

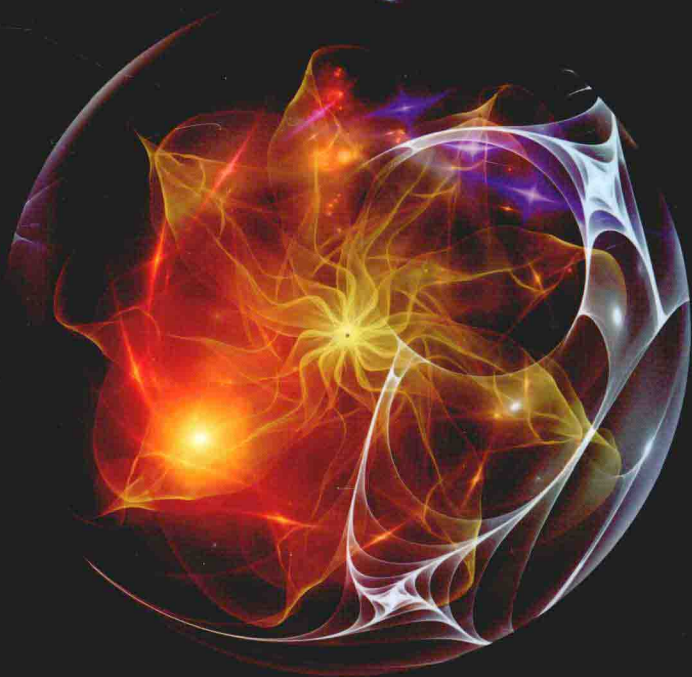


基于Android 4.3以上版本

- 全面细致地剖析了进程/线程模型、内存管理、Binder机制、GUI显示系统、多媒体管理、输入系统等核心模块在Android操作系统中的设计思想
- 通过大量图片与实例引导读者学习，以求尽量在源码分析外，为读者提供更易于理解的思维路径
- 由浅入深，由总体框架再到细节实现，让读者在学习过程中潜移默化地彻底理解Android内核的实现原理

# 深入理解 Android内核设计思想

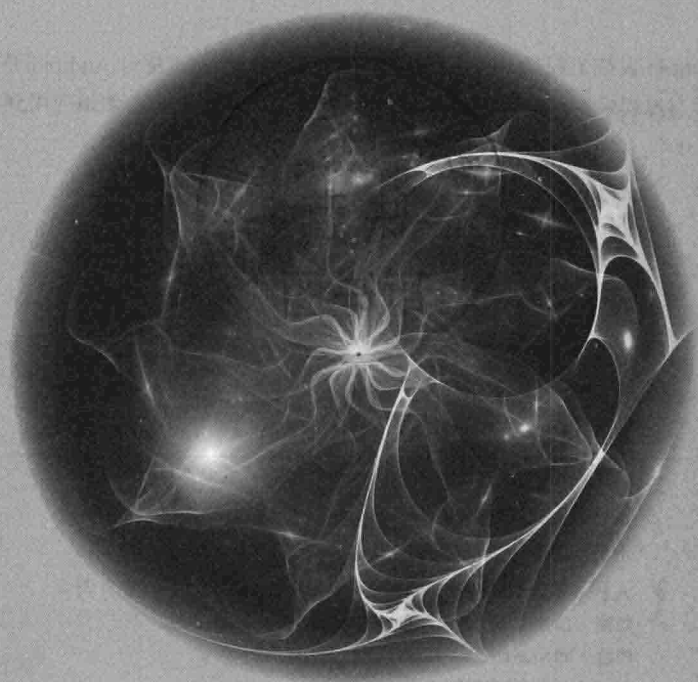
林学森 编著



人民邮电出版社  
POSTS & TELECOM PRESS

# 深入理解 Android内核设计思想

林学森 编著



人民邮电出版社  
北京

## 图书在版编目 (CIP) 数据

深入理解Android内核设计思想 / 林学森编著. —  
北京: 人民邮电出版社, 2014. 5  
ISBN 978-7-115-34841-8

I. ①深… II. ①林… 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(2014)第046163号

## 内 容 提 要

本书适用于 Android 4.3 以上的版本。全书从操作系统的基础知识入手, 全面剖析进程/线程、内存管理、Binder 机制、GUI 显示系统、多媒体管理、输入系统等核心技术在 Android 中的实现原理。书中讲述的知识点大部分来源于工程项目研发, 因而具有较强的实用性, 希望可以让读者“知其然, 更知其所以然”。全书分为编译篇、系统原理篇、应用原理篇、系统工具篇共 4 篇 22 章, 基本涵盖了参与 Android 开发所需具备的知识, 并通过大量图片与实例来引导读者学习, 以求尽量在源代码分析外为读者提供更易于理解的思维方式。

本书既适合 Android 系统工程师, 也适合于应用开发工程师来阅读提升 Android 开发能力。读者可以在本书潜移默化的学习过程中更深刻地理解 Android 系统, 并将所学知识自然地应用到实际开发难题的解决中。

---

◆ 编 著 林学森  
责任编辑 张 涛  
责任印制 彭志环 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号  
邮编 100164 电子邮件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
北京铭成印刷有限公司印刷

◆ 开本: 787×1092 1/16  
印张: 43.75  
字数: 1217千字  
印数: 1-3500册

2014年5月第1版  
2014年5月北京第1次印刷



---

定价: 108.00 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316  
反盗版热线: (010)81055315

# 前言

## 写本书的原因

4次大幅改版，N次修订，前后历时近3年，本书终于要与读者见面了。

在这3年的时间里，Android系统不断更新换代，书本内容也尽可能紧随其步伐——我们总是在第一时间下载到工程源码，然后系统性地比对和研究每次改版后的差异。可以说本书伴随着Android的高速发展，完整地见证了它给大家带来的一次又一次惊喜。

在这么长的写作跨度中，有一个问题始终萦绕在我的脑海中，即“为什么写这本书”？

市面上讲解操作系统的著作很多，主要风格有两种。

- 理论型

高校中采用的操作系统教材多数属于这种类型。它们主要阐述通用的计算机理论与原理，一般不会针对某个具体的操作系统做详细剖析。这类书籍是我们进入计算机科学的“敲门砖”。只有基础打得扎实，研究市面上任何一款操作系统才能做到“有的放矢”。

- 实用型

这类书籍以讲解某个具体的操作系统为主，如市面上就有非常多的关于Windows和Linux系统的。前者因为不开源，谁也不可能深入代码级别进行讲解；而后者则恰恰相反，任何人都能轻松获取到完整的内核源码。在Linux之父经典名言“Read the f\*\*\*king Source Code”的鼓励下，无数有志之士投入到“代码汪洋”的分析中，从中细细感受大师们的设计艺术。

那么本书属于什么类型呢？个人认为更贴切地说，就是上面两种的结合。

本书的一个主要宗旨是希望读者可以由浅入深地逐步理解Android系统的方方面面。因而在每章节内容的编排上，采用由整体到局部的线索铺展开来——先让读者有一个直观感性的认识，明白“是什么”、“有什么用”，然后才剖析“如何做到的”。这样的一个好处是读者在学习过程中不容易产生困惑；否则如果直接切入原理，长篇大论地分析代码，仅一大堆函数调用就可能让人失去学习的方向。这样的结果往往是，读者花了非常多的时间来理清函数关系，但始终不明白代码编写者的意图，甚至连这些函数想实现什么功能都无法完全理解。

本书希望可以从更高的层次，即抽象的、反映设计者思想的角度去理解系统。而在思考的过程中，大部分情况下我们都将从读者容易理解的基础知识开始讲起。就好比画一张素描一样——先给出一张白纸，勾勒出整体框架，然后针对重点部位细细加工，最后才能还原出完整的画面。另外，本书在对系统原理本身进行讲解的同时，也最大程度地结合工程项目中可能遇到的难点，理论联系实际地进行解析。希望这样的方式既能让读者真正学习到Android系统的设计思想，也能学有所用，增加一些实际的项目开发经验和技巧。

## 本书的主要内容

细心的读者会发现本书章节中包含了“Android和OpenGL ES”、“信息安全基础概述”等看似与本书无关的内容——有些人可能会产生疑问，是否有此必要？

根据我们多年的Android项目开发和培训经验，答案就是“非常有必要”。举个例子，Android

的显示系统是围绕 OpenGL ES 来展开的，后者是它的“根基”。但另外，并非所有开发人员都深谙 OpenGL ES。这样导致的结果就是他们在学习显示系统的过程中，有一种“四处碰壁”的感觉——实践证明，正是这些因素直接打击到了大家学习 Android 系统的信心。

因此，我们在讲解系统实现原理之前，会最大程度地为读者提炼出所需的背景知识。有了这样的铺垫，相信对大家学习 Android 内核大有裨益。

本书在内容选择上依据的是“研发人员（包括系统开发和应用程序开发）参与实际 Android 项目所需具备的知识”，因而具有较强的实用性。全书共分为 4 篇，涵盖了编译、系统原理、应用原理和系统工具等多个方面。

其中第一篇不仅详细介绍了 Android 源码的下载及编译过程，为读者呈现了“Hello World”式的入门向导——更为重要的是，结合编译系统的架构和内部原理，为各厂家定制自己的 Android 产品提供了参考范例。

Android 本质上只是市面上众多主流的操作系统之一。所以在系统原理的讲解过程中，我们将首先引导读者从计算机体系结构、经典的操作系统理论（比如进程/线程管理、进程间通信等）的角度来思考问题——包括 Android 在内的任何操作系统内核在实现过程中都“逃”不出这些经典的理论范畴。本书虽然是剖析 Android 系统的，但更希望读者可以从中学到“渔”，而不仅仅是“鱼”。

从动态运行的角度来理解，Android 内核是由众多系统服务组成的，如 ActivityManagerService、GUI 系统中的 SurfaceFlinger、音频系统中的 AudioFlinger、输入系统 InputManagerService 等。而各服务之间通信的基础就是 Binder 机制。本书在阐述它们错综复杂的关系中，遵循由“整体到局部”、“由点及面”的科学方法，将知识点深入浅出地铺展开来，希望为读者全面理解 Android 内核提供“思维捷径”。

与其他讲解 Android 应用程序的书籍不同，本书在分析 APK 应用程序时的立足点是它的内部实现原理。如 Intent 匹配规则、应用程序的资源适配过程、字符编码的处理、Widget 机制、应用程序的编译打包等都是应用开发人员在工作中经常会遇到的难题。通过系统性地解析隐藏在这些实现背后的原理，有助于他们彻底摆脱困惑，加深对应用开发的理解。

不论是系统工程师还是应用开发人员，Android 调试工具都至关重要。但我们在实际工作中发现，不少研发人员对这些工具“只知其一，不知其二”。因而系统工具篇中将针对常用调试工具进行全面解析，希望由此可以让大家学习到如何“举一反三”，真正把它们的作用发挥得“淋漓尽致”。

## 本书的主要特点

(1) 通过大量情景图片与实例引导读者学习，以求尽量在源码分析外为读者提供更易于理解的思维路径。

(2) 作者在展开一个话题时，通常会由浅入深、由总体框架再到细节实现。这样可以保证读者能跟得上分析的节奏，并且“有根有据可循”，尽可能防止部分读者阅读技术书籍时“看了后面忘了前面”的现象。

(3) 目前市面上不少 Android 书籍仍停留在 Android 2.3 或者更早期的版本。虽然原理类似，但对于开发人员来说，他们需要与项目研发相契合的技术书籍。本书希望尽可能紧随 Android 的更新步伐，为读者了解最新的 Android 技术提供帮助。

(4) 本书的出发点仍是操作系统的经典原理，并以此为根基扩展分析 Android 中的具体实现机制——贯穿其中的是经久不衰的理论知识。

(5) 本书所阐述的知识点大部分来源于工程项目研发的经验总结，因而具有较强的实用性，希望可以让读者“知其然，更知其所以然”。做到真正贴近读者，贴近开发需求。

## 致谢

感谢王益民董事长、钟宝英女士长期以来的关心、信任和支持——你们在很多方面都是我们学习的楷模。衷心祝愿王总的企业蒸蒸日上、更创辉煌；衷心祝愿钟小姐事事顺心如意、永葆青春。

感谢人民邮电出版社的编辑，你们的专业态度和处理问题的人性化，是所有作者的“福音”。

感谢我的家人、长辈和朋友林进跃、林美玉、林惠忠、刘冰、林月明、温艳，感谢你们长期以来对我工作和生活上无微不至的关心和支持。

感谢所有读者的支持，是你们赋予了我写作的动力。另外，因为个人能力和水平有限，书中可能还有不足之处，希望读者不吝指教，一起探讨学习，联系方式是：[xuesenlin@alumni.cuhk.net](mailto:xuesenlin@alumni.cuhk.net)。编辑联系邮箱为：[zhangtao@ptpress.com.cn](mailto:zhangtao@ptpress.com.cn)。



# 目 录

第 1 篇 Android 编译篇		3.2.5 dist_files ..... 41
第 1 章 Android 系统简介 ..... 2		3.2.6 Android.mk 的编写规则 ..... 42
1.1 Android 系统发展历程 ..... 2		第 2 篇 Android 原理篇
1.2 Android 系统特点 ..... 4		第 4 章 操作系统基础 ..... 48
1.3 Android 系统框架 ..... 8		4.1 计算机体系结构
第 2 章 Android 源码下载及编译 ..... 10		(Computer Architecture) ..... 48
2.1 Android 源码下载指南 ..... 10		4.1.1 冯·诺依曼结构 ..... 48
2.1.1 基于 Repo 和 Git 的		4.1.2 哈佛结构 ..... 48
版本管理 ..... 10		4.2 什么是操作系统 ..... 49
2.1.2 Android 源码下载流程 ..... 11		4.3 进程间通信的经典实现 ..... 51
2.2 原生态系统编译指南 ..... 12		4.3.1 共享内存
2.2.1 建立编译环境 ..... 13		(Shared Memory) ..... 52
2.2.2 编译流程 ..... 15		4.3.2 管道 (Pipe) ..... 54
2.3 定制产品的编译与烧录 ..... 17		4.3.3 Unix Domain Socket ..... 55
2.3.1 定制新产品 ..... 17		4.3.4 RPC (Remote
2.3.2 Linux 内核编译 ..... 21		Procedure Calls) ..... 58
2.3.3 烧录 ..... 22		4.4 同步机制的经典实现 ..... 58
2.4 Android 系统映像文件 ..... 23		4.4.1 信号量 (Semaphore) ..... 58
2.4.1 boot.img ..... 23		4.4.2 Mutex ..... 59
2.4.2 ramdisk.img ..... 25		4.4.3 管程 (Monitor) ..... 59
2.4.3 system.img ..... 26		4.4.4 同步范例 ..... 60
2.5 OTA 系统升级 ..... 26		4.5 Android 中的同步机制 ..... 61
2.5.1 生成升级包 ..... 27		4.5.1 进程间同步——Mutex ..... 61
2.5.2 获取升级包 ..... 28		4.5.2 条件判断——Condition ..... 62
2.5.3 OTA 升级-RecoveryMode ..... 29		4.5.3 “栅栏、障碍”
2.6 Android 反编译 ..... 31		——Barrier ..... 64
第 3 章 Android 编译系统 ..... 34		4.5.4 加解锁的自动化操作
3.1 Makefile 入门 ..... 34		——Autolock ..... 66
3.2 Android 编译系统 ..... 35		4.6 操作系统内存管理基础 ..... 66
3.2.1 Makefile 依赖树的概念 ..... 36		4.6.1 虚拟内存
3.2.2 树根节点 droid ..... 36		(Virtual Memory) ..... 66
3.2.3 main.mk 解析 ..... 38		4.6.2 内存保护
3.2.4 droidcore 节点 ..... 39		(Memory Protection) ..... 69

4.7	Android 中的 Low Memory Killer	71	6.5	Binder 客户端—— Binder Client	185
4.8	Android 匿名共享内存 (Anonymous Shared Memory)	74	6.6	Android 接口描述语言 ——AIDL	190
4.8.1	Ashmem 设备	74	6.7	匿名 Binder Server	202
4.8.2	Ashmem 应用实例	78	<b>第 7 章</b>	<b>Android 启动过程简析</b>	205
4.9	JNI	83	7.1	第一个系统进程 (init)	205
4.9.1	Java 函数的本地实现	83	7.1.1	init.rc 语法	205
4.9.2	本地代码访问 JVM	86	7.1.2	init.rc 实例分析	208
4.10	学习 Android 系统的两条线索	88	7.2	系统关键服务的启动简析	209
<b>第 5 章</b>	<b>Android 进程/线程管理</b>	89	7.2.1	Android 的“DNS 服务器” ——ServiceManager	209
5.1	Android 进程和线程	89	7.2.2	“孕育”新的线程和进程 ——Zygote	209
5.2	Handler, MessageQueue, Runnable 与 Looper	95	7.2.3	Android 的“系统服务” ——SystemServer	211
5.3	UI 主线程——ActivityThread	102	<b>第 8 章</b>	<b>管理 Activity 和组件运行状态 的系统进程——Activity ManagerService (AMS)</b>	213
5.4	Thread 类	105	8.1	AMS 功能概述	213
5.4.1	Thread 类的内部原理	105	8.2	管理当前系统中 Activity 状态 ——Activity Stack	215
5.4.2	Thread 休眠和唤醒	106	8.3	startActivity 流程	217
5.4.3	Thread 实例	110	8.4	完成同一任务的“集合” ——Activity Task	224
5.5	Android 应用程序的 典型启动流程	112	8.4.1	“后进先出”——Last In, First Out	225
<b>第 6 章</b>	<b>进程间通信——Binder</b>	114	8.4.2	管理 Activity Task	226
6.1	智能指针	117	<b>第 9 章</b>	<b>GUI 系统之 SurfaceFlinger</b>	229
6.1.1	智能指针的设计理念	117	9.1	OpenGL ES 与 EGL	229
6.1.2	强指针 sp	120	9.2	Android 的硬件接口——HAL	231
6.1.3	弱指针 wp	121	9.3	Android 终端显示设备的“化身” ——Gralloc 与 Framebuffer	233
6.2	进程间的数据传递载体 ——Parcel	128	9.4	Android 中的本地窗口	237
6.3	Binder 驱动与协议	135	9.4.1	Framebuffer NativeWindow	239
6.3.1	打开 Binder 驱动 ——binder_open	136	9.4.2	应用程序端的本地窗口 ——Surface	245
6.3.2	binder_mmap	137	9.5	BufferQueue 详解	249
6.3.3	binder_ioctl	140	9.5.1	BufferQueue 的 内部原理	249
6.4	“DNS”服务器——ServiceManager (Binder Server)	141			
6.4.1	ServiceManager 的启动	141			
6.4.2	ServiceManager 的构建	142			
6.4.3	获取 ServiceManager 服务 ——设计思考	147			
6.4.4	ServiceManagerProxy	151			
6.4.5	IBinder 和 BpBinder	153			
6.4.6	ProcessState 和 IPCThreadState	155			



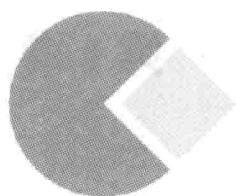
9.5.2	BufferQueue 中的缓冲区分配	252	10.2.3	窗口属性 (LayoutParams)	322
9.5.3	应用程序的典型绘图流程	258	10.3	窗口的添加过程	324
9.5.4	应用程序与 BufferQueue 的关系	263	10.3.1	系统窗口的添加过程	324
9.6	SurfaceFlinger	267	10.3.2	Activity 窗口的添加过程	333
9.6.1	“黄油计划”——Project Butter	267	10.3.3	窗口添加实例	337
9.6.2	SurfaceFlinger 的启动	271	10.4	Surface 管理	340
9.6.3	接口的服务端——Client	275	10.4.1	Surface 申请流程 (relayout)	341
9.7	VSync 的产生和处理	279	10.4.2	Surface 的跨进程传递	344
9.7.1	VSync 信号的产生和分发	279	10.4.3	Surface 的业务操作	347
9.7.2	VSync 信号的处理	285	10.5	performLayoutAndPlace SurfacesLockedInner	347
9.7.3	handleMessage Transaction	287	10.6	窗口大小的计算过程	349
9.7.4	“界面已经过时/无效, 需要重新绘制”——handle MessageInvalidate	291	10.7	启动窗口的添加与销毁	358
9.7.5	合成前的准备工作——preComposition	293	10.7.1	启动窗口的添加	358
9.7.6	可见区域——rebuild LayerStacks	295	10.7.2	启动窗口的销毁	362
9.7.7	为“Composition”搭建环境——setUpHW Composer	299	10.8	窗口动画	363
9.7.8	doDebugFlashRegions	301	10.8.1	窗口动画类型	364
9.7.9	doComposition	302	10.8.2	动画流程跟踪——Window StateAnimator	365
10.8.3	AppWindowAnimator	368	10.8.4	动画的执行过程	370
<b>第 10 章 GUI 系统之“窗口管理员”</b>					
<b>——WMS</b> 309					
10.1	“窗口管理员”——WMS 综述	310	<b>第 11 章 让你的界面炫彩起来的 GUI 系统之 View 体系</b> 377		
10.1.1	WMS 的启动	312	11.1	应用程序中的 View 框架	377
10.1.2	WMS 的基础功能	312	11.2	Activity 中 View Tree 的创建过程	380
10.1.3	WMS 的工作方式	313	11.3	在 WMS 中注册窗口	386
10.1.4	WMS, AMS 与 Activity 间的联系	314	11.4	ViewRoot 的基本工作方式	388
10.2	窗口属性	316	11.5	View Tree 的遍历时机	389
10.2.1	窗口类型与层级	316	11.6	View Tree 的遍历流程	393
10.2.2	窗口策略 (Window Policy)	320	11.7	View 和 ViewGroup 属性	402
			11.7.1	View 的基本属性	402
			11.7.2	ViewGroup 的属性	408
			11.7.3	View, ViewGroup 和 ViewParent	408
			11.7.4	Callback 接口	408
			11.8	“作画”工具集——Canvas	411
			11.8.1	“绘制 UI”——Skia	412

11.8.2	数据中介——Surface.lock Canvas	413	13.3.1	AudioFlinger 服务的 启动和运行	460	
11.8.3	解锁并提交结果—— unlockCanvasAndPost	417	13.3.2	AudioFlinger 对音频设备 的管理	461	
11.9	draw 和 onDraw	417	13.3.3	PlaybackThread 的 循环主体	468	
11.10	View 中的消息传递	423	13.3.4	AudioMixer	473	
11.10.1	View 中 TouchEvent 的 投递流程	423	13.4	策略的制定者 ——AudioPolicyService	475	
11.10.2	ViewGroup 中 TouchEvent 的投递流程	427	13.4.1	AudioPolicyService 概述	475	
11.11	View 动画	430	13.4.2	AudioPolicyService 的 启动过程	477	
<b>第 12 章 “问渠哪得清如许，为有源头活水来” ——InputManager Service 与输入事件</b>			436	13.4.3	AudioPolicyService 与 音频设备	480
12.1	事件的分类	436	13.5	音频流的回放——AudioTrack	482	
12.2	事件的投递流程	439	13.5.1	AudioTrack 应用实例	482	
12.2.1	InputManagerService	440	13.5.2	AudioPolicyService 的 路由实现	489	
12.2.2	InputReaderThread	441	13.6	音频数据流	494	
12.2.3	InputDispatcherThread	441	13.6.1	AudioTrack 中的 音频流	495	
12.2.4	ViewRootImpl 对事件 的派发	445	13.6.2	AudioTrack 和 Audio Flinger 间的数据交互	498	
<b>第 13 章 应用不再同质化 ——音频系统</b>			447	13.6.3	AudioMixer 中的 音频流	504
13.1	音频基础	448	13.7	音量控制	506	
13.1.1	声波	448	13.8	音频系统的上层建筑	510	
13.1.2	音频的录制、存储 与回放	448	13.8.1	从功能入手	510	
13.1.3	音频采样	449	13.8.2	MediaPlayer	511	
13.1.4	Nyquist-Shannon 采样定律	451	13.8.3	MediaRecorder	514	
13.1.5	声道和立体声	451	13.8.4	一个典型的多媒体 录制程序	517	
13.1.6	声音定级——Weber- Fechner law	452	13.8.5	MediaRecorder 源码解析	518	
13.1.7	音频文件格式	453	13.8.6	MediaPlayerService 简析	520	
13.2	音频框架	454	13.9	Android 支持的媒体格式	523	
13.2.1	Linux 中的音频框架	454	13.9.1	音频格式	523	
13.2.2	TinyAlsa	456	13.9.2	视频格式	523	
13.2.3	Android 系统中的 音频框架	457	13.9.3	图片格式	524	
13.3	音频系统的核心 ——AudioFlinger	459	13.9.4	网络流媒体	524	
			13.10	ID3 信息简述	525	

13.11	Android 多媒体文件管理	528	17.1.1	计算机 3D 图形	584
13.11.1	MediaStore	529	17.1.2	图形管线	585
13.11.2	多媒体文件信息的存储 “仓库”——Media Provider	530	17.2	Android 中的 OpenGL ES 简介	587
13.11.3	多媒体文件管理中的 “生产者”——Media Scanner	533	17.3	图形渲染 API—EGL	588
<b>第 3 篇 应用原理篇</b>			17.3.1	EGL 与 OpenGL ES	588
<b>第 14 章</b>	<b>Intent 的匹配规则</b>	538	17.3.2	egl.cfg	588
14.1	Intent 属性	538	17.3.3	EGL 接口解析	590
14.2	Intent 的匹配规则	540	17.3.4	EGL 实例	593
14.3	Intent 匹配源码简析	546	17.4	简化 OpenGL ES 开发 ——GLSurfaceView	593
<b>第 15 章</b>	<b>APK 应用程序的资源适配</b>	551	<b>第 18 章</b>	<b>“系统的 UI”——SystemUI</b>	601
15.1	资源类型	552	18.1	SystemUI 的组成元素	601
15.1.1	状态颜色资源	553	18.2	SystemUI 的实现	603
15.1.2	图形资源	554	18.3	Android 壁纸资源 ——WallpaperService	610
15.1.3	布局资源	555	18.3.1	WallPaperManager Service	611
15.1.4	菜单资源	556	18.3.2	ImageWallpaper	613
15.1.5	字符串资源	556	<b>第 19 章</b>	<b>Android 常用的工具“小插件” ——Widget 机制</b>	616
15.1.6	样式资源	557	19.1	“功能的提供者” ——AppWidgetProvider	616
15.1.7	其他资源	558	19.2	AppWidgetHost	618
15.1.8	属性资源	558	<b>第 20 章</b>	<b>Android 应用程序的 编译和打包</b>	624
15.2	提供可选资源	561	20.1	“另辟蹊径”采用第三方 工具——Ant	624
15.3	最佳资源的匹配流程	565	20.2	通过命令行编译和打包 APK	625
15.4	屏幕适配	567	20.3	APK 编译过程详解	626
15.4.1	屏幕适配的重要参数	567	20.4	信息安全基础概述	628
15.4.2	如何适配多屏幕	569	20.5	应用程序签名	633
<b>第 16 章</b>	<b>Android 字符编码格式</b>	572	20.6	应用程序签名源码简析	636
16.1	字符编码格式背景	572	<b>第 4 篇 Android 系统工具</b>		
16.2	ISO/IEC 8859	573	<b>第 21 章</b>	<b>软件版本管理</b>	642
16.3	ISO/IEC 10646	573	21.1	版本管理简述	642
16.4	Unicode	574	21.2	Git 的安装	643
16.5	String 类型	577	21.2.1	Linux 环境下安装 Git	643
16.5.1	构建 String	578			
16.5.2	String 对多种 编码的兼容	579			
<b>第 17 章</b>	<b>Android 和 OpenGL ES</b>	583			
17.1	3D 图形学基础	584			

21.2.2	Windows 环境下 安装 Git	644
21.3	Git 的使用	644
21.3.1	基础配置	644
21.3.2	新建仓库	646
21.3.3	文件状态	647
21.3.4	忽略某些文件	649
21.3.5	提交更新	649
21.3.6	其他命令	650
21.4	Git 原理简析	650
21.4.1	分布式版本 系统的特点	651
21.4.2	安全散列算法 ——SHA-1	652
21.4.3	四个重要对象	653
21.4.4	三个区域	657
21.4.5	分支的概念与实例	658

第 22 章	系统调试辅助工具	662
22.1	万能模拟器——Emulator	662
22.1.1	QEMU	662
22.1.2	Android 工程中的 QEMU	667
22.1.3	模拟器控制台 (Emulator Console)	670
22.1.4	实例: 为 Android 模拟器 添加串口功能	672
22.2	此 Android 非彼 Android	674
22.3	快速建立与模拟器或真机的 通信渠道——ADB	676
22.3.1	ADB 的使用方法	676
22.3.2	ADB 的组成元素	678
22.3.3	ADB 源代码解析	679
22.3.4	ADB Protocol	684



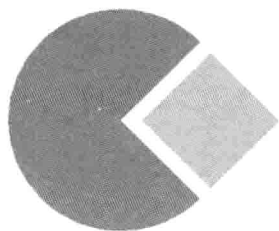
# 第 1 篇

## Android 编译篇

第 1 章 Android 系统简介

第 2 章 Android 源码下载及编译

第 3 章 Android 编译系统



## 第1章 Android 系统简介

美国当地时间 2013 年 5 月 15~17 日,“Google I/O 2013”大会在旧金山市的 Moscone Center 举行。

据会议公布的官方数据:

- 全球已经激活的 Android 设备达到 9 亿次;
- Google Play 中收录了超过 70 万的应用程序;
- 应用程序安装量达到 480 亿次;
- 132 个以上的国家销售 Android 设备;
- 超过 190 个国家可以下载到免费的 Android 应用程序。

从 2008 年 9 月 Google 发布 Android 1.0 版本开始,Android 已经走过了 5 个年头。在这短短的几年间,这个以机器人为 Logo 的操作系统不仅席卷了全球各地的手机市场,而且与 iOS, Windows Phone 形成三足鼎立之势,更渗透到传统与新兴电子产业的方方面面。身边越来越多的产品已开始采用 Android 系统,如 Android 电视、平板电脑、MP4 等与人们日常生活息息相关的电子设备。

那么,Android 势不可当的魅力从何而来呢?本章将试着以 Android 系统的发展历史为主线,先为读者提供最直观的背景知识,从而为以后的“透过现象看本质”打下一定的基础。

### 1.1 Android 系统发展历程

和“Google”这个词所蕴含着的“Geek”们对于顶尖技术的狂热崇拜一样,“Android”一词先天就充满天才们改变世界梦想的味道。虽然一件杰出的作品并不能只靠“名号”,但毋庸置疑的是,一个叫得响又耐人寻味的名称总会使人产生不自觉的亲近感。这或许就是每个 Android 版本都会有个代号的原因。下面来看看各个版本对应的 Android“外号”,如表 1-1 所示。

表 1-1 Android 各版本的代号

Code name	Version	API level
(no code name)	1.0	API level 1
(no code name)	1.1	API level 2
Cupcake (纸杯蛋糕)	1.5	API level 3, NDK 1
Donut (甜甜圈)	1.6	API level 4, NDK 2
Éclair (松饼)	2.0	API level 5
Éclair	2.0.1	API level 6
Éclair	2.1	API level 7, NDK 3
Froyo (冻酸奶)	2.2.x	API level 8, NDK 4
Gingerbread (姜饼)	2.3 - 2.3.2	API level 9, NDK 5



续表

Code name	Version	API level
Gingerbread	2.3.3 - 2.3.7	API level 10
Honeycomb (蜂巢)	3.0	API level 11
Honeycomb	3.1	API level 12, NDK 6
Honeycomb	3.2.x	API level 13
Ice-creamSandwich (冰激凌三明治)	4.0.1 - 4.0.2	API level 14, NDK 7
Ice-creamSandwich	4.0.3 - 4.0.4	API level 15
Jelly Bean (果冻豆)	4.1.x	API level 16
Jelly Bean	4.2.x	API level 17
Jelly Bean	4.3.x	API level 18
KitKat	4.4.x	API level 19

“Android”一词来源于法国作家 Auguste Villiers de l'Isle-Adam 的科幻小说《L'ève future》(未来夏娃),是机器人的意思。因此,最初每个系统版本的命名也都是以全球著名的机器人为参考的,如“AstroBoy”。后来由于版权问题,才改为以食物的方式取名。不过 Android 的 Logo 仍然是机器人的形象,如图 1-1 所示。



▲图 1-1 Android 官方 Logo

和很多著名的科技企业一样,Android 的创始人 Andy Rubin 也是个技术狂人。在创立 Android 公司前,他曾完成多项当时被称为“过于超前”的产品研发,并取得了一定的成绩。而创办 Android,最初的目的是提供一款开放式的移动平台系统。从 2003 年 10 月 Andy Rubin 开始启动这一系统的研究,Android 便正式走上历史的舞台。以下是关于这个系统的一些重要历史事件。

- 2003 年 10 月, Andy Rubin 在加利福尼亚州成立 Android 公司。
- 2005 年 4 月, Google 收购 Android。
- 2007 年 11 月, Google 成立 OHA (Open Handset Alliance) 联盟, 成员包括 Google, Broadcom, HTC, Intel, LG, Marvell, Motorola, NVIDIA, Qualcomm, Samsung 等通信行业和芯片制造领域的巨头。随后几年, 这个联盟又陆续有不少公司加入, 如著名的 Arm 公司、中国的华为等。
- 2007 年 11 月, Google 成立“Android Open Source Project”(AOSP)。这一项目的起步标志着 Android 系统首次公开面向全世界的开发者与使用者。

AOSP 的宗旨是:

Android Open Source Project is to create a successful real-world product that improves the mobile experience for end users.

因为是开源开放的组织,所以意味着每个人都可以参与进来,并为整个项目的发展添砖加瓦。如果读者有意愿成为其中的一员,可以参考该组织的相关说明(<http://source.android.com/source/index.html>)。

- 2007 年 11 月, Android Beta 版本发布。
- 2007 年 11 月, Android 第一个 SDK 版本发布。
- 2008 年 9 月, Android 1.0 版本正式发布。

至此,Android 版本的发布驶入正常轨道,保持着每年多次升级的速度,并加入越来越多的创新功能。

关于 Android 每个版本的特性及改版后一些重要变化的详细说明,请参阅官方网站(<http://source.android.com/source/overview.html>)。相信经过仔细比对各个版本的变化,读者会发现 Android 系统确实一直在秉承其“为终端用户提供更好的移动设备体验”的宗旨。

## 1.2 Android 系统特点

在这一节中，我们将从观察者的角度来客观评价 Android 系统某些突出的特点。这些分析中既包含了其值得肯定的诸多优点，也不吝指出其需要持续改善的地方。只有正确全面地了解一个系统所存在的优缺点，才可能在开发的过程中“知己知彼，百战不殆”，真正让系统为我们所用。

### 1. 开放与扩展性

相对于 iOS 和 Windows Phone 阵营，Android 操作系统最大的特点就是开放性；而且有别于个别开源项目的“藏藏掖掖”、“犹抱琵琶半遮面”，Android 几乎所有源码都可以免费下载到。无论是公司组织，还是个人开发者，Android 对于下载者基本没有限制，也没有下载权限的认证束缚。关于如何下载系统源码的完整描述，请参考下一章节。

当然，这并不代表开发者可以随意使用 Android 源码。事实上，Android 遵循的是 Apache 开源软件许可证。因此，所有跨越许可证规定范畴的行为都将是被禁止的。希望了解更多 Apache 协议条款详情的读者，可以自行查阅其官方说明 (<http://www.apache.org/>)。

不过在大部分情况下，Android 操作系统仍然被认为是“高度自由”的。这也是越来越多的厂商选择 Android 作为下一代产品基础平台最主要的原因之一。可以想象，在其他操作系统对其授权的设备动辄收取每台高达几十甚至数百美元专利费的情况下，采用 Android 开源系统理论上就意味着降低成本。

另外，由于整个操作系统是开源的，从而给诸多产品制造商、软件开发商提供了创新的土壤环境。各厂商可以根据自己的需求，来完成对原生态系统的修改。大多数情况下，这种修改只是基于上层 UI 交互的“二次包装”，而保留底层系统的大框架。这就好比 Google 为大家免费提供了已经盖好的办公大楼，虽然是毛坯房，但相较于“万丈高楼平地起”的艰辛，显然已经为我们节约了大量的项目时间；而且我们可以通过“装修”把主要精力倾注在用户看得到的地方，从而更最大限度地摆脱“产品同质化”的威胁。事实上，目前全球范围内已经有非常多这样的“装修范例”。大到跨国企业、运营商，小到一些初创的设计公司，都选择在 Android 系统上进行“界面”改造，再冠以新的操作系统名号。其中也不乏一些成功者，根据不同的地域环境、文化差异、使用习惯而定制出新的系统——这些具有“本地化”风格的“办公楼”往往比原生生态系统更贴近当地消费者，因此受到热烈追捧。

而这一切，都要归功于 Android 系统的开放性。

### 2. 合理的分层架构

要学习 Android 系统，就不得不提它的分层架构。早期版本的 Android 系统框架包括四层，即 Linux Kernel, Library and Runtime, Application Framework 及 Application。后来因为版权相关原因在 Kernel 层之上新增了一个 Hardware Abstraction Layer。我们会在后续小节对各层功能适当地展开讨论。

由此可见，Android 系统是一个“杂合体”，即便说其“包罗万象”也一点不为过。它包括了 prebuilt, bionic 等在内的不少开源项目。管理这些项目显然不是件容易的事，这也是 Android 系统提供 Repo 工具，而不是直接使用 Git 来进行版本管理的原因之一（详见后续章节的描述）。

在面对这么多独立项目的时候，合理的分层架构就显得异常重要——既要保证系统功能的完整性，也要确保各项目的相对独立性。Android 系统成功地做到了这一点，整个软件栈条理清晰，分工明确。一方面，它将底层复杂性与移植难度尽可能隐藏起来；另一方面，则提供尽可能方便

的上层 API 接口，为开发者设计实现各种应用程序打下了坚实的基础。

### 3. 易用强大的 SDK

SDK (Software Development Kit) 是操作系统与开发者之间的接口，也可以看成一个系统对外的窗口。对于广大的开发者而言，能否借助这个工具在尽可能短的周期内设计出符合需求而又稳定可靠的应用程序，是评判一个操作系统 SDK 好坏的重要标准之一。

Android 系统的大部分应用程序可以基于 Java 来开发。如果读者曾参与过大型的 C/C++ 研发项目 (特别是面向嵌入式系统的)，一定不会忘记加班加点解决内存泄露或者空指针异常的那些无眠夜。Java 语言对这些软件开发中最令人头疼的问题进行了强有力的改造，不但提供了垃圾回收机制，而且彻底隐藏了指针的使用。即便程序出现了崩溃，通常情况下也可以根据调用栈及各种 Log 来定位出问题的根源。这无疑为我们快速解决问题、保证程序稳定性提供了很好的平台基础。

Android 系统通过总结应用程序的开发规律，提供了 Activity, Service, Broadcast Receiver 及 Content Provider 四大组件；并且和 MFC 类似，设计了人性化的向导模式来帮助开发者便捷地生成工程原型。可以说，这些都为项目开发节约了不少宝贵时间。

另外，Android SDK 覆盖面相当广，且仍在持续扩充中。从线程管理、进程间通信等程序设计基础到各种界面组件的应用，只要是开发者能想到的，几乎都可以在 SDK 中找到现成的调用接口。而对一些界面特效的封装，使得开发者可以高效地设计出各种绚丽的 UI 效果，进而让 Android 系统加分良多。

### 4. 不断改进的交互界面

Android 版本的更迭是一件让无数人兴奋的事。除了那些令人眼前一亮的新功能外，不断改进的用户交互界面也是吸引用户的一个重要因素。我们可以明显地从新老版本的对比中找到 Google 追求绝佳用户体验的决心。

下面先来看看 Gingerbread (2.3 版本) 的 Launcher 与 Camera 界面，如图 1-2 所示。

然后来看看后续版本上的变化，如图 1-3 所示。



▲图 1-2 Launcher 和 Camera 界面



▲图 1-3 后续 Android 版本变化

可以发现，新的版本相较于以前，不但在 UI 界面的色彩搭配、布局上有了很大提高，用户交互也更趋于人性化。这种对于用户最直观的“艺术盛宴”展示，促使越来越多的人投身到 Android 阵营中。

### 5. 逐步完善的生态系统

IT 业界长期以来都有一个共识——开发一个操作系统 (OS) 并不是最难的，而基于这个新的操作系统建立完整的生态系统才是最大的难点。用一句老话来说，颇有点“打江山易，守江山难”