

# 深入解析

异步图书  
www.epubit.com.cn

钟世礼 ◆ 编著

# Android

# 虚拟机



## 本书涵盖了 Android 虚拟机原理的主要内容

依次讲解了 Java 虚拟机基础、Android 虚拟机基础，分析 JNI、内存系统、Android 程序的生命周期管理、IPC 进程通信机制、init 进程，讲解了 Dalvik VM 的进程系统和运作流程、Dvlik VM 内存系统、Dalvik VM 垃圾收集机制、Dalvik VM 内存优化机制、Dalvik VM 的启动过程、Dalvik VM 异常处理，以及 Dalvik VM 内存优化和 Dalvik VM 性能优化等内容。

 中国工信出版集团

 人民邮电出版社  
POSTS & TELECOM PRESS

# 深入解析 Android 虚拟机

钟世礼◆编著

人民邮电出版社  
北京

## 图书在版编目 (C I P) 数据

深入解析Android虚拟机 / 钟世礼 编著. — 北京 :  
人民邮电出版社, 2016.9  
ISBN 978-7-115-42353-5

I. ①深… II. ①钟… III. ①移动终端—应用程序—  
程序设计—虚拟处理机 IV. ①TP338

中国版本图书馆CIP数据核字(2016)第238207号

## 内 容 简 介

Android 系统从诞生到现在的短短几年时间里, 凭借其易用性和开发的简洁性, 赢得了广大开发者的支持。在整个 Android 系统中, Dalvik VM 一直是贯穿从底层内核到高层应用开发的核心。本书循序渐进地讲解了 Android 虚拟机系统的基本知识, 并剖析了其整个内存系统的进程和运作流程, 并对虚拟机系统优化和异常处理的知识进行了详细讲解。本书几乎涵盖了 Dalvik VM 系统的所有主要内容, 并且讲解方法通俗易懂, 特别有利于读者学习并消化。

本书适合 Android 初学者、Android 底层开发人员、源代码分析人员和虚拟机开发人员学习, 也可以作为大专院校相关专业师生的学习用书和培训学校的教材。

- 
- ◆ 编 著 钟世礼  
责任编辑 张 涛  
责任印制 沈 蓉 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>  
固安县铭成印刷有限公司印刷
  - ◆ 开本: 787×1092 1/16  
印张: 38  
字数: 1004 千字 2016 年 9 月第 1 版  
印数: 1—2 000 册 2016 年 9 月河北第 1 次印刷
- 

定价: 99.00 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

Android 虚拟机技术——Dalvik VM 是通往 Android 高级开发的必备技术！为了让广大读者深入理解 Android 系统，不再停留在抽象的原理和概念之上，本书对 Android 虚拟机方面的知识进行了细致分析，这样做的目的是“提炼”出 Android 系统的本质，了解 Android 系统究竟是如何运作的，进程和线程之间是如何协调并进的，内存之间是如何分配并存的。并以此为基础，详细讲解了内存优化、垃圾收集和系统优化方面的基本原理和具体实现。

## 本书的内容

本书共 24 章，循序渐进地讲解了 Android 虚拟机系统的基本知识，从获取并编译 Android 源码开始，依次讲解了 Java 虚拟机基础、Android 虚拟机基础、分析 JNI、分析内存系统、Android 程序的生命周期管理、IPC 进程通信机制、init 进程、Dalvik VM 的进程系统、Dalvik VM 运作流程、DEX 文件、Dvlik VM 内存系统、Dalvik VM 垃圾收集机制、Dalvik VM 内存优化机制、Dalvik VM 的启动过程、注册 Dalvik VM 并创建线程、Dalvik VM 异常处理、JIT 编译、Dalvik VM 内存优化、Dalvik VM 性能优化等内容。

## 本书特色

在内容的编写上，本书具有以下特色。

### (1) 结构合理

从用户的实际需要出发，科学安排知识结构，详细讲解了 Android 虚拟机的各方面知识，内容循序渐进、由浅入深。

### (2) 遵循“基础讲解—源码分析—核心技术剖析”这一主线

为了使广大读者彻底弄清楚 Android 虚拟机中的各个知识点，剖析了与 Android 虚拟机相关的进程运行机制、内存系统、生命周期管理等核心知识，并讲解了读者关心的系统优化技术。

### (3) 易学易懂

本书内容条理清晰、语言简洁，可以帮助读者快速掌握每个知识点。使读者既可以按照本书编排的章节顺序进行学习，也可以根据自己的需求对某一章节进行有针对性地学习。

## 本书参考资料

由于 Android 虚拟机系统十分深奥，加上市面上的相关资料十分稀缺。作者在写作过程中对每一段文字都进行了深入研究和推敲，并参阅了国内外大师们的经典资料，对这些资料进行了深入地研读。在作者的写作过程中，从下面 4 部分资料中获得了帮助。

### (1) Oracle 官方资料

<http://docs.oracle.com/javase/7/docs/>

<http://docs.oracle.com/javase/6/docs/>

<http://www.oracle.com/technetwork/java/>

上述资料是 Oracle 官方提供的 Java 虚拟机资料，这些资料也是国内外读者学习 Java 虚拟机的第一手资料。

#### (2) 国外经典名著

《The Java Language Specification, Third Edition》

《The Java Virtual Machine Specification》

上述资料是国外大师们根据 Oracle 官方资料而著成的经典名著，也是国内外读者学习 Java 虚拟机的参考资料。在国内的一些开源论坛中，有很多热心网友进行了翻译。

#### (3) Google 官方资料

Google I/O 2010 - A JIT Compiler for Android's Dalvik VM

Dalvik VM Internals - Presentation from Google I/O 2008, by Dan Bornstein

Detailed Dalvik specifications documents

上述资料是 Google 公司《Google I/O 讲座系列》的内容，讲解了 Android 虚拟机优化和内存系统的知识，对广大初学者来说有很强的借鉴作用。当然，Google 提供的 Android 源码更是人们分析 Dalvik VM 的第一手资料。

#### (4) 国内著作

《解析 Java 虚拟机开发：权衡优化、高效和安全的最优方案》清华大学出版社，张善香，2013-06-01。

这是国内技术高人的一本著作，可以说是讲解 Java 虚拟机方面较全的一本参考书。里面介绍的很多内容对写作本书有很大启发，想了解这方面内容的读者可以参考一下。

## 读者对象

- 初学 Android 编程的自学者
- Linux 开发人员
- 大中专院校的老师和学生
- 做毕业设计的学生
- Android 编程爱好者
- 相关培训机构的老师和学员
- 从事 Android 开发的程序员

本书在编写过程中，我的家人在我写作时给予了巨大支持，在此表示深深的感谢。另外，由于本人水平有限，书中如有纰漏和不尽如人意之处在所难免，诚请读者提出意见或建议，以便今后修订并使之更臻完善。另外为本书提供了售后支持网站：<http://www.toppr.net/>，读者如有疑问可以在此提出，一定会得到满意的答复。编辑联系邮箱：[zhangtao@ptpress.com.cn](mailto:zhangtao@ptpress.com.cn)。

作者



# 目 录

第 1 章 获取并编译 Android 源码	1
1.1 获取 Android 源码	1
1.1.1 在 Linux 系统获取 Android 源码	1
1.1.2 在 Windows 平台获取 Android 源码	2
1.1.3 Windows 获取 Android L 源码	4
1.2 分析 Android 源码结构	6
1.3 编译 Android 源码	8
1.3.1 搭建编译环境	8
1.3.2 开始编译	9
1.3.3 在模拟器中运行	10
1.3.4 常见的错误分析	10
1.3.5 实践演练——演示两种编译 Android 程序的方法	11
1.4 编译 Android Kernel	14
1.4.1 获取 Goldfish 内核代码	14
1.4.2 获取 MSM 内核代码	17
1.4.3 获取 OMAP 内核代码	17
1.4.4 编译 Android 的 Linux 内核	17
第 2 章 Java 虚拟机基础	19
2.1 虚拟机的作用	19
2.2 Java 虚拟机概述	20
2.2.1 JVM 的数据类型	20
2.2.2 Java 虚拟机体系结构	21
2.2.3 JVM 的生命周期	25
2.3 JVM 的安全性	26
2.3.1 JVM 的安全模型	26
2.3.2 沙箱模型的 4 种组件	27
2.3.3 分析 Java 的策略机制	28
2.4 网络移动性	29
2.4.1 现实需要网络移动性	29
2.4.2 网络移动性	30
2.5 内存异常和垃圾处理	31
2.5.1 内存分配中的栈和堆	31
2.5.2 运行时的数据区域	33
2.5.3 对象访问	34
2.5.4 内存泄露	35
2.5.5 JVM 的垃圾收集策略	36
2.5.6 垃圾收集器	37
2.6 Java 内存模型	37
2.6.1 Java 内存模型概述	38
2.6.2 主内存与工作内存	38
2.6.3 内存间交互操作	39
第 3 章 Dalvik 和 ART 基础	40
3.1 Dalvik VM 和 JVM 的差异	40
3.2 Dalvik 虚拟机的主要特征	41
3.3 Dalvik VM 架构	42
3.3.1 Dalvik 虚拟机的代码结构	42
3.3.2 dx 工具	44
3.3.3 Dalvik VM 的进程管理	44
3.3.4 Android 的初始化流程	44
3.4 Dalvik VM 控制 VM 命令详解	45
3.4.1 基本命令	45
3.4.2 扩展的 JNI 检测	45
3.4.3 断言	46
3.4.4 字节码校验和优化	46
3.4.5 Dalvik VM 的运行模式	47
3.4.6 死锁预测	47
3.4.7 dump 堆栈追踪	48
3.4.8 dex 文件和校验	48
3.4.9 产生标志位	48
3.5 ART 机制基础	48
3.5.1 什么是 ART 模式	48
3.5.2 ART 优化机制基础	50
第 4 章 分析 JNI	52
4.1 JNI 的本质	52
4.2 分析 Java 层	54
4.2.1 加载 JNI 库	54
4.2.2 实现扫描工作	55
4.2.3 读取并保存信息	56
4.2.4 删除 SD 卡外的信息	58
4.2.5 直接转向 JNI	58
4.2.6 扫描函数 scanFile	59
4.2.7 JNI 中的异常处理	59
4.3 分析 JNI 层	60

4.3.1	将 Native 对象的指针保存到 Java 对象	60	6.2.4	管理 Activity 的生命周期	119
4.3.2	创建 Native 层的 MediaScanner 对象	60	6.2.5	Activity 的实例化与启动	120
4.4	Native (本地) 层	61	6.2.6	Activity 的暂停与继续	120
4.4.1	注册 JNI 函数	61	6.2.7	Activity 的关闭/销毁与重新运行	121
4.4.2	完成注册工作	63	6.2.8	Activity 的启动模式	121
4.4.3	动态注册	64	6.3	进程与线程	122
4.4.4	处理路径参数	65	6.3.1	进程	122
4.4.5	扫描文件	66	6.3.2	线程	123
4.4.6	添加 TAG 信息	66	6.3.3	线程安全的方法	123
4.4.7	总结函数 JNI_OnLoad() 与函数 JNI_OnUnload() 的用途	67	6.3.4	Android 的线程模型	123
4.4.8	Java 与 JNI 基本数据类型转换	67	6.4	测试生命周期	125
4.4.9	JNIEnv 接口	69	6.5	Service 的生命周期	129
4.4.10	JNI 中的环境变量	70	6.5.1	Service 的基本概念和用途	129
6.5.2	Service 的生命周期详解	129	6.5.2	Service 的生命周期详解	129
6.5.3	Service 与 Activity 通信	129	6.5.3	Service 与 Activity 通信	129
6.6	Android 广播的生命周期	133	6.6	Android 广播的生命周期	133
6.6.1	Android 的广播机制	133	6.6.1	Android 的广播机制	133
6.6.2	编写广播程序	133	6.6.2	编写广播程序	133
6.7	ART 进程管理	135	6.7	ART 进程管理	135
第 5 章	分析内存系统	71	第 7 章	IPC 进程通信机制	147
5.1	分析 Android 的进程通信机制	71	7.1	Binder 机制概述	147
5.1.1	Android 的进程间通信 (IPC) 机制 Binder	71	7.2	Service Manager 是 Binder 机制的上下文管理者	148
5.1.2	Service Manager 是 Binder 机制的上下文管理者	72	7.2.1	入口函数	148
5.1.3	Service Manager 服务	86	7.2.2	打开 Binder 设备文件	149
5.2	匿名共享内存子系统详解	89	7.2.3	创建设备文件	149
5.2.1	基础数据结构	89	7.2.4	管理内存映射地址空间	154
5.2.2	初始化处理	90	7.2.5	发生通知	156
5.2.3	打开匿名共享内存设备文件	91	7.2.6	循环等待	161
5.2.4	内存映射	93	7.3	内存映射	162
5.2.5	读写操作	94	7.3.1	实现内存分配功能	162
5.2.6	锁定和解锁	95	7.3.2	分配物理内存	164
5.2.7	回收内存块	100	7.3.3	释放物理页面	166
5.3	C++ 访问接口层详解	101	7.3.4	分配内核缓冲区	167
5.3.1	接口 MemoryBase	101	7.3.5	释放内核缓冲区	168
5.3.2	接口 MemoryBase	108	7.3.6	查询内核缓冲区	170
5.4	Java 访问接口层详解	111	第 8 章	init 进程详解	171
第 6 章	Android 程序的生命周期管理	115	8.1	init 基础	171
6.1	Android 程序的生命周期	115	8.2	分析入口函数	172
6.1.1	进程和线程	115	8.3	配置文件详解	174
6.1.2	进程的类型	116	8.3.1	init.rc 简介	174
6.2	Activity 的生命周期	116	8.3.2	分析 init.rc 的过程	176
6.2.1	Activity 的几种状态	117	8.4	解析 service	179
6.2.2	分解剖析 Activity	117			
6.2.3	几个典型的场景	119			

8.4.1	Zygote 对应的 service action	179	10.2.2	初始化跟踪显示系统	262
8.4.2	init 组织 service	180	10.2.3	初始化垃圾回收器	263
8.4.3	函数 parse_service 和 parse_line_service	181	10.2.4	初始化线程列表和主线程 环境参数	263
8.5	字段 on	184	10.2.5	分配内部操作方法的表格 内存	264
8.5.1	Zygote 对应的 on action	184	10.2.6	初始化虚拟机的指令码相关 的内容	264
8.5.2	init 组织 on	185	10.2.7	分配指令寄存器状态的 内存	264
8.5.3	解析 on 用到的函数	186	10.2.8	分配指令寄存器状态的内存 和最基本用的 Java 库	265
8.6	在 init 控制 service	186	10.2.9	初始化使用的 Java 类库 线程类	266
8.6.1	启动 Zygote	186	10.2.10	初始化虚拟机使用的异常 Java 类库	267
8.6.2	启动 service	187	10.2.11	初始化其他对象	268
8.6.3	4 种启动 service 的方式	191	10.3	启动 Zygote	276
8.7	控制属性服务	194	10.3.1	在 init.rc 中配置 Zygote 启动 参数	276
8.7.1	引入属性	194	10.3.2	启动 Socket 服务端口	276
8.7.2	初始化属性服务	197	10.3.3	加载 preload-classes	277
8.7.3	启动属性服务	197	10.3.4	加载 preload-resources	277
8.7.4	处理设置属性的请求	200	10.3.5	使用 folk 启动新进程	278
第 9 章	Dalvik VM 的进程系统	202	10.4	启动 SystemServer 进程	278
9.1	Zygote (孕育) 进程详解	202	10.4.1	启动各种系统服务线程	279
9.1.1	Zygote 基础	202	10.4.2	启动第一个 Activity	280
9.1.2	分析 Zygote 的启动过程	203	10.5	加载 class 类文件	281
9.2	System 进程详解	216	10.5.1	DexFile 在内存中的映射	281
9.2.1	启动 System 进程前的准备 工作	216	10.5.2	ClassObject——Class 在 加载后的表现形式	283
9.2.2	分析 SystemServer	217	10.5.3	加载 Class 并生成相应 ClassObject 的函数	283
9.2.3	分析 EntropyService	220	10.5.4	加载基本类库文件	284
9.2.4	分析 DropBoxManagerService	222	10.5.5	加载用户类文件	284
9.2.5	分析 DiskStatsService	227	第 11 章	DEX 文件详解	285
9.2.6	分析 DeviceStorageManager Service	231	11.1	DEX 文件介绍	285
9.2.7	分析 SamplingProfilerService	233	11.2	DEX 文件的格式	285
9.2.8	分析 ClipboardService	241	11.2.1	map_list	286
9.3	应用程序进程详解	247	11.2.2	string_id_item	288
9.3.1	创建应用程序	247	11.2.3	type_id_item	291
9.3.2	启动线程池	256	11.2.4	proto_id_item	292
9.3.3	创建信息循环	257	11.2.5	ield_id_item	293
第 10 章	Dalvik VM 运作流程详解	259	11.2.6	method_id_item	293
10.1	Dalvik VM 相关的可执行程序	259	11.2.7	class_def_item	294
10.1.1	dalvikvm、dvz 和 app_process 简介	259			
10.1.2	对比 app_process 和 dalvikvm 的 执行过程	260			
10.2	初始化 Dalvik 虚拟机	262			
10.2.1	开始虚拟机的准备工作	262			



11.3 DEX 文件结构	297	15.2.4 创建 JNIEnvExt 对象	378
11.3.1 文件头 (File Header)	297	15.2.5 设置当前进程和进程组 ID	382
11.3.2 魔数字段	298	15.2.6 注册 Android 核心类的 JNI 方法	382
11.3.3 检验码字段	298	15.2.7 创建 javaCreateThreadEtc 钩子	385
11.3.4 SHA-1 签名字段	300		
11.3.5 map_off 字段	300		
11.3.6 string_ids_size 和 off 字段	301		
11.4 DEXFile 接口详解	303	<b>第 16 章 注册 Dalvik VM 并创建线程</b>	387
11.4.1 构造函数	303	16.1 注册 Dalvik VM 的 JNI 方法	387
11.4.2 公共方法	304	16.1.1 设置加载程序	387
11.5 DEX 和动态加载类机制	306	16.1.2 加载 so 文件并验证	387
11.5.1 类加载机制	306	16.1.3 获取描述类	392
11.5.2 具体加载	306	16.1.4 注册 JNI 方法	392
11.5.3 代码加密	308	16.1.5 实现 JNI 操作	394
11.6 动态加载 jar 和 DEX	309	16.2 创建 Dalvik VM 进程	395
<b>第 12 章 Dvlik VM 内存系统详解</b>	310	16.2.1 分析底层启动过程	395
12.1 如何分配内存	310	16.2.2 创建 Dalvik VM 进程	395
12.2 内存管理机制详解	312	16.2.3 初始化运行的 Dalvik VM	398
12.3 优化 Dalvik 虚拟机的堆内存分配	326	16.3 创建 Dalvik VM 线程	399
<b>第 13 章 Dalvik VM 垃圾收集机制</b>	328	16.3.1 检查状态值	399
13.1 引用计数算法	328	16.3.2 创建线程	399
13.2 Mark Sweep 算法	328	16.3.3 分析启动过程	402
13.3 和垃圾收集算法有关的函数	330	16.3.4 清理线程	404
13.4 垃圾回收的时机	346	<b>第 17 章 Dalvik VM 异常处理详解</b>	407
13.5 调试信息	347	17.1 Java 异常处理机制	407
13.6 Dalvik VM 和 JVM 垃圾收集机制的区别	348	17.1.1 方法调用栈	407
<b>第 14 章 Dalvik VM 内存优化机制详解</b>	350	17.1.2 Java 提供的异常处理类	409
14.1 sp 和 wp 简介	350	17.2 Java VM 异常处理机制详解	409
14.1.1 sp 基础	350	17.2.1 Java 语言及虚拟机的异常处理机制	410
14.1.2 wp 基础	351	17.2.2 COSIX 虚拟机异常处理的设计与实现	410
14.2 智能指针详解	351	17.3 分析 Dalvik 虚拟机异常处理的源码	414
14.2.1 智能指针基础	352	17.3.1 初始化虚拟机使用的异常 Java 类库	414
14.2.2 轻量级指针	353	17.3.2 抛出一个线程异常	415
14.2.3 强指针	355	17.3.3 持续抛出进程	415
14.2.4 弱指针	365	17.3.4 找出异常原因	416
<b>第 15 章 分析 Dalvik VM 的启动过程</b>	369	17.3.5 找出异常原因	417
15.1 Dalvik VM 启动流程概览	369	17.3.6 清除挂起的异常和等待初始化的异常	420
15.2 Dalvik VM 启动过程详解	370	17.3.7 包装“现在等待”异常的不同例外	420
15.2.1 创建 Dalvik VM 实例	370	17.3.8 输出跟踪当前异常的错误信息	421
15.2.2 指定一系列控制选项	371		
15.2.3 创建并初始化 Dalvik VM 实例	376		

17.3.9	搜索和当前异常相匹配的方法	421	19.2.3	通过 Runtime 类实现	465
17.3.10	获取匹配的捕获块	423	19.2.4	使用 DDMS 工具获取	465
17.3.11	进行堆栈跟踪	424	19.2.5	其他方法	469
17.3.12	生成堆栈跟踪元素	425	19.3	Android 的内存泄露	472
17.3.13	将内容添加到堆栈跟踪日志中	426	19.3.1	什么是内存泄露	472
17.3.14	将内容添加到堆栈跟踪日志中	427	19.3.2	为什么会发生内存泄露	473
17.4	常见异常的类型与原因	428	19.3.3	shallow size、retained size	474
17.4.1	SQLException: 操作数据库异常类	428	19.3.4	查看 Android 内存泄露的工具——MAT	475
17.4.2	ClassCastException: 数据类型转换异常	428	19.3.5	查看 Android 内存泄露的方法	478
17.4.3	NumberFormatException: 字符串转换为数字类型时抛出的异常	428	19.3.6	Android (Java) 中常见的容易引起内存泄漏的不良代码	480
17.5	调用堆栈跟踪分析异常	429	19.4	常见的引起内存泄露的坏习惯	480
17.5.1	解决段错误	429	19.4.1	查询数据库时忘记关闭游标	481
17.5.2	跟踪 Android Callback 调用堆栈	431	19.4.2	构造 Adapter 时不习惯使用缓存的 convertView	481
第 18 章	JIT 编译	434	19.4.3	没有及时释放对象的引用	482
18.1	JIT 简介	434	19.4.4	不在使用 Bitmap 对象时调用 recycle() 释放内存	482
18.1.1	JIT 概述	434	19.5	解决内存泄露实践	483
18.1.2	Java 虚拟机主要的优化技术	436	19.5.1	使用 MAT 根据 heap dump 分析 Java 代码内存泄漏的根源	483
18.1.3	Dalvik 中 JIT 的实现	436	19.5.2	演练 Android 中内存泄露代码优化及检测	489
18.2	Dalvik VM 对 JIT 的支持	436	第 20 章	Dalvik VM 性能优化	491
18.3	汇编代码和改动	438	20.1	加载 APK/DEX 文件优化	491
18.3.1	汇编部分代码	438	20.1.1	APK 文件介绍	492
18.3.2	对 C 文件的改动	438	20.1.2	DEX 文件优化	493
18.4	Dalvik VM 中的 JIT 源码	439	20.1.3	使用类动态加载技术实现加密优化	493
18.4.1	入口文件	439	20.2	SD 卡优化	496
18.4.2	核心函数	447	20.3	虚拟机优化详解	497
18.4.3	编译文件	450	20.3.1	平台优化——ARM 的流水线技术	497
18.4.4	BasicBlock 处理	458	20.3.2	Android 对 C 库优化	501
18.4.5	内存初始化	459	20.3.3	优化创建的进程	504
18.4.6	对 JIT 源码的总结	462	20.3.4	渲染优化	504
第 19 章	Dalvik VM 内存优化	463	第 21 章	分析 ART 的启动过程	508
19.1	Android 内存优化的作用	463	21.1	运行环境的转换	508
19.2	查看 Android 内存和 CPU 使用情况	464	21.2	运行 app_process 进程	509
19.2.1	利用 Android API 函数查看	464	21.3	准备启动	512
19.2.2	直接对 Android 文件进行解析查询	464	21.4	创建运行实例	518

21.5	注册本地 JNI 函数	519	23.2	主函数 main	549
21.6	启动守护进程	520	23.3	调用初始化函数	550
21.7	解析参数	521	23.4	创建 PackageManagerService 服务	553
21.8	初始化类、方法和域	528	23.5	扫描并解析	554
<b>第 22 章 执行 ART 主程序</b>		534	23.6	保存解析信息	570
22.1	进入 main 主函数	534	<b>第 24 章 ART 环境安装 APK 应用程序</b>		572
22.2	查找目标类	535	24.1	Android 安装 APK 概述	572
22.2.1	函数 LookupClass()	535	24.2	启动时安装	572
22.2.2	函数 DefineClass()	537	24.3	ART 安装	581
22.2.3	函数 InsertClass()	540	24.4	实现 dex2oat 转换	586
22.2.4	函数 LinkClass()	541	24.4.1	参数解析	586
22.3	类操作	543	24.4.2	创建 OAT 文件指针	588
22.4	实现托管操作	544	24.4.3	dex2oat 准备工作	588
<b>第 23 章 安装 APK 应用程序</b>		549	24.4.4	提取 classes.dex 文件	589
23.1	PackageManagerService 概述	549	24.4.5	创建 OAT 文件	594
			24.5	APK 文件的转换	595

# 第 1 章 获取并编译 Android 源码

在本章中，将详细讲解获取并编译 Android 源码的基本知识，介绍各个目录中主要文件的功能，为读者步入本书后面知识的学习打下基础。

## 1.1 获取 Android 源码

要想研究 Android 系统的源码，需要先获取其源码。目前市面上主流的操作系统有 Windows、Linux、Mac OS 的操作系统，由于 Mac OS 源自于 Linux 系统，因此本书将讲解分别在 Windows 系统和 Linux 系统中获取 Android 源码的知识。

### 1.1.1 在 Linux 系统获取 Android 源码

在 Linux 系统中，通常使用 Ubuntu 来下载和编译 Android 源码。由于 Android 的源码内容很多，Google 采用了 Git 的版本控制工具，并对不同的模块设置不同的 Git 服务器，可以用 repo 自动化脚本来下载 Android 源码，下面介绍获取 Android 源码的过程。

#### (1) 下载 repo。

在用户目录下创建存放 repo 的 bin 文件夹，并把该路径设置到环境变量中去，具体命令如下所示：

```
$ mkdir ~/bin
$ PATH=~/.bin:$PATH
```

下载用于执行 repo 的 repo 的脚本，具体命令如下所示：

```
$ curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
```

设置可执行权限，命令如下所示：

```
$ chmod a+x ~/bin/repo
```

#### (2) 初始化一个 repo 的客户端。

在用户目录下创建一个空目录，用于存放 Android 源码，命令如下所示：

```
$ mkdir AndroidCode
$ cd AndroidCode
```

进入到 AndroidCode 目录，并运行 repo 下载源码，下载主线分支的代码，主线分支包括最新修改的 bug，以及并未正式发布版本的最新源码，命令如下所示：

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

下载其他分支，建议下载正式发布的版本，可以通过添加 -b 参数来下载，例如下载 Android 4.3 正式版的命令如下所示：

```
$ repo init -u https://android.googlesource.com/platform/manifest -b
android-4.3_r1
```

在下载过程中会需要填写 Name 和 E-mail，填写完毕之后，选择 Y 进行确认。最后提示 repo 初始化完成，这时可以开始同步 Android 源码了。同步过程非常漫长，需要大家耐心等待。执行下面命令开始同步代码：

```
$ repo sync
```

经过上述步骤后，便开始下载并同步 Android 源码了，界面效果如图 1-1 所示。

```
Checking out files: 100% (2497/2497), done. out files: 31% (790/2497)
Checking out files: 100% (1654/1654), done. out files: 39% (649/1654)
Checking out files: 100% (3471/3471), done.
Checking out files: 100% (24607/24607), done. ut files: 21% (5269/24607)
Checking out files: 100% (2431/2431), done. out files: 32% (800/2431)
Checking out files: 100% (18696/18696), done.
Checking out files: 100% (4276/4276), done. out files: 48% (2058/4276)
Checking out files: 100% (2216/2216), done. out files: 49% (1093/2216)
Checking out files: 100% (857/857), done. ng out files: 13% (115/857)
Checking out files: 100% (1141/1141), done. out files: 2% (28/1141)
Checking out files: 100% (431/431), done. ng out files: 10% (46/431)
Checking out files: 100% (175/175), done. ng out files: 10% (19/175)
Checking out files: 100% (135/135), done.
Checking out files: 100% (378/378), done.
Checking out files: 100% (433/433), done.
Checking out files: 100% (2407/2407), done. out files: 30% (724/2407)
Checking out files: 100% (2489/2489), done.
Checking out files: 100% (2493/2493), done.
Checking out files: 100% (177/177), done. ng out files: 15% (27/177)
Checking out files: 100% (137/137), done.
Checking out files: 100% (40775/40775), done. ut files: 5% (2199/40775)
Checking out files: 100% (93/93), done.
Checking out files: 100% (450/450), done.
Checking out files: 100% (5265/5265), done. out files: 41% (2167/5265)
Syncing work tree: 100% (329/329), done.
```

图 1-1 下载同步界面

### 1.1.2 在 Windows 平台获取 Android 源码

在 Windows 平台获取源码与在 Linux 上原理相同，但是需要预先在 Windows 平台上搭建一个 Linux 环境，此处需要用到 Cygwin 工具。Cygwin 的作用是构建一套在 Windows 上的 Linux 模拟环境，下载 Cygwin 工具的地址如下所示：

```
http://cygwin.com/install.html
```

下载成功后会得到一个名为“setup.exe”的可执行文件，通过此文件可以更新和下载最新的工具版本，具体流程如下所示。

- (1) 启动 Cygwin，如图 1-2 所示。
- (2) 单击“下一步”按钮，选择第一个选项：从网络下载安装，如图 1-3 所示。

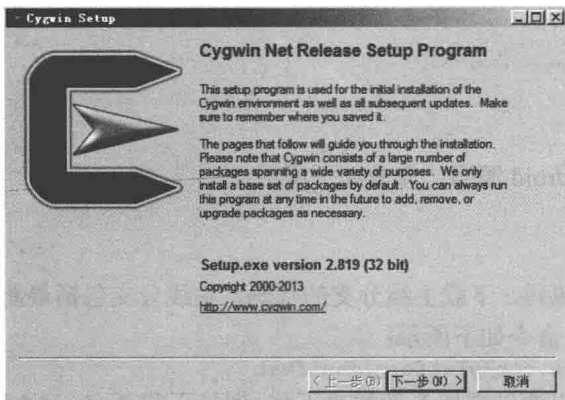


图 1-2 启动 Cygwin

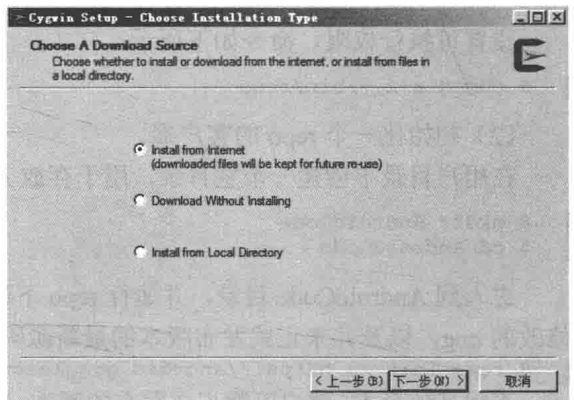


图 1-3 选择从网络下载安装

- (3) 单击“下一步”按钮，选择安装根目录，如图 1-4 所示。
- (4) 单击“下一步”按钮，选择临时文件目录，如图 1-5 所示。

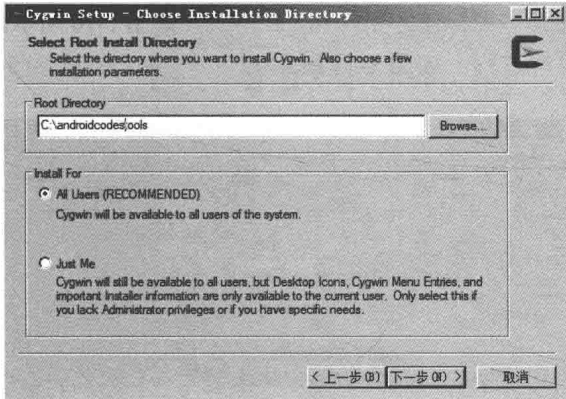


图 1-4 选择安装根目录

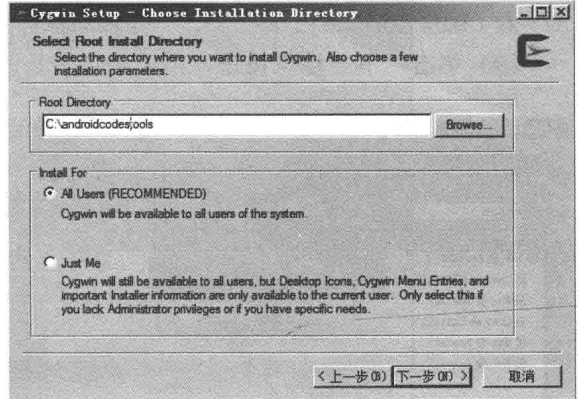


图 1-5 选择临时文件目录

(5) 单击“下一步”按钮，设置网络代理。如果所在网络需要代理，则在这一步进行设置，如果不用代理，则选择直接下载，如图 1-6 所示。

(6) 单击“下一步”按钮，选择下载站点。一般选择离得比较近的站点，速度会比较快，如图 1-7 所示。

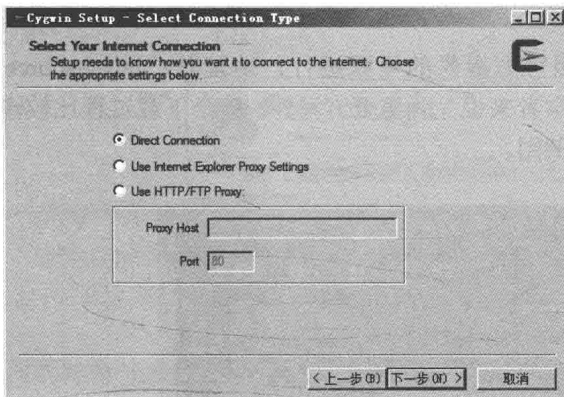


图 1-6 设置网络代理

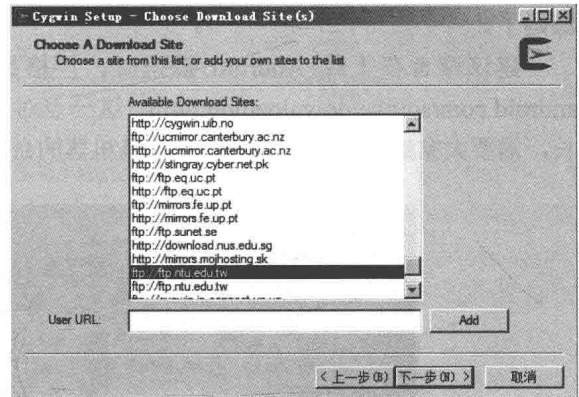


图 1-7 选择下载站点

(7) 单击“下一步”按钮，开始更新工具列表，如图 1-8 所示。

(8) 单击“下一步”按钮，选择需要下载的工具包。在此需要依次下载 curl、git、python 这些工具，如图 1-9 所示。

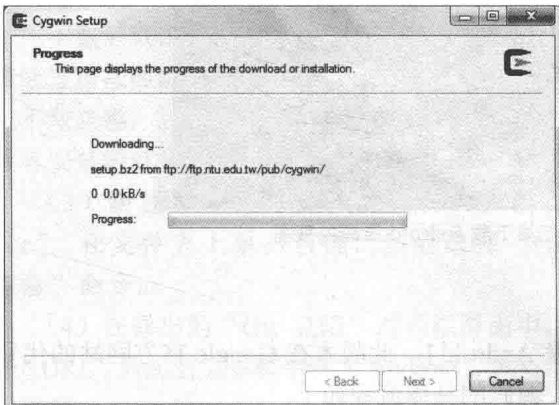


图 1-8 更新工具列表

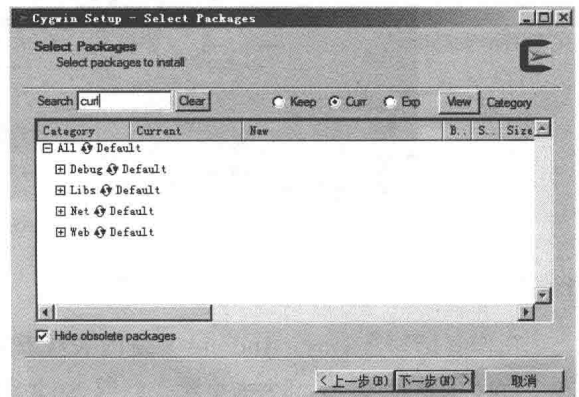


图 1-9 依次下载工具





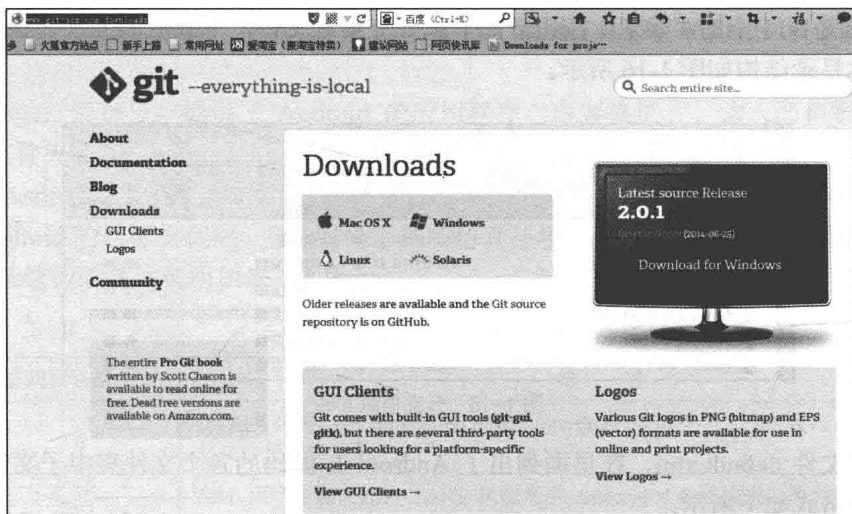


图 1-13 下载 Git 工具

下载后双击可执行文件进行安装，在安装过程按照默认选项安装即可。

(2) 下载并安装 TortoiseGit 工具，下载地址是 <https://code.google.com/p/tortoisegit/>，如图 1-14 所示。

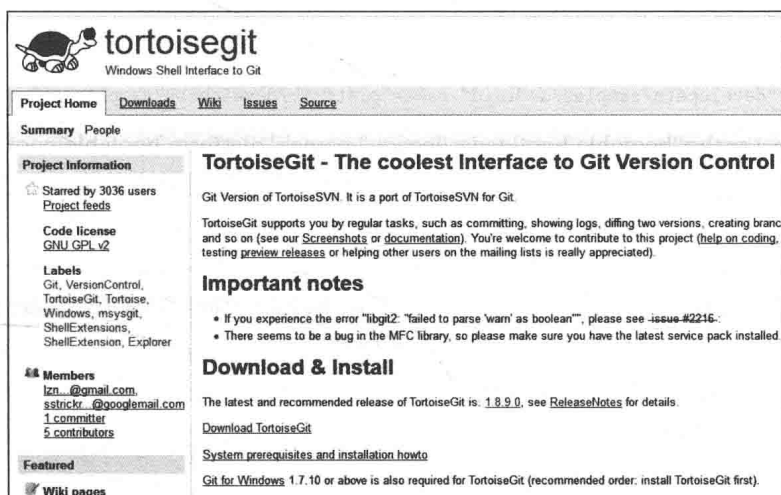


图 1-14 下载 TortoiseGit 工具

下载后双击下载后的可执行文件进行安装，安装过程按照默认选项安装即可。如果读者对英文不敢兴趣，可以下载 TortoiseGit 的中文版本，笔者安装的就是 TortoiseGit 中文版。

(3) 新建一个保存源码的文件夹，例如“cc”，在文件夹上单击右键，然后选择“Git 克隆”命令。

(4) 在弹出的“Git 克隆”对话框界面中，在“URL”后面的文本框中输入 Android L 项目下载路径：<https://android.googlesource.com/platform/manifest.git>，如图 1-15 所示。

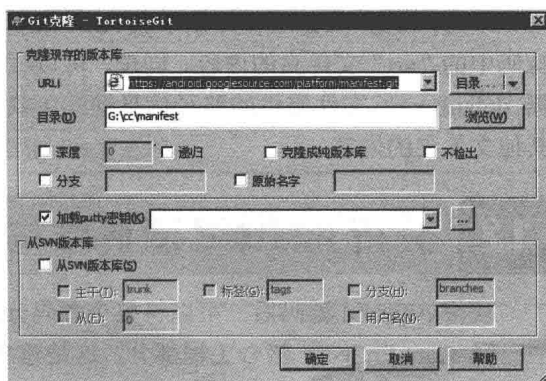


图 1-15 输入 Android L 项目下载路径

(5) 单击图 1-15 中的“确定”按钮开始下载分支信息文件，下载后的文件被保存在“cc”文件夹中，具体目录结构如图 1-16 所示。



图 1-16 分支信息文件

(6) 打开文件 default.xml，在里面列出了 Android L 源码的各个文件夹中子文件夹目录的分支信息，具体格式如下所示：

```
<project path="abi/cpp" name="platform/abi/cpp" groups="pdk"/><project path="art" name="platform/art"/>
<project path="bionic" name="platform/bionic" groups="pdk"/>
<project path="bootable/bootloader/legacy" name="platform/bootable/bootloader/legacy"/>
<project path="bootable/diskinstaller" name="platform/bootable/diskinstaller"/>
<project path="bootable/recovery" name="platform/bootable/recovery" groups="pdk"/>
<project path="cts" name="platform/cts" groups="cts"/>
<project path="dalvik" name="platform/dalvik"/>
<project path="developers/build" name="platform/developers/build"/>
<project path="developers/demos" name="platform/developers/demos"/>
<project path="developers/docs" name="platform/developers/docs"/>
<project path="developers/samples/android" name="platform/developers/samples/android"/>
```

例如“<project path="bootable/bootloader/legacy" name="platform/bootable/bootloader/legacy"/>”表示在 Android L 的源码中，存在了一个名为“bootable”的根目录文件夹，而在“bootable”文件夹中又包含了一个名为“bootloader”的子文件夹，而在“bootloader”文件夹下又包含了一个名为“legacy”的文件夹。

(7) 开始下载 Android L 源码，在文件夹“cc”上单击右键，然后选择“Git 克隆”命令。在弹出界面中输入 Android L 某个分支的下载路径，例如 path=“art”表示“art”，此文件夹的下载地址是：<https://android.googlesource.com/a/art.git>。然后勾选“分支”复选框，并在后面填写“1-preview”分支，如图 1-17 所示。

单击“确定”按钮后将开始下载 Android L 源码中的“art”文件夹的内容。同理，可以根据 default.xml 文件提供的路径信息继续下载其他文件夹的内容。

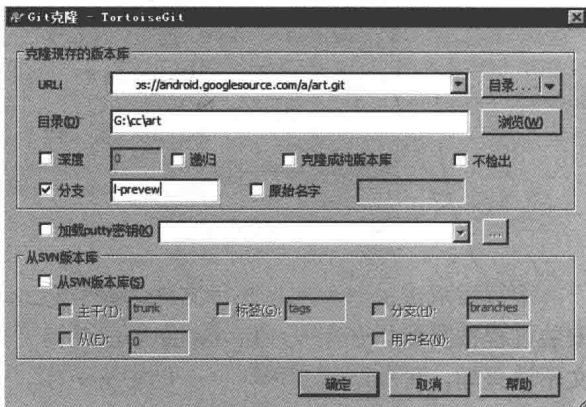


图 1-17 开始下载“art”文件夹的内容

## 1.2 分析 Android 源码结构

获得 Android 源码后，可以将整个源码分为如下 3 个部分。

□ Core Project: 核心工程部分，这是建立 Android 系统的基础，被保存在根目录的各个文件夹中。