

## <<Android底层开发技术实战详解>>

### 图书基本信息

书名：<<Android底层开发技术实战详解>>

13位ISBN编号：9787121175930

10位ISBN编号：7121175932

出版时间：2012-8

出版时间：电子工业出版社

作者：王振丽

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

## <<Android底层开发技术实战详解>>

### 内容概要

本书从底层原理开始讲起，结合真实的案例向读者详细介绍了Android内核、移植和驱动开发的整个流程。

全书分为19章，依次讲解驱动移植的必要性，何为HAL层深入分析，Goldfish、MSM、MAP内核和驱动解析，显示系统、输入系统、振动器系统、音频系统、视频输出系统的驱动，OpenMax多媒体、多媒体插件框架，传感器、照相机、Wi-Fi、蓝牙、GPS和电话系统等。

在每一章中，重点介绍了与Android驱动开发相关的底层知识，并对Android源代码进行了剖析。

## 书籍目录

## 第1章 Android底层开发基础1

## 1.1 什么是驱动1

## 1.1.1 驱动程序的魅力1

## 1.1.2 电脑中的驱动2

## 1.1.3 手机中的驱动程序2

## 1.2 开源还是不开源的问题3

## 1.2.1 雾里看花的开源3

1.2.2 从为什么选择Java谈为什么不  
开源驱动程序3

## 1.2.3 对驱动开发者来说是一把双刃剑4

## 1.3 Android和Linux4

## 1.3.1 Linux简介5

## 1.3.2 Android和Linux的关系5

## 1.4 简析Linux内核8

## 1.4.1 内核的体系结构8

1.4.2 和Android密切相关的Linux  
内核知识10

## 1.5 分析Linux内核源代码很有必要14

## 1.5.1 源代码目录结构14

## 1.5.2 浏览源代码的工具16

1.5.3 为什么用汇编语言编写内核  
代码17

## 1.5.4 Linux内核的显著特性18

## 1.5.5 学习Linux内核的方法26

## 第2章 分析Android源代码31

## 2.1 搭建Linux开发环境和工具31

## 2.1.1 搭建Linux开发环境31

## 2.1.2 设置环境变量32

## 2.1.3 安装编译工具32

## 2.2 获取Android源代码33

## 2.3 分析并编译Android源代码35

## 2.3.1 Android源代码的结构35

## 2.3.2 编译Android源代码40

## 2.3.3 运行Android源代码42

2.3.4 实践演练——演示编译Android  
程序的两种方法43

## 2.4 编译Android Kernel47

## 2.4.1 获取Goldfish内核代码47

## 2.4.2 获取MSM内核代码50

## 2.4.3 获取OMAP内核代码50

## 2.4.4 编译Android的Linux内核50

## 2.5 运行模拟器52

## 2.5.1 Linux环境下运行模拟器的方法53

## 2.5.2 模拟器辅助工具——adb54

## 第3章 驱动需要移植57

## &lt;&lt;Android底层开发技术实战详解&gt;&gt;

- 3.1 驱动开发需要做的工作57
- 3.2 Android移植59
  - 3.2.1 移植的任务60
  - 3.2.2 移植的内容60
  - 3.2.3 驱动开发需要做的工作61
- 3.3 Android对Linux的改造61
  - 3.3.1 Android对Linux内核文件的改动62
  - 3.3.2 为Android构建 Linux的操作系统63
- 3.4 内核空间和用户空间接口是一个媒介64
  - 3.4.1 内核空间和用户空间的相互作用64
  - 3.4.2 系统和硬件之间的交互64
  - 3.4.3 使用Relay实现内核到用户空间的数据传输66
- 3.5 三类驱动程序70
  - 3.5.1 字符设备驱动程序70
  - 3.5.2 块设备驱动程序79
  - 3.5.3 网络设备驱动程序82
- 第4章 HAL层深入分析84
  - 4.1 认识HAL层84
    - 4.1.1 HAL层的发展84
    - 4.1.2 过去和现在的区别86
  - 4.2 分析HAL层源代码86
    - 4.2.1 分析HAL module86
    - 4.2.2 分析mokoid工程89
  - 4.3 总结HAL层的使用方法98
  - 4.4 传感器在HAL层的表现101
    - 4.4.1 HAL层的Sensor代码102
    - 4.4.2 总结Sensor编程的流程104
    - 4.4.3 分析Sensor源代码看Android API 与硬件平台的衔接104
  - 4.5 移植总结116
    - 4.5.1 移植各个Android部件的方式116
    - 4.5.2 移植技巧之一——不得不说的辅助工作117
- 第5章 Goldfish下的驱动解析125
  - 5.1 staging驱动125
    - 5.1.1 staging驱动概述125
    - 5.1.2 Binder驱动程序126
    - 5.1.3 Logger驱动程序135
    - 5.1.4 Lowmemorykiller组件136
    - 5.1.5 Timed Output驱动程序137
    - 5.1.6 Timed Gpio驱动程序139
    - 5.1.7 Ram Console驱动程序139

## <<Android底层开发技术实战详解>>

- 5.2 wakelock和early\_suspend140
  - 5.2.1 wakelock和early\_suspend的原理140
  - 5.2.2 Android休眠141
  - 5.2.3 Android唤醒144
- 5.3 Ashmem驱动程序145
- 5.4 Pmem驱动程序148
- 5.5 Alarm驱动程序149
  - 5.5.1 Alarm简析149
  - 5.5.2 Alarm驱动程序的实现150
- 5.6 USB Gadget驱动程序151
- 5.7 Android Paranoid驱动程序153
- 5.8 Goldfish设备驱动154
  - 5.8.1 FrameBuffer驱动155
  - 5.8.2 键盘驱动159
  - 5.8.3 实时时钟驱动程序160
  - 5.8.4 TTY终端驱动程序161
  - 5.8.5 NandFlash驱动程序162
  - 5.8.6 MMC驱动程序162
  - 5.8.7 电池驱动程序162
- 第6章 MSM内核和驱动解析164
  - 6.1 MSM基础164
    - 6.1.1 常见MSM处理器产品164
    - 6.1.2 Snapdragon内核介绍165
  - 6.2 移植MSM内核简介166
  - 6.3 移植MSM168
    - 6.3.1 Makefile文件168
    - 6.3.2 驱动和组件170
    - 6.3.3 设备驱动172
    - 6.3.4 高通特有的组件174
- 第7章 OMAP内核和驱动解析177
  - 7.1 OMAP基础177
    - 7.1.1 OMAP简析177
    - 7.1.2 常见OMAP处理器产品177
    - 7.1.3 开发平台178
  - 7.2 OMAP内核178
  - 7.3 移植OMAP体系结构180
    - 7.3.1 移植OMAP平台180
    - 7.3.2 移植OMAP处理器183
  - 7.4 移植Android专用驱动和组件188
  - 7.5 OMAP的设备驱动190
- 第8章 显示系统驱动应用195
  - 8.1 显示系统介绍195
    - 8.1.1 Android的版本195
    - 8.1.2 不同版本的显示系统195
  - 8.2 移植和调试前的准备196
    - 8.2.1 FrameBuffer驱动程序196

## <<Android底层开发技术实战详解>>

- 8.2.2 硬件抽象层198
- 8.3 实现显示系统的驱动程序210
  - 8.3.1 Goldfish中的Framebuffer驱动程序210
  - 8.3.2 使用Gralloc模块的驱动程序214
- 8.4 MSM高通处理器中的显示驱动实现224
  - 8.4.1 MSM中的Framebuffer驱动程序225
  - 8.4.2 MSM中的Gralloc驱动程序227
- 8.5 OMAP处理器中的显示驱动实现235
- 第9章 输入系统驱动应用239
  - 9.1 输入系统介绍239
    - 9.1.1 Android输入系统结构元素介绍239
    - 9.1.2 移植Android输入系统时的工作240
  - 9.2 Input ( 输入 ) 驱动241
  - 9.3 模拟器的输入驱动256
  - 9.4 MSM高通处理器中的输入驱动实现257
    - 9.4.1 触摸屏驱动257
    - 9.4.2 按键和轨迹球驱动264
  - 9.5 OMAP处理器平台中的输入驱动实现266
    - 9.5.1 触摸屏驱动267
    - 9.5.2 键盘驱动267
- 第10章 振动器系统驱动269
  - 10.1 振动器系统结构269
    - 10.1.1 硬件抽象层271
    - 10.1.2 JNI框架部分272
  - 10.2 开始移植273
    - 10.2.1 移植振动器驱动程序273
    - 10.2.2 实现硬件抽象层274
  - 10.3 在MSM平台实现振动器驱动275
- 第11章 音频系统驱动279
  - 11.1 音频系统结构279
  - 11.2 分析音频系统的层次280
    - 11.2.1 层次说明280
    - 11.2.2 Media库中的Audio框架281
    - 11.2.3 本地代码284
    - 11.2.4 JNI代码288
    - 11.2.5 Java代码289
  - 11.3 移植Audio系统的必备技术289
    - 11.3.1 移植Audio系统所要做的  
工作289
    - 11.3.2 分析硬件抽象层290

## <<Android底层开发技术实战详解>>

- 11.3.3 分析AudioFlinger中的Audio硬件抽象层的实现291
- 11.4 真正实现Audio硬件抽象层298
- 11.5 MSM平台实现Audio驱动系统298
  - 11.5.1 实现Audio驱动程序298
  - 11.5.2 实现硬件抽象层299
- 11.6 OSS平台实现Audio驱动系统304
  - 11.6.1 OSS驱动程序介绍304
  - 11.6.2 mixer305
- 11.7 ALSA平台实现Audio系统312
  - 11.7.1 注册音频设备和音频驱动312
  - 11.7.2 在Android中使用ALSA声卡313
  - 11.7.3 在OMAP平台移植Android的ALSA声卡驱动322
- 第12章 视频输出系统驱动326
  - 12.1 视频输出系统结构326
  - 12.2 需要移植的部分328
  - 12.3 分析硬件抽象层328
    - 12.3.1 Overlay系统硬件抽象层的接口328
    - 12.3.2 实现Overlay系统的硬件抽象层331
    - 12.3.3 实现接口3

## 章节摘录

版权页：插图：Wi-Fi系统Java层的核心是根据IWifiManger接口所创建的Binder服务器端和客户端，服务器端是WifiService，客户端是WifiManger。

编译IWifiManger.aidl生成文件IWifiManger.java，并生成IWifiManger.Stub（服务器端抽象类）和IWifiManger.Stub.Proxy（客户端代理实现类）。

WifiService通过继承IWifiManger.Stub实现，而客户端通过getService（）函数获取IWifiManger.Stub.Proxy（即Service的代理类），将其作为参数传递给WifiManger，供其与WifiService通信时使用。

Wi-Fi系统Java部分的核心是根据IWifiManager接口所创建的Binder服务器端和客户端，服务器端是WifiService，客户端是WifiManager。

具体结构如图17—4所示。

图17—4中主要构成元素的具体说明如下所示。

（1）WifiManger是Wi-Fi部分与外界的接口，用户通过它来访问Wi-Fi的核心功能。

WifiWatchdogService这一系统组件也是用WifiManger来执行一些具体操作。

（2）WifiService是服务器端的实现，作为Wi-Fi的核心，处理实际的驱动加载、扫描、链接/断开等命令，以及底层上报的事件。

对于主动的命令控制，Wi-Fi是一个简单的封装，针对来自客户端的控制命令，调用相应的WifiNative底层实现。

当接收到客户端的命令后，一般会将其转换成对应的自身消息塞入消息队列中，以便客户端的调用可以及时返回，然后在WifiHandler的handleMessage（）中处理对应的消息。

而底层上报的事件，WifiService则通过启动WifiStateTracker来负责处理。

WifiStateTracker和WifiMonitor的具体功能如下所示。

WifiStateTracker除了负责WiFi的电源管理模式等功能外，其核心是WifiMonitor所实现的事件轮询机制，以及消息处理函数handleMessage（）。

WifiMonitor通过开启一个MonitorThread来实现事件轮询，轮询的关键函数是前面提到的阻塞式函数WifiNative.waitForEvent（）。

获取事件后，WifiMonitor通过一系列的Handler通知给WifiStateTracker。

这里WifiMonitor的通知机制是将底层事件转换成WifiStateTracker所能识别的消息，塞入WifiStateTracker的消息循环中，最终在handleMessage（）中由WifiStateTracker完成对应的处理。



## <<Android底层开发技术实战详解>>

### 编辑推荐

《Android底层开发技术实战详解:内核、移植和驱动》适合Android研发人员及Android爱好者学习，也可以作为相关培训学校和大专院校相关专业的教学用书。

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>