Android Dalvik虚拟机结构及机制剖析

——第1卷 Dalvik虚拟机结构剖析

张国印 吴艳霞 编著

清华大学出版社

Android Dalvik虚拟机结构及机制剖析

——第1卷 Dalvik虚拟机结构剖析

张国印 吴艳霞 编著

清华大学出版社
北京

内容简介

本系列丛书共分2卷，本书为第1卷，是一本以情景方式对Android的源代码进行深入分析的书，内容广泛，主要从Dalvik虚拟机整体结构、获取和编译Dalvik虚拟机的源码、源码分析辅助工具使用、.dex文件及Dalvik字节码格式解析、Dalvik虚拟机下的系统工具介绍及Dalvik虚拟机执行流程简述等方面进行阐述，帮助读者从宏观上了解Dalvik虚拟机的架构设计，为有兴趣阅读Dalvik虚拟机源码的读者提供必要的入门指导。

第1卷共6章：第1章为准备工作，在这一章中主要介绍了Dalvik虚拟机的功用、分析Dalvik源码所用到的主要方法以及如何搭建Dalvik源码分析环境；第2章为源码分析辅助工具介绍，包括Vim、Doxygen、GDBSERVER等；第3章为Dex文件以及Dalvik字节码格式分析；第4章为系统工具介绍，在这一章中主要介绍了Dalvik虚拟机的一些重要系统工具，通过对系统工具的介绍，让读者对虚拟机内部的实现机制更加清晰；第5章为Dalvik虚拟机执行流程简述，通过这一章的介绍，旨在让读者对Dalvik虚拟机的整体功能架构有一个宏观的认识，为后续进一步掌握各个功能模块的原理功能做好相应的知识铺垫；第6章为调试支撑模块，在这一章中主要介绍了调试支撑模块的基本原理。

通过阅读本书，让读者了解Dalvik虚拟机在Android应用程序运行过程中所扮演的重要角色及其不可替代的价值；同时对Android应用程序的执行过程有更加细致的了解，可以帮助读者优化自己编写的应用程序，更加合理地设计应用程序结构，有效提高应用程序的运行速度。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

Android Dalvik虚拟机结构及机制剖析. 第1卷　Dalvik虚拟机结构剖析/张国印，吴艳霞编著. --北京：清华大学出版社，2014
ISBN 978-7-302-36103-9

Ⅰ. ①A…　Ⅱ. ①张…　②吴…　Ⅲ. ①移动终端－应用程序－程序设计－虚拟处理机－研究
Ⅳ. ①TP338

中国版本图书馆CIP数据核字(2014)第069719号

责任编辑：袁勤勇　薛　阳
封面设计：傅瑞学
责任校对：李建庄
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：7.5　　**字　　数**：187千字
版　　次：2014年11月第1版　　**印　　次**：2014年11月第1次印刷
印　　数：1～2000
定　　价：25.00元

产品编号：057014-01

前言

随着移动互联网的不断发展，业务移动化已逐渐被人们接受，移动电子商务、移动办公、移动生活越发深入人心。作为目前市场占有率最高的 Android 操作系统，当之无愧受到广大程序开发人员的青睐。Android 是由 Google 公司基于移动设备而开发的嵌入式系统，具有优良的性能表现以及较低的硬件配置需求，因此使其迅速成为目前移动终端之上的主流操作系统。这种优势的体现主要得益于 Google 对作为 Android 系统基石的 Dalvik 虚拟机所做出的大量优化。对于高阶程序开发人员来说，要想让自己开发的应用程序在数十万应用程序中脱颖而出，就必须掌握整个 Android 系统运行时环境，这其中最为关键的就是 Dalvik 虚拟机。

本书详细地介绍了 Dalvik 虚拟机的结构及其运行机制，尤其针对类数据加载、内存管理、本地方法、反射机制、解释器、即时编译等关键功能模块的设计原理、功能架构以及执行流程进行了介绍，并结合关键代码加以细致讲解。力求让读者了解 Dalvik 虚拟机是如何在底层对 Android 应用程序进行解释执行，并可以结合 Dalvik 虚拟机技术特性对自己的应用程序加以优化改进，以达到进一步提高应用程序安全性、稳定性、高效性的目的。

全书共分为 6 章：

第 1 章为准备工作，主要介绍 Dalvik 虚拟机的定义以及它的功用，分析 Dalvik 源码所用到的主要方法以及如何搭建 Dalvik 源码分析环境。

第 2 章为源码分析辅助工具介绍，主要介绍一些辅助源码分析的工具，包括 Vim、Doxygen、GDBSERVER，并介绍了其使用的方法，为后期的阅读和分析打下基础。

第 3 章为 Dex 文件以及 Dalvik 字节码格式分析，主要介绍 Dex 文件中所涉及的各个数据结构以及相关函数的具体定义，并结合一个 Dex 文件实例对原理内容进行讲解。同时还对 Dalvik 字节码进行了全面的介绍，主要包括字节码设计、字节码格式等内容。另外，在这章的最后还对 Dex 文件的优化产物 Odex 文件功能原理与实际应用进行了简单的介绍，为后续进一步深入讨论 Dex 文件的优化机制做好相关准备。

第 4 章为系统工具介绍，主要介绍 Dalvik 虚拟机的一些重要系统工具，这些工具主要应用于 Dex 文件优化，封装的 apk 文件进行或对 Dex 进行反编译，调试分析 Android 程序源码内存泄漏问题，分析 Android 程序运行过程中生成的 trace 文件等。通过对系统工具的介绍，让读者更清楚虚拟机内部的实现机制。

第 5 章为 Dalvik 虚拟机执行流程简述，主要介绍 Dalvik 虚拟机的整体执行流程以及各个模块所扮演的功能角色。通过这一章的介绍，旨在让读者对 Dalvik 虚拟机的整体功能架构有一个宏观的认识，为后续进一步掌握各个功能模块的原理功能做好相应的知识铺垫。

第 6 章为调试支撑模块，主要介绍调试支撑模块的基本原理，随后，着重介绍 DDM 协

议、JDWP 协议、Debugger 调试器三者的原理及实现，以帮助读者更加清晰地理解调试支撑这部分内容。

本书主要由哈尔滨工程大学张国印、吴艳霞编写，参与本书编写和校核工作的还有汪永峰、王彦璋、谢东良、于成、张婷婷、许圣明、苗施亮、檀凯，这里对他们的辛苦工作表示衷心的感谢。

本书主要是针对高级 Android 应用开发工程师、Android 系统开发工程师、Android 移植工程师及对 Android Dalvik 虚拟机源码实现感兴趣的读者。

作　者

2014 年 6 月

试读结束：需要全本请在线购买： www.ertongbook.com

目录

第1章　准备工作 …… 1
1.1　本章概述 …… 1
1.1.1　什么是 Dalvik 虚拟机 …… 1
1.1.2　Dalvik 虚拟机的功能 …… 3
1.1.3　Dalvik 虚拟机与 Java 虚拟机的区别 …… 6
1.1.4　Dalvik 虚拟机的特性 …… 7
1.2　Ubuntu Linux 系统安装 …… 8
1.3　工作目录设置 …… 11
1.4　下载、编译和运行 Android 内核源代码 …… 12
1.4.1　下载 Android 内核源代码 …… 12
1.4.2　整体编译 Android 源代码 …… 15
1.4.3　运行 Android 模拟器 …… 16
1.5　编译经过修改的 Android 源码 …… 17
1.6　开发第一个 Android 应用程序 …… 17
小结 …… 21

第2章　源码分析辅助工具 …… 22
2.1　本章概述 …… 22
2.2　Vim 源码阅读环境搭建 …… 22
2.3　Doxygen 工具 …… 25
2.4　GDBSERVER 工具 …… 29
小结 …… 32

第3章　Dex 文件及 Dalvik 字节码格式解析 …… 33
3.1　本章概述 …… 33
3.2　Dex 文件格式 …… 34
3.2.1　Dex 文件中的数据结构 …… 34
3.2.2　Dex 文件结构分析 …… 35
3.3　Dalvik 字节码介绍 …… 46
3.3.1　Dalvik 字节码总体设计 …… 46

3.3.2 Dalvik 字节码指令格式 …… 47
3.4 Odex 文件简介 …… 48
3.4.1 什么是“优化文件” …… 49
3.4.2 Odex 文件结构 …… 49
3.4.3 Odex 文件加速系统运行速度 …… 51
3.4.4 手机“减负”问题再讨论 …… 51
小结 …… 52

第 4 章 系统工具 …… 53

4.1 本章概述 …… 53
4.2 dexdump 工具 …… 54
4.2.1 dexdump 工具简介 …… 54
4.2.2 dexdump 工具使用方法 …… 54
4.3 dexdeps 工具 …… 64
4.3.1 dexdeps 工具简介 …… 64
4.3.2 dexdeps 工具使用方法 …… 64
4.4 dexlist 工具 …… 67
4.4.1 dexlist 工具简介 …… 67
4.4.2 dexlist 工具使用说明 …… 67
4.5 dexopt 工具 …… 72
4.5.1 dexopt 工具简介 …… 72
4.5.2 dexopt 工具使用方法 …… 72
4.6 dvz 工具 …… 73
4.6.1 dvz 工具简介 …… 73
4.6.2 dvz 工具使用方法 …… 73
小结 …… 74

第 5 章 开发分析工具 …… 75

5.1 本章概述 …… 75
5.2 trace 文件分析工具 …… 75
5.2.1 trace 文件分析工具简介 …… 75
5.2.2 trace 文件分析工具使用方法 …… 76
5.3 Heap Profile 工具 …… 78
5.3.1 Heap Profile 工具简介 …… 78
5.3.2 Heap Profile 工具使用方法 …… 79
5.4 DDMS 工具 …… 83
5.4.1 启动 DDMS …… 84
5.4.2 DDMS 原理和特性 …… 86
5.4.3 DDMS 具体功能 …… 86

5.4.4 进程监控 …… 87
5.4.5 使用文件浏览器 …… 90
5.4.6 模拟器控制 …… 91
5.4.7 应用程序日志 …… 92
小结 …… 93

第6章 Dalvik 虚拟机执行流程详解 …… 94

6.1 本章概述 …… 94
6.2 Dalvik 虚拟机的入口点介绍 …… 95
6.2.1 Dalvik 虚拟机在 x86 平台运行的入口点 …… 95
6.2.2 Dalvik 虚拟机运行在 ARM 平台的入口点 …… 96
6.2.3 Dalvik 虚拟机的初始化 …… 97
6.3 Zygote 进程 …… 97
6.4 Dalvik 虚拟机运行应用程序过程 …… 109
6.4.1 apk 文件生成 …… 109
6.4.2 Dalvik 虚拟机运行应用程序的主要流程 …… 109
小结 …… 111

第1章

准备工作

本章主要内容

- 你知道 Dalvik 虚拟机吗?
- 开发者有必要了解 Dalvik 虚拟机吗?
- 如何分析 Dalvik 虚拟机源码?
- 如何搭建源码分析环境?

随着移动互联网的不断发展,业务移动化已逐渐被人们接受,移动电子商务、移动办公、移动生活越发深入人心。智能手机间的竞争已从硬件的竞争中逐渐走出,取而代之的是操作系统的竞争。作为目前市场占有率最高的 Android 操作系统,受到广大程序开发人员的青睐。对于程序开发人员来说,如果能使自己开发的应用程序在数十万应用程序中脱颖而出,无疑是对自身能力最好的肯定。要想达到这种水平,必须深入理解应用开发的各个细节,不仅包括用户体验,还包括代码的质量和性能。想要提高 Android 应用程序的执行效率,就一定要深入理解 Android 应用程序是如何执行的,并且对于整个 Android 系统运行时环境的学习也是十分必要的,这就不得不提到 Dalvik 虚拟机。

1.1 本章概述

当你翻开这本书时,想必一定多次见到过如图 1.1 所示的 Android 系统架构图,在 Android 运行时环境部分(Android Runtime)可以找到接下来将要介绍的 Dalvik 虚拟机(Dalvik Virtual Machine)。

1.1.1 什么是 Dalvik 虚拟机

Dalvik 虚拟机是 Google 等厂商合作开发的 Android 移动设备平台的核心组成部分之一。Dalvik 由 Dan Bornstein 编写,名字来源于他的祖先曾经居住过的名叫 Dalvik 的小渔村,村子位于冰岛。

很多人认为 Dalvik 虚拟机就是一个 Java 虚拟机,因为 Android 的编程语言恰恰就是 Java 语言。但是这种说法并不准确,因为 Dalvik 虚拟机并不是按照 Java 虚拟机的规范来实现的,两者并不兼容,对于这一点后面还会介绍。

那么到底什么是 Dalvik 虚拟机呢?首先,它是一个虚拟机,也就是一个虚构出来的计算机,是通过在实际的计算机上仿真模拟各种计算机功能来实现的。它有自己完善的硬件架构,如处理器、堆栈、寄存器等,还具有相应的指令系统。第二,这台虚拟出来的计算机主要负责运行 Android 应用程序,它是 Android 应用程序中 Java 代码的运行基础。其指令集

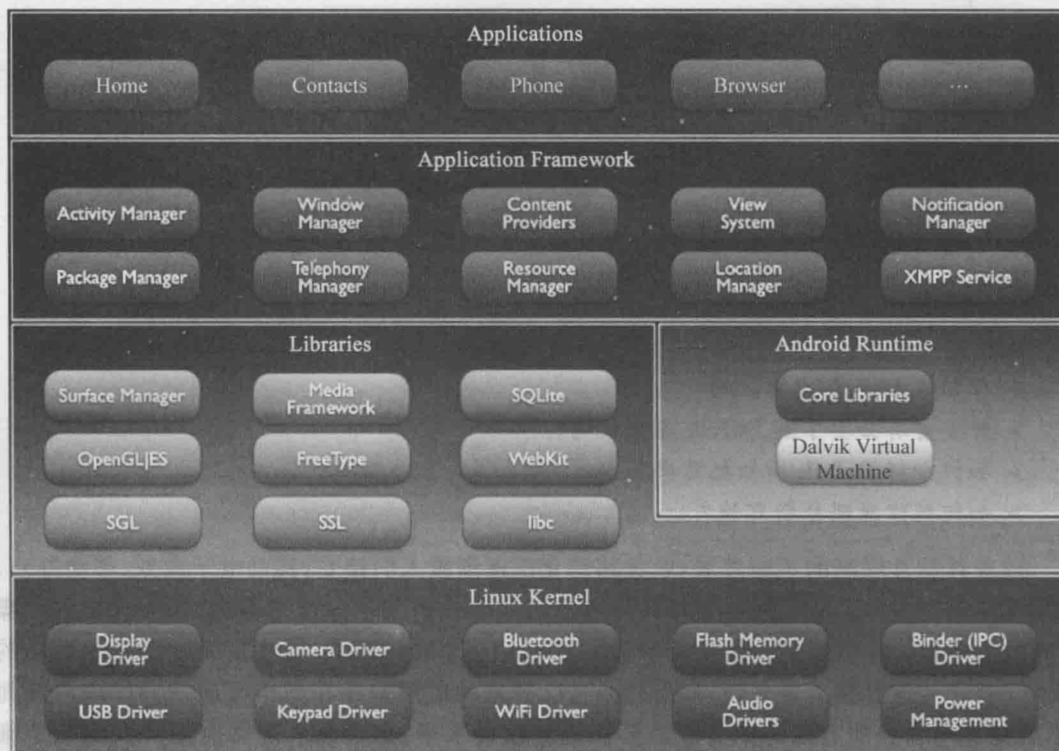

图 1.1 Android 系统架构图

基于寄存器架构，执行其特有的文件格式——Dex 字节码来完成对象生命周期管理、堆栈管理、线程管理、安全异常管理、垃圾回收等重要功能。它的核心内容是实现库(libdvm. so)，大体由 C 语言实现。依赖于 Linux 内核的一部分功能——线程机制、内存管理机制、能高效使用内存以及在低速 CPU 上表现出的高性能。每一个 Android 应用在底层都会对应一个独立的 Dalvik 虚拟机实例，其代码在虚拟机的解释下得以执行。

Android 系统架构可以分为 4 层，分别是应用程序层、应用程序框架层(Framework)，核心库层与 Dalvik VM 以及 Linux 内核。Dalvik VM 和核心库在同一层级，作为承上启下的重要一环。所有 Android 应用程序都运行在 Dalvik VM 之上，如果 Android 应用程序需要调用核心库中的库函数，Dalvik VM 将调用本地接口(Native Interface)并执行核心库中的函数。在 Dalvik VM 之上运行的程序或应用是跨平台的，但 Dalvik VM 是和操作系统及硬件相关的。其依赖操作系统掌管的一些功能，如线程调度、内存管理等。和操作系统与底层硬件相关一样，Dalvik VM 的解释器和 JIT 都因硬件的不同而有不同的实现，如 ARM 系列、MIPS 以及 Intel X86 等平台都对应有不同的实现。

作为 Android 平台至关重要的中间件，Dalvik VM 的输入是经过 dx 工具打包好的 Dex 文件，输出是程序执行结果。图 1.2 以 Hello.java 为例，给出了一个普通应用程序在 Dalvik 虚拟机中的执行流程。

Google 推荐 Android 应用程序使用的编程语言是 Java。如果编写性能要求高的程序，也可使用 C/C++。Dalvik VM 在 API 上和 Oracle 公司的 Java API 是兼容的，减少了应用程序开发难度。编写好的 Java 程序可以直接用 PC 上的 Java 编译器编译为 class 文件。

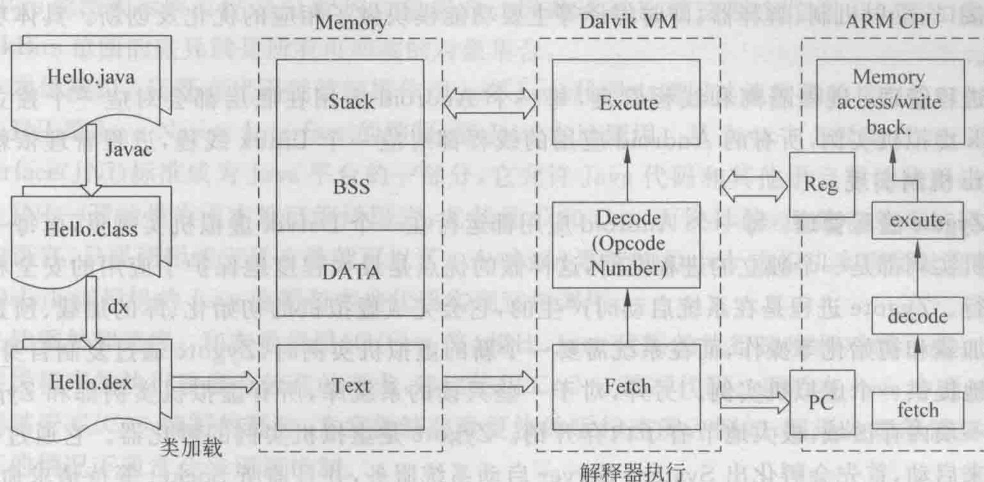

图 1.2 Dalvik VM 执行应用程序流程图

之后，需要使用 Dalvik VM 提供的 dx 工具对其进行转换。Dalvik VM 最初是基于 ARM 这个 RISC 架构设计的，和传统的基于栈的 Java 虚拟机不同，其是基于寄存器的。基于寄存器的虚拟机相比于基于栈的虚拟机，有更少的 losd/store 之类的访存类指令，也就是更少的内存访问和指令分派次数。根据统计，相比于 JVM，在 Dalvik VM 上运行的应用程序要少 47% 的指令数。dx 工具解析 class 文件，合并多个 class 文件，转换为基于寄存器的字节码，并优化字节码，最终生成 Dex 文件。生成的 Dex 文件将作为 Dalvik VM 的输入。

Dalvik VM 启动并初始化后，Dex 文件将被映射到内存区，解释器开始将 Dex 文件中的每一条字节码解释为本地代码并运行。如图 1.2 所示，解释器的工作流程和真实 CPU 的工作原理非常相像，都包括取指、解码以及执行。具体的实现原理较为简单，以一个循环来完成一条字节码的解释工作。Dalvik 解释器从内存中取得字节码，并对字节码进行解码（获取字节码号），之后跳转到对应的代码段执行。无论是 C 语言编写的解释器还是汇编语言编写的解释器，每一条字节码都有一段与之等价的最终执行。如果有 JIT 的支持，JIT 将编译热代码，并将编译后的 Native Code 安装至内存区。再次执行时，解释器将跳转至相应 Native Code 执行，这将大幅度提高执行速度。

点拨 Dalvik 虚拟机的设计者 Dan Bornstein 的个人主页是 http://www.milk.com/home/danfuzz/。Dalvik 虚拟机源码目录下的 docs 目录中也有对虚拟机进行相关介绍的文档，可予以参考。

1.1.2 Dalvik 虚拟机的功能

1.1.1 节中已经简要介绍了 Dalvik 虚拟机的功能，概括来说，Dalvik 虚拟机主要完成对象生命周期的管理、堆栈的管理、线程管理、安全和异常的管理以及垃圾回收等重要功能。在 Dalvik 设计的过程中充分利用了 Linux 进程管理的特点，使其可以同时运行多个进程，这就使得在 Android 系统上可以同时运行多个应用程序，每一个应用程序都对应后台一个独立的虚拟机进程。

Dalvik 虚拟机考虑到运行环境资源相对紧张的特点，对线程管理、类加载、内存管理、

试读结束：需要全本请在线购买： www.ertongbook.com

本地接口、反射机制、解释器、即时编译等主要功能模块做了相应的优化及创新。具体功能如下。

进程管理：进程隔离和线程管理，每一个 Android 应用在底层都会对应一个独立的 Dalvik 虚拟机实例，所有的 Android 应用的线程都对应一个 Linux 线程，进程管理依赖于 Zygote 机制实现。

Zygote 线程管理：每一个 Android 应用都运行在一个 Dalvik 虚拟机实例里，而每一个虚拟机实例都是一个独立的进程空间，这样做的优点是最大程度地保护了应用的安全和独立运行。Zygote 进程是在系统启动时产生的，它会完成虚拟机的初始化、库的加载、预置类库的加载和初始化等操作，而在系统需要一个新的虚拟机实例时，Zygote 通过复制自身，最快速地提供一个虚拟机实例。另外，对于一些只读的系统库，所有虚拟机实例都和 Zygote 共享一块内存区域，极大地节省了内存开销。Zygote 是虚拟机实例的孵化器。它通过 init 进程来启动，首先会孵化出 System_Server 启动系统服务，并且监听 Socket 等待请求命令，当有一个应用程序启动时，Zygote 会调用相应函数 fork 出一个新的进程来执行应用程序。Zygote 进行 fork 时有以下三种不同的方式。

(1) fork()，fork 一个普通的进程，该进程属于 Zygote 进程。

(2) forkAndSpecialize()，fork 一个特殊的进程，该进程不再是 Zygote 进程。

(3) forkSystemServer()，fork 一个系统服务进程。

类加载：解析 Dex 文件并加载 Dalvik 字节码。

对 Android 源码经过 Android SDK 编译后生成的 APK 文件进行一系列的处理，再在展开的 APK 文件中找到 classes. dex 文件并从 Dex 文件中加载 Dalvik 字节码供虚拟机执行模块调用。

类加载器在虚拟机中负责查找并加载字节码文件，即通过提取二进制的字节码文件，并将其存入该类的运行时数据结构，供解释器执行。在加载目标类时，还需要将该类的所有超类和超类实现的接口，需要加载的类分为基础类库和用户自定义类。当虚拟机装载某个类型时，类加载器会定位相应的字节码文件，然后读入这个字节码文件，提取其中的数据信息，并将这些信息存储到对应的内存中。

Android 系统启动时，类加载器会加载所有基础类库，用户自定义类是在虚拟机运行时才载入的。当虚拟机在运行时需要调用的一个成员方法或者一个成员变量所属的类没有被解析的时候，虚拟机会调用类加载模块，对这个类、超类以及这些类的相关接口进行加载和连接。

内存管理：分配系统启动初始化和应用程序运行时需要的内存资源。

Dalvik 虚拟机内存管理分为内存分配和垃圾回收。内存分配的底层依赖是基于 Doug Lea 编写的 dlmalloc 内存分配器，在 Heap 上完成，按照分配规则，每分配一个内存区域经过数次尝试。如果分配不成功，就启动垃圾收集按照相应策略进行垃圾收集。Dalvik 虚拟机在垃圾回收时使用 Mark Sweep 算法，该算法一般分为 Mark 阶段和 Sweep 阶段。Mark 阶段就是标记出活动对象，使用栈来保存根集合，然后对栈中的每一个元素，递归追踪所有可访问的对象，对于所有可访问的对象，在 markBits 位图中该将对象的内存起始地址对应的位设为 1。这样当栈为空时，markBits 位图就是所有可访问的对象集合。垃圾收集的第二步就是回收内存，在 Mark 阶段通过 markBits 位图可以得到所有可访问的对象集合，而

liveBits位图表示所有已经分配的对象集合。因此通过比较这两个位图，liveBits位图和markBits位图的差异就是所有可回收的对象集合。

本地接口：让既有代码继续发挥作用。在Java代码中调用其他代码的接口。

JNI是Java Native Interface的缩写，即Java本地调用。从Java 1.1开始，Java Native Interface(JNI)标准成为Java平台的一部分，它允许Java代码和其他语言写的代码进行交互。JNI一开始是为了本地已编译语言，尤其是C和C++而设计的，但是它并不妨碍使用其他语言，只要调用约定受支持就可以了。Android系统的Dalvik虚拟机实现了这套接口，供Dalvik虚拟机的Java应用与本地代码实现互相调用。

注重处理速度：和本地代码(C/C++等)相比，Java代码的执行速度相对慢一些。如果对某段程序的执行速度有较高的要求，建议使用C/C++编写代码。而后在Java中通过JNI调用基于C/C++编写的部分，常常能够获得更快的运行速度。例如，图形处理等需要大量计算的情况下通常会采用该机制。

直接进行硬件控制：为了更好地控制硬件，硬件控制代码通常使用C语言编写。而后借助JNI将其与Java层连接起来，从而实现对硬件的控制。Dalvik虚拟机使用一些本地代码编写的已编译的代码库与硬件、操作系统直接进行交互。

对既有本地代码的复用：在程序编写过程中，常常会使用一些已经编写好的本地代码(如C/C++代码)，既提高了编程效率，又确保了程序的安全性与健壮性。在复用这些本地代码时，就要通过JNI本地调用接口来实现。

从整体来看，使用JNI主要在于可以直接重用一些数量庞大的本地代码，对于那些性能要求比较高的代码而言，通过JNI机制使用本地代码可以增强程序的性能，减少功耗，特别是对于内存较小、CPU运算速度有限和电池电量不够充沛的嵌入式移动手机设备而言，提高性能减少功耗这一点就显得尤为重要。然而，使用JNI调用接口也会带来一些负面影响，比如有些时候失去了平台的可移植性，但是从综合角度考虑，在有些情况下，JNI机制带来的优点要大于它的缺点。

反射机制：能动态查看、调用、更改任意类中的方法和属性，并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。

反射机制是Dalvik虚拟机中的核心机制之一，也算作一类工具，合理地使用反射机制能使Java代码变得更加简洁、灵活。同时，反射机制是Java被当作准动态语言的一个关键性质。反射机制允许程序在运行的过程中通过反射机制的API取得任何一个已知名称的类的内部信息，包括其中的描述符、超类，也包括属性和方法等所有信息，并且可以在程序运行时改变属性的相关内容或调用其内部的方法。反射机制在实现其功能时首先通过上层应用API运用JNI本地调用机制调用本地方法集中的函数，再向下层调用Dalvik虚拟机中的内部函数，最后将结果逐层返回到最上层的应用。

解释器：根据自身的指令集Dalvik ByteCode解释字节码。

解释器是Dalvik虚拟机的执行引擎，它负责解释执行Dex字节码。在Android 4.04版本的虚拟机中，解释器共有两种实现，分别是C语言实现和汇编语言的实现，分别称作可移植型(Portable)解释器和快速型(Fast)解释器。在字节码加载已经完毕后，Dalvik虚拟机解释器开始取指解释字节码。解释器根据指令进行相应的操作，然后返回相应的结果。

在mterp目录下有一个重要的部分，即out目录，存储的是针对各个平台的解释器程序

和C语言实现的通用解释器。在解释器执行时，仿照真实机器执行，分别有取指、执行过程。在每个操作码的解释程序完成后，就取下一条指令，并跳转执行，以提高效率。解释器的入口代码位于 interp 目录下的 Interp.cpp 中的 dvmInterpret 函数，在该函数中，根据系统参数的不同，会分别选择 Fast 解释器和 Portable 解释器来执行 dvmMterpStd 和 dvmInterpretPortable。

执行引擎是虚拟机的核心，负责执行字节码或者本地方法。Dalvik 虚拟机主要采用解释执行的方式，Android 4.04 版本的虚拟机同时支持 JIT 编译器技术。

解释执行方式是指虚拟机在执行过程中将每一条字节码指令解释成本地代码运行，其工作原理比较简单，字节码的解释过程是一个循环结构，每次循环完成一条字节码的执行工作：取指令、执行功能和跳转。简言之，解释执行方式就是把字节码指令的功能用特定平台上的语言来实现。解释执行是利用该平台的资源，不需要进行附加的硬件设计，利用软件来实现虚拟机的方式。

即时编译：将反复执行的热代码编译成本地码，降低解释器压力。

JIT 技术是将字节码编译成本地代码执行，当某一个方法第一次被调用时，JIT 编译器将对虚拟机方法表所指向的字节码进行编译，编译后表中的指针将指向编译生成的机器码，如果程序再次执行该方法时，将执行经过编译的代码，提高了执行速度。

众所周知，程序执行有两种方式，分别为解释和编译。解释方式是在逐句读取源程序逐句翻译成机器码再执行；而编译方式则是在运行程序前，整个翻译为等价的目标程序，计算机直接执行该目标程序。JIT(Just-In-Time)，中文含义为即时编译，又称为动态编译，执行时动态地编译程序，以缓解解释器的低效工作。JIT 混合了两种技术，解释器解释时，编译部分程序，并在下次直接执行该编译后的源程序。

对于 Java 这类语言来说，不论是解释器还是 JIT 模块，都是在中间代码基础上做文章。Java 语言编译后，生成和平台无关的中间代码。不同平台上的虚拟机(JVM)都可以执行相同的 Java 中间代码，同时需要处理和操作系统和硬件平台相关的部分。这也是大多数解释型语言采用的模型。虚拟机中最主要的是解释器，负责解释执行中间代码。

虽有跨平台的优势，但同时也带来了运行效率低的直观感受。究其原因，是因为其执行原理是一句一句翻译字节码。比如，字节码中出现循环，根据解释器的原理，它需要重复地解释并执行这一组程序。这使得纯粹基于解释器运行的程序效率十分低下，造成了浪费。

JIT 是解决这个缺陷的一种有效手段。通过将字节码编译为 Native Code，让解释器不再重复执行这些热点代码片段。而且，相比于解释器，JIT 编译器可以更高效地利用 CPU 和寄存器。同时在编译的过程中，可以进行部分低级代码优化，比如常数传播、取消范围检查、复制传播等。尽可能地生成媲美编译器编译的二进制代码。之后执行编译生成的 Native Code，从而达到加速执行应用程序的目的。

1.1.3 Dalvik 虚拟机与 Java 虚拟机的区别

Dalvik 虚拟机并不是按照 Java 虚拟机的规范来实现的，二者并不兼容；二者的最大差异是架构上的差异，即 Dalvik 虚拟机的设计是基于寄存器的，Java 虚拟机的设计是基于栈的。

那为何Java虚拟机选择基于栈的设计，而Dalvik虚拟机选择基于寄存器的设计呢？这和Dalvik运行的硬件环境有关。众所周知，Dalvik运行于手持设备之上，手持设备大多采用的是ARM或是相似的RISC结构的CPU，这些硬件有个特点，相对来说，RISC架构有更丰富的寄存器。如果Dalvik的字节码也采用RISC架构设计，在中间语言这一层，就有丰富的寄存器。这就使得在解释时，避免了过多的访存指令。

Dalvik虚拟机和Java虚拟机另外一个显著的区别是两个虚拟机上运行的文件格式不同，前者具有自己专有的文件格式(.dex)，后者则是字节码文件(.class)。在Java程序中，Java类会被编译成一个或多个字节码文件，然后打包为.jar文件，Java虚拟机从相应的.class和.jar文件中获取相应的字节码。Android应用程序也是用Java语言编写的，但在编译成.class文件后，还会通过dx工具将所有.class文件统一封装成一个.dex文件，Dalvik虚拟机从中读取指令和数据。

点拨　掌握Dalvik虚拟机与Java虚拟机的联系和区别对于学习Dalvik虚拟机是非常有帮助的，目前除源码目录中的一小部分技术文档外，官方并没有给出关于Dalvik虚拟机的技术文档，建议在学习Dalvik虚拟机前类比Java虚拟机来学习，相信可以达到事半功倍的效果。其官方(The Java® Virtual Machine specification)的网址为http://docs.oracle.com/javase/specs/jvms/se7/html/index.html。

1.1.4　Dalvik虚拟机的特性

Dalvik虚拟机非常适合在移动终端上使用，相对于在桌面系统和服务器系统运行的虚拟机而言，它不需要很快的CPU速度和大量的内存空间。根据Google的测算，64MB的RAM已经能够令系统正常运转了。其中24MB被用于底层系统的初始化和启动，另外20MB被用于启动高层服务。当然，随着系统服务的增多和应用功能的扩展，其所消耗的内存也势必越来越大。

Android是由Google公司基于移动设备而开发的嵌入式系统，具有优良的性能表现以及较低的硬件配置需求，因此使其迅速成为目前移动终端之上的主流操作系统。这种优势的体现主要得益于Google对作为Android系统基石的Dalvik虚拟机所做出的大量优化。实际上，Dalvik虚拟机并不是一个标准的Java虚拟机，因为它不符合Java虚拟机设计规范。Dalvik虚拟机是一个针对嵌入式系统中低速CPU和内存受限等特点，经过专门设计优化而实现的Java语言虚拟机。

Dalvik虚拟机是基于寄存器架构的，相比基于堆栈的标准Java虚拟机，基于寄存器设计的Dalvik虚拟机的操作指令更长，因此用于实现相同程序的指令数则更少，同时对于较大的程序，其花费的编译时间也更少。研究表明，寄存器架构虚拟机比起堆栈虚拟机，相同功能程序字节码减少了47%；代码总量增加25%；内存访问次数减少37.42%；程序整体执行时间减少32.30%。Dalvik虚拟机使用专用的ByteCode字节码，在Android 2.1中，共有218个字节码，分布在Ox00与Oxff之间，中间保留了38个无效字节码。字节码的存储方式为16位比特的无符号数。同时，为了能在资源受限的嵌入式系统中实现更高的性能，Dalvik虚拟机使用专门定制的Dex文件作为可执行文件。Dex文件是对多个Class文件进行高效整合的产物，使各个类共享相同的数据，减少了冗余性，结构十分紧凑。

1.2 Ubuntu Linux 系统安装

在系统安装前，需要获取一个版本为 Ubuntu 10.04 的系统镜像，该镜像大小为695MB，目前网络上的资源很多且下载速度较快，读者自行下载即可。在获取了安装程序之后，需要根据需求选择安装方式。

安装方式主要分为两种：

① 在本机上直接安装 Ubuntu 系统；

② 在虚拟机中安装 Ubuntu 系统。

两种方式各有利弊，简单来说，在本机上直接安装会较少地占用系统资源，系统的工作效率较高，适合机器配置较低的用户，但对于不常用 Linux 系统的用户来说可能会有较多的不便；在虚拟机中安装相对来说更加安全且简便，尤其对于长期使用 Windows 的用户来说，不会干扰其习惯的工作模式，但在虚拟机中运行 Ubuntu 系统对机器的性能要求较高，建议机器配置较高的用户选择这种安装方式。

点拨 Ubuntu 10.04 的硬件门槛很低，但为了保证学习的顺利，建议读者参照如下标准配置设备。①处理器：尽量选用 Inter I 系列 64 位多核处理器，或者其他支持 64 位虚拟化技术的处理器。②内存：4GB 以上。③硬盘：至少 60GB。④图形性能不做要求。

下面通过图示简单介绍一下系统的安装流程。

(1) 将烧录好的系统盘插入计算机，并启动计算机，正确读取系统盘后会出现如图 1.3 所示的界面。

图 1.3 系统启动

这里需要指出，选择直接安装的用户可以将系统镜像烧录到 DVD 光盘或是 U 盘中，以制成系统启动盘。关于启动盘的制作方法，网上有大量的资料，在此就不再赘述；选择虚拟机安装的用户可以直接加载系统镜像，其后安装步骤和直接安装完全相同。

(2) 经过一段时间的等待，就正式开始系统的安装了。首先，需要对系统语言、键盘布局以及时区等基本信息进行设定，如图 1.4 所示。

(3) 完成相关的设定之后，将要为目标系统选择相应的安装位置，可以直接选择将系统安装到一块硬盘中，也可以通过分区工具对目标硬盘进行分区，并将系统装入指定分区中，如图 1.5 所示。

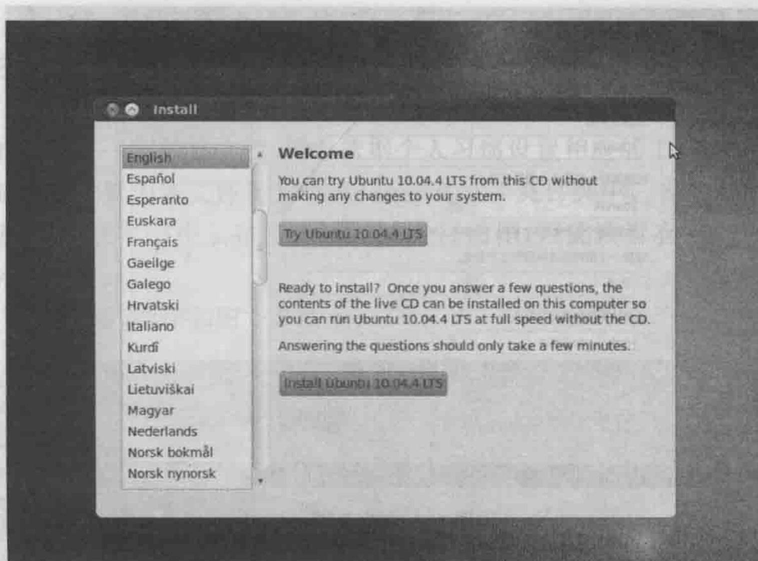

图1.4 语言、键盘及时区设置

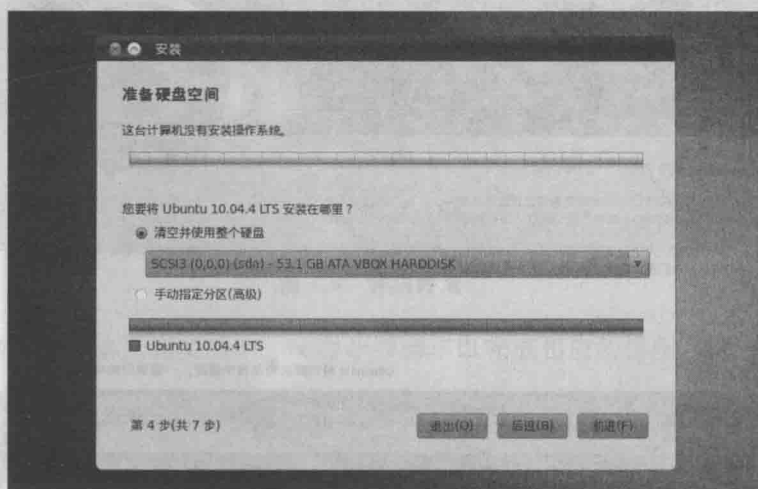

图1.5 系统分区设置

(4) 在这个界面中，需要对用户名、计算机名以及密码等信息进行设定。值得注意的是，密码一定要认真记录，在Linux操作系统中经常需要输入密码以获取操作权限，如图1.6所示。

(5) 在完成用户名、密码等信息设定之后，系统将开始自动安装，如图1.7所示。

(6) 系统安装结束后会提示将重新启动，在启动完毕后正确输入之前设定的用户名以及密码则可以登录到系统，如图1.8所示。

至此，Ubuntu系统的安装就简要介绍完毕。这也完成了Android源码调试分析的第一步工作。

试读结束：需要全本请在线购买： www.ertongbook.com