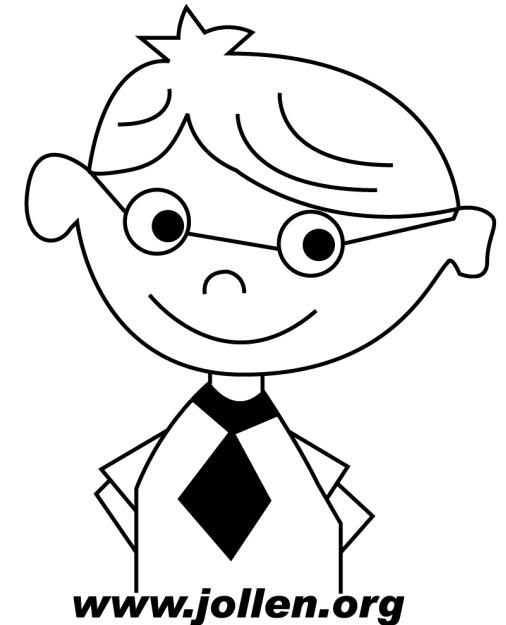
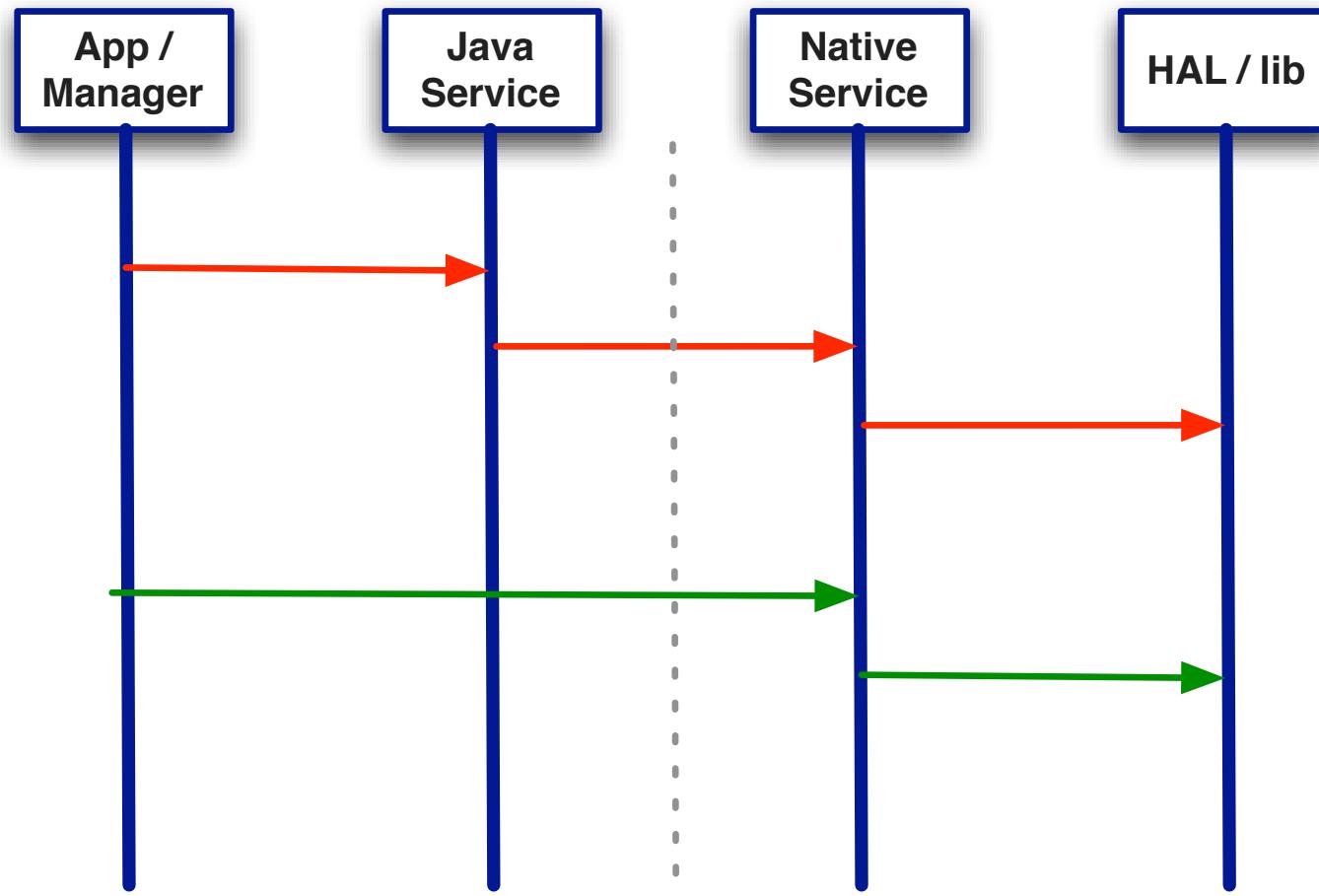
Android HAL 架構



Android Service 介紹



Manager API 與 Android Service

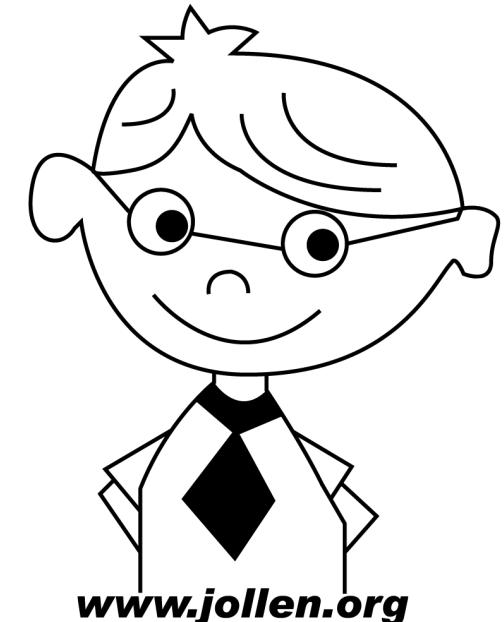
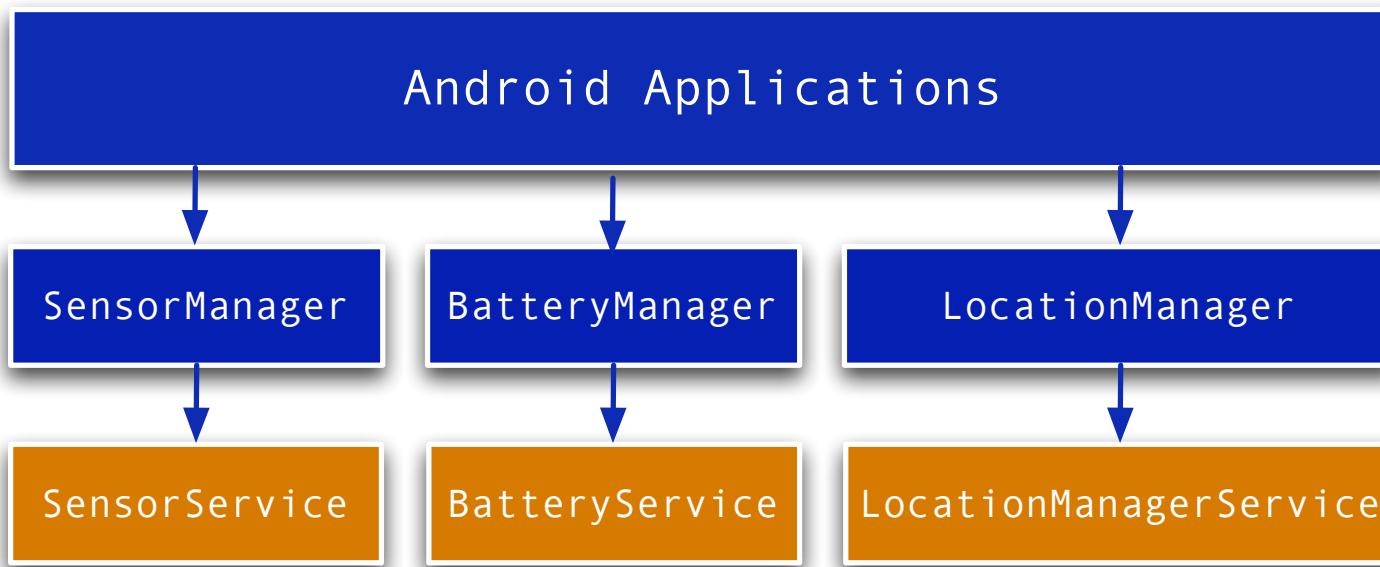


www.jollen.org

Android Service 的用途

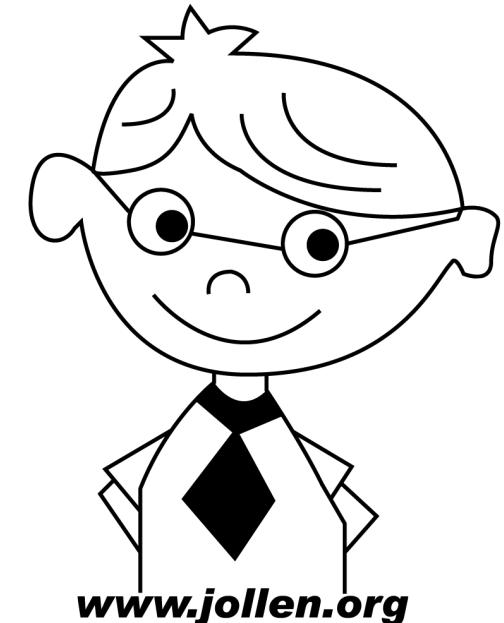
□ 控制硬體的物件

- SensorService
- BatteryService



實例探討：SensorService

- 閱讀 Android Framework 原始碼
 - SensorService.java
 - SystemServer.java
- 依此架構實作的 mokoid 範例、為 Android 框架新增了 Led 硬體控制功能
- 修改 Anroid 框架、優化 mokoid-led 程式碼
 - 採取完全標準的 Android 框架架構
 - 於 Android Framework 課程另做說明



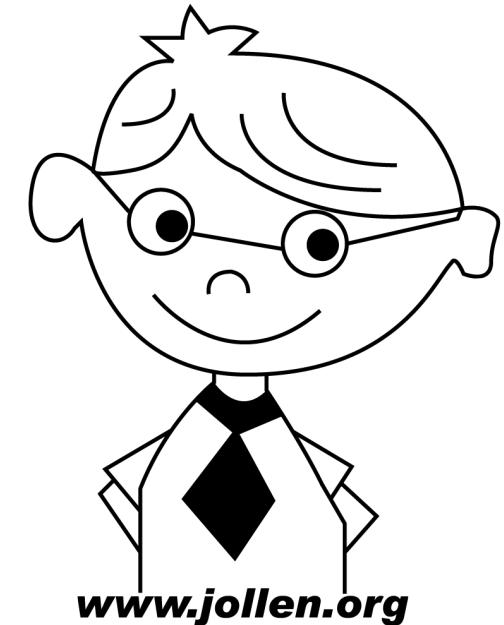


嵌入式產品設計 專業培訓 廠訓諮詢

第03堂課：HAL Development

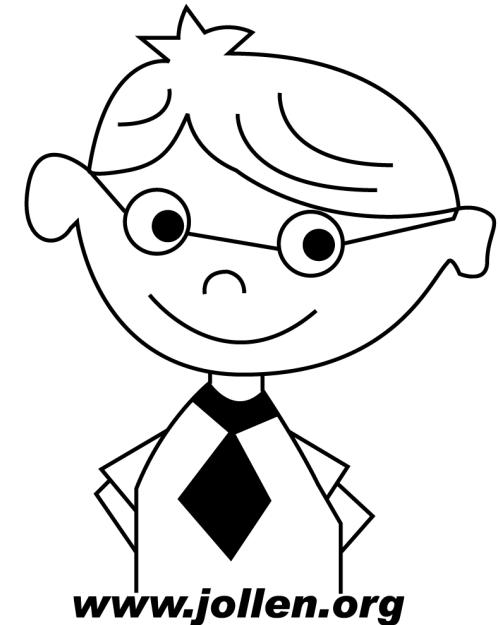
本堂主題

第三堂	HAL Development
3.1	HAL Stub Analysis and Design (OOAD)
3.2	HAL Stub Class
3.3	HAL Stub Interface
3.4	專題討論：開發 LED 的 HAL 模組

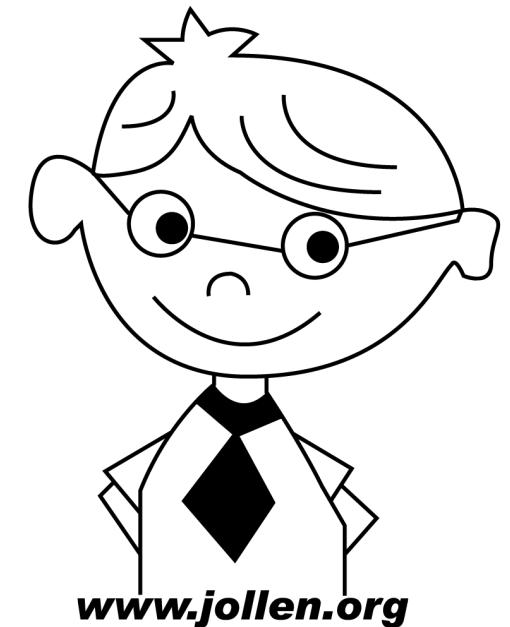
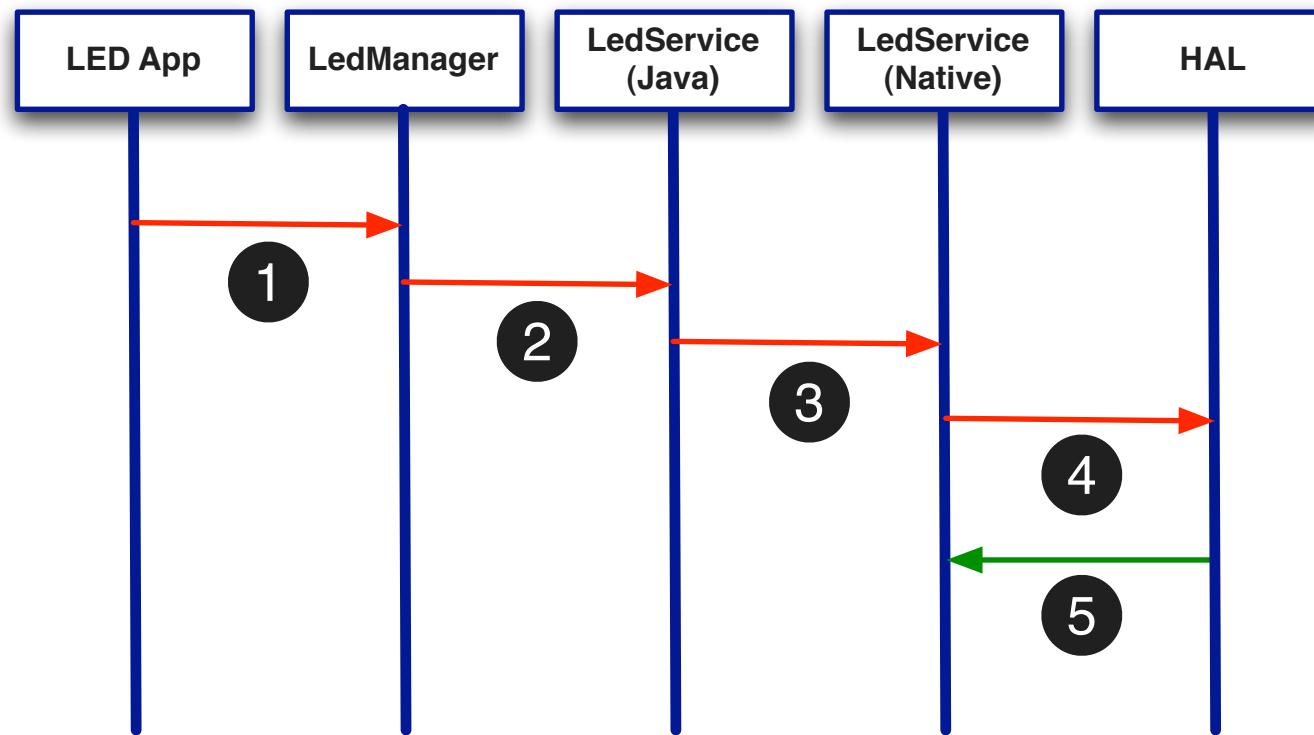


HAL Stub & API 開發流程

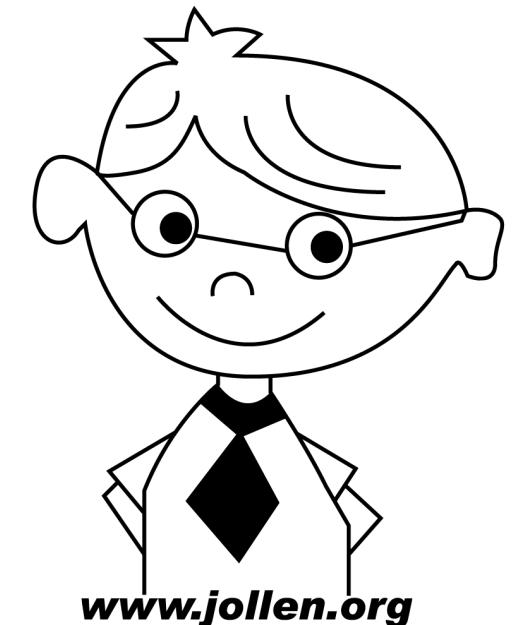
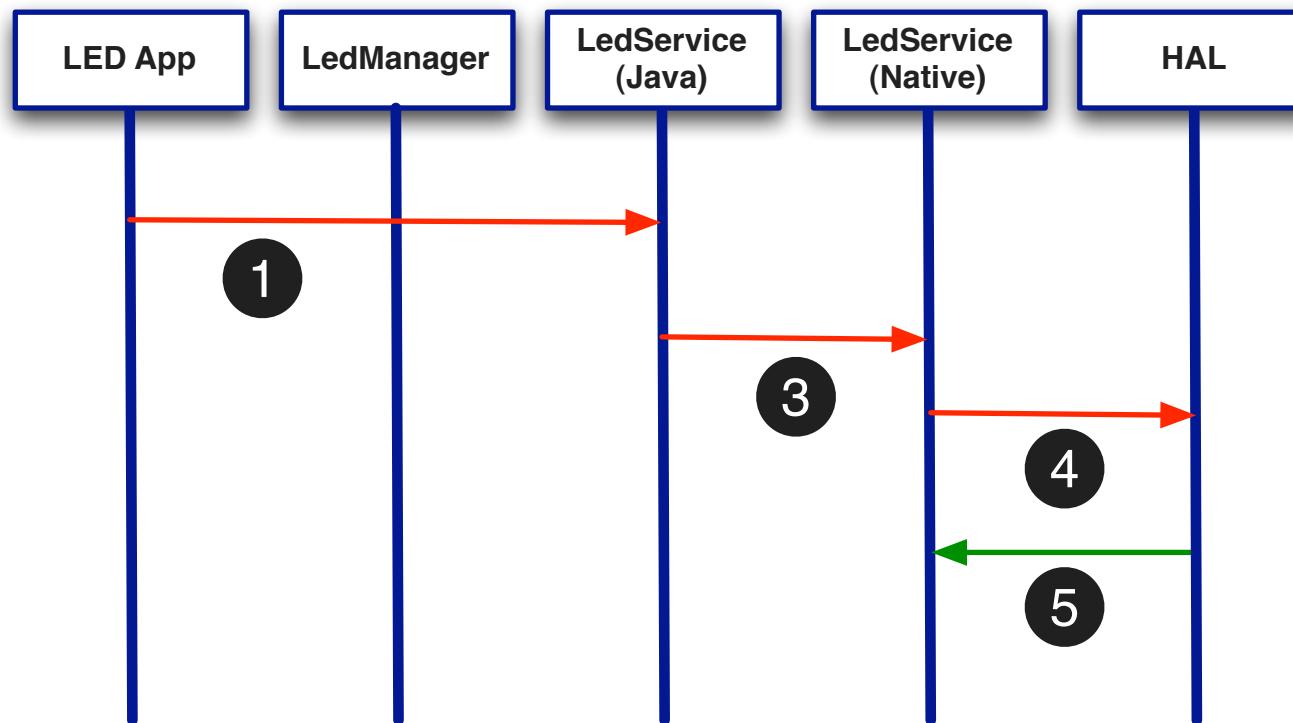
- 1. 開發 LedService API
- 2. 設計好的 LED Stub 與 LedService 整合
- 3. 設計 API Test 應用程式
- 4. 後續優化 (Refinement)



LedTest 整體架構

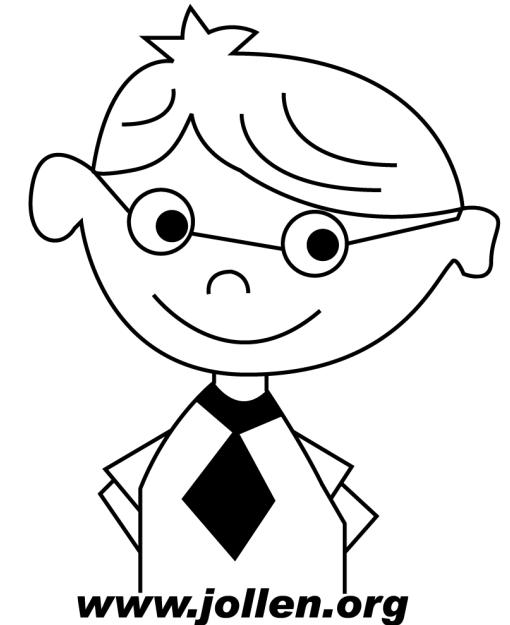


LedTest 整體架構

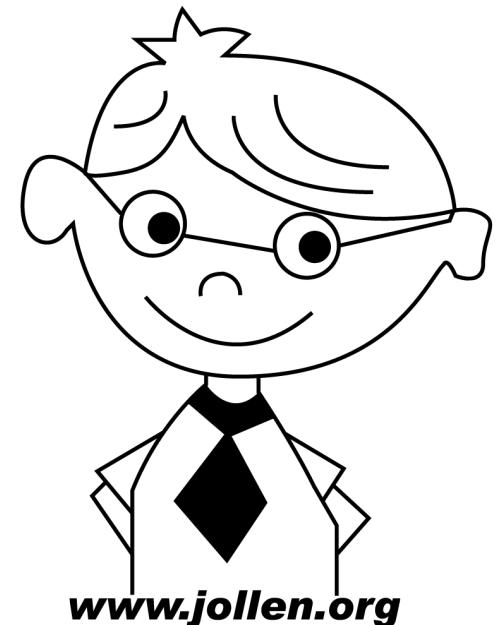
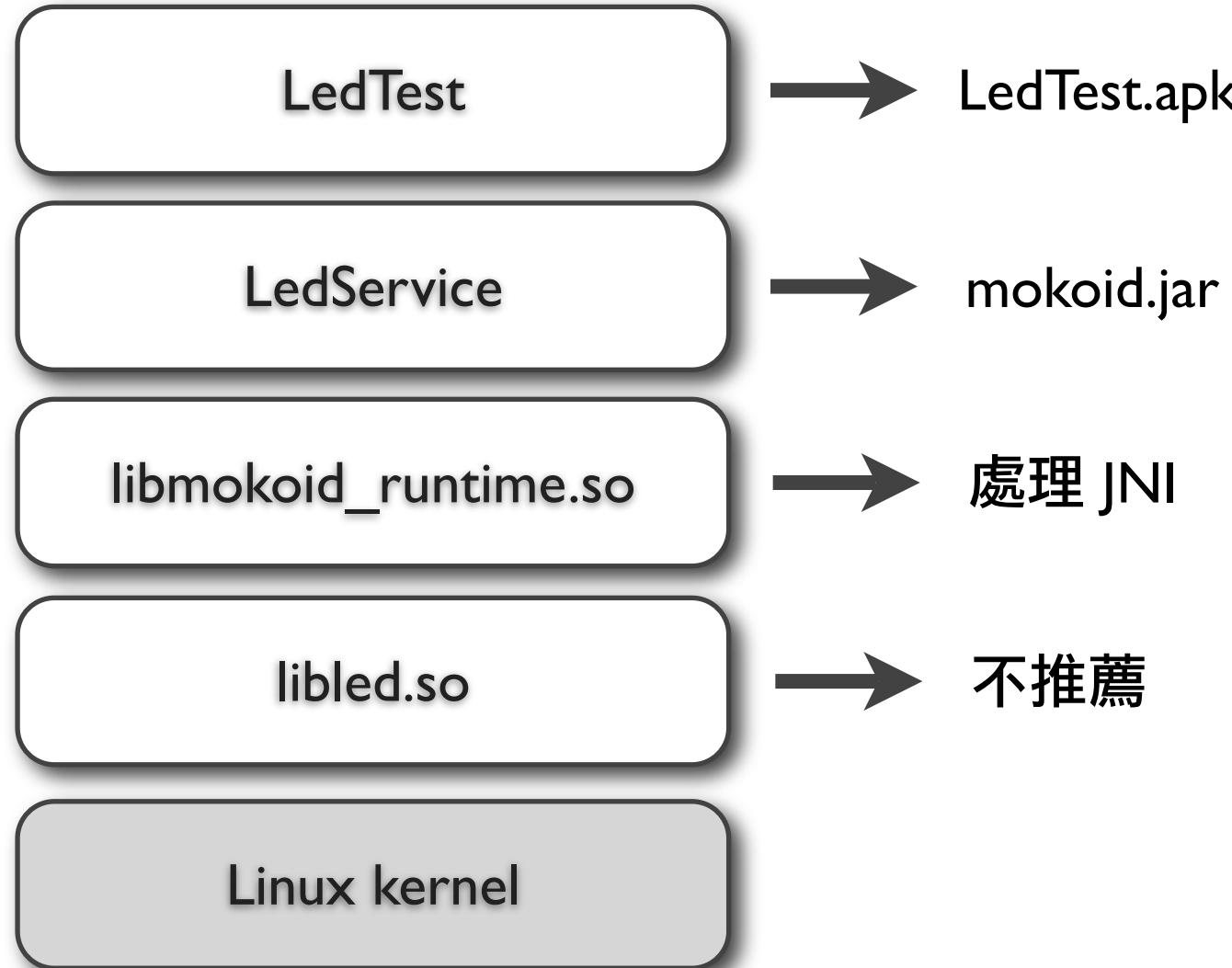


LedTest 控制 LED

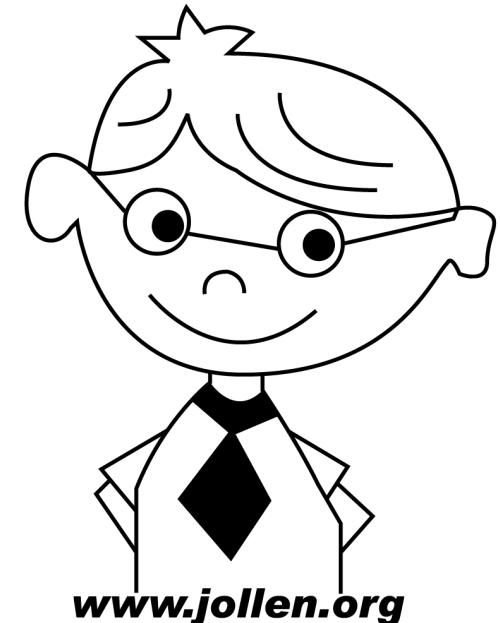
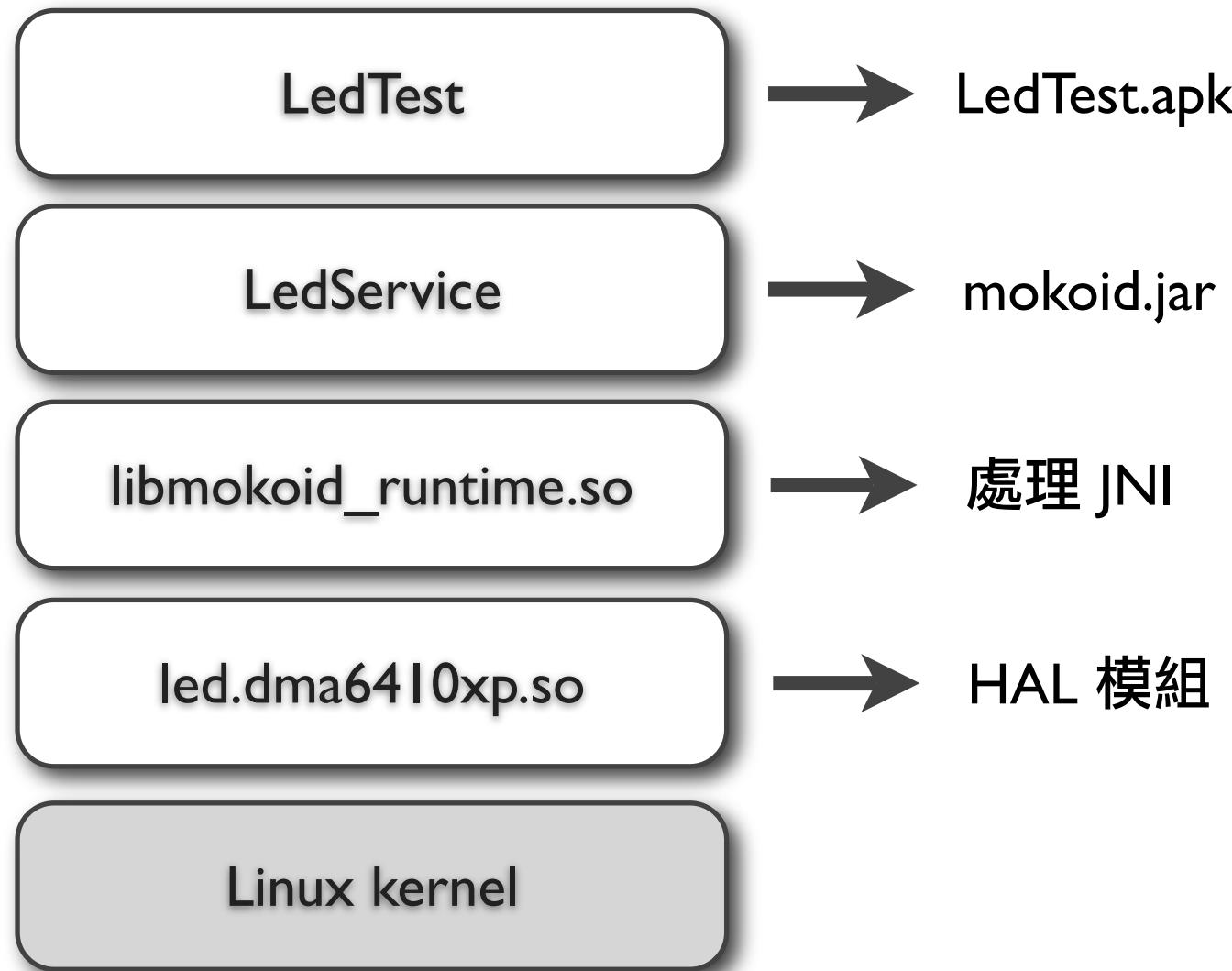
```
public class LedClient extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Call an API on the library.  
        LedService ls = new LedService();  
        ls.setOn(1);  
  
        TextView tv = new TextView(this);  
        tv.setText("LED 0 is on.");  
        setContentView(tv);  
    }  
}
```



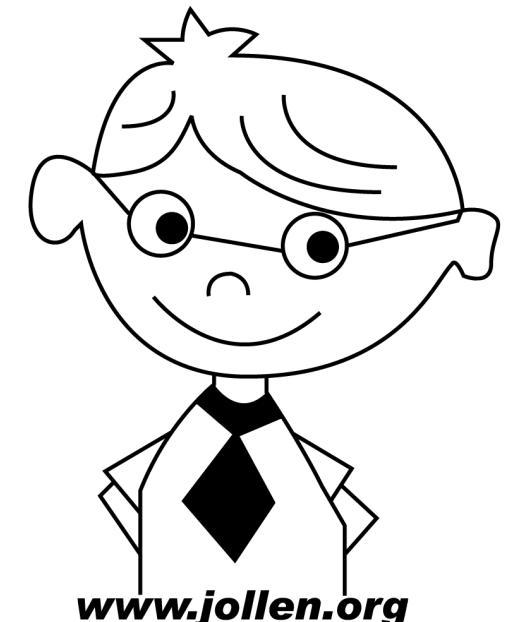
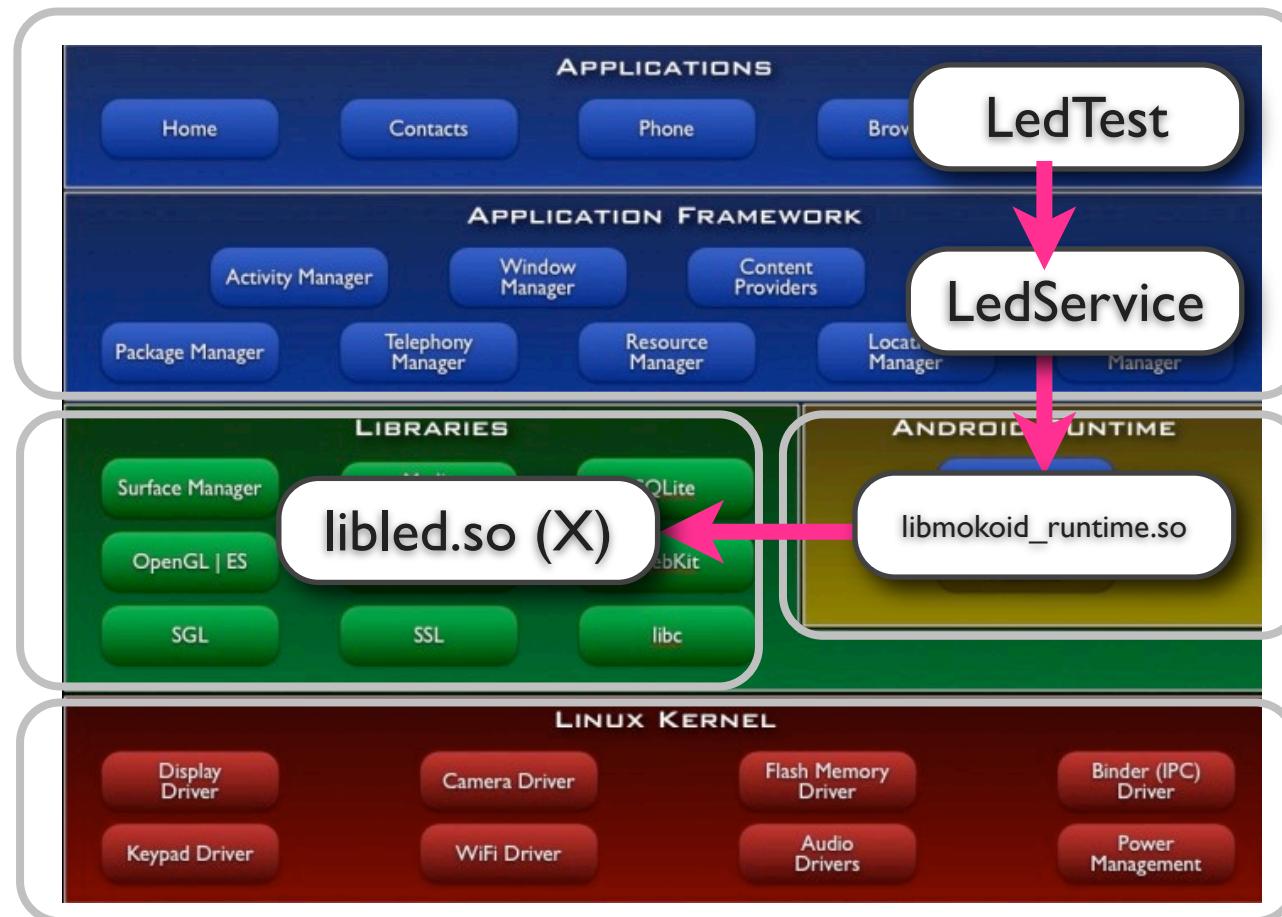
實例探討：LedTest 應用程式



實例探討：LedTest 應用程式

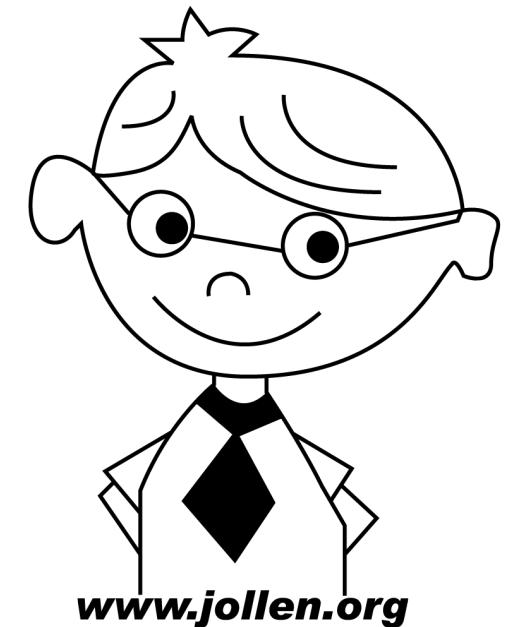


LED 移植到 Android



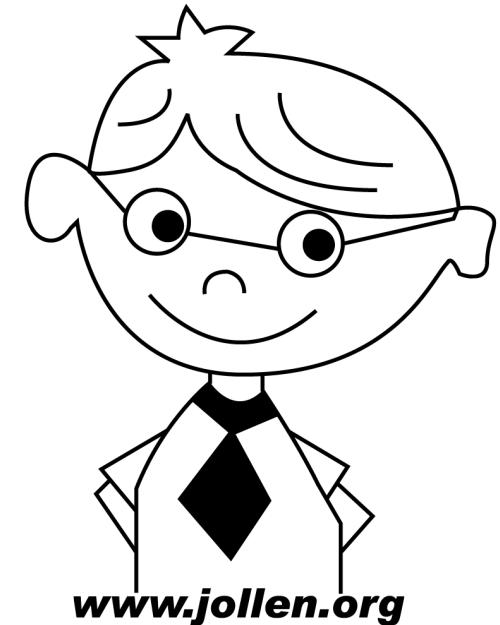
HAL Stub 實作

```
static struct hw_module_methods_t led_module_methods = {  
    open: led_device_open  
};  
  
const struct led_module_t HAL_MODULE_INFO_SYM = {  
    common: {  
        tag: HARDWARE_MODULE_TAG,  
        version_major: 1,  
        version_minor: 0,  
        id: LED_HARDWARE_MODULE_ID,  
        name: "Sample LED stub",  
        author: "The Mokoid Open Source Project",  
        methods: &led_module_methods,  
    }  
    /* supporting APIs go here */  
};
```



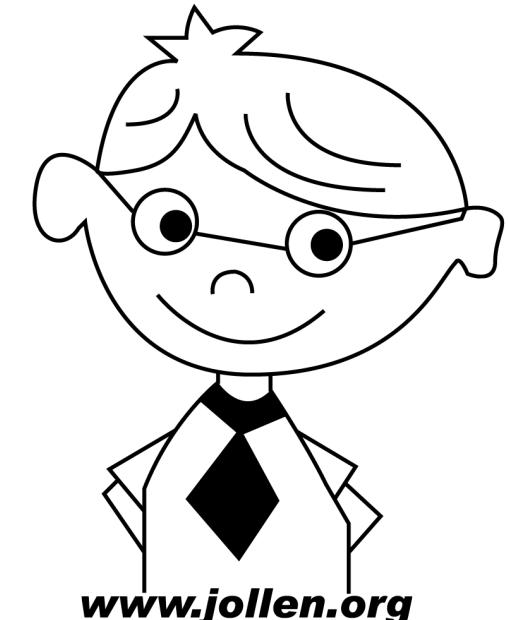
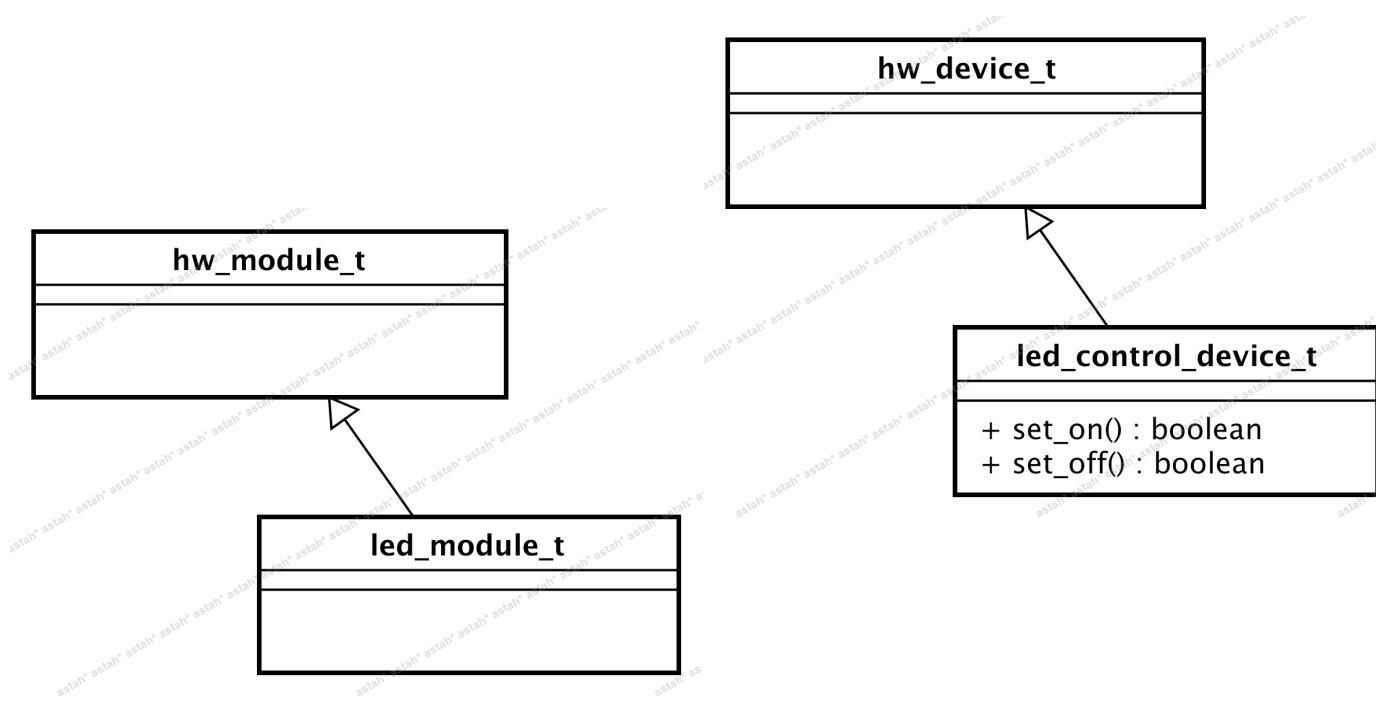
Supporting API 有二類

- Supporting API
 - 「輔助的API」
- 初始化時間 (hw_get_module) 的 supporting APIs
- 控制的 supporting APIs



HAL Stub 實作步驟

- 在 led_moduel_t 定義自己的初始化時期 Supporting API
- 在 led_control_device_t 定義控制的 Supporting API

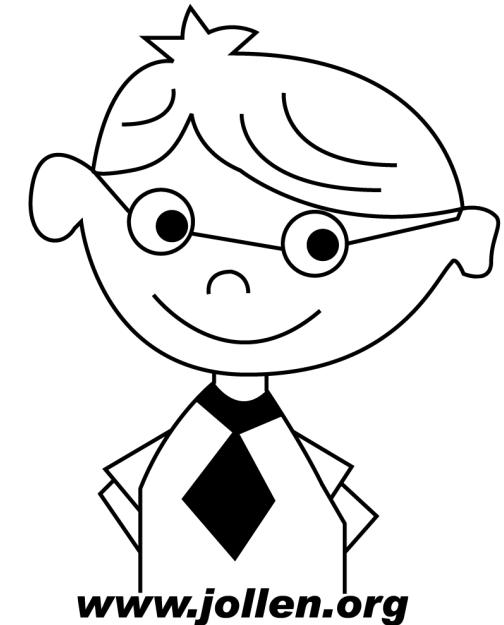


www.jollen.org

控制的Supporting API

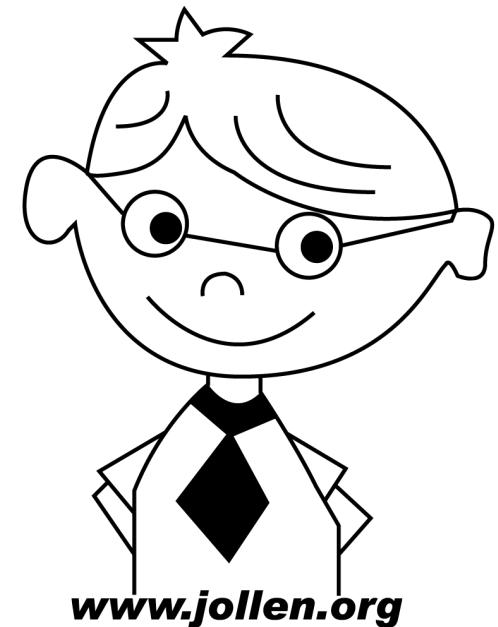
```
struct led_control_device_t {  
    struct hw_device_t common;  
    /* supporting control APIs go here */  
  
    int (*set_on)(struct led_control_device_t *dev, int32_t led);  
    int (*set_off)(struct led_control_device_t *dev, int32_t led);  
};
```

API	用途
set_on()	點亮LED
set_off()	熄滅LED



提供 Supporting API

- 控制的 supporting API 在 HAL stub 的 open callback 裡提供
- HAL stub 的 open operation 必須提供 supporting API 給 runtime



Stub 的 Operations

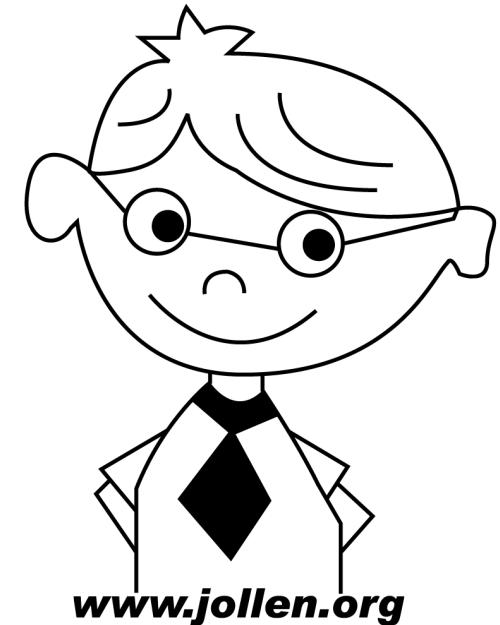
□ 實作 Interface

```
int led_on(struct led_control_device_t *dev, int32_t led)
{
    LOGI("LED Stub: set %d on.", led);

    return 0;
}

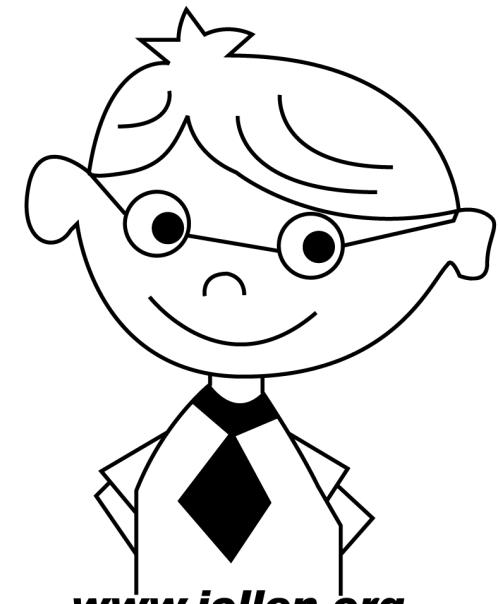
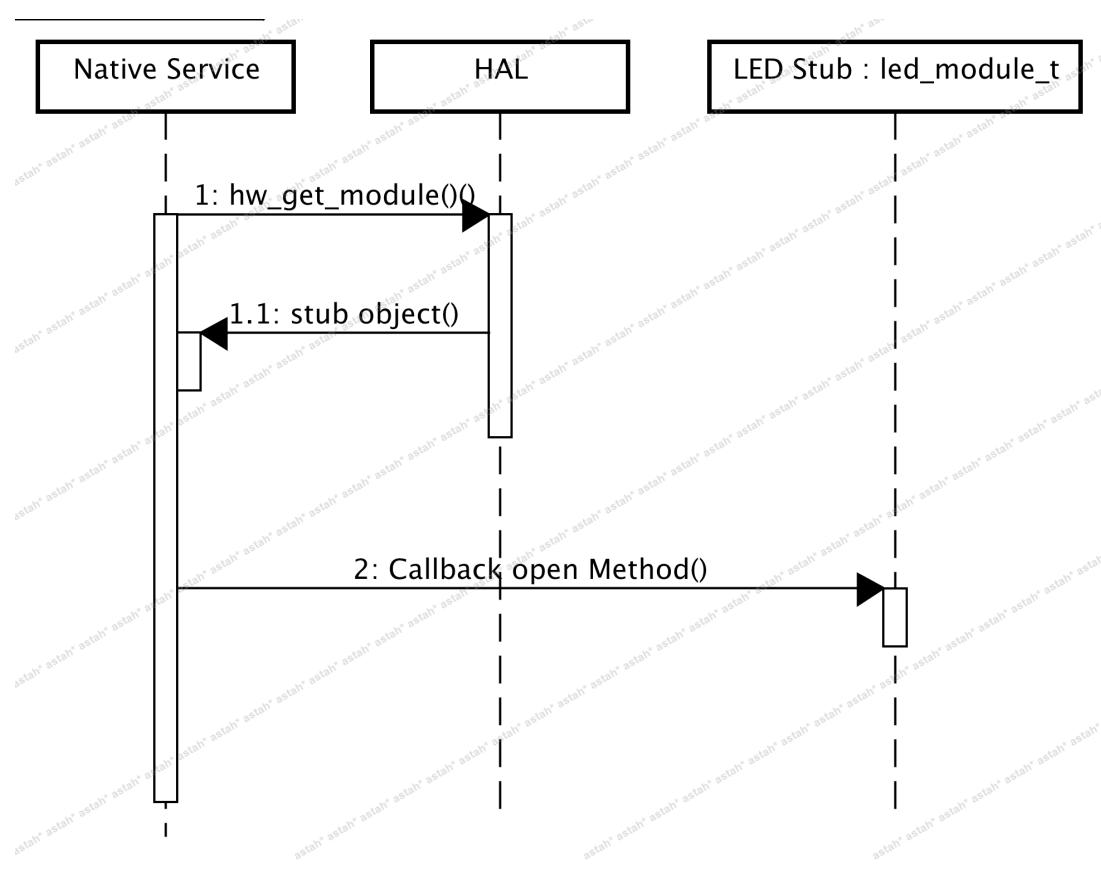
int led_off(struct led_control_device_t *dev, int32_t led)
{
    LOGI("LED Stub: set %d off.", led);

    return 0;
}
```



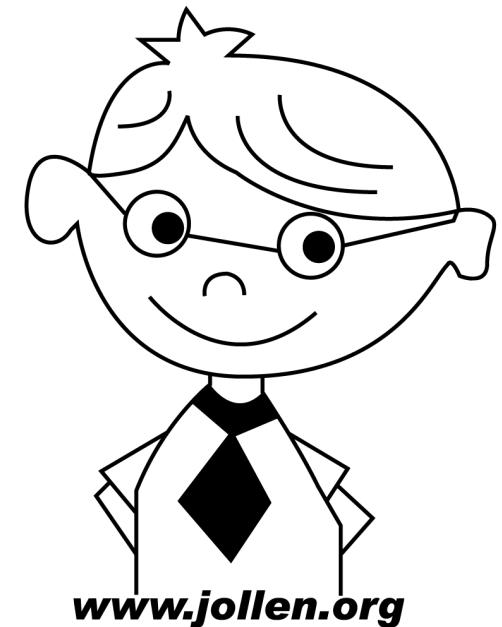
HAL 的 Open Interface

- Callback Open interface取得HAL Stub的operations



實作 Open Interface

```
static int led_device_open(const struct hw_module_t* module, const char*  
name, struct hw_device_t** device)  
{  
    struct led_control_device_t *dev;  
  
    dev = (struct led_control_device_t *)malloc(sizeof(*dev));  
    memset(dev, 0, sizeof(*dev));  
  
    dev->common.tag = HARDWARE_DEVICE_TAG;  
    dev->common.version = 0;  
    dev->common.module = module;  
    dev->common.close = led_device_close;  
  
    dev->set_on = led_on;  
    dev->set_off = led_off;  
  
    *device = &dev->common;  
  
success:  
    return 0;  
}
```

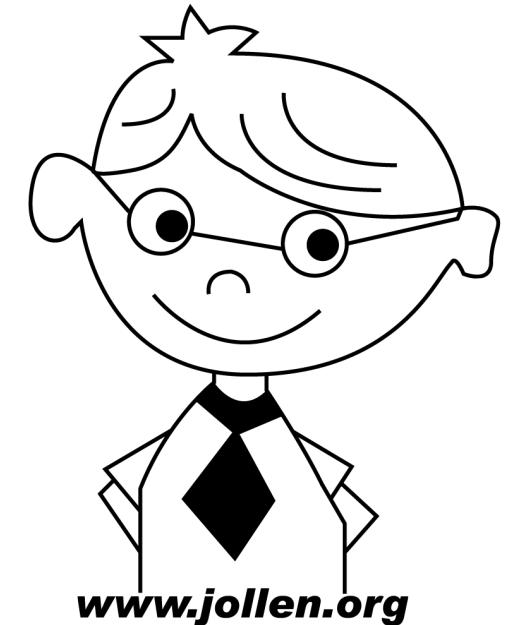


HAL Stub 控制硬體

```
int led_on(struct led_control_device_t *dev, int32_t led)
{
    LOGI("LED Stub: set %d on.", led);

    /* Put system calls here. */

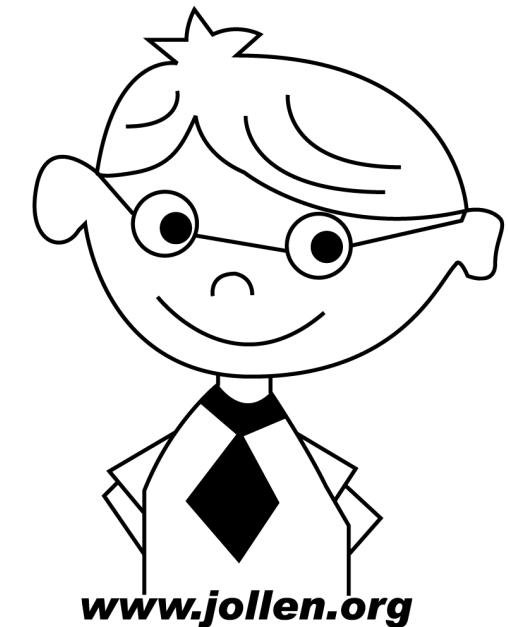
    return 0;
}
```



實機畫面, #1

File Edit View Terminal Tabs Help

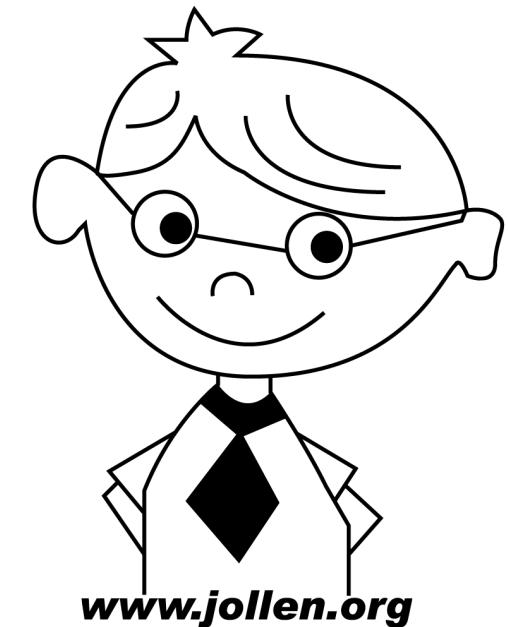
```
PLATFORM_VERSION=1.6
TARGET_PRODUCT=dma6410xp
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=Donut
=====
build/core/copy_headers.mk:15: warning: overriding
 commands for target `out/target/product/dma6410xp
 /obj/include/libpv/getactualaacconfig.h'
build/core/copy_headers.mk:15: warning: ignoring o
ld commands for target `out/target/product/dma6410
xp/obj/include/libpv/getactualaacconfig.h'
make: Nothing to be done for `PRODUCT-dma6410xp-en
g'.
jollen@android:~/try/mokoid$ 
```



實機畫面, #2

```
File Edit View Terminal Tabs Help
xp/obj/include/libpv/getactualaacconfig.h'
make: Nothing to be done for `PRODUCT-dma6410xp-eng'.
jollen@android:~/try/mokoid$ cd out/target/product/dma6410xp/
jollen@android:~/try/mokoid/out/target/product/dma6410xp$ ls
android-info.txt          ramdisk-uboot.img
clean_steps.mk            ramdisk-uboot.sh
data                      root
installed-files.txt       symbols
obj                       system
previous_build_config.mk  system.img
ramdisk.img                userdata.img
jollen@android:~/try/mokoid/out/target/product/dma6410xp$ ~/work/android-sdk-linux_x86-1.6_r1/tools/emulator -system system.img -data userdata.img -ramdisk ramdisk.img
Warning: No DNS servers found

```

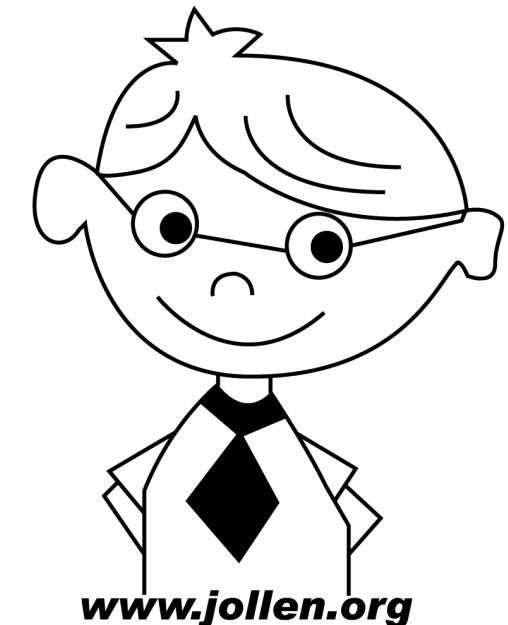


實機畫面, #3

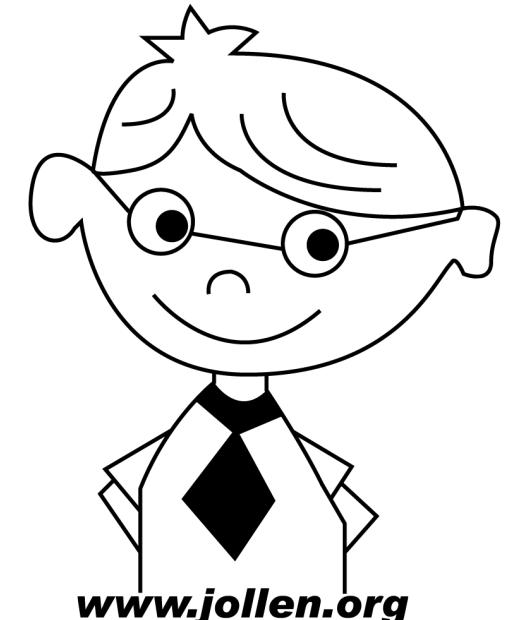
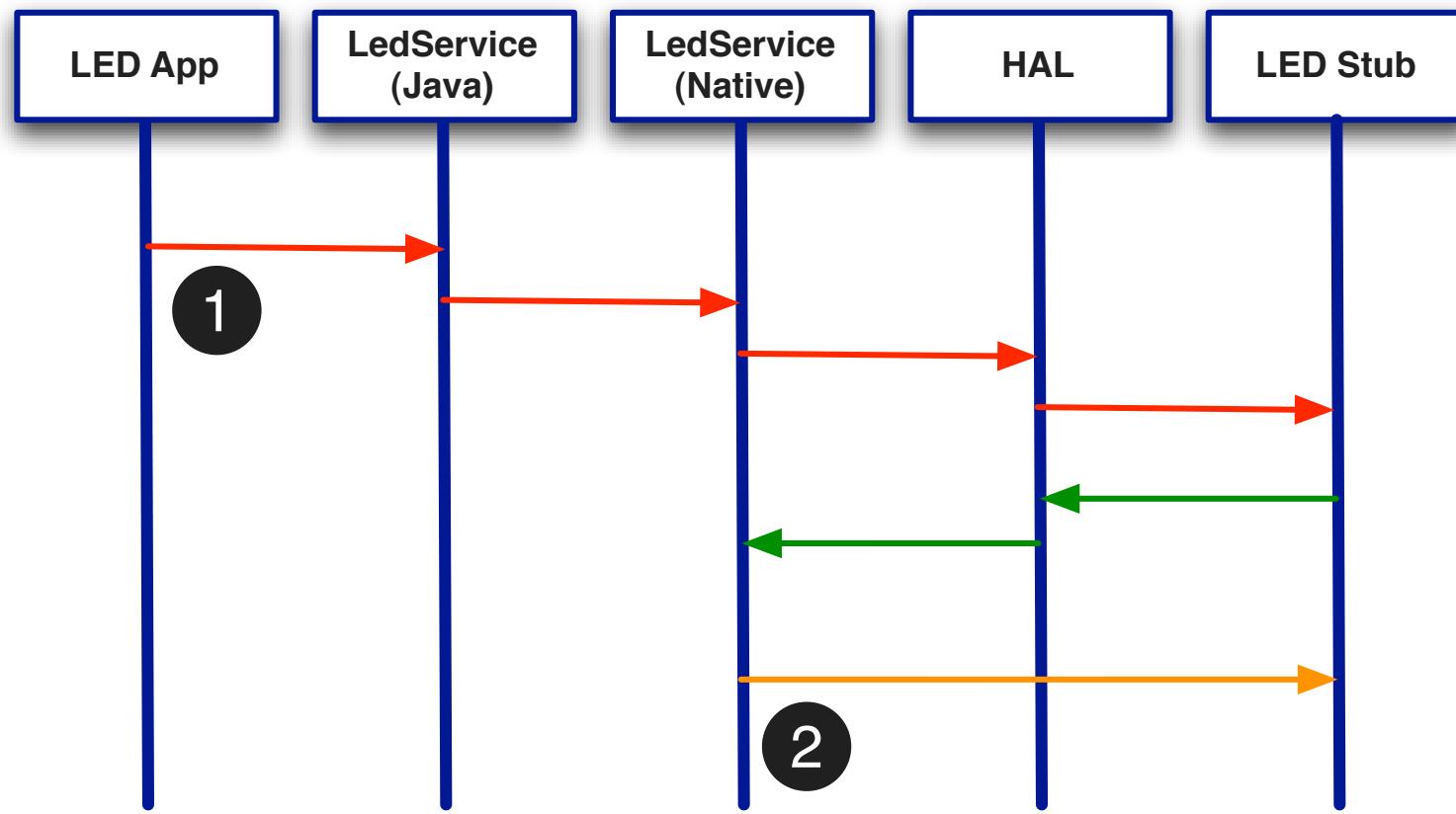


實機畫面, #4

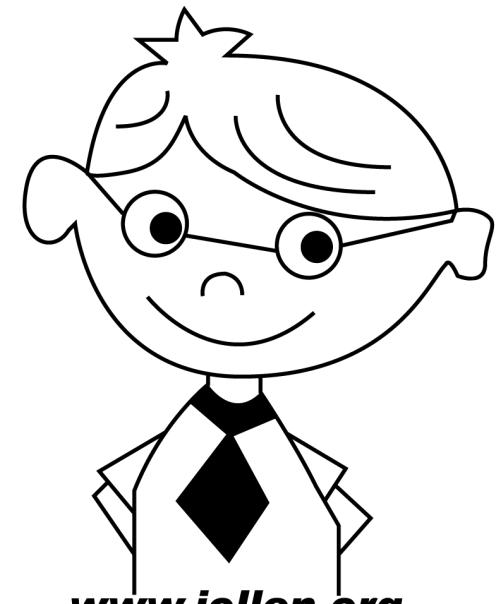
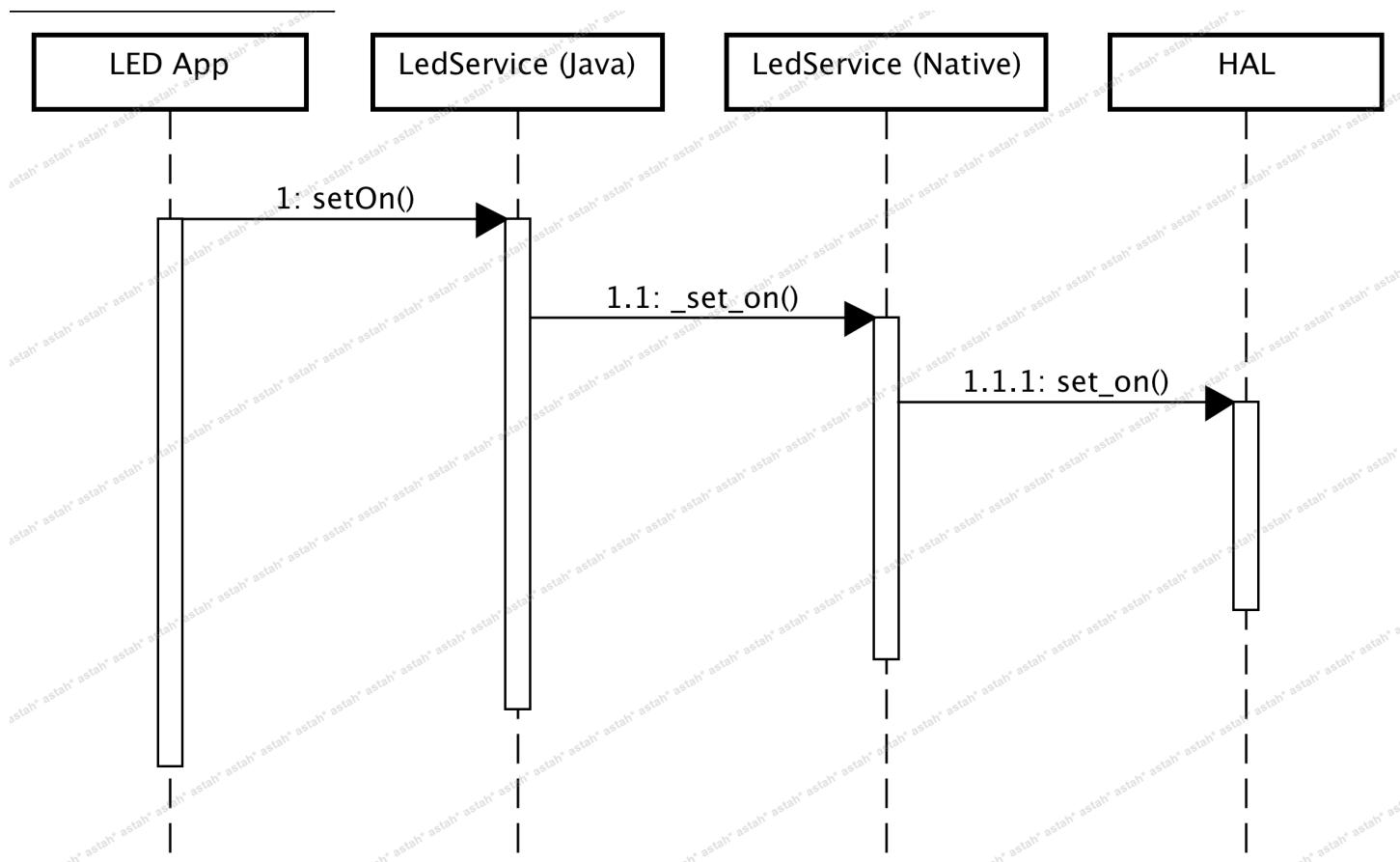
```
File Edit View Terminal Tabs Help
jollen@android: ~/try/mokoid/out/target/product/dma641... jollen@android: ~/try/mokoid/out/target/product/dma641...
80517000-80518000 rw-p 00017000 1f:00 430 /
system/lib/libomx_amrenc_sharedlibrary.so
80600000-80615000 r-xp 00000000 1f:00 452 /
system/lib/libomx_sharedlibrary.so
80615000-80616000 rw-p 00015000 1f:00 452 /
system/lib/libomx_sharedlibrary.so
80700000-80701000 r-xp 00000000 1f:00 493 /
system/lib/libmokoid_runtime.so
80701000-80702000 rw-p 00001000 1f:00 493 /
system/lib/libmokoid_runtime.so
80800000-80801000 r-xp 00000000 1f:00 491 /
system/lib/hw/led.goldfish.so
80801000-80802000 rw-p 00001000 1f:00 491 /
system/lib/hw/led.goldfish.so
9a200000-9a258000 r-xp 00000000 1f:00 482 /
system/lib/libsrec_jni.so
9a258000-9a259000 rw-p 00058000 1f:00 482 /
system/lib/libsrec_jni.so
9d800000-9d808000 r-xp 00000000 1f:00 466 /
system/lib/libdrm1.so
```



Callback HAL Stub



Callback set_on()



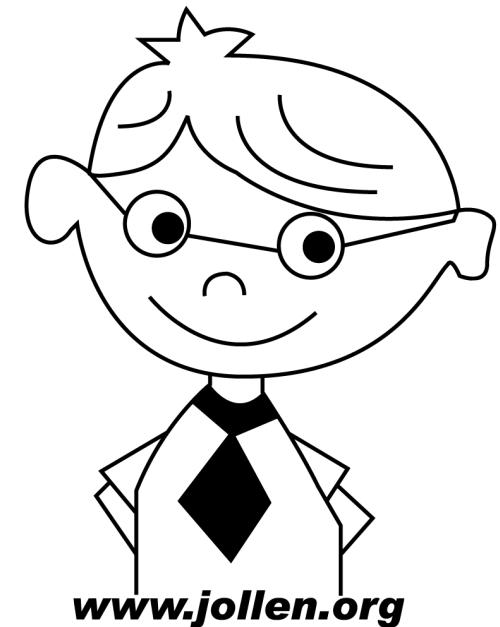


嵌入式產品設計 專業培訓 廠訓諮詢

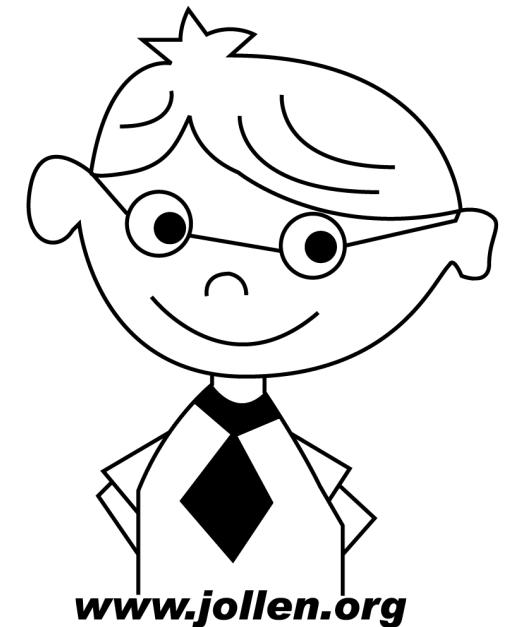
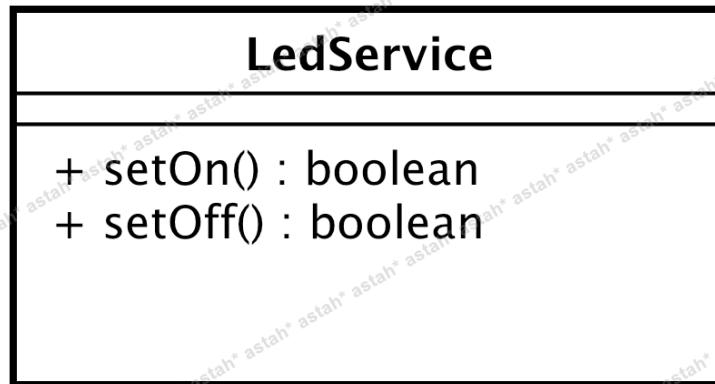
第04堂課： Extend Android API

本堂主題

第四堂	Extend Android API
4.1	如何加入 API 至 Android Framework
4.2	如何編譯並製作獨立 JAR 檔
4.3	上機實驗：開發LedService API 與製作mokoid.jar 程式庫



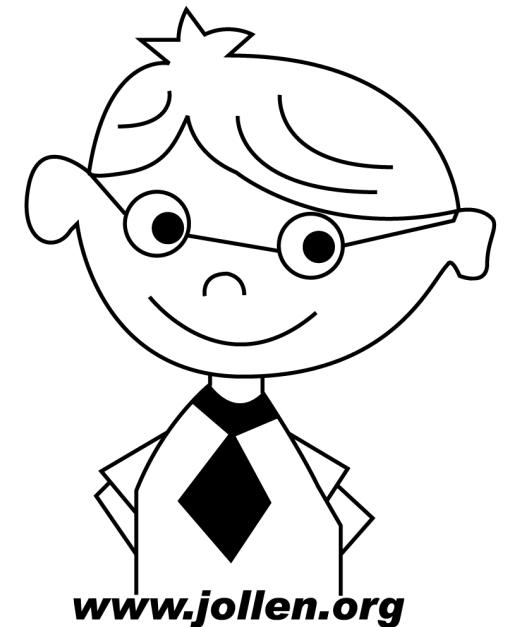
設計 LedService Class



www.jollen.org

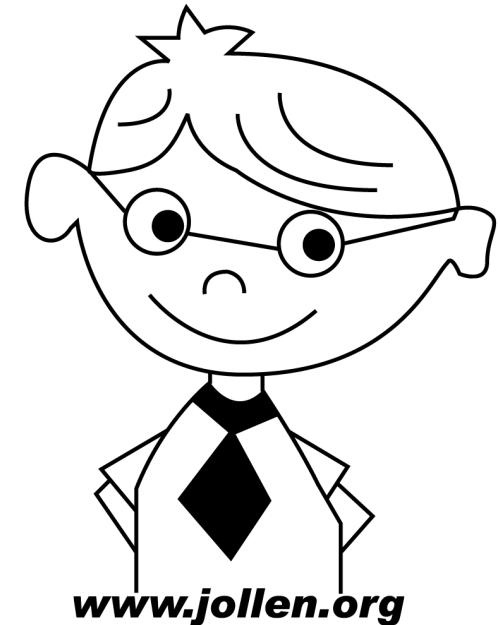
LedService Class 實作

```
public final class LedService {  
  
    /*  
     * Mokoid LED native methods.  
     */  
  
    public boolean setOn(int led) {  
        Log.i("MokoidPlatform", "LED On");  
    }  
  
    public boolean setOff(int led) {  
        Log.i("MokoidPlatform", "LED Off");  
    }  
}
```



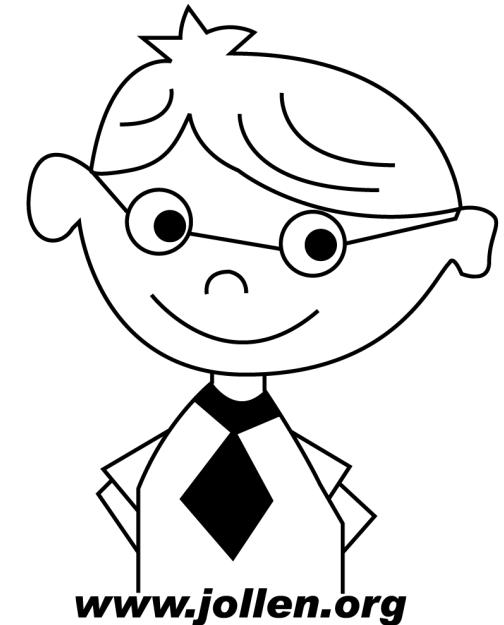
加入 Constructor

```
public final class LedService {  
  
    static {  
        System.loadLibrary("mokoid_runtime");  
    }  
  
    public LedService() {  
        _init();  
    }  
  
    ...  
}
```



建構子時期

```
public final class LedService {  
    ...  
    public LedService() {  
        _init();  
    }  
    ...  
}  
  
static const JNINativeMethod gMethods[] = {  
    {"_init",  
        "(OZ",  
        (void*)mokoid_init},  
    {"_set_on",  
        "(I)Z",  
        (void*)mokoid_setOn },  
    {"_set_off",  
        "(I)Z",  
        (void*)mokoid_setOff },  
};
```



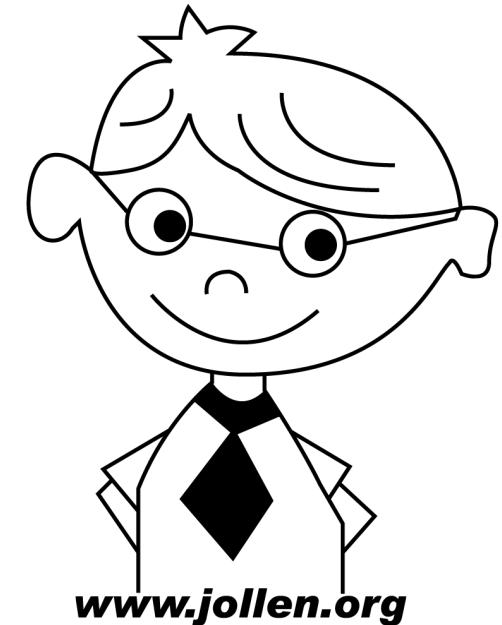


嵌入式產品設計 專業培訓 廠訓諮詢

第05堂課： JNI & Runtime Library

本堂主題

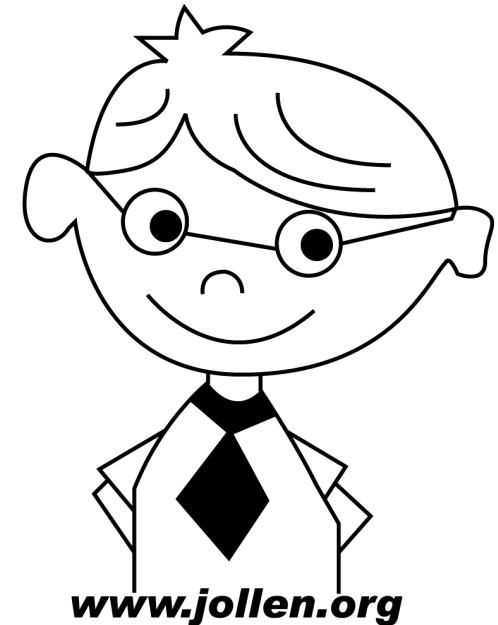
第五堂	JNI & Runtime Library
5.1	什麼是 JNI
5.2	如何撰寫 JNI & Native Method
5.3	如何製作 Android Runtime Library
5.4	專題討論：如何開發與製作Runtime Library



JNI & Native Method

- Dalvik VM 與 Host OS (Linux) 溝通的介面
- Native Method :

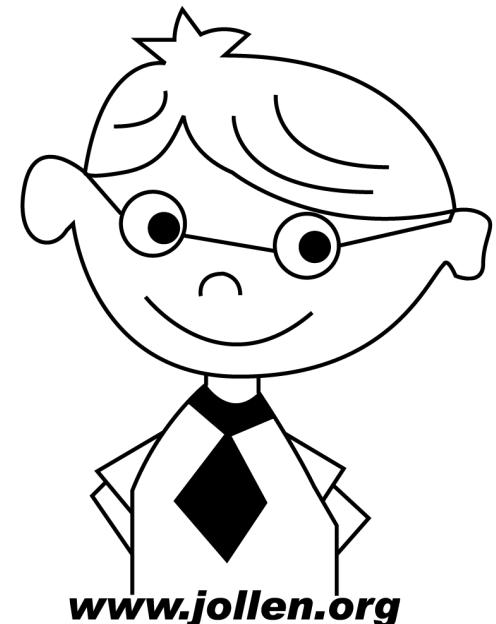
```
public final class LedService {  
    ...  
    private static native boolean _init();  
    private static native boolean _set_on(int led);  
    private static native boolean _set_off(int led);  
}
```



Native Method 對應 Native Function

- Dalvik VM 透過 JNI 呼叫 Host OS 的 Function
- 在 Linux 下以 (大部份) 存在於 shared library
- 根據 JNI 1.4 Spec：
 - shared library 裡加入 `JNI_OnLoad` 符號來註冊 JNI Table

```
static const JNINativeMethod gMethods[] = {  
    {"_init",                 "()Z",  
     (void*)mokoid_init},  
    {"_set_on",                "(I)Z",  
     (void*)mokoid_setOn },  
    {"_set_off",               "(I)Z",  
     (void*)mokoid_setOff },  
};
```



JNI_OnLoad Symbol

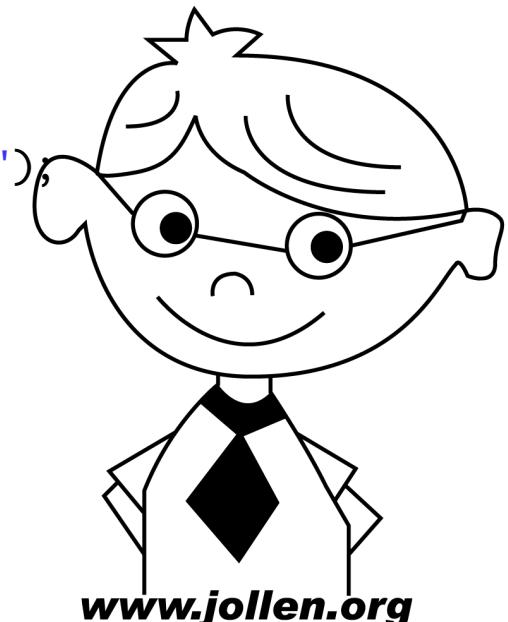
```
/*
 * This is called by the VM when the shared library is first loaded.
 */
jint JNI_OnLoad(JavaVM* vm, void* reserved) {
    JNIEnv* env = NULL;
    jint result = -1;

    if (vm->GetEnv((void**) &env, JNI_VERSION_1_4) != JNI_OK) {
        LOGE("ERROR: GetEnv failed\n");
        goto bail;
    }
    assert(env != NULL);

    if (registerMethods(env) != 0) {
        LOGE("ERROR: PlatformLibrary native registration failed\n");
        goto bail;
    }

    /* success -- return valid version number */
    result = JNI_VERSION_1_4;

bail:
    return result;
}
```



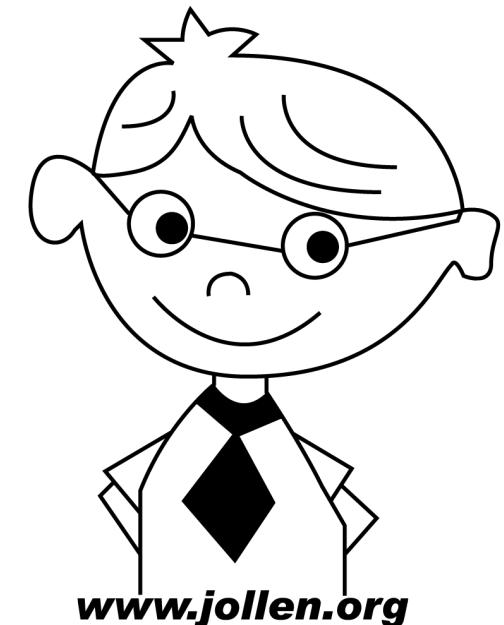
註冊JNI Table到VM

```
static int registerMethods(JNIEnv* env) {
    static const char* const kClassName =
        "com/mokoid/server/LedService";
    jclass clazz;

    /* look up the class */
    clazz = env->FindClass(kClassName);
    if (clazz == NULL) {
        LOGE("Can't find class %s\n", kClassName);
        return -1;
    }

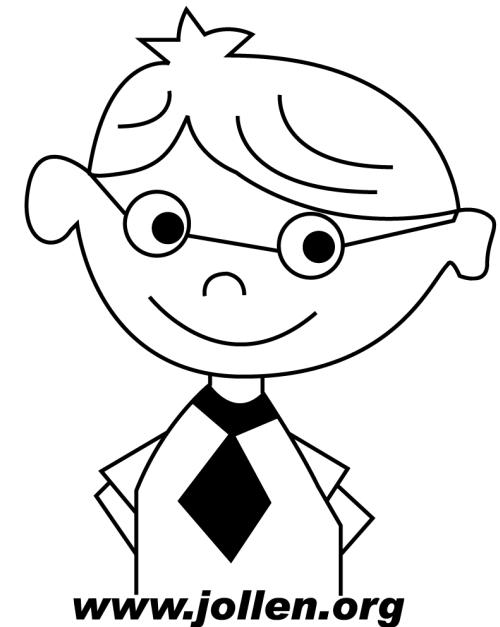
    /* register all the methods */
    if (env->RegisterNatives(clazz, gMethods,
                               sizeof(gMethods) / sizeof(gMethods[0])) != JNI_OK)
    {
        LOGE("Failed registering methods for %s\n", kClassName);
        return -1;
    }

    /* fill out the rest of the ID cache */
    return 0;
}
```



libhardware 的角色

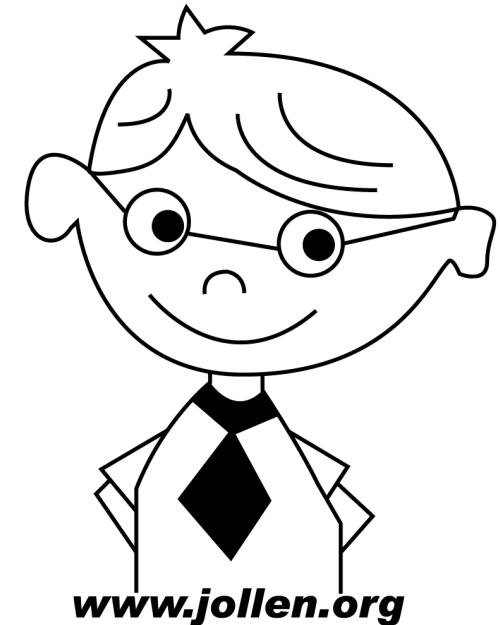
- libhardware讓驅動程式開發者撰寫HAL stub
- Native service呼叫hw_get_module API取得 HAL stub的callback ops
- Native service透過callback ops與驅動程式溝通
 - indirect function call



取得 HAL Stub

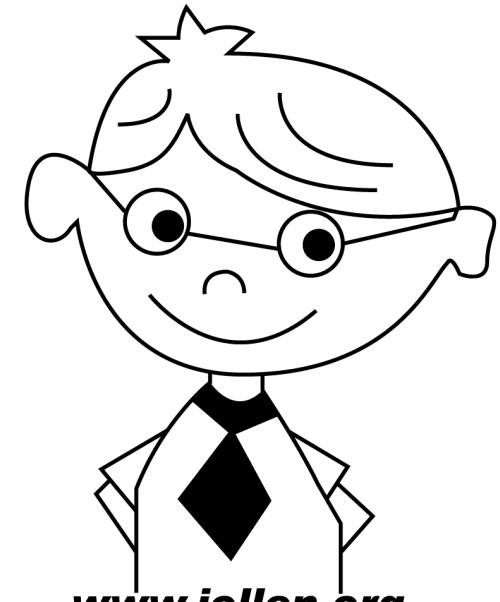
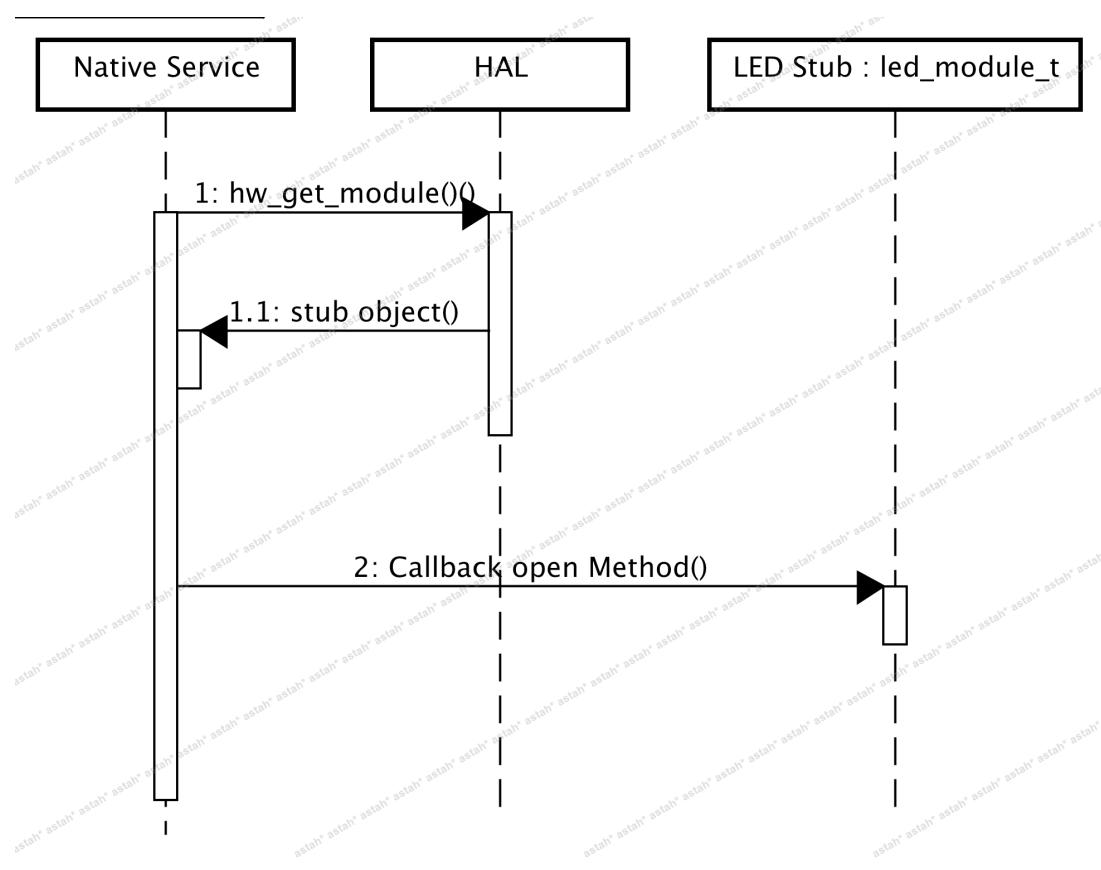
```
int hw_get_module(const char *id,  
                  const struct hw_module_t **module)
```

- **id: HAL module ID**
- ***module: HAL stub operations**



取得 Stub Operations

- Runtime callback HAL 的 Open interface
 - 取得 Stub operations



Callback Open Interface

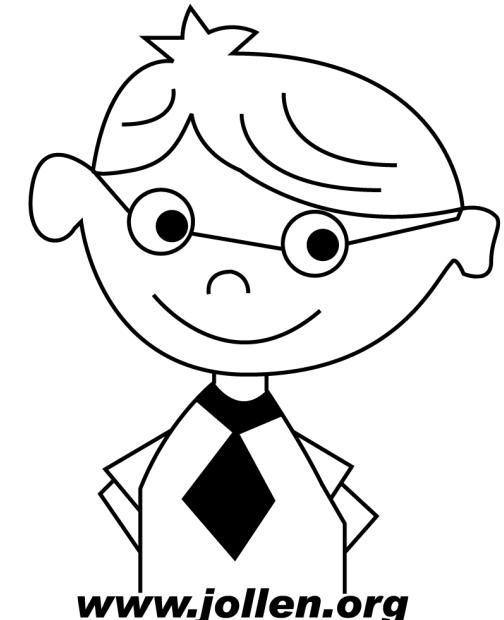
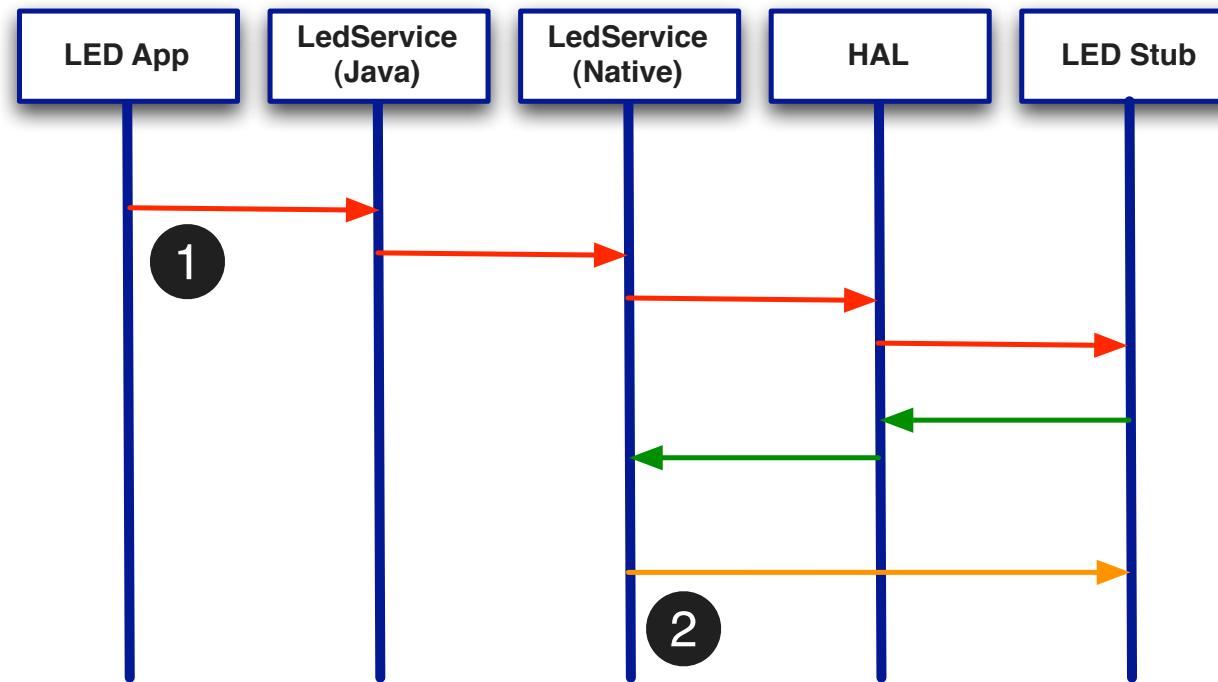
```
/** helper APIs */
static inline int led_control_open(const struct hw_module_t* module,
    struct led_control_device_t** device) {
    return module->methods->open(module,
        LED_HARDWARE_MODULE_ID, (struct hw_device_t**)device);
}

static jint
mokoid_init(JNIEnv *env, jclass clazz)
{
    led_module_t* module;

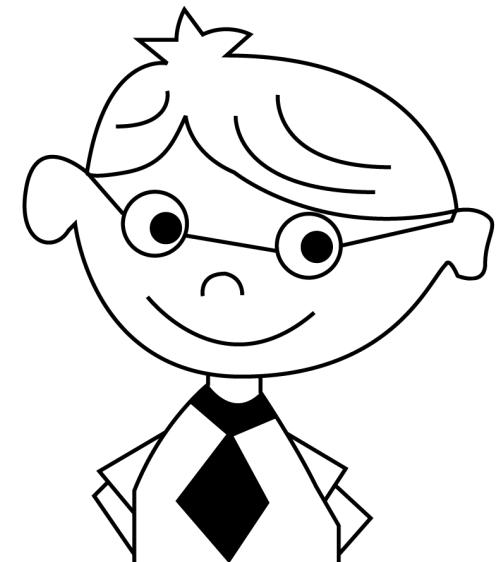
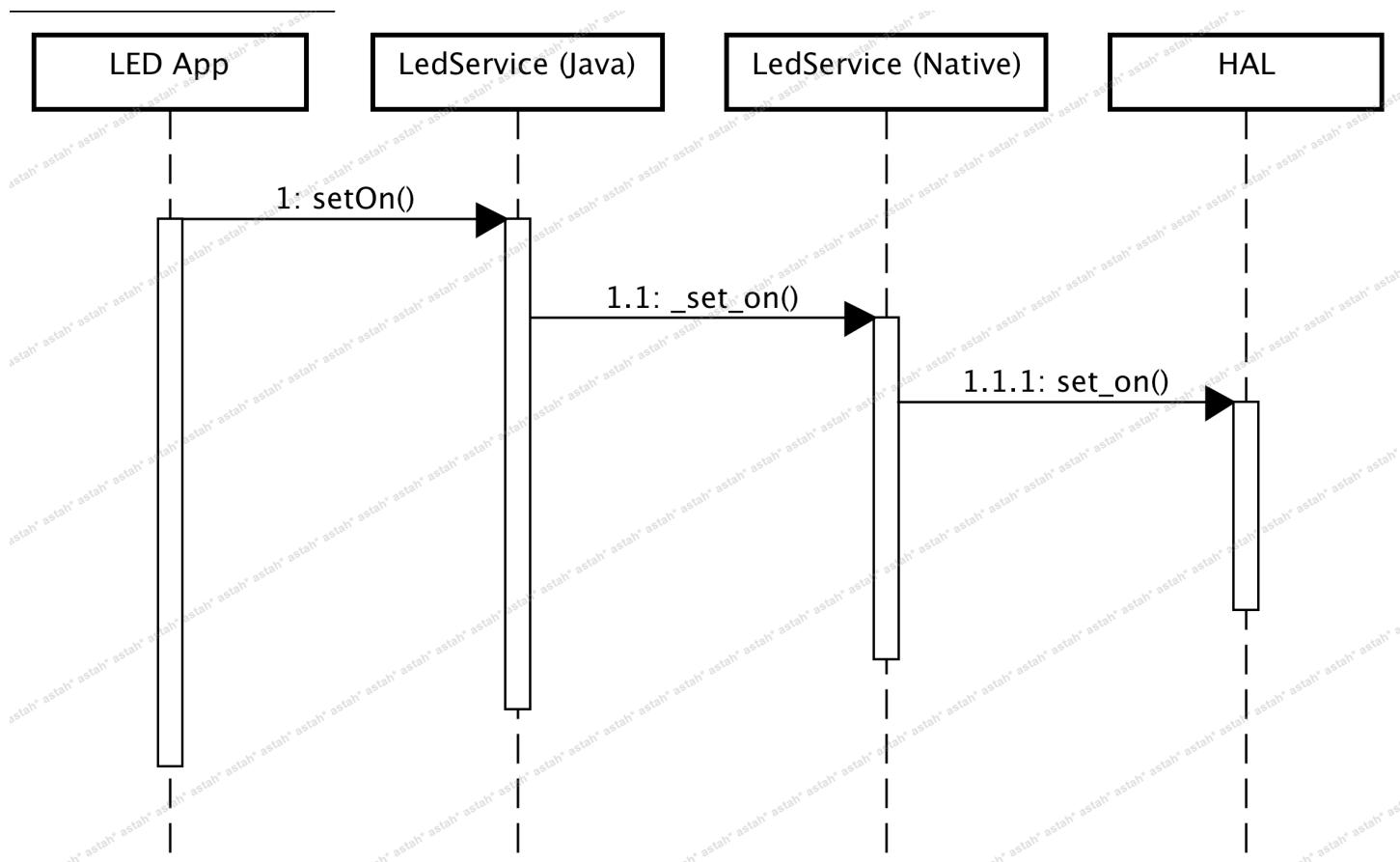
    if (hw_get_module(LED_HARDWARE_MODULE_ID, (const hw_module_t**)&module) == 0) {
        if (led_control_open(&module->common, &sLedDevice) == 0) {
            return -1;
        }
    }
    return 0;
}
```

Callback Stub Operation

- 取得 Stub 後、即可呼叫 HAL Stub operation
- 錯誤處理



呼叫 LedService.setOn() API



www.jollen.org

Callback HAL Interface

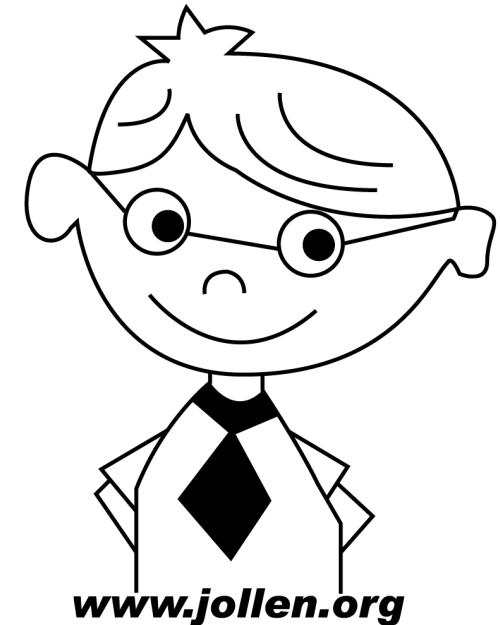
```
struct led_control_device_t *sLedDevice = NULL;

static jboolean mokoid_setOn(JNIEnv* env, jobject thiz, jint led) {
    LOGI("LedService JNI: mokoid_setOn() is invoked.");

    if (sLedDevice == NULL) {
        LOGI("LedService JNI: sLedDevice was not fetched correctly.");
        return -1;
    }

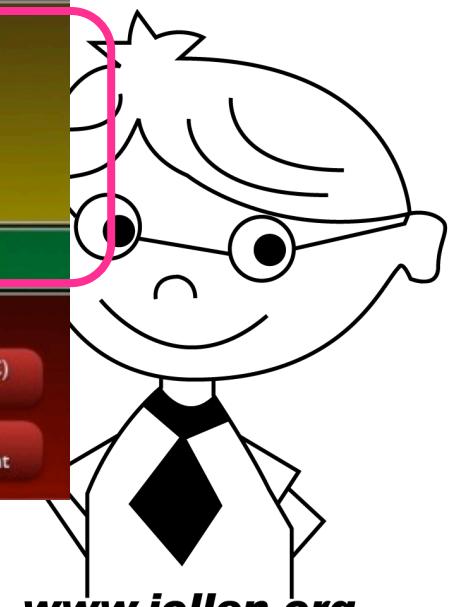
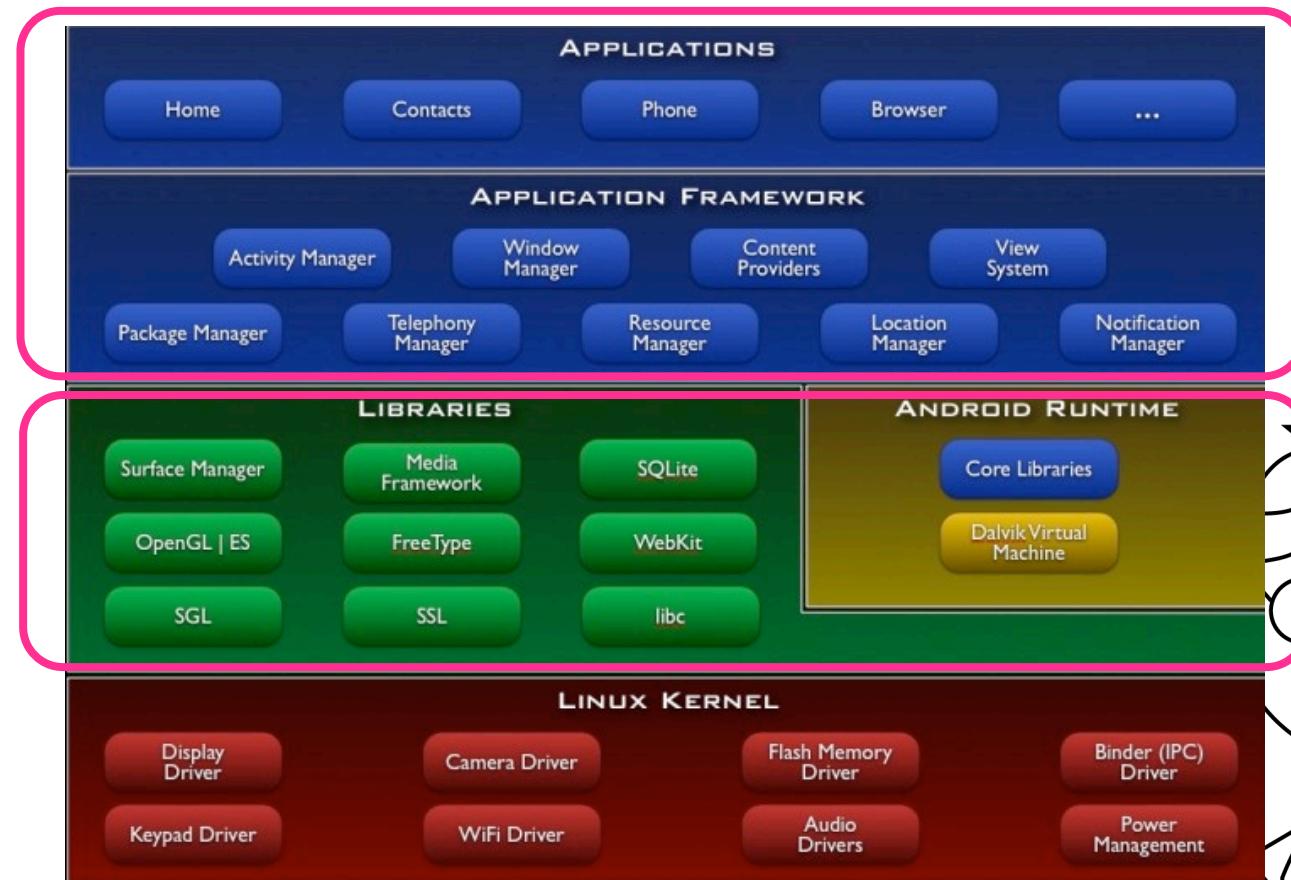
    if (sLedDevice->set_on == NULL)
        return -1;

    return sLedDevice->set_on(sLedDevice, led);
}
```



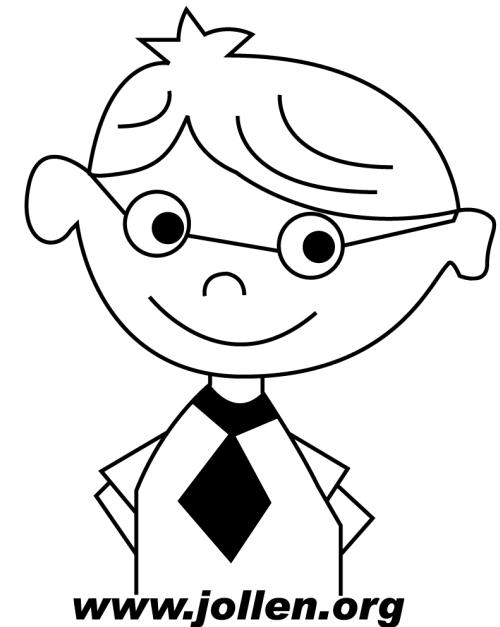
Managed Code

Managed Code



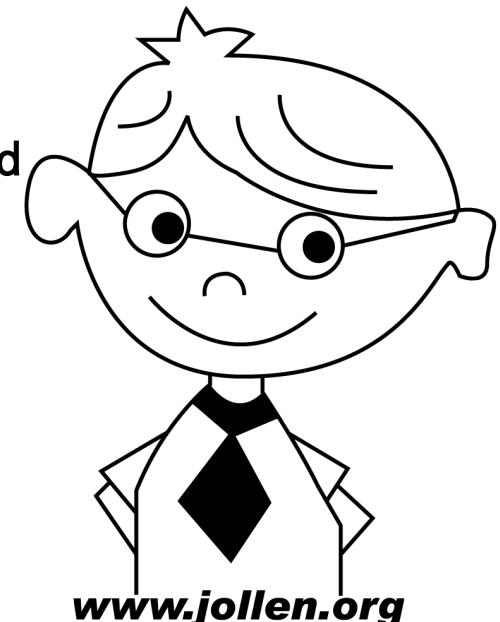
ANR

- Android Not Response
- ANR 僅出現在 Managed Code 端
- Native Code 不受管理
 - 仔細考量
 - 謹慎實作



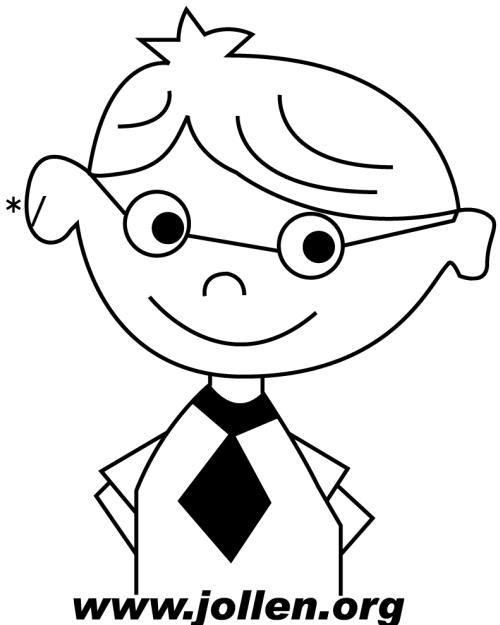
Blocking & Long Operations

- Kernel-space (Device Driver) 經常有 long operation 或是 blocking operation (Blocking I/O)
- Native method 呼叫的 native function 為 long operation 或 blocking I/O 時：
 - 應用程式稱此類 method 為同步 method
 - 呼叫此類 method 的 method (如 API) 亦為同步
 - 在 secondary thread (Java thread) 裡呼叫同步 method 才是正確做法 (ANR)
 - 或是在獨立的 background process 裡呼叫 (android.app.Service)



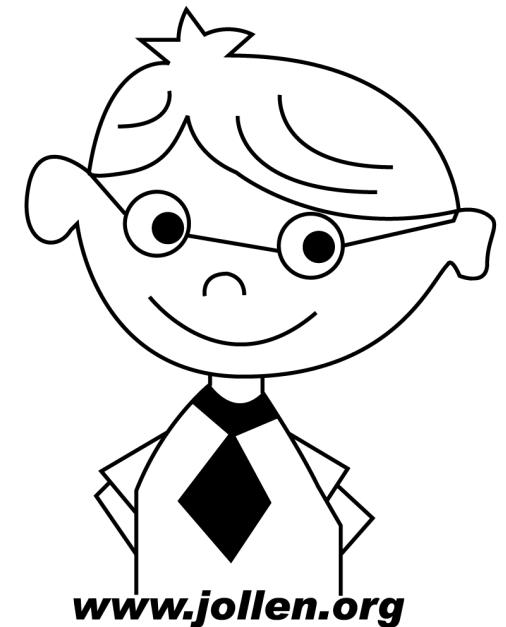
Doing Async Method Call

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    Button btn = (Button)findViewById(R.id.playBtn);  
  
    btn.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View view) {  
            /* Perform block or long operations. */  
            new Thread() {  
                @Override  
                public void run() {  
                    super.run();  
  
                    /* Invoke synchronious methods here. */  
                }  
            }.start();  
        }  
    });  
}
```



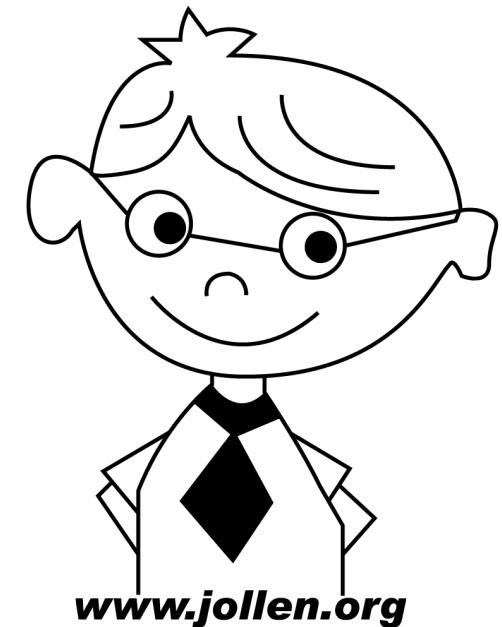
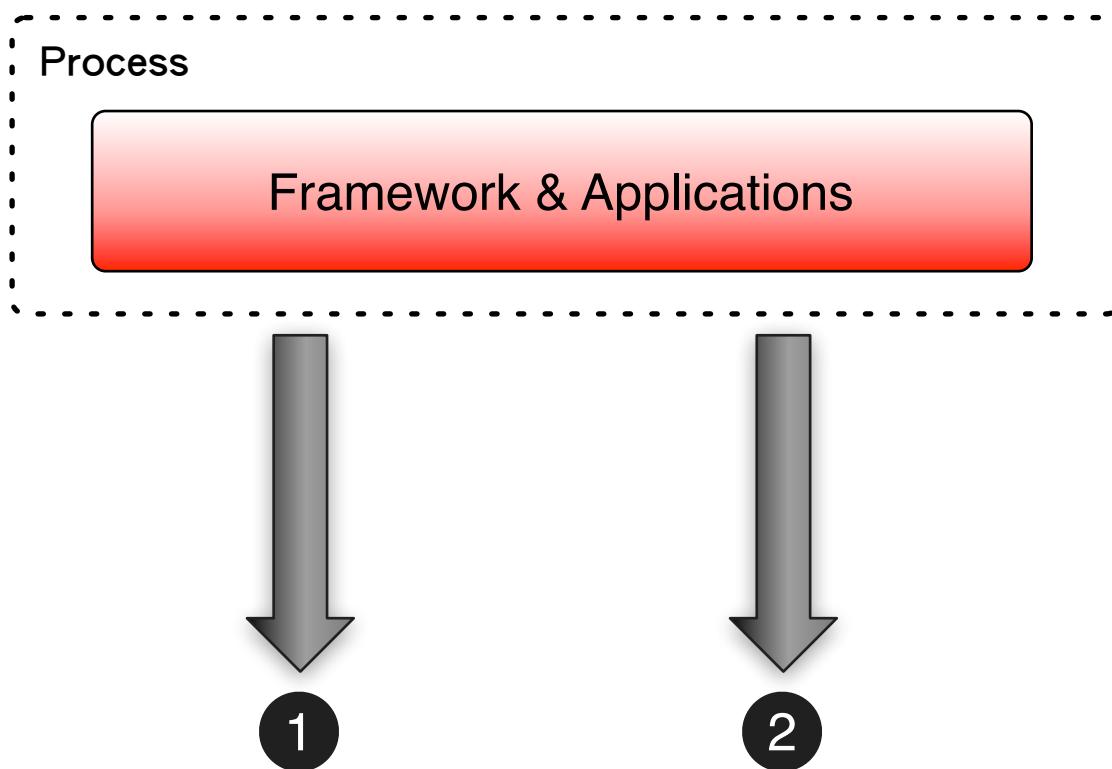
細節存在 API

- 將上述的考慮與實作細節移至 Framework
- Manager API 設計時、考量各種細節
- 讓 Application 簡單化



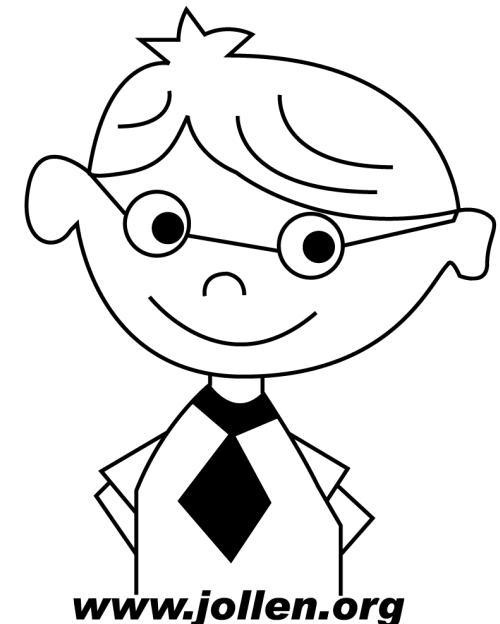
應用程式有自己的Dalvik

- JNI以上為一個process
- 意即每一個Android應用程式都有一台虛擬機(Dalvik)



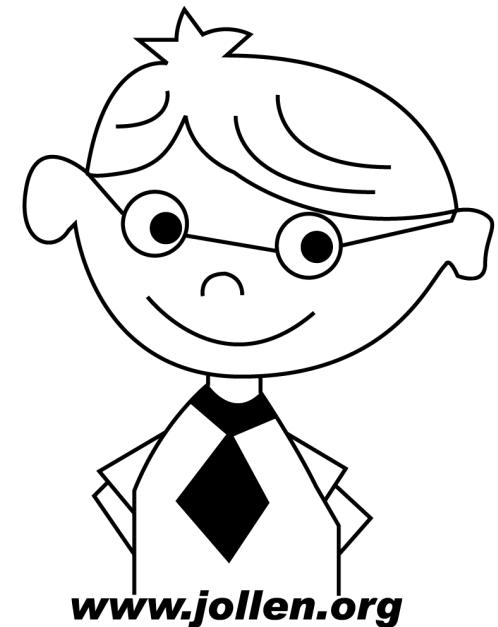
Shared Library Mapping

- 典型的C/C++程式以dynamic linking方式將*.so mapping到自己的virtual address space
- 編譯時需要「能」參考到*.so檔裡的符號(symbol)
- 那麼Android的process呢？

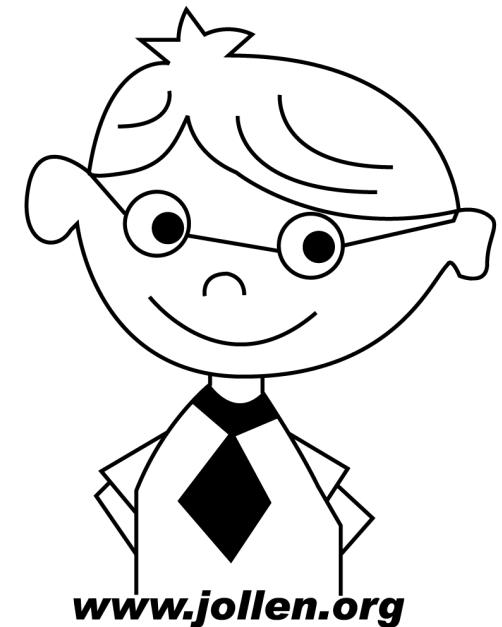
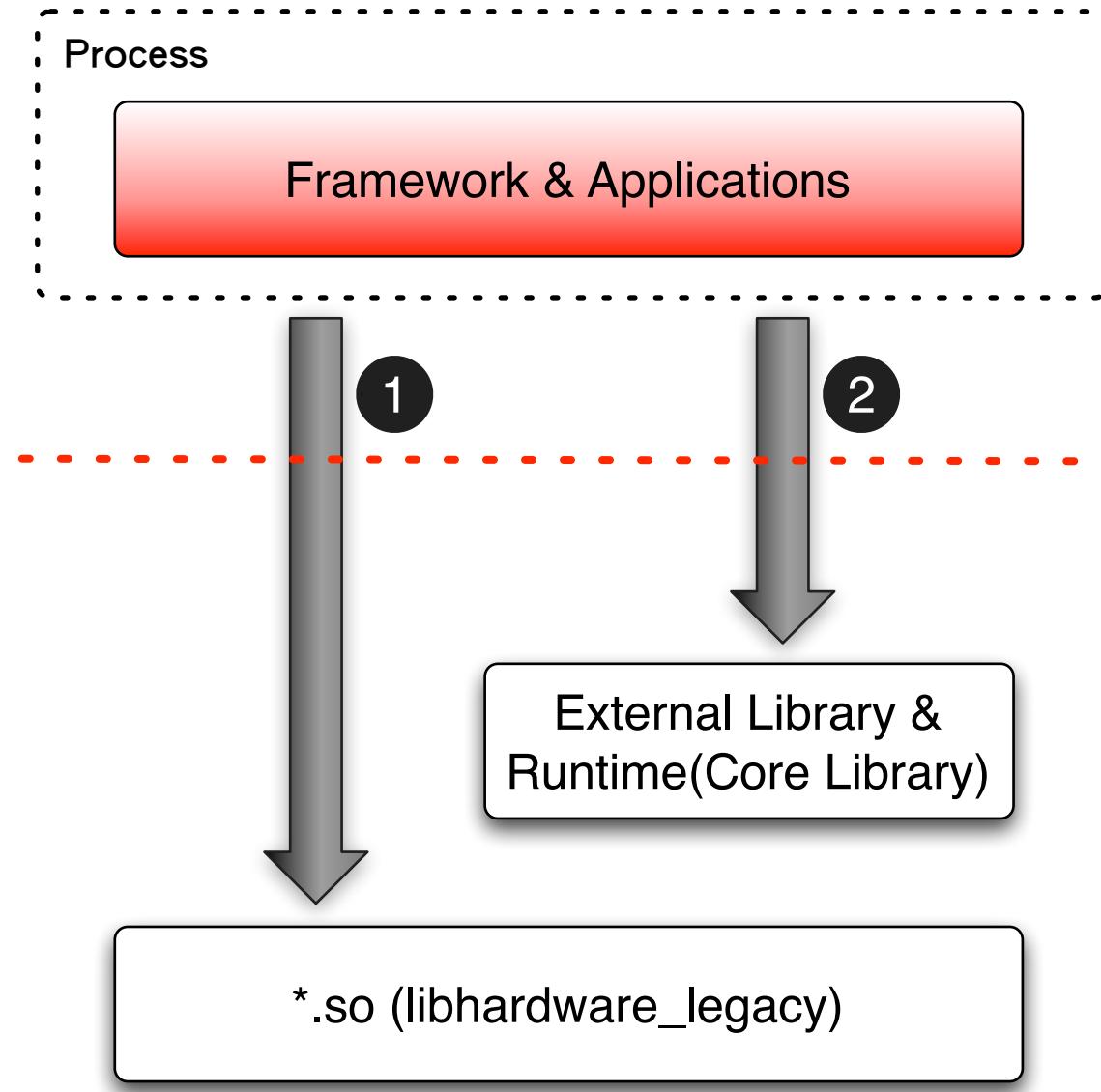


Dynamic Linker 探討

- Android process直接透過JNI呼叫*.so裡的function；或是
- 先經由Runtime裡的core libraries，再呼叫*.so裡的function

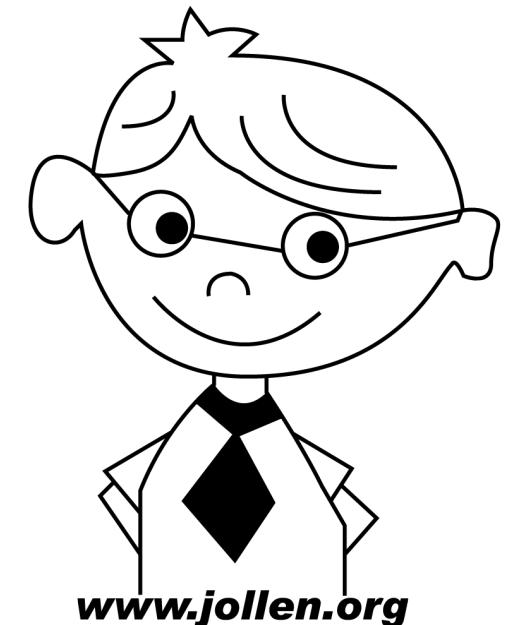
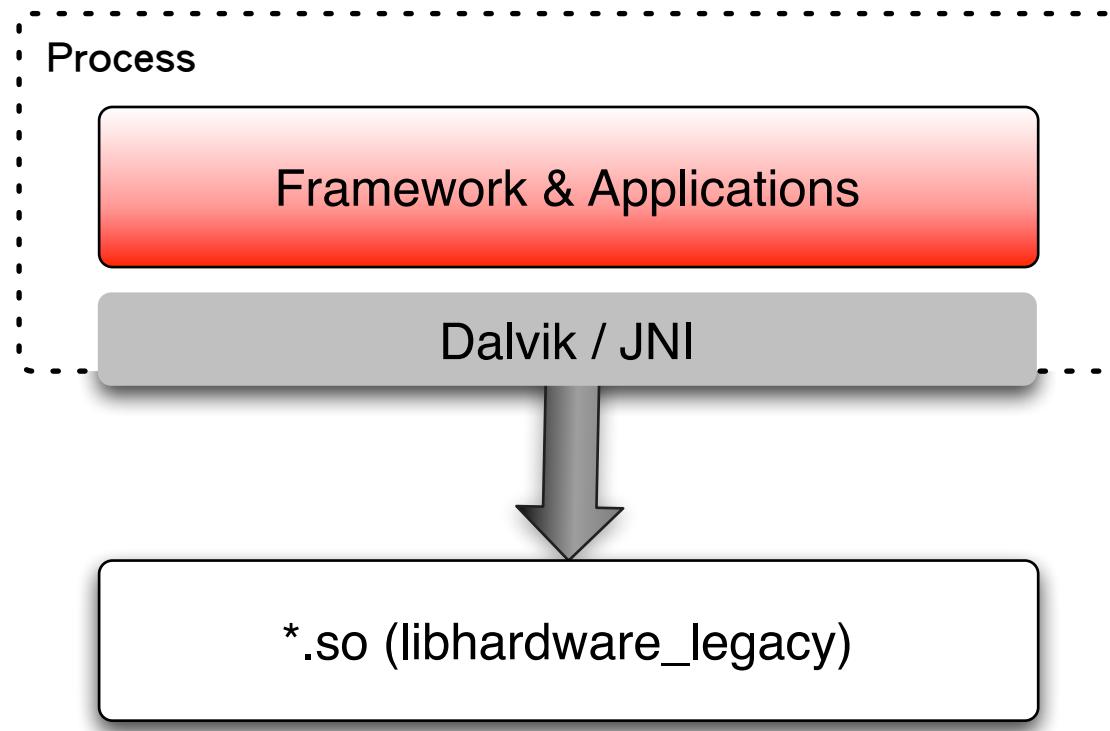


透過 JNI_OnLoad 符號



NDK 的做法

- NDK 的做法
- 繼存於現有架構直至 HAL 完善



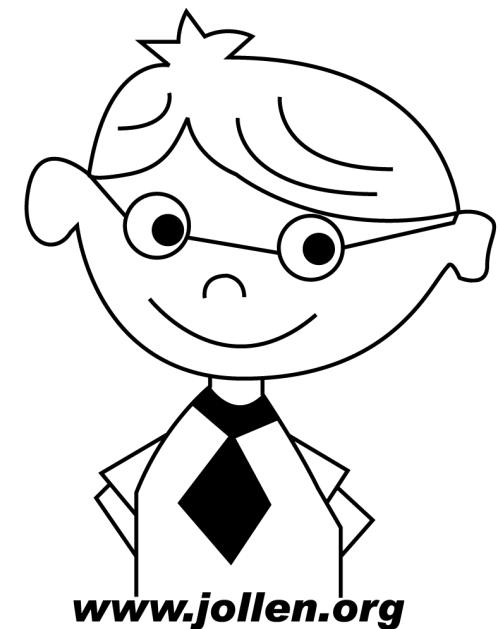
Android Runtime

□ Runtime(Core library)

- 包含 Native Service
- 目前為簡單的 Native Function 實作
- C/C++ 程式碼

□ Java Service

- Java Service 往上接 Manager API
- 往下接 JNI、Runtime



Android Service

□ 往上接 Manager API

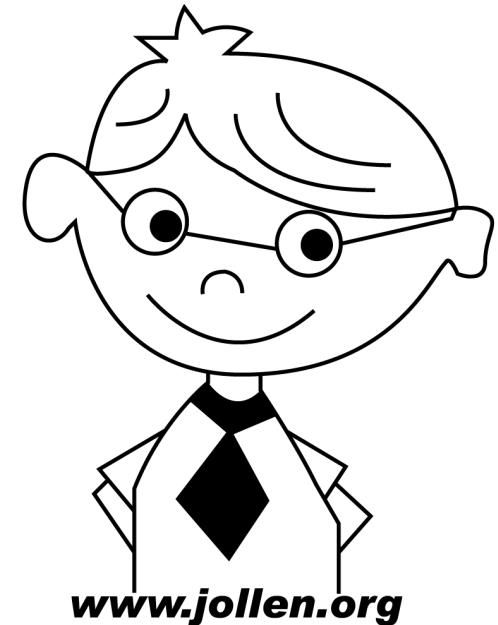
- 例如：SensorManager

□ Java Service 相對應一套 Manager API

- SensorService vs SensorManager
- WifiService vs WifiManager

□ Remote Object 關係

- AIDL
- Proxy Object





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。

Q&A