计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(7):246−251 [doi: 10.15888/j.cnki.csa.006996]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

安卓应用中无声音频的收集与检测①

颜宏冰, 熊　焰, 黄文超, 孟昭逸

(中国科学技术大学 计算机科学与技术学院, 合肥 230027)

摘　要: 在安卓系统中, 一些安卓应用为了避免被系统杀死, 会通过各种方式在后台占用系统的 CPU, 内存等资源, 实现后台保活. 这类行为会加速安卓系统的电量消耗. 其中一种后台保活的方式是在后台持有 Audiomix 锁并播放无声音频. 针对这种行为, 本文设计了相应的方案来检测这个问题. 通过对安卓源码进行修改, 收集到安卓应用正在播放的音频数据, 再通过检测脚本对音频进行实时检测, 来判断安卓应用是否在后台播放无声音频来实现保活. 实验分析了 50 个安卓应用, 结果表明该方法可以有效检测此类行为.

关键词: 后台保活; 音频播放; 脉冲编码调制; 音量调节; 安卓系统

引用格式: 颜宏冰,熊焰,黄文超,孟昭逸.安卓应用中无声音频的收集与检测.计算机系统应用,2019,28(7):246–251. http://www.c-s-a.org.cn/1003-3254/6996.html

Collection and Detection of Soundless Audio in Android Applications

YAN Hong-Bing, XIONG Yan, HUANG Wen-Chao, MENG Shao-Yi

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: In Android system, in order to avoid being killed by the system, some Android applications will occupy the system's CPU, memory and other resources in the background in various ways to achieve background live preservation. This kind of behavior accelerates Android's power consumption. One way to keep alive in the background is to hold Audiomix lock in the background and play silent audio data. Considering this kind of behavior, we designs corresponding schemes to detect this problem. By modifying the Android system source code, we collect the audio data that Android applications are playing, and then use the program to check whether the Android application keeps alive in the background by playing silent audio data in realtime. We analyze 50 Android applications in the experiment, and the results show that this method can detect such behaviors effectively.

Key words: background live preservation; audio playing; pulse coding modulation; volume regulation; android system

目前安卓手机的市场份额越来越大, 2017 年已经到达了 85.9%. 如此大的市场份额也意味着巨大的利益, 很多厂商都进入了这个市场, 这使得安卓应用的数量和安卓手机的功能迅速增长. 现在安卓手机的计算能力, 存储能力, 通信能力已经达到一个相当高的地步, 甚至已经能和个人 PC 相媲美. 但安卓手机性能的上升是以电池电量的加速消耗为代价的, 而电池电量的增长速度跟不上手机性能的提升, 这导致了现在手机的待机时间越来越短.

为了解决这个问题, 许多研究都尝试去优化安卓的电量消耗的情况. 方葵[1]考虑到手机开启数据流量时耗电比平时高, 设计了关闭屏幕时关闭数据流量的软件; Martins 等[2]对安卓系统进行了修改, 降低耗电事件的发生频率; Hoffmann 等[3]则是将应用中的一些静态配

① 基金项目: 国家自然科学基金 (61520106007)
Foundation item: National Natural Science Foundation of China (61520106007)
收稿时间: 2019-01-17; 修改时间: 2019-02-03; 采用时间: 2019-02-26; csa 在线出版时间: 2019-07-01

置参数改成可以调节的动态参数, 达到节省电量的目的.

这些方案在一定程度上优化了安卓的耗电问题, 但安卓系统中仍有很多的耗电问题亟待解决. 目前一些安卓应用为了在后台长期保活, 会在后台占用系统资源. 其中一种典型情况就是在后台持有 Audiomix 锁并播放一些无声数据. 这种行为会加快手机的电量消耗, 同时由于占用了系统资源, 还可能导致手机运行速度变慢, 所以这种情况下应当终止应用的后台占用. 但在安卓系统中, 音频数据的收集并不容易, 虽然安卓提供了音频播放的函数, 但并没有提供音频收集相关的函数. 同时很多安卓应用并不使用安卓系统提供的音频播放函数, 而是自己实现了音频播放的函数. 针对这个问题, 本文提出了一种收集并检测安卓中的音频数据的方法.

1 背景知识

1.1 安卓系统架构

安卓系统是在 Linux 内核的基础上构建的, 为了满足移动设备的需求, 安卓系统引入了新的组件. 如图 1 所示, 安卓系统主要分为 4 层, 从上到下分别是应用程序层、应用程序框架层、安卓原生库和运行时层、Linux 内核层[4,5].

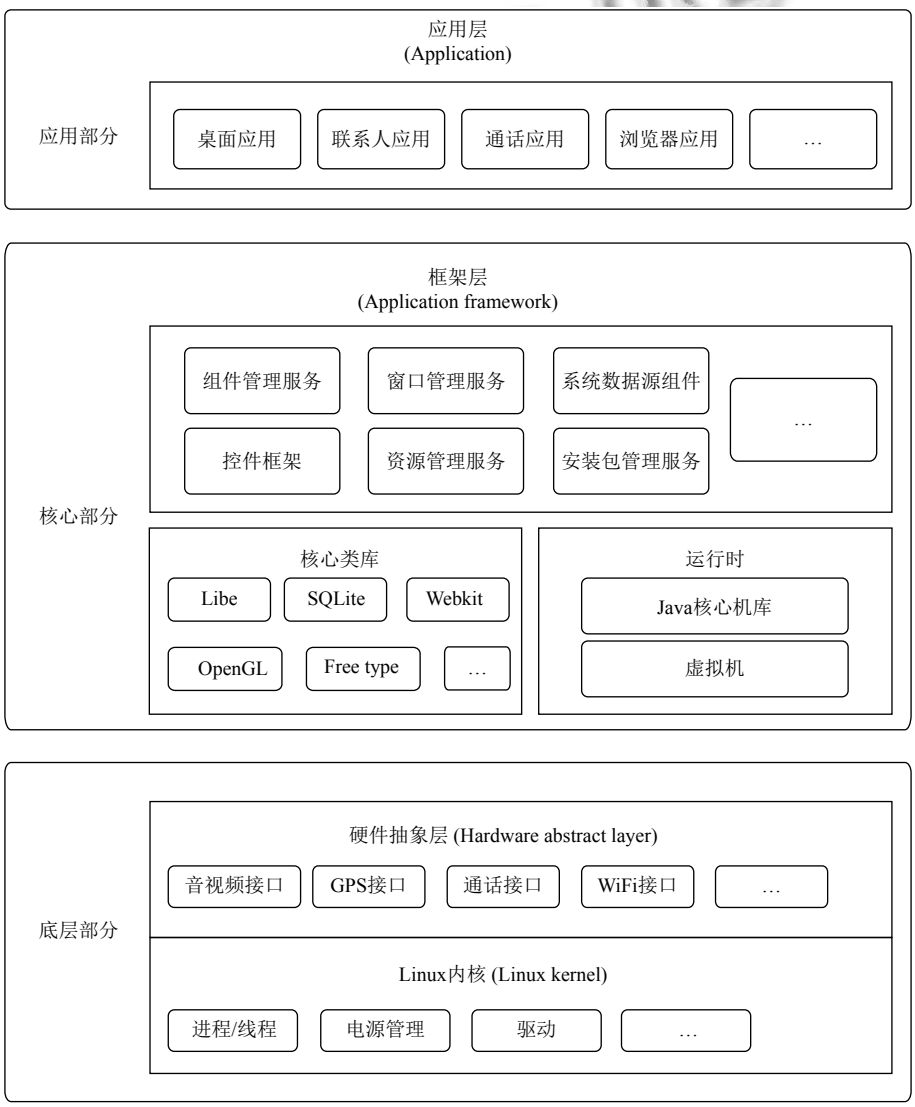

图 1 安卓系统架构

最上层的应用层由直接和用户交互的应用组成, 包括系统应用比如日历、浏览器、联系人管理程序和普通应用比如微博, 微信, 这些应用都是 Java 语言开发的.

下一层的应用程序框架层主要包括了能被移动应用重用和共享的一系列系统服务, 包括 View System(构建安卓应用的 UI, 包括列表, 网格, 文本框等等), Resource Manager(访问非代码资源, 如本地化的字符串, 布局文件) 等.

第三层中的安卓原生库基本是由 C/C++编写的, 负责特定的硬件设备结合或者响应 UI 的输入请求, 比如 OpenGL 和 SGL, 分别负责 3D 和 2D 图形的渲染. 安卓运行时则包括虚拟机和核心 Java 库, 安卓 5.0 之前使用的是 Dalvik 虚拟机, 5.0 之后则使用更快的 ART 虚拟机, 其作用与 Java 虚拟机类似, 但运行的是.dex 格式的文件. 核心库则提供了 Java 编程语言核心库的大多数功能, 比如文件访问, 网络访问等.

最下层的 Linux 内核层则是整个安卓系统的基础, 安卓系统的功能最终都是通过 Linux 内核完成. 它提供了硬件设备的抽象接口, 以供上层使用.

1.2 安卓音频播放

Audio 是整个安卓系统非常重要的一个组成部分, 负责音频数据的采集和输出、音频流的控制、音频设备的管理、音量调节等, 整体结构如图 2 所示.

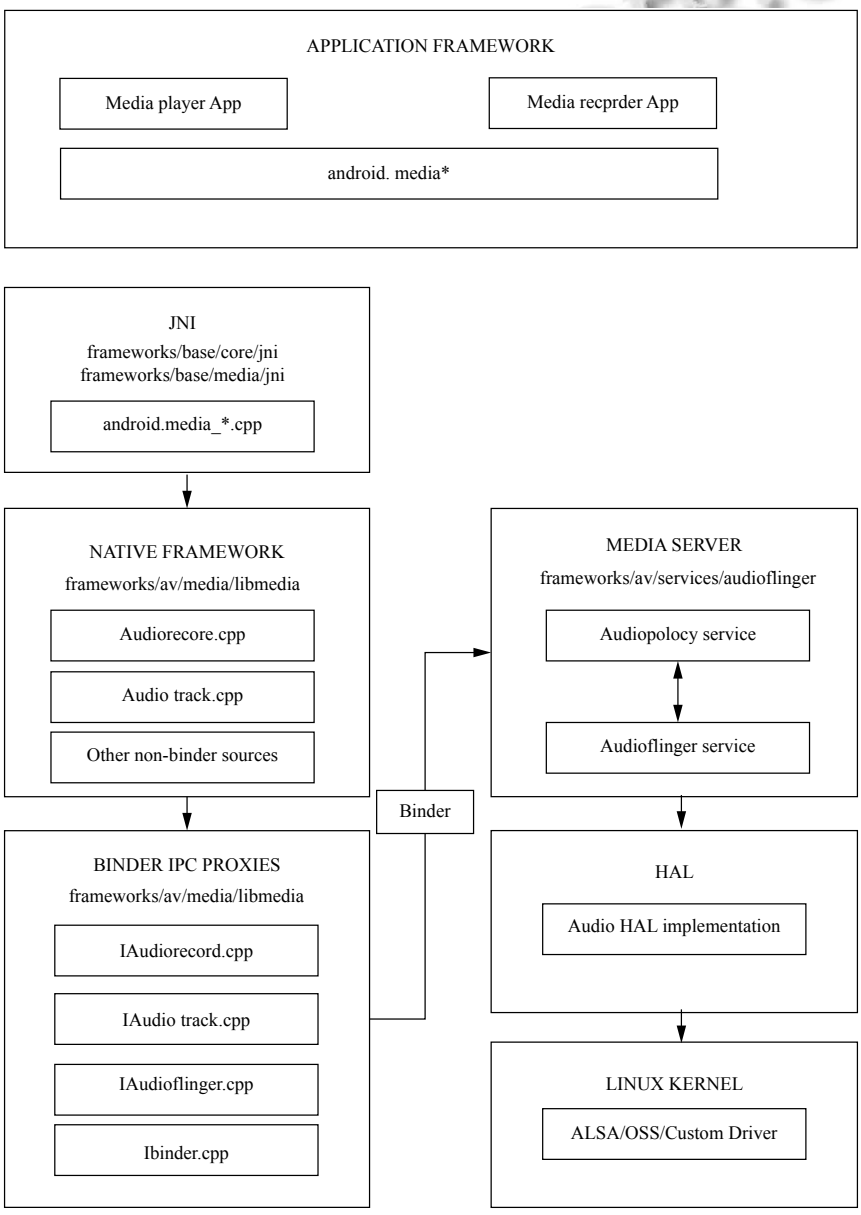

图 2 安卓音频系统结构

Audio 主要包括以下几个部分: 音频应用框架 (Audio Application Framework), 音频本地框架 (Audio Native Framework), 音频服务 (Audio Service), 音频硬件抽象层 (Audio HAL). 其中, 音频应用框架和音频本地框架负责音频数据的播放和采集, 以及音频事务的综合管理. 音频服务则由 AudioPolicyService 和 AudioFlinger 组成, 前者负责制定音频策略, 比如音频设备切换的策略选择, 音量调节等. 后者则负责执行策略, 比如输入输出流设备的管理及音频流数据的处理传输. 硬件抽象层则负责与音频硬件设备交互, 由 AudioFlinger 直接调用.

安卓中音频播放主要有两种方式: 第一种是使用 MediaPlayer 播放, 第二种是使用 AudioTrack 播放. 这两种播放方式最大的区别是 MediaPlayer 会在应用程序框架层创建对应的音频解码器, 对播放的源文件进行解码, 故 MediaPlayer 可以播放多种格式的声音文件, 例如 MP3, AAC, WAV, OGG, MIDI 等. 而 AudioTrack 不创建解码器, 故只能播放已经解码的 PCM 流, 其对应的文件格式是 WAV 格式的音频文件. MediaPlayer 和 AudioTrack 之间还是有联系的, MediaPlayer 在应用程序框架层仍需要创建 AudioTrack, 把解码后的 PCM 数据流传递给 AudioTrack, AudioTrack 再传递给 AudioFlinger 进行混音, 然后才传递给硬件播放.

脉冲编码调制 (PCM) 是一种编码方式, 一般用于对连续变化的模拟信号进行抽样、量化和编码产生数字信号. 安卓中的音频一般都是 PCM 编码的.

PCM 文件的格式如图 3, 根据采样位数分为 8 位, 16 位, 32 位, 根据声道数分为单声道和双声道. 文件中每个样本值包含在一个数中, 根据位数的不同, 这个数占 1、2 或者 4 个字节, 并且其表示的范围也有区别, 比如 8 位的 PCM 文件每个样本值的范围是 0 到 255. 位数越多, 表示声音的音质越高.

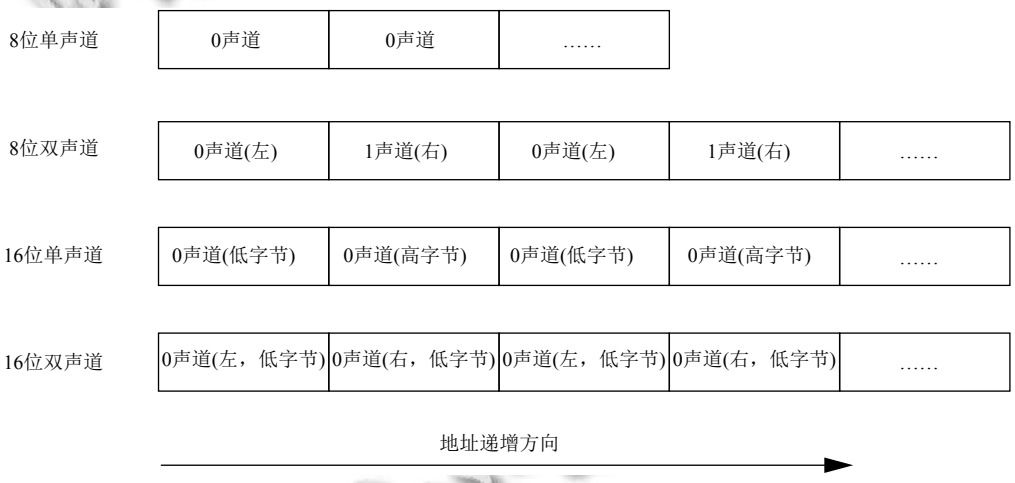

图 3 PCM 文件格式

2 方案设计

为了达到收集并检测安卓系统中无声音频的目的, 有以下两个挑战: 第一, 安卓系统中没有现成的收集音频数据的接口或函数, 并且安卓应用可能不使用安卓应用程序框架提供的音频函数, 需要找到一个合适的位置插入代码才能收集数据; 第二, 对于无声音频, 没有一个确定的识别标准, 需要根据实验结果制定一个标准. 针对以上问题, 本文设计了一个收集和检测无声音频的系统, 如图 4 所示.

图中的安卓系统经过修改后, 可以收集到安卓应用播放的 PCM 音频数据, 这些数据再被送到一个系统应用中进行检测, 这个系统应用中包含了自定义的检测脚本, 用来判断音频数据是否是无声的. 因为需要实时检测音频数据, 所以将检测脚本做成系统应用集成在安卓系统中. 经过系统应用检测后, 结果将返回给用户. 接下来就音频数据收集和音频数据检测做详细介绍.

2.1 音频数据采集

安卓中没有可以直接收集音频的函数, 为了能收集安卓应用播放的音频, 需要对安卓的源码进行相应修改. 但实验中发现, 如果在 Java 层添加代码来收集音频数据, 约有 40% 的安卓应用是收集不到音频数据的. 通过进一步分析, 这些应用没有使用安卓系统中

Java 层的 MediaPlayer 来播放音频, 而是在自定义的动态链接库中实现了相关的函数, 进一步调用底层的 Native 函数播放音频.

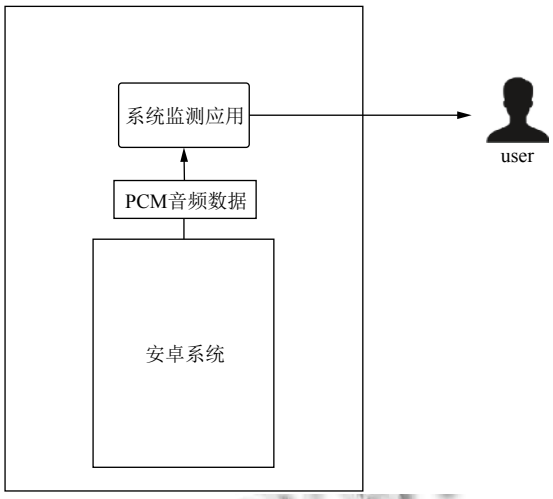

图 4 音频收集和检测系统框架

通过对安卓系统源码以及这些安卓应用的分析发现, 虽然它们实现的播放音频的函数细节上有区别, 但是最终都是由 Audiotrack 调用 write 函数或者 processAudioBuffer 函数, 向共享内存中写入需要播放的音频数据, 再传递到 AudioFlinger 进行播放. 所以只需要在 write 函数和 processAudioBuffer 函数中插入截取音频数据的代码, 就可以采集到相应的音频数据. 此外, 为了将音频数据与应用建立对应关系, 还需要收集播放此音频数据的安卓应用的相关信息, 故在 AudioTrack 中添加相应的代码将应用的信息也保存下来.

2.2 音频数据识别

收集到安卓应用播放的音频数据后, 需要对音频数据进行识别, 判断其是否是无声的音频数据. 但 PCM 音频数据是否无声这个问题没有具体的判别标准, 因为 PCM 音频数据描述的是时域振动, 本文考虑的情况是如果 PCM 音频数据波动的数据点占总数的比例小于某个阈值时, 就认为 PCM 数据是无声的音频数据. 根据此规则, 可以对 PCM 文件进行解析, 并判断文件中包含的是否是无声的音频数据.

由于有些安卓应用会将音量调节为 0, 故即使音频数据不是无声的音频数据, 还要进行进一步的检测. 这里采用的方式是追踪函数序列. 安卓系统在调节音量时, 会产生特定的函数调用序列. 在安卓 7.0 系统中, 当用户按下音量侧键调节音量时, 系统会先调用 frameworks/base/core/java/com/android/internal/policy/PhoneWindow.java 中的 onKeyDown() 函数, 在 onKeyDown() 函数中, 又调用了./base/media/java/android/media/session/MediaSessionLegacyHelper.java 中的 MediaSessionLegacyHelper.getHelper(), 通过 getHelper 静态函数实例化 MediaSessionLegacyHelper, 再调用其成员函数 sendAdjustVolumeBy(), 在 sendAdjustVolumeBy() 函数中通过 binder 机制调用./base/services/core/java/com/android/server/media/MediaSessionService.java 中 MediaSessionService 的成员函数 dispatchAdjustVolume(), 最后在此函数中间接调用了 adjustSuggestedStreamVolume(). 而如果应用直接调节音量, 则不会产生这个函数调用的序列. 因此, 只要追踪是否有此函数调用序列, 便可判断应用是否在用户未知的情况下对音量进行了调节.

3 实验分析

实验使用的安卓手机是 Google Nexus 6p, 安卓系统的版本是 7.0. 实验中做为检测样本的安卓应用来自于豌豆荚应用市场.

3.1 安卓应用选取

考虑到要收集安卓应用中的音频数据, 检测的应用主要是游戏和音乐这两个分类下的, 因为这两个分类下的安卓应用可以比较好的收集音频数据.

3.2 实验结果分析

实验共检测了 50 个安卓应用. 实验中发现, 音频文件的长度以及检测音频数据脚本中设置的 PCM 数据波动的数据点占总数的比例阈值对于最后的实验结果都会产生影响, 故实验中通过设置不同的值来寻找最合适的参数值.

表 1 是设置收集的音频文件的长度不同值时的检测结果. 对于表中所列的每个音频长度, 实验中对每个应用都随机截取了三次音频数据, 综合三次分析的结果, 来判断音频数据是否是无声的.

表 1 音频长度与检测正确率

音频长度 (秒)	正确率 (%)
5	82
10	92
15	100
20	100
25	100

可以看到, 当音频长度为 5 秒时, 检测的正确率比较低. 原因是音频播放时可能会出现某一小段音频恰好没有声音的情况, 如果截取的音频刚好是这一小段中的一部分, 就可能会导致错误的判断. 这种情况下, 虽然截取的音频文件时无声的, 但实际上就整个音频来看的话还是要判断为不是无声的. 为了减少这种情况的发生, 应该尽可能截取比较长的音频文件. 从表中数据能得出结论: 截取的音频文件的长度越长, 检测的正确率越高. 因为音频文件越长意味着其囊括的范围越大, 这样更能反映音频的真实情况.

表 2 是设置 PCM 数据波动的数据点占总数的比例阈值为不同的值时得到的结果.

表 2 界定阈值和正确率

阈值 (%)	正确率 (%)
0.1	74
1	84
5	100
10	96
20	90

可以看到, 当阈值设置的很低, 比如 0.1% 时, 检测的正确率也会比较低. 当阈值升高, 检测的正确率也会提升, 但阈值过大时, 又会出现正确率下降的情况. 分析音频数据发现, 阈值很低时, 如果音频数据波动的数据点集中在某一段, 而数据点数目又不足以产生声音, 这种情况下无声的音频数据就可能被识别为有声的. 而当阈值很高时, 如果有声音的音频数据段比较短, 其包含的数据点数目占总体数据点的比值达不到阈值, 就会将有声的音频数据误判为无声的.

因为检测脚本以系统 Service 的形式运行在后台, 经测试, 修改后的系统对用户使用体验没有影响.

4 结论与展望

本文主要关注的问题是安卓中无声音频的收集和检测, 因为安卓系统本身并未提供音频数据的收集手段, 并且对于无声音频也没有确定的衡量标准, 故本文通过修改安卓源码, 并调节无声数据检测脚本中的相关参数, 解决了安卓中无声数据的收集和检测问题. 在实验中发现, 仅凭后台播放无声数据这一点还不能确定安卓应用是否是恶意在后台占用系统资源来实现后台保活的, 故接下来的研究工作是要收集更多安卓应用运行时的相关信息, 结合本文实验的实验结果, 提高检测的准确率.

参考文献

1 方葵. 基于 Android 系统网络耗电量优化方法的研究. 通信技术, 2012, 45(10): 33–35. [doi: 10.3969/j.issn.1002-0802.2012.10.013]

2 Martins M, Cappos J, Fonseca R. Selectively taming background android apps to improve battery lifetime. Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference. Berkeley, CA, USA. 2015. 563-575.

3 Hoffmann H, Sidiroglou S, Carbin M, *et al*. Dynamic knobs for responsive power-aware computing. ACM SIGARCH Computer Architecture News, 2011, 39(1): 199–212. [doi: 10.1145/1961295]

4 赵静. Android 系统架构及应用程序开发研究. 自动化与仪器仪表, 2017, (1): 86–87, 90.

5 黄吉华. Android 系统架构研究与应用. 电子技术与软件工程, 2016, (7): 49.