Android系统：整体

1 综合

1.1 ++

--

Android系统概述

提到Android,很多人可能以为它只是一个写APP应用的工具，只能开发手机上运行的应用程序，实际上Android是一种基于Linux的自由及开放源代码的操作系统。基于Android操作系统平台，可以快速开发功能强大的各种终端设备，例如手机、平板、智能家电、汽车智能座舱、无人机、门禁系统等等。

--

Android代号

从2009年5月开始，Android操作系统改用甜点来作为版本代号，这些版本按照从C大写字母开始的顺序来进行命名：纸杯蛋糕(Cupcake)、甜甜圈(Donut)、闪电泡芙(Eclair))、冻酸奶(Froyo)、姜饼(Gingerbread)、蜂巢(Honeycomb)、冰淇淋三明治(Ice Cream Sandwich)、果冻豆(Jelly Bean)、奇巧(KitKat)、棒棒糖(Lollipop)、棉花糖(Marshmallow)、牛轧糖Nougat)、奥利奥(Oreo)、馅饼(Pie)。 各个版本的代号：

Android milestone builds (with Astro Boyand Benderfloating around in here somewhere)
Android1.0(没有开发代号)
Android 1.1 –Petit Four
Android 1.5–Cupcake
Android1.6–Donut
Android 2.0/2.1 –Eclair
Android 2.2–Froyo
Android 2.3–Gingerbread
Android 3.0/3.1/3.2–Honeycomb
Android 4.0–Ice Cream Sandwich
Android 4.1/4.2/4.3 Jelly Bean
Android 4.4–KitKat
Android 5.0/5.1 Lollipop(Android L)
Android 6.0–Marshmallow(Android M)
Android 7.0–Nougat(Android N)[1]
Android 8.0–Oreo(Android O)
Android 9.0–Pie (Android P)
Android 10.0–Android Q(不再使用甜品代号)
Android 11.0–Red Velvet Cake(Android R)
Android 12.0Snow Cone(Android S)

--

Android的演进

系统演进趋势：每个Android大版本的更新迭代前行，历经10余年，在用户体验、流畅性、续航、安全、隐私、机器学习等方面都取得较大的改进。图中是每个大版本中最具代表性的特征标记在图中，并不代表着该版本全部特征，同样专项计划也不是只在某一个版本执行，比如续航和性能优化，每一个版本都在持续改进中，Treble计划也一直在迭代至今。

- 从Android 1.0发展到Android 4.0，系统各项功能和特性迭代到一个较完善的阶段；
- Android 4.1系统，Google开展了黄油计划（Project Butter），为了让Android系统摆脱UI交互上的严重滞后感，希望能像“黄油”一样顺滑。 核心原理是系统框架中的渲染和动画统一采用垂直同步技术(VSYNC)，以及三重缓冲技术(Triple Buffer)，让滑动、翻页等操作更加一致与顺滑。
- Android 4.4系统，Google开展了瘦身计划（Project Svelte），力求降低安卓系统的内存使用，解决低端机型升级难的问题，让Android 4.4可正常运行在所有Android手机，从而减少安卓系统继续碎片化。UI设计上，支持新的“沉浸式模式”，用户界面由过去的黑色与蓝色为主的色调转向带有透明度的浅色系，视觉语言变得更加明亮与现代化。
- Android 5.0系统，Google开展了伏特计划（Project Volta），力求提升续航能力，这方面Google落后于业界厂商，厂商直面用户对续航尤为迫切，往往系统资源管控更为严格。另外，系统采用全新的ART，抛弃Dalvik虚拟机，大幅提升运行效率。UI设计上，使用全新的扁平化Material Design设计风格，更加清新与质感的设计，统一Android设备的外观和使用体验。
- Android 6.0系统，Google引入新的运行时权限，让用户能够更好地了解和控制权限；引入了Doze模式，进一步提升电池续航能力。UI设计上，新增夜间模式，大幅改进通知栏，让通知更简洁。
- Android 7.0系统，引入新的JIT编译器，对AOT编译器的补充，可节省存储空间和加快更新速度；进一步优化Doze唤醒机制；UI设计上，支持分屏功能；
- Android 8.0系统，Google开展了计划（Project Treble），重新架构Android，将安卓系统框架与Vendor层解耦，力求彻底解决安卓碎片化这一老大难的问题，这是安卓系统架构最大的变化。系统层面加强对后台服务、广播、位置的管控限制。UI设计上，改进通知栏，智能文本选择和自动填充功能。
- Android 9.0系统，引入神经网络API，采用机器学习的思路来预测用户使用习惯来做省电优化，继续强化Treble计划；文件系统(sdcardf/F2FS)持续提升；私有API的限制进一步规范化Android生态，强化隐私和安全，硬件安全性模块以及统一生物识别身份验证界面。 UI设计上，新的手势导航，加强支持刘海屏，UI搜索界面使用到机器学习，AI正在逐步强化Android系统。
- Android 10.0系统，Google开展了主线计划（Project Mainline），相关模块（Modules）不允许厂商直接修改，只能由Google应用商店来更新升级，强化用户隐私、系统安全与兼容性。支持脸部生物识别。

1.2 Android与嵌入式Linux的关系

——

Linux与android关系密切，具体来说有以下三点

- Android采用Linux作为内核
- Android对Linux内核做了修改，目的是适应其在移动设备上的应用
- Android内核一开始是作为Linux内核的一个分支，后来由于无法并入Linux内核的主开发树，已被Linux 内核组从开发树中剔除

——

Android是在Linux内核基础上运行的，提供的核心服务包括安全、内存管理、进程管理和驱动模型等。内核部分相当于介于硬件层和系统其他软件组之间的一个抽象层次。
因为Android内核是由标准Linux内核修改而来，所以很自然继承了Linux内核的很多优点，并保留了Linux内核的架构。Android本身有很多创新，按照移动设备的需求在文件系统、内存管理、进程间通信和电源管理方面进行了修改，根据需要添加了很多相关的新驱动和新功能。
总而言之，Android很大程度保留了Linux基本框架。

1.3 Android与Linux内核区别

Android系统层面的底层是Linux,并且在中间加上了一个叫做Dalvik的Java虚拟机，从表面层看是Android运行库。每个Android应用都运行在自己的进程上，享有Dalvik虚拟机为它分配的专有实例。为了支持多个虚拟机在同一设备上高效运行，dalvik被改写过。Dalvik虚拟机执行的是Dalvik格式的可执行文件（.dex)–该格式经过优 化，以将内存好用降到最低。 下面我们来看一下Android内核和Linux内核的一些主要差别：

--

Android Binder

Android Binder基于Openbinder框架，用于提供Android平台进程间的通信(IPC)机制，整个Android Binder的 实现包括Java层、Native层及驱动层。原来的Linux系统上层应用的进程间通信主要是D–bus,采用消息总线的方 式来进行IPC。Android binder驱动层源代码位于 drivers/staging/android/binder.c

--

Android电源管理(PM)

Android电源管理是一个基于标准Linux电源管理系统的轻量级Andorid电源管理驱动，针对嵌入式设备做了很 多优化。利用锁和定时器来切换系统状态，控制设备在不同状态下的功耗，以达到节能的目的。其源码位于 kernel/power/earlysuspend.c kernel/power/consoleearlysuspend.c kernel/power/fbearysuspend.c kernel/power/wakelock.c kernel/power/userwakelock.c

--

低内存管理器(Low memory Killer)

Android中低内存管理器和Linux标准的OOM相比，机制更加灵活，可以根据需要杀死进程来释放需要的内存 Low memory Killer的代码非常简单，里面关键是函数Lowmem shrinker().作为一个模块在初始化时调用 register_shrike注册一个Lowmen_shriker,它会被vm在内存紧张的情况下调用。源码位于 drivers/staging/android/lowmemorykiller.c

--

匿名共享内存(Ashmem)

匿名共享内存为进程间提供大块共享内存，同时为内核提供回收和管理这个内存的机制。如果一个程序尝试访问Kernel释放的一个共享内存块，它将会受到一个错误提示，然后重新分配内存并重载数据。其源码位于 mm/ashmem.c

--

Android PMEM(Phsical)

PMEM用于向用户空间提供连续的物理内存区域，DSP和某些设备只能工作在连续的物理内存上。驱动中提供 mmap、open/release和ioctl等接口。

--

Android Logger

Android Logger是一个轻量级的日志设备，用于抓取Android系统的各种日志，是Linux锁没有的

Android Alarm

Android Alarm提供了一个定时器用于把设备从睡眠状态唤醒，同时它也提供了一个即使在设备睡眠是也会运行的时钟基准。其源码位于 driver/rtc/alarm.c drivers/rtc/alarm-dev.c

---

USB Gadget驱动

此驱动是一个具有标准Linux USB gadget驱动框架的设备驱动，Android的USB驱动是基于gadget框架的。 其源码位于如下文件： drivers/usb/gadget/android.c drivers/usb/gadget/f_adb.c drivers/usb/gadget/f_mass_storage.c

---

Android Ram Console

为了提供调试功能，Android允许将调试日志信息写入一个被称为RAM Console的设备里，它是一个基于RAM的Buffer其源码位于drviers/staging/android/ram_console.c

---

Android timed device

Android timed device提供了对设备进行定时控制功能，目前仅仅支持vibrator和LED设备。其源码为 drviers/staging/adnroid/timed output.c

---

Yaffs.2文件系统

在Android系统中，采用Yaffs2作为MTD NAND FLASH文件系统。Yaffs2是一个快速稳定的应用于NAND和NOR FLash的跨平台的嵌入式设备文件性，同其他Flash文件系统相比，Yaffs2使用更小的内存来保存运行状态因此它占用内存小；Yaffs2的垃圾回收非常简单而且快速，因此能够达到更好的性能；其源代码位于fs/yaffs22目录。 需要注意的是，随着Android内核和Linux内核的快速发展，它们之间的差异也会在变化。一个功能的实现， 可能是由APP、Framework、Native层、驱动层等一起配合完成。

2 Android源码

2.1 Android源码查看

2.1.1 查看网站

Android Code Search

AOSPXRef

Opersys AOSP Portal

2.1.2 使用方法

OpenGrok search搜索方法

在此处可以进行一些源代码的搜索，不过其搜索体验就完全没有官方的网站好了。图中的各个搜索框对应功能如下：

字段	作用
Full Search	全文搜索，搜索索引中的所有文本标记（单词，字符串，标识符，数字）。搜索符号：覆盖符号的定义及使用，包括注释出现该符号
Definition	仅查找符号定义。例如搜索xx函数在哪些类中有定义
Symbol	仅查找符号。包括该符号的定义及使用位置
File Path	源文件的路径。搜索源码文件名中包含给定字符串的文件。（类级别）支持输入类名等。方法名通过前三种方式搜索。
History	历史记录日志注释。
Type	限制文件类型。

2.1.3 目录结构

> 源码基于android-12.0.0_r3

--

整体结构

各个版本的源码目录基本是类似，如果是编译后的源码目录会多增加一个out文件夹，用来存储编译产生的文件。

Android源码根目录	描述
art	Android Runtime，一种App运行模式，区别于传统的Dalvik虚拟机，旨在提高Android系统的流畅性
bionic	基础C库源代码，Android改造的C/C++库
bootable	Android程序启动导引，适合各种bootloader的通用代码，包括一个recovery目录
build	存放系统编译规则及generic等基础开发包配置
compatibility	Android兼容性计划
cts	Android兼容性测试套件标准
dalvik	Android Dalvik虚拟机相关内容
developers	Android开发者参考文档
development	Android应用开发基础设施相关
device	Android支持的各种设备及相关配置
external	Android中使用的外部开源库
frameworks	应用程序框架，Android系统核心部分，由Java和C++编写
hardware	硬件适配接口，主要是硬件抽象层的代码
kernel	Linux Kernel，不过Android默认不提供，需要单独下载，只有一个tests目录
libcore	Android Java核心类库
libnativehelper	Android动态库，实现JNI库的基础
out	编译完成后代码输出在此目录
packages	应用程序包
pdk	Plug Development Kit 的缩写，本地开发套件
platform_testing	Android平台测试程序
prebuilts	x86和arm架构下预编译的一些资源
sdk	Android的Java层sdk
system	Android底层文件系统库、应用和组件
test	Android Vendor测试框架
toolchain	Android工具链文件
tools	Android工具文件
Android.bp	Android7.0开始代替Android.mk文件，它是告诉ndk将jni代码编译成动态库的一个脚本
Makefile	全局Makefile文件，用来定义编译规则

--

应用层部分

应用层位于整个Android系统的最上层，开发者开发的应用程序以及系统内置的应用程序都是在应用层。源码根目录中的packages目录对应着系统应用层。

packages目录	描述
apps	核心应用程序
inputmethods	输入法目录
providers	内容提供者目录
screensavers	屏幕保护
services	通信服务
wallpapers	墙纸

--

应用框架层部分

应用框架层是系统的核心部分，一方面向上提供接口给应用层调用，另一方面向下与C/C++程序库以及硬件抽象层等进行衔接。 应用框架层的主要实现代码在/frameworks/base和/frameworks/av目录下，其中/frameworks/base目录结构如表所示。

/frameworks/base目录	描述
av	多媒体框架
base	Android源码的主要核心目录
compile	编译相关
ex	文件解析器
hardware	硬件适配接口
layoutlib	布局相关
minikin	Android原生字体，连体字效果
multidex	多dex加载器
native	native实现
opt	一些软件
proto_logging	
rs	Render Script，可创建3D接口
wilhelm	基于Khronos的OpenSL ES/OpenMAX AL的audio/multimedia实现

--

应用框架核心层部分

/frameworks/base目录	描述
apct-tests	性能优化测试
api	android应用框架层声明类、属性和资源
cmds	android系统启动时用到的commands

cmds	android系统启动时用到的commands
core	framework的核心框架组件
docs	android项目说明
drm	实现权限管理，数字内容解密等模块的工作
errorprone	
graphics	图像渲染模块
identity	
keystore	秘钥库
libs	库信息(界面、存储、USB)
location	位置信息
lowpan	
media	手机媒体管理(音频、视频等)
mime	
mms	
native	本地方法实现(传感器、输入、界面、窗体)
nfc-extras	近场通讯
obex	蓝牙
opengl	2D和3D图形绘制
packages	框架层的实现(界面、服务、存储)
proto	协议框架
rs	资源框架
samples	例子程序
sax	xml解析器
services	各种服务程序
startop	
telecomm	telecomm通信框架
telephony	电话通讯框架
test-base	
test-legacy	
test-mock	

test-runner	
tests	各种测试
tools	
wifi	wifi模块

--

C/C++程序库部分

系统运行库层（Native)中的 C/C++程序库的类型繁多，功能强大，C/C++程序库并不完全在一个目录中，这里给出几个常用且比较重要的C/C++程序库所在的目录位置。

目录位置	描述
bionic/	Google开发的系统C库，以BSD许可形式开源。
/frameworks/av/media	系统媒体库
/frameworks/native/opengl	第三方图形渲染库
/frameworks/native/services/surfaceflinger	图形显示库，主要负责图形的渲染、叠加和绘制等功能
external/sqlite	轻量型关系数据库SQLite的C++实现

讲完 C/C++程序库部分，剩下的部分我们在之前表已经给出：
Android运行时库的代码放在art/目录中。硬件抽象层的代码在hardware/目录中，这一部分是手机厂商改动最大的一部分，根据手机终端所采用的硬件平台会有不同的实现。