



# Android

## 深度探索 (卷1)

### HAL与驱动开发

李宁 编著

第一本结合 S3C6410 开发板学习 Android  
HAL 开发和驱动开发的实战书籍



- **3大完整的驱动开发案例：** 蜂鸣器驱动
- **3大真实的实验环境：** Ubuntu Linux 12.04 LTS、Android 模拟器和 S3C6410 开发板
- **必知必会的驱动开发高级技术：** 并发控制、阻塞和非阻塞 I/O、异步编程、Linux 中断和底半部、时间管理、内存管理和 I/O 访问等
- **6大核心 Linux 驱动代码分析与实战：** RTC 驱动、LCD 驱动、音频驱动、块设备驱动、网络设备驱动和 USB 驱动

人民邮电出版社  
POSTS & TELECOM PRESS



# Android

深度探索 (卷1)



## HAL与驱动开发

李宁 编著



YZLI0890169342

人民邮电出版社  
北京

## 图书在版编目(CIP)数据

Android深度探索. 第1卷, HAL与驱动开发 / 李宁编  
著. — 北京: 人民邮电出版社, 2013.1  
ISBN 978-7-115-29802-7

I. ①A… II. ①李… III. ①移动终端—应用程序—  
程序设计 IV. ①TN929.53

中国版本图书馆CIP数据核字(2012)第249939号

## 内 容 提 要

全书分为4篇,分别从搭建开发环境, Linux驱动和 Android HAL的基础知识, 开发 Linux驱动的高级技术和分析典型的 Linux驱动源代码4个方面介绍 Android和 Linux的底层开发。本书使用的试验环境是 Ubuntu Linux12.04 LTS、Android模拟器和 S3C6410开发板。在第1篇详细介绍了如何搭建和使用这3个试验环境。第2篇通过3个 Linux驱动的完整案例(统计单词个数驱动、LED驱动和蜂鸣器驱动)从不同角度来讨论如何开发一个完整的 Linux驱动。并且通过完整的案例介绍了如何编写 Android HAL, 以及如何与 Linux驱动交互。第3篇则介绍了开发 Linux驱动所需要的高级技术, 这些技术包括并发控制、阻塞和非阻塞 I/O、异步编程、Linux中断和底半部、时间管理、内存管理和 I/O访问。最后一部分分析了一些典型 Linux驱动的源代码(RTC驱动、LCD驱动、音频驱动、块设备驱动、网络设备驱动和 USB驱动)。

本书注重理论和实践相结合。在介绍了大量的基础知识的同时, 为每一个知识点提供了完整的案例, 使读者可以通过实际的代码更好地理解 Linux驱动和 Android底层技术。

为了使读者更好地实践本书提供的实例代码, 在随书光盘中除了提供源代码文件外, 还提供了一个 VMWare Ubuntu Linux12.04 LTS的虚拟环境。读者可以在 Windows、Linux和 Mac OS X上, 通过 VMWare打开该虚拟机文件来学习和测试本书的例子(虚拟环境中也带了一套本书提供的例子代码)。

本书适合底层开发的程序员和编程爱好者使用, 也适合作为相关培训学校的 Android底层开发培训教材。

### Android 深度探索 (卷1): HAL 与驱动开发

- ◆ 编 著 李 宁  
责任编辑 张 涛
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号  
邮编 100061 电子邮件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
北京鑫正大印刷有限公司印刷
- ◆ 开本: 800×1000 1/16  
印张: 40.75  
字数: 969千字 2013年1月第1版  
印数: 1-3000册 2013年1月北京第1次印刷

ISBN 978-7-115-29802-7

定价: 99.00元(附光盘)

读者服务热线: (010)67132692 印装质量热线: (010)67129223

反盗版热线: (010)67171154

广告经营许可证: 京崇工商广字第0021号

# 目 录

第一篇	Android 驱动开发前的准备	1	2.4.4	导入 Android NDK 的例子	22
第 1 章	Android 系统移植与 驱动开发概述	3	2.4.5	配置 Android NDK 的 集成开发环境	22
1.1	Android 系统架构	3	2.5	安装交叉编译环境	25
1.2	Android 系统移植的主要工作	4	2.6	小结	28
1.3	查看 Linux 内核版本	5	第 3 章	Git 使用入门	29
1.4	Linux 内核版本号的定义规则	6	3.1	安装 Git	29
1.5	如何学习 Linux 驱动开发	7	3.2	查看 Git 文档	30
1.6	Linux 设备驱动	8	3.3	源代码的提交与获取	31
1.6.1	设备驱动的发展和作用	8	3.3.1	创建版本库: git init	31
1.6.2	设备的分类及特点	9	3.3.2	将文件提交到本地 版本库: git commit	32
1.7	见识一下什么叫 Linux 驱动: LED	9	3.3.3	创建本地分支: git branch	33
1.8	小结	11	3.3.4	切换本地分支: git checkout	34
第 2 章	搭建 Android 开发环境	12	3.3.5	在 GitHub 上创建 开源项目	34
2.1	Android 底层开发需要 哪些工具	12	3.3.6	上传源代码到 GitHub: git push	36
2.2	安装 JDK	13	3.3.7	从 GitHub 下载源代码: git clone	39
2.3	搭建 Android 应用程序 开发环境	13	3.4	小结	39
2.3.1	安装 Android SDK	13	第 4 章	源代码的下载和编译	41
2.3.2	安装 Eclipse	15	4.1	下载、编译和测试 Android 源代码	41
2.3.3	安装 ADT	16	4.1.1	配置 Android 源代码 下载环境	41
2.3.4	配置 ADT	18	4.1.2	Android 源代码目录 结构解析	43
2.3.5	建立 AVD	19			
2.4	安装 Android NDK 开发环境	20			
2.4.1	下载 Android NDK	20			
2.4.2	安装 CDT	21			
2.4.3	命令行方式编译 Android NDK 程序	21			

4.1.3	下载 Android 源代码中的一部分	44	6.3.6	实现统计单词数的算法	84
4.1.4	编译 Android 源代码	46	6.3.7	编译、安装、卸载 Linux 驱动程序	87
4.1.5	out 目录结构分析	48	6.4	使用多种方式测试 Linux 驱动	88
4.1.6	将自己的 APK 作为 Android 内置程序发布	49	6.4.1	使用 Ubuntu Linux 测试 Linux 驱动	88
4.1.7	用模拟器测试 system.img 文件	50	6.4.2	在 Android 模拟器上通过原生 (Native) C 程序测试 Linux 驱动	90
4.2	下载和编译 Linux 内核源代码	51	6.4.3	使用 Android NDK 测试 Linux 驱动	93
4.2.1	下载 Linux 内核源代码	51	6.4.4	使用 Java 代码直接操作设备文件来测试 Linux 驱动	98
4.2.2	Linux 内核源代码的目录结构	51	6.4.5	使用 S3C6410 开发板测试 Linux 驱动	100
4.2.3	安装 Android 内核的编译环境	52	6.4.6	将驱动编译进 Linux 内核进行测试	101
4.2.4	配置和编译 Linux 内核	53	6.5	使用 Eclipse 开发和测试 Linux 驱动程序	105
4.3	小结	57	6.5.1	在 Eclipse 中开发 Linux 驱动程序	105
<b>第 5 章</b>	<b>搭建 S3C6410 开发板的测试环境</b>	<b>58</b>	6.5.2	在 Eclipse 中测试 Linux 驱动	109
5.1	S3C6410 开发板简介	58	6.6	小结	110
5.2	安装串口调试工具: minicom	60	<b>第 7 章</b>	<b>LED 将为我闪烁: 控制发光二级管</b>	<b>111</b>
5.3	烧写 Android 系统	62	7.1	LED 驱动的实现原理	111
5.4	配置有线网络	65	7.2	编写 LED 驱动	112
5.5	小结	66	7.2.1	体验 LED 驱动的奇妙	112
<b>第二篇</b>	<b>Android 底层开发入门</b>	<b>67</b>	7.2.2	创建 LED 驱动的设备文件	113
<b>第 6 章</b>	<b>第一个 Linux 驱动程序: 统计单词个数</b>	<b>69</b>	7.2.3	卸载 LED 驱动的设备文件	118
6.1	Linux 驱动到底是个什么东西	69	7.2.4	设置寄存器与初始化 LED 驱动	118
6.2	编写 Linux 驱动程序步骤	70	7.2.5	控制 LED	121
6.3	第一个 Linux 驱动: 统计单词个数	71	7.2.6	LED 驱动的模块参数	123
6.3.1	编写 Linux 驱动程序前的准备工作	72	7.2.7	LED 驱动的完整代码	125
6.3.2	编写 Linux 驱动程序的骨架 (初始化和退出驱动)	72			
6.3.3	指定与驱动相关的信息	75			
6.3.4	注册和注销设备文件	78			
6.3.5	指定回调函数	80			

7.3	测试 LED 驱动	129	9.3.7	编写调用 Service 的 Java 库	186
7.3.1	编写测试 I/O 控制命令的 通用程序	130	9.3.8	测试 LED 驱动	187
7.3.2	使用 NDK 测试 LED 驱动	132	9.4	小结	188
7.3.3	使用 Java 测试 LED 驱动	135	<b>第 10 章</b>	<b>嵌入式 Linux 的调试技术</b>	189
7.4	LED 驱动的移植	136	10.1	打印内核调试信息: printk	189
7.5	小结	138	10.2	防止 printk 函数降低 Linux 驱动性能	192
<b>第 8 章</b>	<b>让开发板发出声音: 蜂鸣器驱动</b>	139	10.3	通过虚拟文件系统 (/proc) 进行数据交互	195
8.1	Linux 驱动的代码重用	139	10.4	调试工具	200
8.1.1	编译是由多个文件组成的 Linux 驱动	139	10.4.1	用 gdb 调试用户 空间程序	200
8.1.2	Linux 驱动模块的依赖 (导出符号)	142	10.4.2	用 gdbserver 远程调试 用户空间程序	201
8.2	强行卸载 Linux 驱动	146	10.4.3	用 kgdb 远程调试 内核程序	203
8.3	蜂鸣器 (PWM) 驱动	151	10.5	小结	204
8.3.1	蜂鸣器驱动的原理	151	<b>第三篇</b>	<b>Linux 驱动开发高级技术</b>	205
8.3.2	实现蜂鸣器驱动	152	<b>第 11 章</b>	<b>Linux 驱动程序中的并发控制</b>	207
8.3.3	测试蜂鸣器驱动	155	11.1	并发和竞态	207
8.4	小结	155	11.2	原子操作	208
<b>第 9 章</b>	<b>硬件抽象层: HAL</b>	156	11.2.1	整型原子操作	208
9.1	为什么要在 Android 中 加入 HAL	156	11.2.2	64 位整型原子操作	210
9.2	Android HAL 架构	157	11.2.3	位原子操作	211
9.3	为 LED 驱动增加 HAL	158	11.2.4	用原子操作阻止设备 文件被多个进程打开	212
9.3.1	编写一款支持 HAL 的 Linux 驱动程序的步骤	159	11.3	自旋锁 (Spin Lock)	214
9.3.2	颠覆 Linux 驱动的设计 理念: 精简 LED 驱动	159	11.3.1	自旋锁的使用方法	215
9.3.3	测试读写寄存器操作	166	11.3.2	使用自旋锁保护临界区	217
9.3.4	编写调用 LED 驱动的 HAL 模块	169	11.3.3	读写自旋锁	220
9.3.5	编写调用 HAL 模块的 Service	178	11.3.4	使用读写自旋锁保护 临界区	223
9.3.6	HAL 模块的存放路径和命名 规则	182	11.3.5	顺序锁 (seqlock)	228
			11.3.6	使用顺序锁写入正在 读取的共享资源	230

11.4	读—复制—更新 (RCU) 机制	232	13.2.3	Linux 驱动中的异步函数 ( <code>aio_read</code> 和 <code>aio_write</code> )	282
11.4.1	RCU 的原理	232	13.2.4	接收信号时异步 读取数据	283
11.4.2	RCU API	234	13.2.5	AIO 中的回调函数	285
11.4.3	RCU 的应用	237	13.3	小结	286
11.5	信号量 (Semaphore)	238	<b>第 14 章 Linux 中断和底半部</b> 287		
11.5.1	信号量的使用	239	14.1	什么是中断	287
11.5.2	信号量用于同步	240	14.2	中断处理程序	288
11.5.3	读写信号量	242	14.3	Linux 中断处理的核心: 顶半部和底半部	288
11.5.4	使用读写信号量保护 临界区	243	14.4	获取 Linux 系统的中断 统计信息	289
11.6	互斥体 (Mutex)	245	14.5	Linux 中断编程	290
11.7	完成量 (Completion)	248	14.5.1	注册中断处理程序	290
11.8	小结	252	14.5.2	注销中断处理程序	293
<b>第 12 章 Linux 驱动程序中的 阻塞和非阻塞 I/O</b> 253			14.5.3	编写中断处理函数	293
12.1	等待队列	253	14.5.4	共享中断处理程序	294
12.1.1	等待队列原理	253	14.5.5	禁止和激活中断	294
12.1.2	等待队列的 API	254	14.5.6	禁止和激活中断线	295
12.1.3	等待队列的使用方法	258	14.5.7	获取中断系统的状态	296
12.1.4	支持休眠和唤醒的 Linux 驱动	258	14.5.8	与中断编程相关 的函数和宏	296
12.2	轮询操作	261	14.6	实例: S3C6410 实时钟中断	297
12.2.1	用户空间的 <code>select</code> 函数	261	14.7	中断中上下文	299
12.2.2	内核空间的 <code>poll</code> 函数	261	14.8	中断的实现原理	300
12.2.3	以非阻塞的方式访问 Linux 驱动	262	14.9	底半部	303
12.3	小结	266	14.9.1	为什么要使用底半部	304
<b>第 13 章 Linux 驱动程序中的 异步编程</b> 267			14.9.2	实现底半部的机制	304
13.1	信号与异步通知	267	14.9.3	软中断	305
13.1.1	Linux 信号	267	14.9.4	Tasklet	309
13.1.2	接收 Linux 信号	268	14.9.5	实例: Tasklet 演示	313
13.1.3	发送信号	271	14.9.6	软中断处理线程 ( <code>ksoftirqd</code> )	314
13.2	异步 I/O (AIO)	276	14.9.7	工作队列 (work queue)	315
13.2.1	异步操作的 API	277	14.9.8	与工作队列相关的 API	321
13.2.2	异步读写本地文件	280	14.9.9	实例: 工作队列演示	322
			14.10	小结	324

<b>第 15 章 时间管理</b> .....	325	<b>16.4 全局缓存 (slab)</b> .....	370
15.1 Linux 内核中的时间概念.....	325	16.4.1 Slab 层的实现原理 .....	371
15.1.1 时钟频率 .....	326	16.4.2 Slab 分配器 .....	373
15.1.2 提高时钟频率的 优点和缺点 .....	326	16.4.3 示例: 从 Slab 高速缓存 中分配和释放对象 .....	375
15.2 节拍总数 (jiffies) .....	327	<b>16.5 Linux 内存池</b> .....	376
15.2.1 访问 jiffies.....	328	16.5.1 内存池的实现原理 .....	377
15.2.2 jiffies、时间和时钟 频率之间的转换 .....	328	16.5.2 示例: 从内存池 获取对象 .....	381
15.2.3 jiffies 的回绕.....	331	<b>16.6 虚拟地址与物理地址   之间的转换</b> .....	384
15.2.4 用户空间和时钟频率 .....	332	<b>16.7 设备 I/O 端口与 I/O 内存</b> .....	385
15.3 实时时钟和定时器 .....	333	16.7.1 读写 I/O 端口 .....	385
15.4 时钟中断处理程序的实现 .....	334	16.7.2 读写 I/O 内存 .....	385
15.5 读写本地时间 .....	337	16.7.3 将 I/O 端口映射为 I/O 内存.....	387
15.6 内核定时器 .....	340	16.7.4 申请和释放设备 I/O 端口和 I/O 内存.....	387
15.6.1 如何使用内核定时器 .....	341	16.7.5 使用设备 I/O 端口和 I/O 内存的一般步骤 .....	388
15.6.2 实例: 秒表定时器 .....	343	<b>16.8 内核空间与用户空间   共享数据</b> .....	389
15.7 内核延迟 .....	347	16.8.1 内存映射与 VMA .....	390
15.7.1 忙等待 .....	347	16.8.2 示例: 用户程序读取 内核空间数据 .....	392
15.7.2 短延迟 .....	348	<b>16.9 I/O 内存静态映射</b> .....	395
15.7.3 休眠延迟 (schedule_timeout) .....	349	<b>16.10 小结</b> .....	397
15.8 小结 .....	351	<b>第四篇 Linux 设备驱动与 Android   底层开发</b> .....	399
<b>第 16 章 内存管理与 I/O 访问</b> .....	352	<b>第 17 章 RTC 驱动</b> .....	401
16.1 内存管理模式 .....	352	17.1 实时时钟 (RTC) 结构与移植 内容 .....	401
16.1.1 内存的基本单位: 页 (Page) .....	352	17.1.1 RTC 系统的结构.....	401
16.1.2 页的逻辑划分: 区 (zone) .....	354	17.1.2 RTC 驱动主要的 移植工作 .....	403
16.1.3 获取页 .....	361	<b>17.2 RTC 系统中的 Android 部分</b> .....	403
16.1.4 释放页 .....	363	17.2.1 警报管理: AlarmManager.....	403
16.2 分配连续的内存空间 (Kmalloc) .....	364		
16.2.1 gfp_mask 标志 .....	365		
16.2.2 释放内存 (kfree) .....	368		
16.3 分配不连续的内存空间 (vmalloc) .....	369		



17.2.2	警报服务: AlarmManagerService.....	406	驱动设计与实现.....	447
17.2.3	直接与 Alarm 驱动交互的 JNI 代码.....	409	18.3.1	FrameBuffer 设备.....
17.3	Alarm 驱动的分析与移植.....	411	18.3.2	示例: 通过 dd 命令与 FrameBuffer 设备文件 交互.....
17.3.1	Alarm 驱动简介.....	411	18.3.3	示例: 编写访问 FrameBuffer 设备 文件的程序.....
17.3.2	Alarm 驱动中的关键 数据结构.....	412	18.3.4	FrameBuffer 驱动的 架构.....
17.3.3	Alarm 驱动的应用层接口 (alarm_dev.c) 代码分析	414	18.3.5	FrameBuffer 驱动主要的 数据结构.....
17.3.4	Alarm 驱动的通用文件 (alarm.c) 代码分析.....	419	18.3.6	如何在 Linux 内核中查找指 定的内容.....
17.4	RTC 驱动的分析与移植.....	423	18.3.7	FrameBuffer 驱动设备事件 的处理 (fbmem.c) .....
17.4.1	实时时钟 (RTC) 的 特性.....	423	18.3.8	FrameBuffer 驱动源代码分 析与移植.....
17.4.2	RTC 的结构.....	423	18.4	FrameBuffer 驱动的 HAL 层 分析.....
17.4.3	RTC 芯片的寄存器.....	425	18.4.1	Gralloc 库.....
17.4.4	RTC 驱动的用户 空间接口.....	427	18.4.2	初始化 HAL Gralloc 的 核心结构体.....
17.4.5	RTC 系统组件之间的 调用关系.....	428	18.4.3	获取 Gralloc HAL 模块.....
17.4.6	设备文件 (/dev/rtc0) 的 I/O 命令.....	435	18.4.4	与 FrameBuffer 设备文件 交互.....
17.4.7	sysfs 虚拟文件 处理函数.....	437	18.5	调用 Gralloc HAL 库.....
17.4.8	proc 虚拟文件 处理函数.....	438	18.6	小结.....
17.5	小结.....	440	第 19 章	音频驱动.....
第 18 章	LCD 驱动.....	441	19.1	音频驱动基础.....
18.1	LCD 简介.....	441	19.1.1	数字音频简介.....
18.1.1	液晶的工作原理.....	441	19.1.2	ALSA 架构简介.....
18.1.2	LCD 的种类.....	442	19.1.3	ALSA 设备文件.....
18.1.3	LCD 的技术参数.....	444	19.1.4	数字采样与数字录音.....
18.1.4	LCD 时序图.....	444	19.1.5	混音器.....
18.2	LCD 驱动结构分析和 移植要点.....	446	19.1.6	音频驱动的目录结构.....
18.3	帧缓冲 (FrameBuffer)		19.1.7	音频设备硬件接口.....
			19.1.8	ALSA 架构支持的

	声卡芯片	488		19.5.7	ASoC 架构中的 Platform	531
19.2	AC97 芯片的寄存器	489		19.6	音频驱动的 HAL 分析	534
	19.2.1 控制寄存器	489		19.6.1	实现 HAL Library	534
	19.2.2 状态寄存器	490		19.6.2	调用 HAL Library	541
	19.2.3 编解码器命令寄存器	490		19.7	小结	545
	19.2.4 编解码器状态寄存器	491		<b>第 20 章</b>	<b>Linux 块设备驱动</b>	<b>546</b>
	19.2.5 PCM 输出/输入通道 FIFO 数据寄存器	491		20.1	块设备简介	546
	19.2.6 MIC 输入通道 FIFO 地址寄存器	491		20.2	块设备的体系架构	546
	19.2.7 PCM 输出/输入通道 FIFO 数据寄存器	492		20.3	块设备的数据结构与相关操作	549
	19.2.8 MIC 输入通道 FIFO 数据寄存器	492		20.3.1	磁盘设备 (gendisk 结构体)	549
19.3	创建声卡	492		20.3.2	block_device_operations 结构体	550
	19.3.1 声卡的顶层数据结构	492		20.3.3	I/O 请求 (request 结构体)	551
	19.3.2 创建声卡的步骤	493		20.3.4	请求队列 (request_queue 结构体)	553
	19.3.3 示例: 基于 ARM 的 AC97 音频驱动	496		20.3.5	块 I/O (bio 结构体)	555
19.4	音频逻辑设备	501		20.4	块设备的加载和卸载	557
	19.4.1 创建 PCM 设备	501		20.5	块设备的打开和释放	559
	19.4.2 创建录音和播放设备文件节点	504		20.6	块设备的 ioctl 函数	559
	19.4.3 创建 Control 设备数据结构	508		20.7	块设备驱动的 I/O 请求处理	560
	19.4.4 创建 Control 设备	514		20.7.1	依赖请求队列	560
	19.4.5 注册与打开音频字符设备	515		20.7.2	不依赖请求队列	561
19.5	嵌入式设备中的 ALSA (ASoC)	516		20.8	实例 1: 依赖请求队列的 RamDisk	562
	19.5.1 什么是 ASoC	517		20.9	在嵌入式设备上测试块设备驱动	567
	19.5.2 ASoC 的硬件架构	517		20.9.1	编译、配置和安装 Busybox	567
	19.5.3 ASoC 的软件架构	518		20.9.2	测试块设备驱动	569
	19.5.4 如何确定 S3C 开发板使用了哪个音频驱动	518		20.10	实例 2: 不依赖请求队列的 RamDisk	570
	19.5.5 ASoC 架构中的 Machine	520		20.11	扇区与磁盘碎片整理	576
	19.5.6 ASoC 架构中的 Codec	529		20.12	小结	577
				<b>第 21 章</b>	<b>网络设备驱动</b>	<b>578</b>

21.1	Linux 网络设备驱动的结构	578	22.2.1	端点 (Endpoint)	608
21.1.1	网络协议接口层	579	22.2.2	接口 (Interfaces)	609
21.1.2	网络设备接口层	583	22.2.3	配置 (Config)	609
21.1.3	设备驱动功能层	585	22.3	USB 设备的核心数据结构	610
21.1.4	网络设备与媒介层	585	22.3.1	USB 设备: usb_device 结构体	610
21.2	网络设备驱动设计与实现	586	22.3.2	USB 驱动: usb_driver 结构体	611
21.2.1	网络设备的注册与注销	586	22.3.3	识别 USB 设备: usb_device_id 结构体	612
21.2.2	网络设备的初始化	587	22.3.4	USB 端点: usb_host_endpoint 结构体	613
21.2.3	网络设备的打开与释放	588	22.3.5	USB 接口: usb_interface 结构体	613
21.2.4	发送数据	589	22.3.6	USB 配置: usb_host_config 结构体	614
21.2.5	接收数据	590	22.4	描述符数据结构	615
21.2.6	网络连接状态	590	22.4.1	设备描述符	615
21.3	示例: DM9000 网卡设备驱动	591	22.4.2	配置描述符	615
21.3.1	如何确定 S3C6410 开发板 使用的网络设备	591	22.4.3	接口描述符	615
21.3.2	DM9000 网卡硬件描述	591	22.4.4	端点描述符	616
21.3.3	网络设备驱动的定义与 安装	592	22.4.5	字符串描述符	616
21.3.4	初始化 DM9000 网卡 设备驱动	593	22.4.6	查看描述符信息	616
21.3.5	移出网络设备	597	22.5	USB 和 sysfs	618
21.3.6	打开和停止 DM9000 网卡	598	22.6	URB (USB 请求块)	620
21.3.7	发送数据	600	22.6.1	URB 结构体	621
21.3.8	接收数据	602	22.6.2	URB 的处理流程	622
21.3.9	设置广播地址	605	22.6.3	简单的批量与 控制 URB	625
21.4	小结	606	22.7	USB 驱动程序的结构	626
<b>第 22 章</b>	<b>USB 驱动</b>	<b>607</b>	22.8	鼠标驱动分析	633
22.1	USB 设备简介	607	22.9	小结	637
22.2	USB 驱动与 USB 核心之间的 交互	608			

# 第一篇

## Android 驱动开发前的准备

Android 系统移植与驱动开发概述

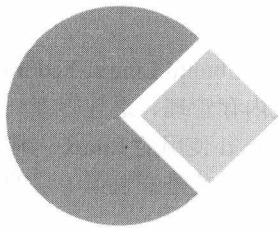
搭建 Android 开发环境

Git 使用入门

源代码的下载和编译

搭建 S3C6410 开发板的测试环境





# 第1章 Android 系统移植与 驱动开发概述

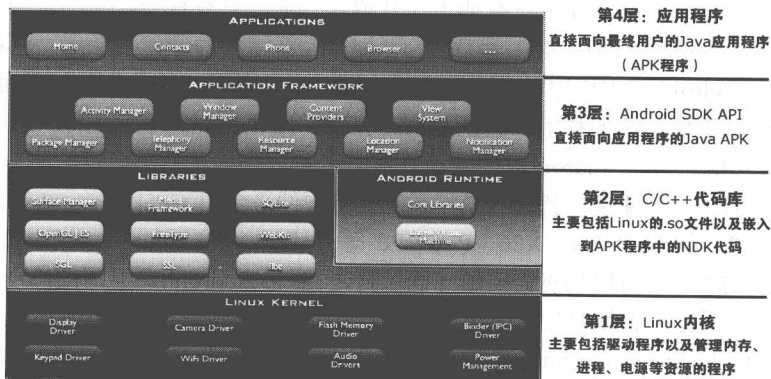
毋庸置疑，Android 已经成为当前智能手机操作系统的老大，市场占有率已遥遥领先于 iOS (iPhone 和 iPad)。Android 在几年时间发展如此神速，在很大程度上取决于任何人都可以利用 Android 的源代码定制完全属于自己的嵌入式系统，而不需要向 Google 交一分钱。

由于 Android 原生的代码支持的设备并不多，因此，要想在自己的设备（包括手机、MP4、智能电视、平板电脑、车载系统等）上完美运行 Android，就需要另外开发一些程序，使得 Android 可以识别相应设备中的硬件（显示器、蓝牙、音频、Wi-Fi 等）。这个为特定设备定制 Android 的过程被称为“移植”。那么，在移植的过程中开发得最多的就是支持各种硬件设备的 Linux 驱动程序（Android 是基于 Linux 内核的）。因此，讲移植就必须讲驱动开发。

本章作为学习 Linux 驱动的第一道门，将对 Android 以及 Linux 驱动做一个总体的介绍，以便使读者对开发 Linux 驱动有一个感性的认识，并为更好地学习 Linux 驱动的方法和技巧打下基础。

## 1.1 Android 系统架构

Android 是一个非常优秀的嵌入式操作系统。经过几年的发展和演进，Android 已经形成了非常完善的系统架构，如图 1-1 所示。



▲图 1-1 Android 系统架构

从图 1-1 可以看出，Android 的系统架构分为 4 层。这 4 层所包含的内容如下。

### 第 1 层: Linux 内核

由于 Android 是基于 Linux 内核的, 因此, Android 和其他 Linux 系统 (如 Ubuntu Linux、Fedora Linux 等) 的核心部分差异非常小。这一层主要包括 Linux 的驱动程序以及内存管理、进程管理、电源管理等程序。Android 使用 Linux 2.6 作为其内核。不过不同版本的 Android 使用的 Linux 内核版本有细微的差异, 所以不同 Android 版本的驱动可能并不通用。本书主要讲的就是开发第 1 层的驱动程序, 以及如何在不同 Linux 版本、硬件平台移植驱动程序。

### 第 2 层: C/C++ 代码库

这一层主要包括使用 C/C++ 编写的代码库 (Linux 下的 .so 文件), 也包括 Dalvik 虚拟机的运行时 (Runtime)。

### 第 3 层: Android SDK API

由于 Android SDK API 是用 Java 语言编写的, 因此, 这一层也可称为 Java API 层。实际上, 这一层就是用 Java 编写的各种 Library。只不过这些 Library 是基于 Dalvik 虚拟机格式的。笔者所著《Android 开发权威指南》主要就是介绍了这一层的 Android SDK API 的使用方法 & 技巧。

### 第 4 层: 应用程序

这一层是所有的 Android 用户 (包括程序员和非程序员) 都要接触到的。因为这一层相当于 Android 的 UI。所有的 Android 应用程序 (包括拍照、电话、短信、Android 的桌面、浏览器以及各种游戏) 都属于这一层。而这一层主要依靠第 3 层中的 Android SDK API 来完成各种功能。

## 12 Android 系统移植的主要工作

Android 移植可分为两部分: 应用移植和系统移植。应用移植是指将如图 1-1 所示第 4 层的应用程序移植到某一个特定硬件平台上。由于不同硬件平台之间的差异, Android SDK API 也有可能存在差异 (有的厂商会修改部分 Android SDK API 以适应自身硬件的需要), 或者将应用程序从低版本 Android 移植到高版本的 Android 上。为了保证应用程序可以在新的硬件平台正常运行, 需要对源代码进行一些修改。当然, 如果没有或无法获取源代码, 只有重新在新的平台上实现了。一般 Android 应用移植并不涉及驱动和 HAL 程序库 (Android 新增加的硬件抽象层, 将在后面的章节介绍) 的移植, 而且 Android 应用程序移植也不在本书讨论的范围内, 因此, 本书后面出现的 Android 移植都是指 Android 操作系统的移植 (包括 Linux 驱动、HAL 程序库的移植)。

Android 系统移植是指让 Android 操作系统在某一个特定硬件平台上运行。使一个操作系统在特定硬件平台上运行的一个首要条件就是该操作系统支持硬件平台的 CPU 架构。Linux 内核本身已经支持很多常用的 CPU 架构 (ARM、X86、PowerPC 等), 因此, 将 Android 在不同的 CPU 架构之间移植并不需要做过多的改动 (有时仍然需要做一些调整)。要想 Android 在不同硬件平台上正常运行, 只支持 CPU 架构还不行, 必须要让 Android 可以识别平台上的各种硬件 (如声卡、显示器、蓝牙设备等)。这些工作主要也是由 Linux 内核完成的。其中的主角就是 Linux 驱动。因此, 系统移植除了移植 CPU 架构外, 最重要的就是移植 Linux 驱动。例如, 为硬件平台增加了一个新型的 Wi-Fi 模块, 就需要为这个 Wi-Fi 模块编写新的驱动程序, 或修改原来的驱动程序, 已使得 Linux 内核可以与 Wi-Fi 模块正常交互。

除了 Linux 驱动需要移植外，在 Android 系统中还增加了一个硬件抽象层（HAL，Hardware Abstraction Layer），为了方便，本书后面的部分都使用 HAL 表示硬件抽象层。

HAL 位于如图 1-1 所示的第 2 层，也是普通的 Linux 程序库（.so 文件），只是 Android SDK 通过 HAL 直接访问 Linux 驱动。也就是说，Android 并不像其他的 Linux 系统一样由应用程序直接访问驱动，而是中间通过 HAL 隔了一层。Google 这样设计的原因很多，例如，由于 Linux 内核基于 GPL 开源协议，而很多驱动厂商不想开放源代码，所以增加了 HAL 层后，可以将 Linux 驱动的业务逻辑放在 HAL 层，这样处理 Linux 驱动开源技术，也只是一个空架子而已。关于 Android 支持 HAL 的原因将在后面的章节详细介绍。

如果为 Android 增加了新的驱动或修改原来的驱动代码，HAL 中的代码就要做相应的调整。因此，Android 移植的主要工作如下：

- 移植 Linux 驱动；
- 移植 HAL。

移植的工作也可能不多，当然，也可能非常多。如果要移植的 Android 系统提供了驱动源代码，那就好办多了，直接根据移植的目标平台修改驱动代码就可以了。不过很多时候由于某些原因，无法获得驱动的源代码，或者要实现的驱动程序所对应的硬件是自己特有的，这就需要从头开始编写驱动程序以及相关的配置文件。对于 HAL 的移植也和 Linux 驱动差不多。总之，Android 移植的基本原则是尽可能找到驱动和 HAL 的源代码，在源代码的基础上改要比从头开始编写容易得多，实在无法获取源代码，就只有从头开始做起了。不过在了解了编写 Linux 驱动和 Android HAL 程序库的步骤和规则以后，看着也没那么复杂。因为驱动和 HAL 的代码远没有 Android SDK 和 Android 应用程序的代码量大。

### 注意

Android 移植在很大程度上是 Linux 内核的移植。Linux 内核移植主要就是移植驱动程序。不同 Linux 版本的驱动程序不能通用，需要重新修改源代码，并在新的 Linux 内核下重新编译才可以运行在新的 Linux 内核版本下。Android 版本和 Linux 版本不同。无论哪个 Android 版本，其 Linux 内核版本都是 Linux 2.6 或 Linux 3.0（将来有可能使用更高版本的 Linux 内核），只是小版本号不同。由于 Android 开放源代码，所以就算同一个 Android 版本，Linux 的内核也可能不同（有很多自制的 ROM 会更换不同的 Linux 内核，以至于和官方同一 Android 版本的 Linux 内核不同），例如，笔者曾见过有的 Android 2.3 使用了 Linux 2.6.29，而官方的 Android 2.3 使用了 Linux 2.6.35。在移植 Linux 驱动时，主要应考虑 Linux 内核的版本，就算 Android 版本不同，只要 Linux 内核版本相同，Linux 驱动就可以互相替换（有时也需要考虑 HAL 是否和 Linux 驱动兼容）。

## 13 查看 Linux 内核版本

目前 Linux 内核主要维护 3 个版本：Linux 2.4、Linux 2.6 和 Linux 3.x，大多数 Linux 系统都使用了这 3 个版本的内核，其中 Linux 2.6 是目前使用最广泛的 Linux 内核版本，Android 就使用了该





▲图 1-2 查看 Android 的 Linux 内核版本

内核版本。而 Linux 2.4 由于其内部设计缺陷（主要是进程调度上的缺陷），除了一些遗留 Linux 系统，已很少有新的 Linux 系统使用 Linux 2.4 了。Linux 3.x 是最新推出的 Linux 内核版本。最新的 Android 4.x 采用了这个新的 Linux 3.0.8 内核版本，还有很多新推出的 Linux 系统（如 Ubuntu Linux 11.10）都使用了 Linux 3.0。读者可在 Android 系统中的“设备”>“关于手机”中查看当前 Android 系统所采用的 Linux 内核版本，如图 1-2 所示。

如果想查其他 Linux 系统的内核版本，可使用下面两种方法。

### 方法 1

在 Linux 终端执行下面的命令。

```
uname -a
```

如果当前系统是 Ubuntu Linux 11.10, 会在 Linux 终端输出如图 1-3 所示的信息。白框内是 Linux 内核的版本。

```
root@mainubuntu:~# uname -a
Linux mainubuntu [3.0.0-14-generic] #23-Ubuntu SMP Mon Nov 21 20:28:43 UTC 2011 x86_64 x86_64 x86_64 GNU/Linux
root@mainubuntu:~#
```

▲图 1-3 使用 uname 命令查看 Linux 内核版本

### 方法 2

在 Linux 终端执行下面的命令。

```
cat /proc/version
```

在 Linux 终端输出如图 1-4 所示的信息。白框内是 Linux 内核的版本。

```
root@mainubuntu:~# cat /proc/version
Linux version 3.0.0-14-generic [(build@allspice) (gcc version 4.6.1 (Ubuntu/Linux ro 4.6.1-9ubuntu3))] #23-Ubuntu SMP Mon Nov 21 20:28:43 UTC 2011
root@mainubuntu:~#
```

▲图 1-4 查看 /proc/version 文件获取 Linux 内核版本

## 注意

/proc 不是普通的文件系统，而是系统内核的映像，也就是说，该目录中的文件是存放在系统内存之中的，它以文件系统的方式为访问系统内核数据的操作提供接口。而 uname 命令就是从 /proc/version 文件中获取信息的，当然直接查看 /proc/version 文件的内容（方法 2）也可以获取同样的信息。uname 命令加上参数“-a”可以获取更多的信息，否则只显示当前的系统名，也就是只会输出“Linux”。

## 14

## Linux 内核版本号的定义规则

Linux 内核版本号由下面几部分组成。

- 主版本号；