Android Anatomy and Physiology

By Google IO

Android 架构剖析和机能分析

翻译：JeefJiang

2009 年 8 月 14 日

· **Android Anatomy　Android 架构剖析**

· Linux Kernel　Linux 内核

· Native Libraries　本地库

· Android Runtime　Android 运行时

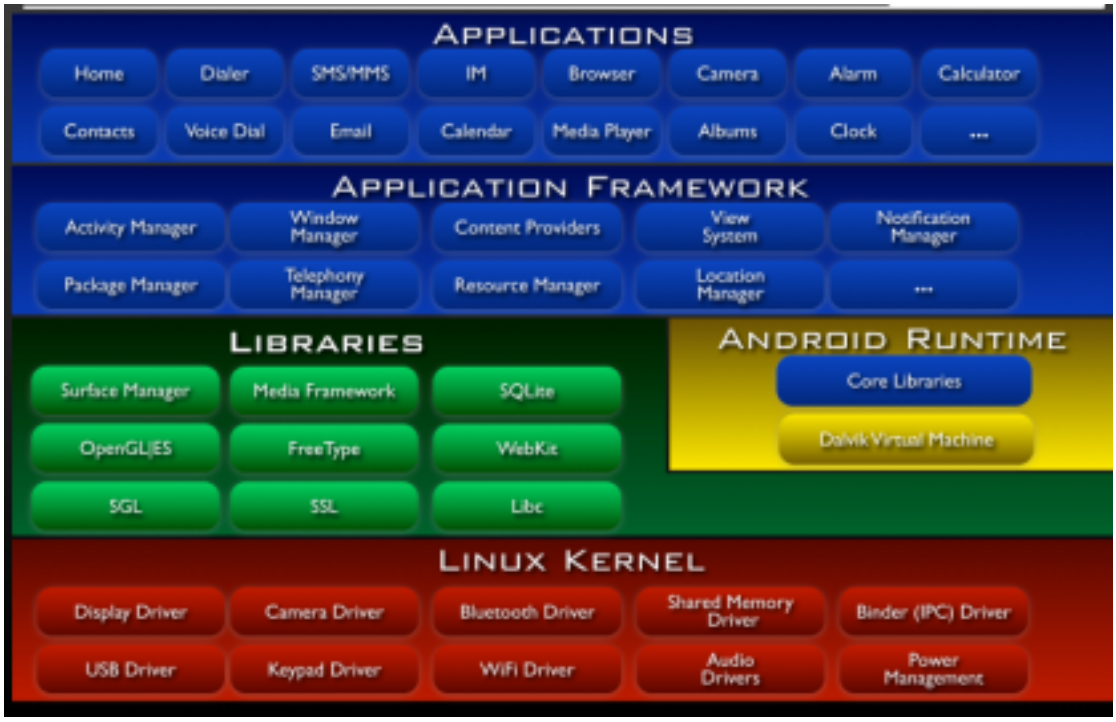
· Application Framework　应用程序框架

· **Android Physiology　Android 机能**

· Start-up Walkthrough　启动流程

· Layer Interaction　层次交互

译者注：Android Anatomy 是从静态的角度分析 Android 的架构，而 Android Physiology 是从动态的分析 Android 是如何启动以及各个层次是如何交互的，以下是 Android 的架构图。我们按照自底向下的方法来分析。



Android 架构——内核：

· Android is built on the Linux kernel, but Android is not Linux

Android 是基于 Linux 内核，当 Android 本身不是 Linux

· No native windowing system

没有本地的窗口系统

· No glibc support

没有 Glibc 的支持

· Does not include the full set of standard Linux utilities

没有包括完整的标准的 Linux 工具集

· Standard Linux 2.6.24 Kernel

标准的 Linux2.6.24 内核（笔者翻译时已经是 2.6.27）

· Patch of “kernel enhancements” to support Android

为了支持 Android 打上了内核优化补丁

为什么使用 Linux

· Great memory and process management 优越的内存和进程管理功能

· Permissions-based security model 基于权限的安全模式

· Proven driver model 被认可的驱动模式

· Support for shared libraries 支持共享库

· It s already open source! 代码开源

内核优化

为了支持 Android，Android 的主线内核做了相应优化，主要体现在下面几个方面：

· Alarm

· Ashmem

· Binder

· Power Management

· Low Memory Killer

· Kernel Debugger

· Logger

Binder

为什么要引入 Binder

· Applications and Services may run in separate processes but must communicate and share data

应用程序虽然以独立的进程运行，但相互之间需要通信

· IPC can introduce significant processing overhead and security holes

IPC 会增加进程的开销并导致安全漏洞

解决方法:

· Driver to facilitate inter-process communication (IPC) 用驱动程序来推进进程间通信

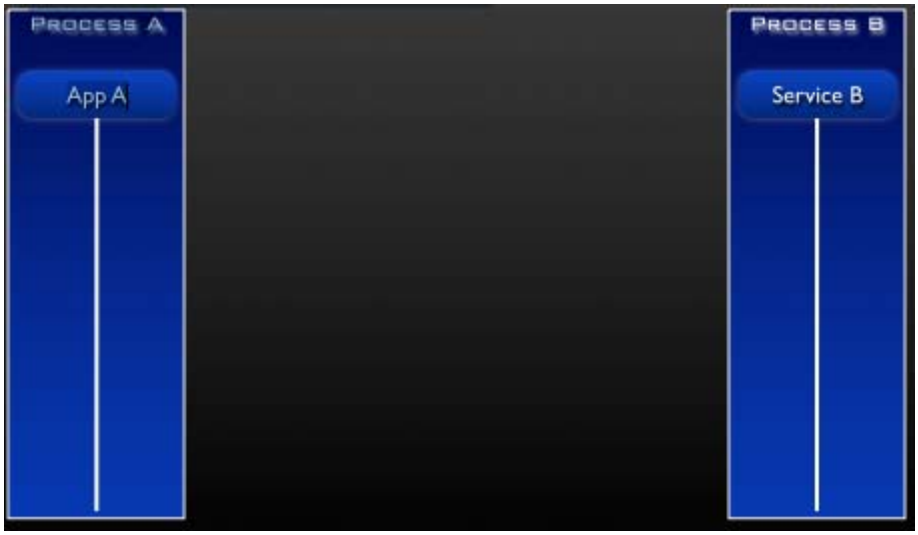
· High performance through shared memory 通过共享内存来提高性能

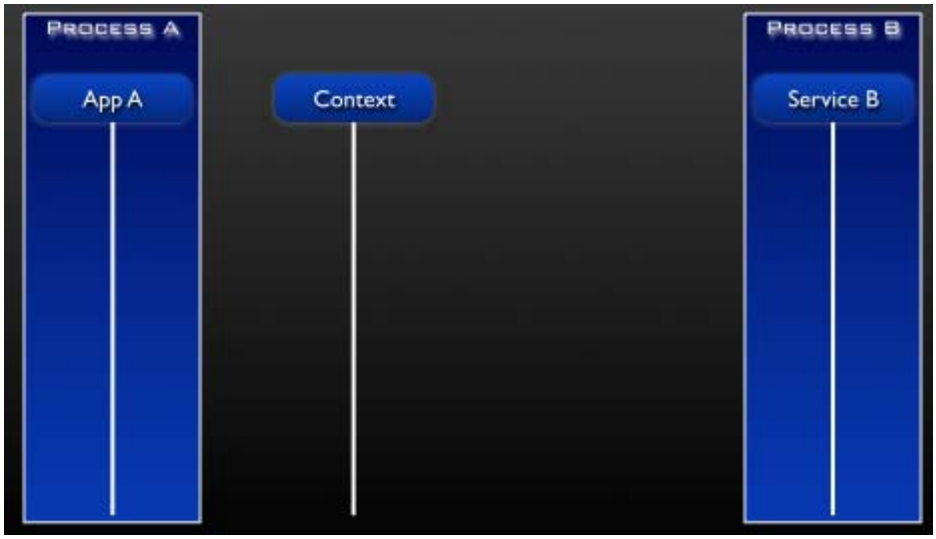
· Per-process thread pool for processing requests 为进程请求分配每进程线程池

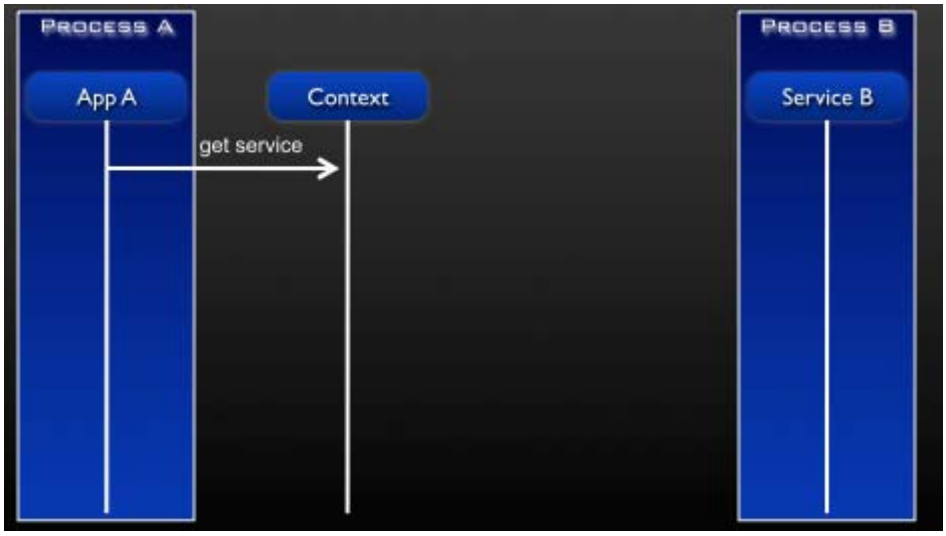
· Reference counting, and mapping of object references across processes 引用计数、跨进程的对象引用映射

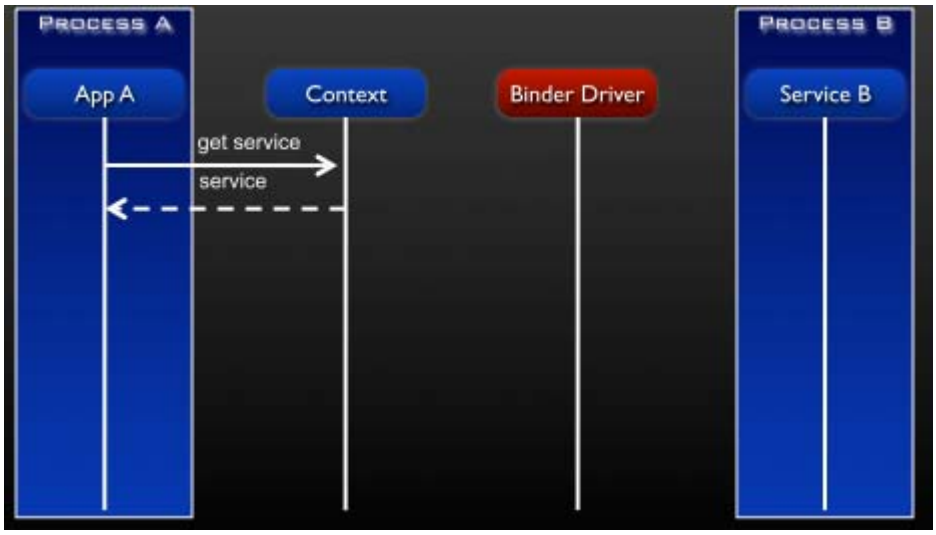
· Synchronous calls between processes 进程间同步调用

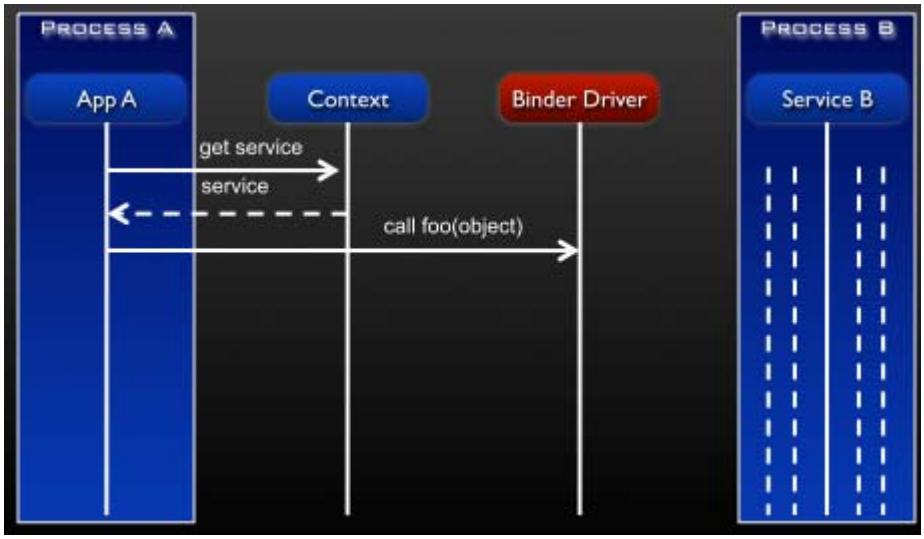
Binder 如何实现：

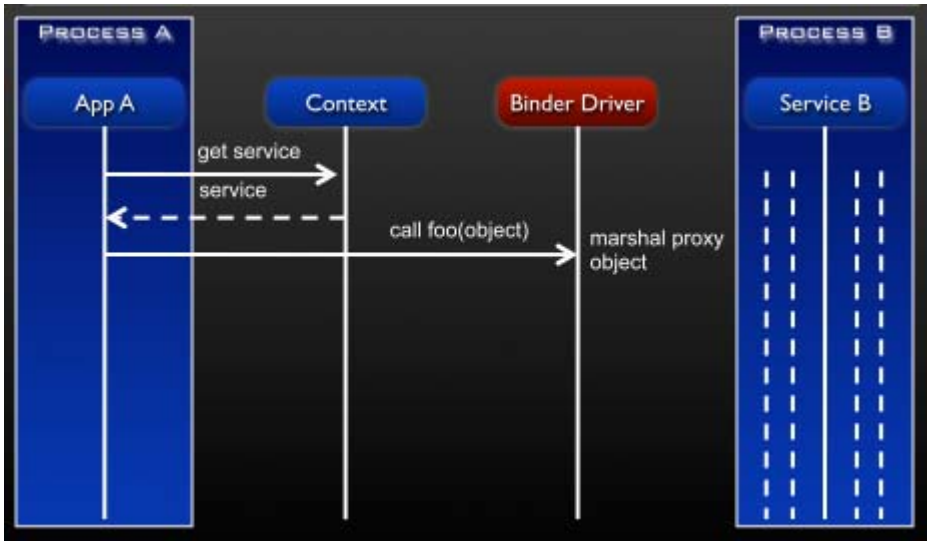

Process A
App A
Process B
Service B

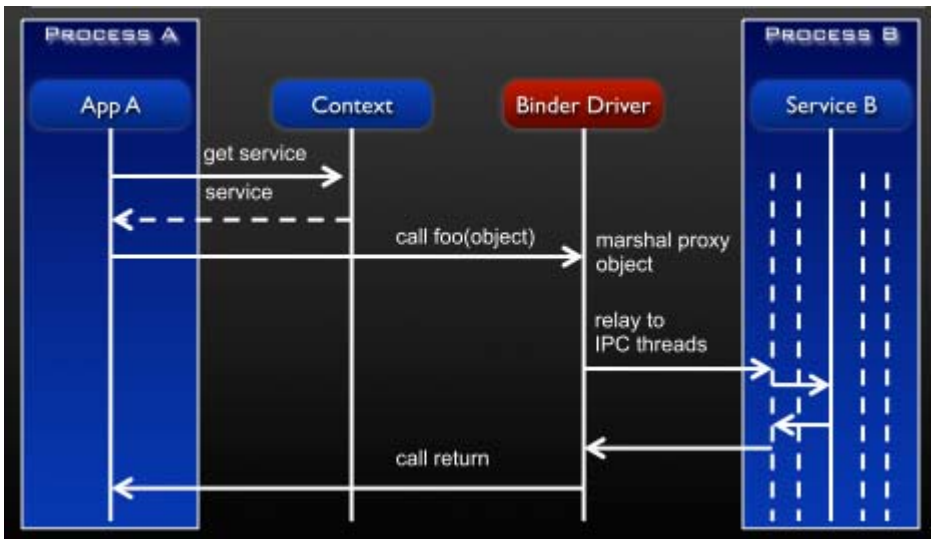

Process A
App A
Context
Process B
Service B


Process A
App A
Context
get service
Process B
Service B


PROCESS A
PROCESS B
App A
Context
Binder Driver
Service B
get service
service


PROCESS A
PROCESS B
App A
Context
Binder Driver
Service B
get service
service
call foo(object)


PROCESS A
PROCESS B
App A
Context
Binder Driver
Service B
get service
service
call foo(object)
marshal proxy
object



Android Interface Definition Language (AIDL)

· http://code.google.com/android/reference/aidl.html

Since each application runs in its own process, and you can write a service that runs in a different process from your Application's UI, sometimes you need to pass objects between processes. On the Android platform, one process can not normally access the memory of another process. **So to talk, they need to decompose their objects into primitives that the operating system can understand, and "marshall" the object across that boundary for you.**

为了两个进程之间能交互数据，必须先将对象分解为操作系统的基本类型并且将对象传递给需求的进程

The code to do that marshalling is tedious to write, so we provide the AIDL tool to do it for you.

AIDL (Android Interface Definition Language) is an IDL language used to generate code that enables two processes on an Android-powered device to talk using interprocess communication (IPC). If you have code in one process (for example, in an Activity) that needs to call methods on an object in another process (for example, a Service), you would use AIDL to generate code to marshall the parameters.

The AIDL IPC mechanism is interface-based, similar to COM or Corba, but lighter weight. **It uses a proxy class to pass values between the client and the implementation.**

AIDL IPC 机制是面向接口的，和 COM 以及 corba 类似，但相比之下更加轻量级，它用 proxy 类来传递值

PM Problem：

· Mobile devices run on battery power 移动设备通过电池供能

· Batteries have limited capacity 电池容量有限

Android 的解决方案：

· Built on top of standard Linux Power Management (PM)

基于标准的 Linux 能耗管理

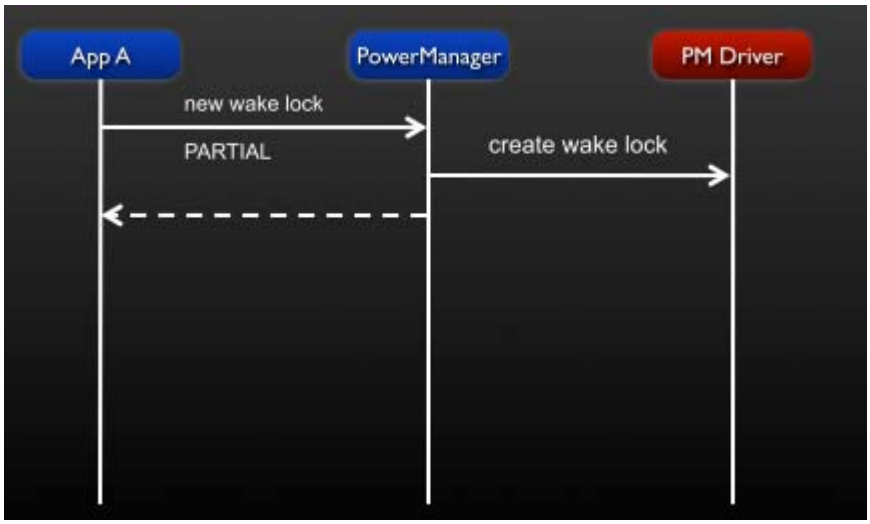
· More aggressive power management policy

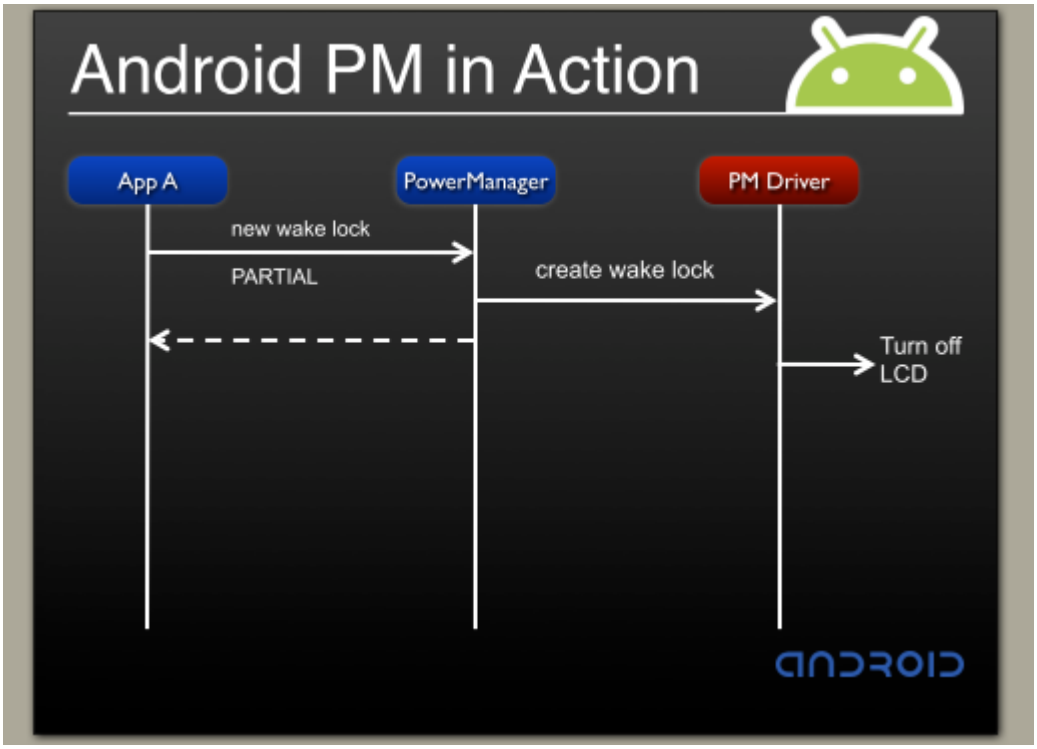
增加了更加好的能耗管理策略

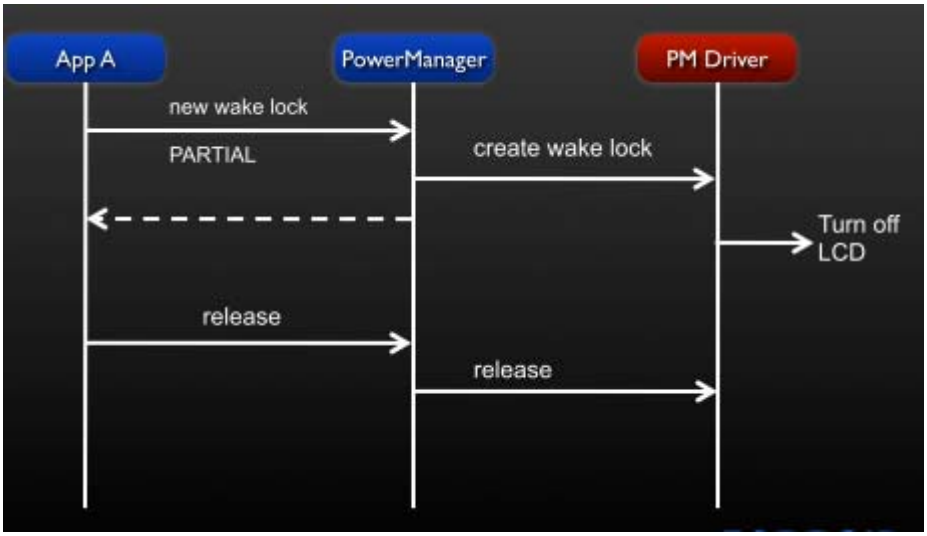
· Components make requests to keep the power on through “ wake locks ”

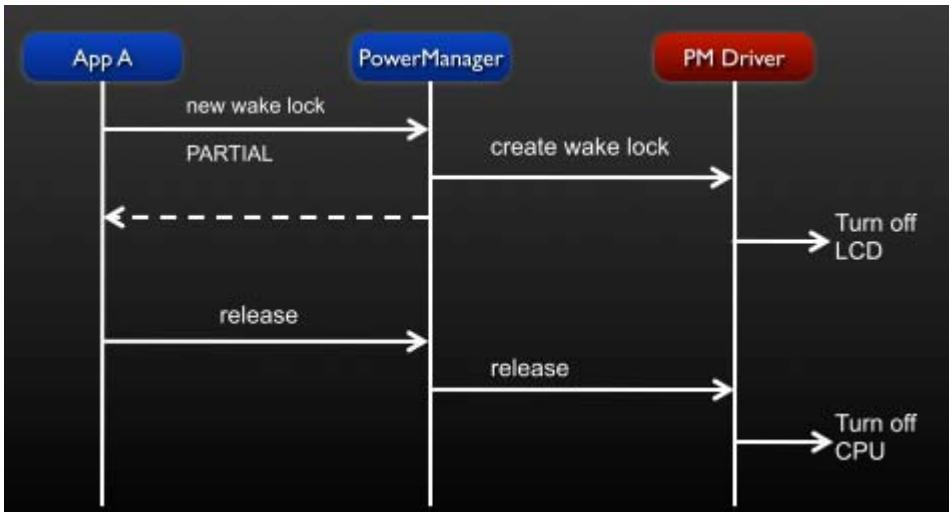
组件通过申请唤醒锁来保持电源打开

· Supports different types of wake locks 支持不同类型的唤醒锁

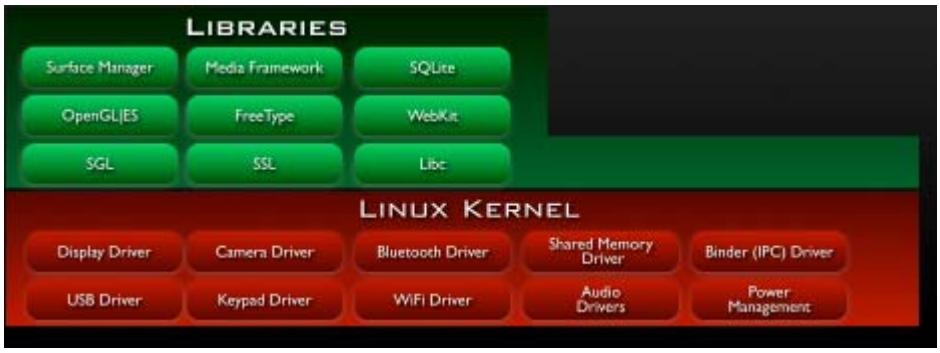



Android PM in Action
App A
PowerManager
PM Driver
new wake lock
PARTIAL
create wake lock
Turn off
LCD
android


App A
PowerManager
PM Driver
new wake lock
PARTIAL
create wake lock
Turn off
LCD
release
release



Android 架构——本地库：



本地库包括四个部分：

· Bionic Libc

· Function Libraries

· Native Servers

· Hardware Abstraction Libraries

Bionic Libc

· What is bionic? 什么是 bionic

· Custom libc implementation, optimized for embedded use.定制的 C 库函数实现，并为嵌入式应用做了优化

Why build a custom libc library?为什么定制 libc 库

- License: we want to keep GPL out of user-space 在用户空间不使用 GPL
- Size: will load in each process, so it needs to be small 最大程度的减少代码空间
- Fast: limited CPU power means we need to be fast 有限的 CPU 能耗，要求我们能运行的更快
- Built-in support for important Android-specific services 内置的对 Android 特有的服务的支持
- system properties 对系统属性的支持函数

  getprop("my.system.property", buff, default);
- log capabilities 对日志系统的支持函数

  LOGI("Logging a message with priority 'Info'");
  - Doesn't support certain POSIX features 不支持模型 POSIX 特征
  - Not compatible with Gnu Libc (glibc) 与 GNU libc 不兼容
  - All native code must be compiled against bionic 所有的本地代码都必须依靠 bionic 支持，而不是 glibc

功能库：

Webkit:

Based on open source WebKit browser: http://webkit.org 基于开源的 Webkit 浏览器

- Renders pages in full (desktop) view
- Full CSS, Javascript, DOM, AJAX support 支持 CSS，JAVASCRIPT,DOM 等脚本语言
- Support for single-column and adaptive view rendering

Media Framework

- Based on PacketVideo OpenCORE platform 基于 PacketVideo OpenCORE 平台
- Supports standard video, audio, still-frame formats 支持标准视频、音频、静态帧格式
- Support for hardware / software codec plug-ins 支持硬件/软件编解码插件

SQLite

- Light-weight transactional data store
- Back end for most platform data storage

Native Servers 本地服务器

皮肤管理：Surface Flinger

- Provides system-wide surface "composer", handling all surface rendering to frame buffer device

提供系统范围的皮肤设计，处理所有与帧缓冲设备相关的皮肤

- Can combine 2D and 3D surfaces and surfaces from multiple applications

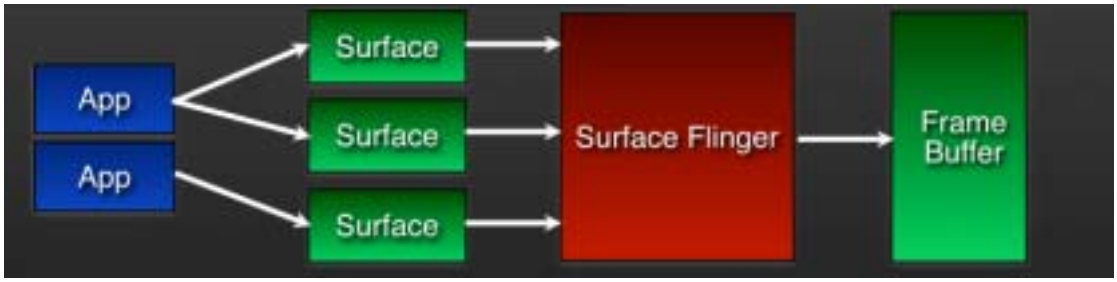
能够合并 2D 和 3D 皮肤，以及来自多个应用程序的皮肤

· Surfaces passed as buffers via Binder IPC calls 皮肤通过 Binder IPC 传递给缓冲器

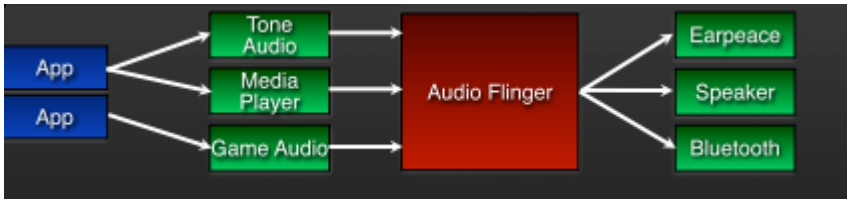
· Can use OpenGL ES and 2D hardware accelerator for its compositions

可以使用 OpenGL ES 和 2D 硬件加速器进行皮肤设计

· Double-buffering using page-flip 双缓冲技术



Audio Flinger：音频管理



· Manages all audio output devices

管理所有的音频输出设备

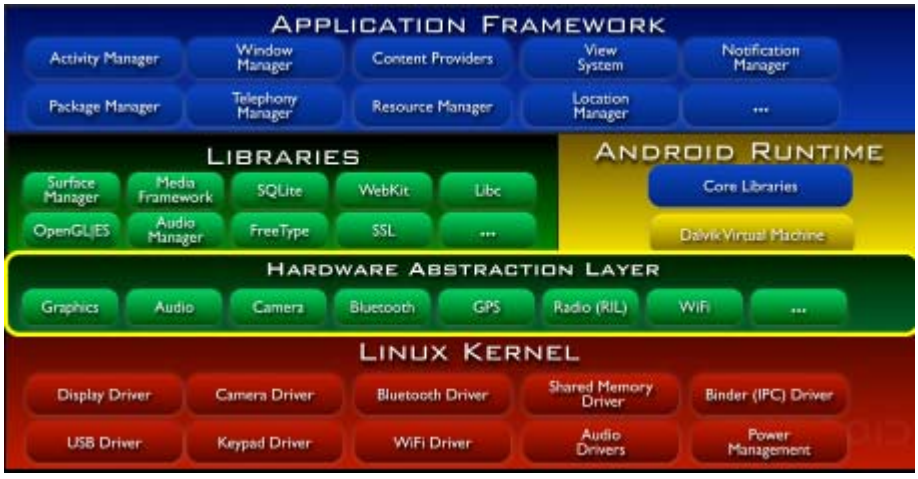
· Processes multiple audio streams into PCM audio out paths

处理 PCM 音频输出通道的多音频流

· Handles audio routing to various outputs

处理音频从不同途径输出

Hardware Abstraction Libraries 硬件抽象库

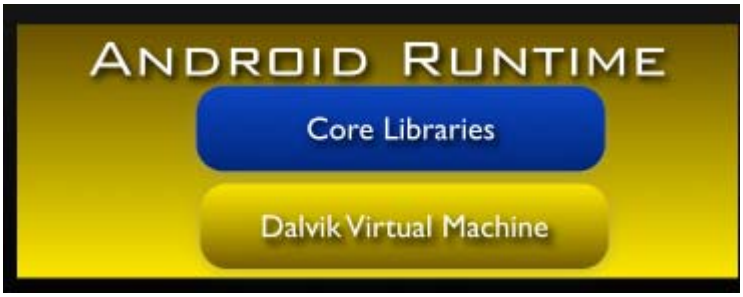


· User space C/C++ library layer 用户空间的 C/C++库
· Defines the interface that Android requires hardware “drivers” to implement
定义了 Android 对硬件驱动要求的接口实现
· Separates the Android platform logic from the hardware interface
将 Android 的平台逻辑和硬件接口分离
Why do we need a user-space HAL?
为什么需要一个用户空间的 HAL
· Not all components have standardized kernel driver interfaces
不是所有的部件都有标准的内核驱动接口
· Kernel drivers are GPL which exposes any proprietary IP
· Android has specific requirements for hardware drivers
Android 对硬件驱动有特定的要求

Andorid 运行时

Dalvik Virtual Machine Dalvik 虚拟机

· Android’s custom clean-room implementation virtual machine
Android 特有的纯净的虚拟机实现
· Provides application portability and runtime consistency
保障了可移植性和运行时的一致性
· Runs optimized file format (.dex) and Dalvik bytecode
以优化的.dex 文件格式运行
· Java .class / .jar files converted to .dex at build time
Jave 的.class / .jar 文件在编译时转化为.dex



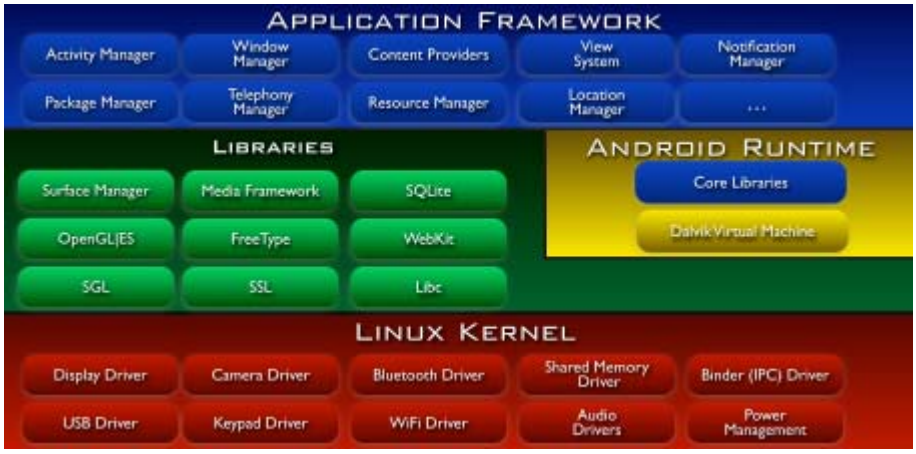
· Designed for embedded environment 为嵌入式开发环境设计
· Supports multiple virtual machine processes per device 支持多虚拟机进程
· Highly CPU-optimized bytecode interpreter 高度 CPU 优化的字节码解释程序
· Uses runtime memory very efficiently 能够高效的使用运行时内存

Core Libraries 核心库

· Core APIs for Java language provide a powerful, yet simple and familiar development platform 核心的 Java 语言 API,提供了一个强大的、简单的、属性的开发平台

· Data structures
· Utilities
· File access
· Network Access
· Graphics
· …

Application Framework 应用程序框架



Core Platform Services 核心平台服务

· Services that are essential to the Android platform 提供 Android 平台的关键服务
· Behind the scenes - applications typically don t access them directly
在后台工作-----应用程序一般不直接访问
包括以下的服务：
· Activity Manager 活动管理
· Package Manager 包管理
· Window Manager 窗口管理
· Resource Manager 资源管理
· Content Providers 内容管理
· View System 视窗系统

Hardware Services 硬件服务

· Provide access to lower-level hardware APIs 提供对底层硬件访问的 API
· Typically accessed through local Manager object 一般通过本地 Manager 对象访问
包括以下服务
· Telephony Service 电话服务

· Location Service 定位服务
· Bluetooth Service 蓝牙服务
· WiFi Service WiFi 服务
· USB Service USB 服务
· Sensor Service 红外传感器服务

关于应用程序框架的详细内容可以参照一下文档：
· At Google I/O
· “Inside the Android Application Framework”

对于 Android 的架构分析到这里就完了，接下来动态分析 Android 的机理

· Android Physiology

· Start-up Walkthrough

· Layer Interaction

包括启动流程和层间交互量部分

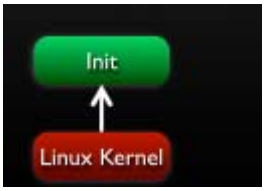
Runtime Walkthrough

It all starts with init…

Similar to most Linux-based systems at startup, the
bootloader loads the Linux kernel and starts the init process.

和其他的基于 Linux 的系统一样，bootloader 加载内核以后，启动 init 进程

我们可以再 init.rc 中看到



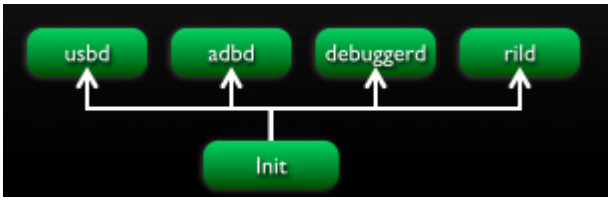
Init starts Linux daemons, including: init 启动后台程序

· USB Daemon (usbd) to manage USB connections USB 后台程序管理 USB 连接
· Android Debug Bridge (adbd) to manage ADB connections ADB 程序管理 ADB 连接
· Debugger Daemon (debuggerd) to manage debug processes requests (dump memory, etc.)

调试后台程序，管理调试进程请求

· Radio Interface Layer Daemon (rild) to manage communication with the radio

无线接口层后台程序，管理无线通信



Init process starts the zygote process:启动结合体进程

· A nascent process which initializes a Dalvik VM instance

这个进程用来启动一个 Dalvik VM 实例

· Loads classes and listens on socket for requests to spawn VMs

为生成 VM 的请求加载类并监听套接字

· Forks on request to create VM instances for managed processes

生成创建管理进程的 VM 实例请求

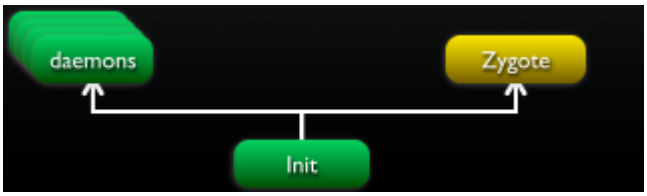
· Copy-on-write to maximize re-use and minimize footprint

通过写时复制来实现最大化的复用

service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server

socket zygote stream 666

onrestart write /sys/android_power/request_state wake



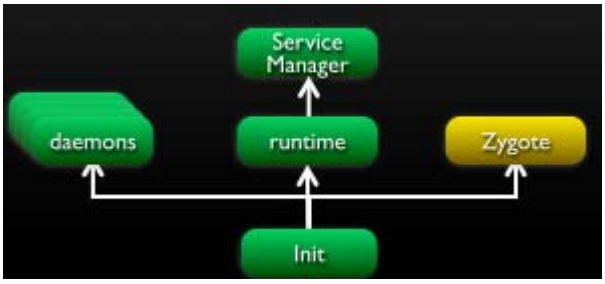
Init starts runtime process:Init 初始化运行时进程

· Initializes Service Manager – the context manager for Binder that handles service registration and lookup

初始化服务管理器

· Registers Service Manager as default context manager for Binder services

注册服务管理为默认的绑定服务上下文管理



service servicemanager /system/bin/servicemanager

user system

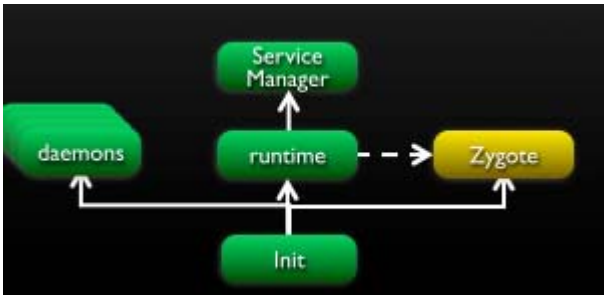
critical

onrestart restart zygote

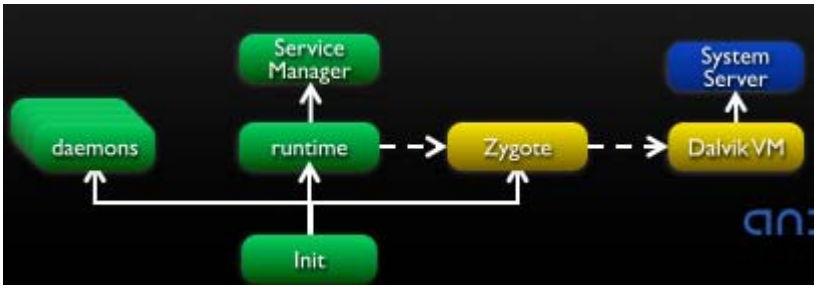
onrestart restart media

Runtime process sends request for Zygote to start System Service

运行时进程发送请求给 Zygote 启动系统服务

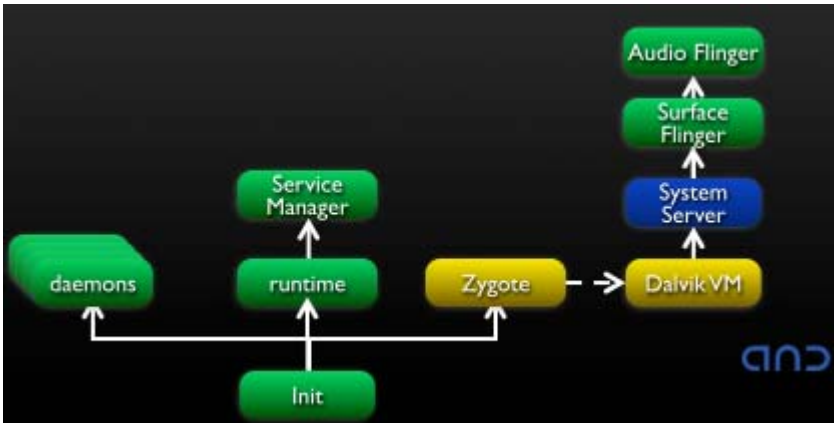


· Zygote forks a new VM instance for the System Service process and starts the service Zygote 为系统服务进程生成一个新的 VM 实例并启动服务

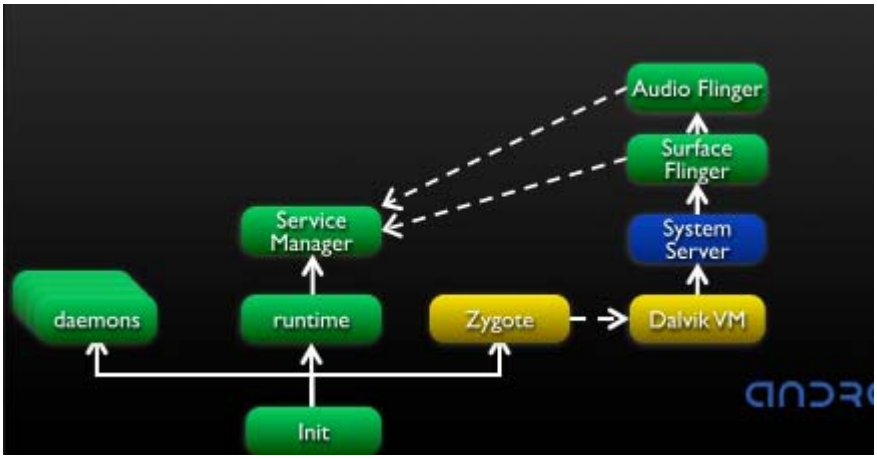


系统服务启动两个本地 Surface Flinger 和　Audio Flinger
System Service starts the native system servers, including:

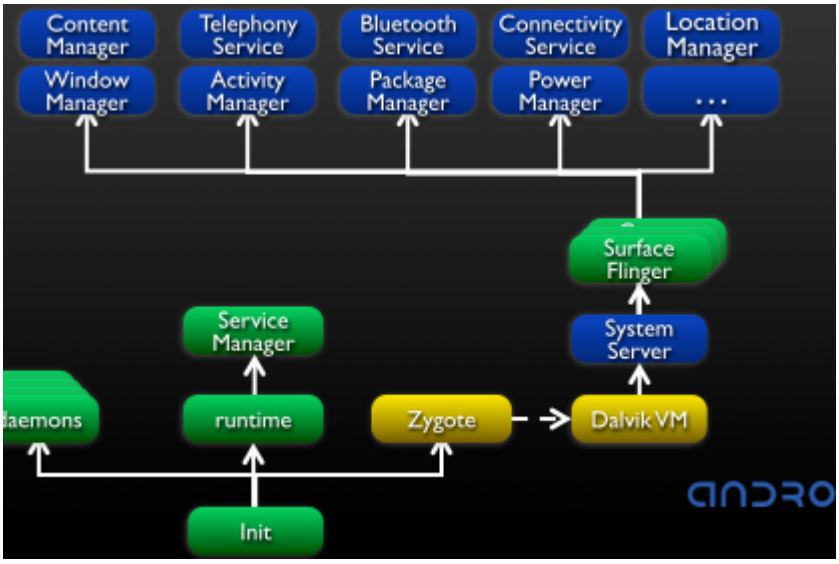
- Surface Flinger
- Audio Flinger



Native system servers register with Service Manager as IPC service targets:
两个本地系统服务项服务管理器注册成为 IPC 服务对象

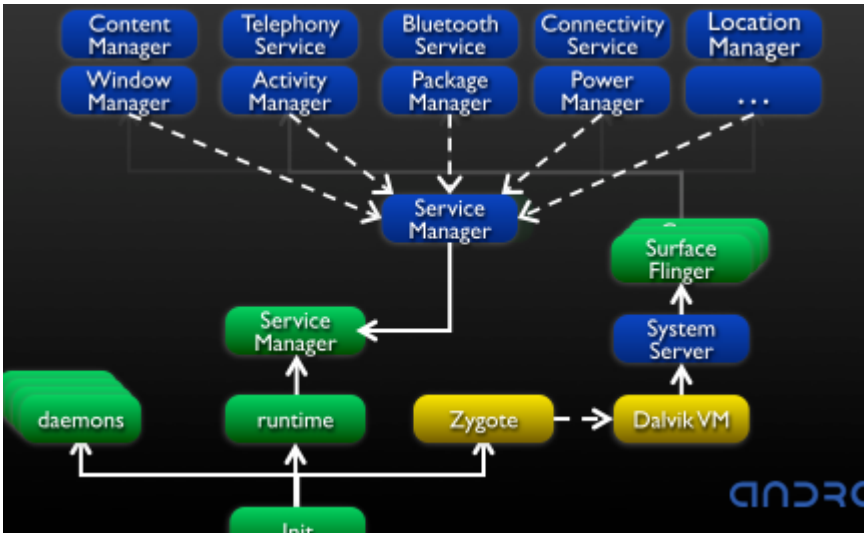


System Service starts the Android managed services
系统服务启动 Android 管理服务：

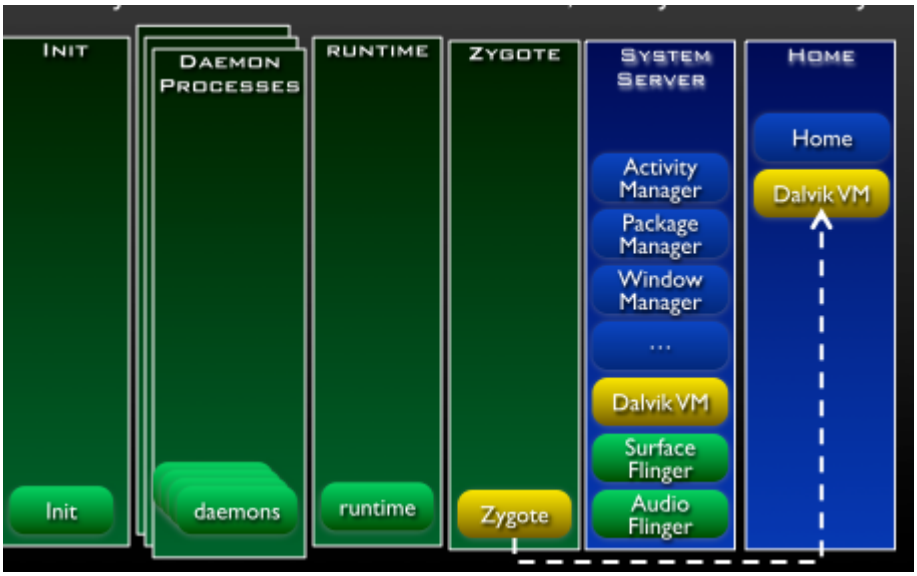


Android managed Services register with Service Manager

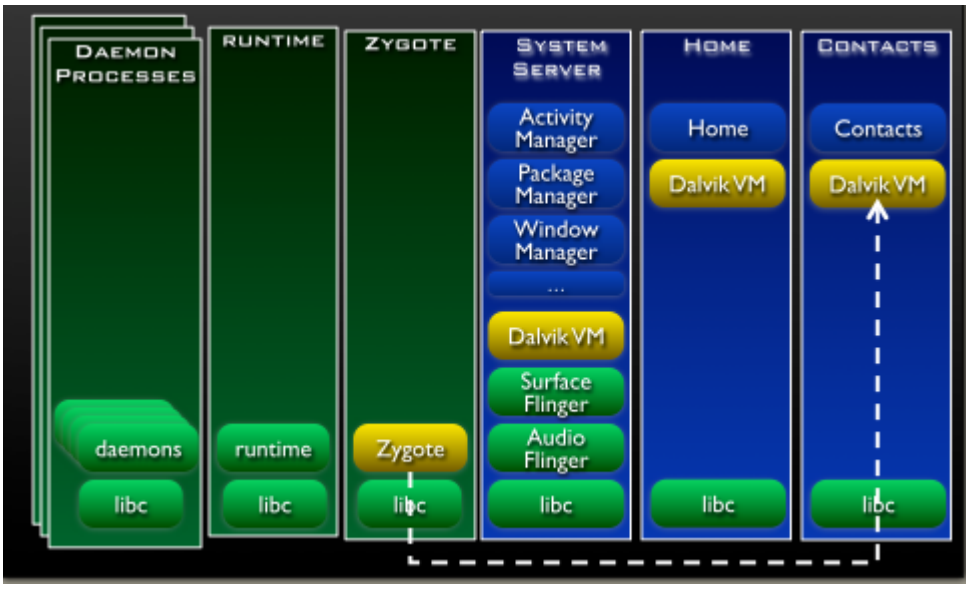
Android 管理的服务项服务管理器注册(包括我们前面设计的硬件服务和核心平台服务)



After system server loads all services, the system is ready…当系统服务加载了所有服务后系统就准备好了。



启动 home 界面



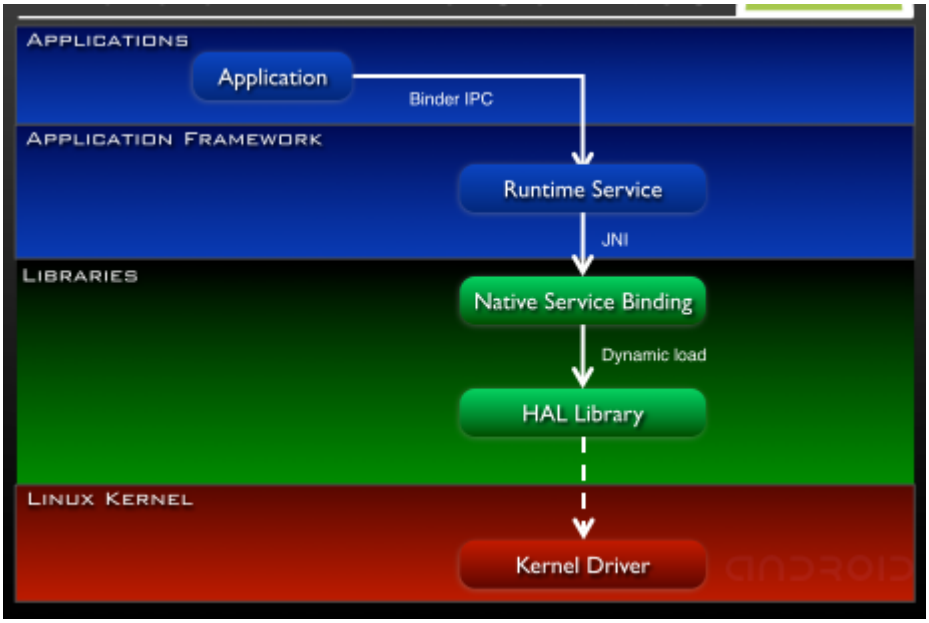
启动联系人程序

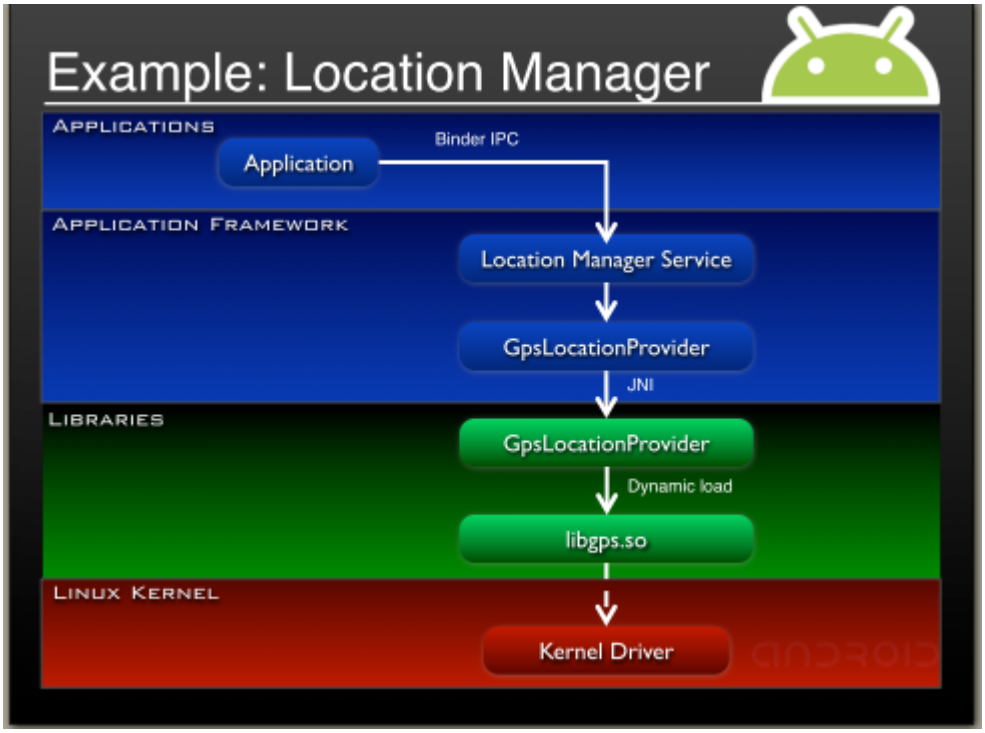
层次交互 Layer Interaction

There are 3 main flavors of Android layer cake:三种形式的层次交互

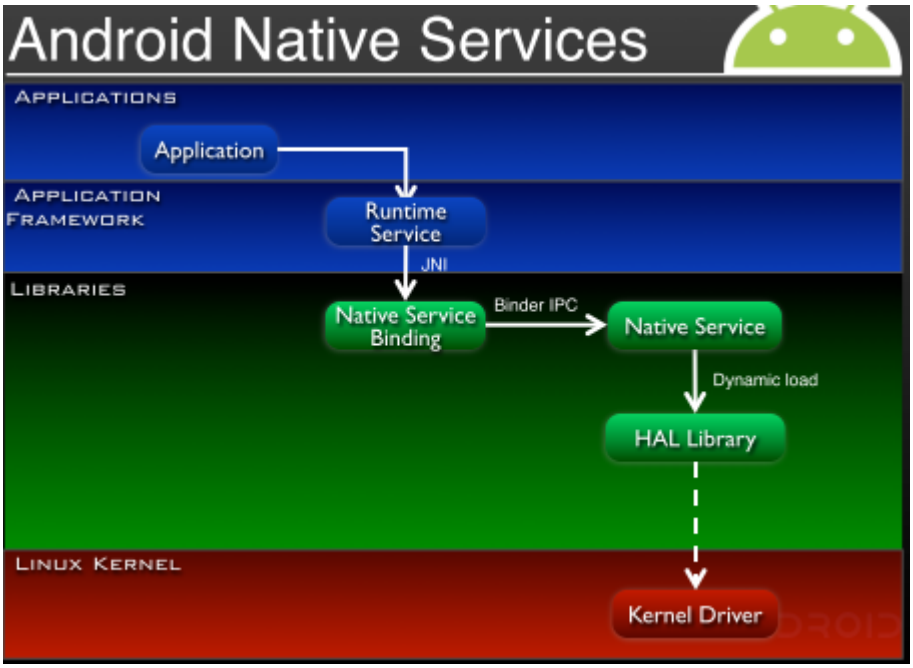
- App Runtime Service lib
- App Runtime Service Native Service lib
- App Runtime Service Native Daemon lib

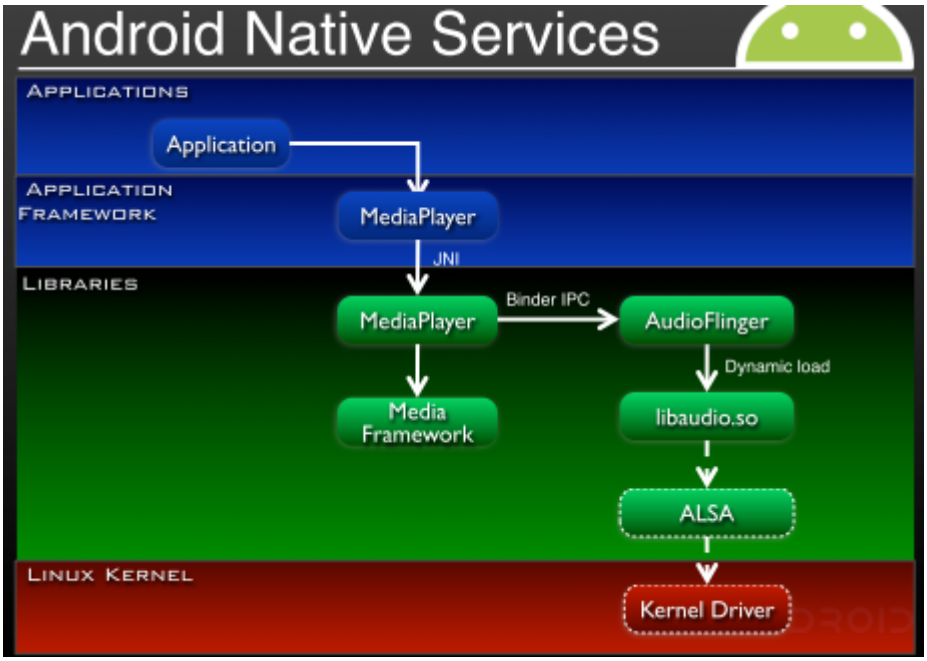
App Runtime Service lib (Android Runtime Services)



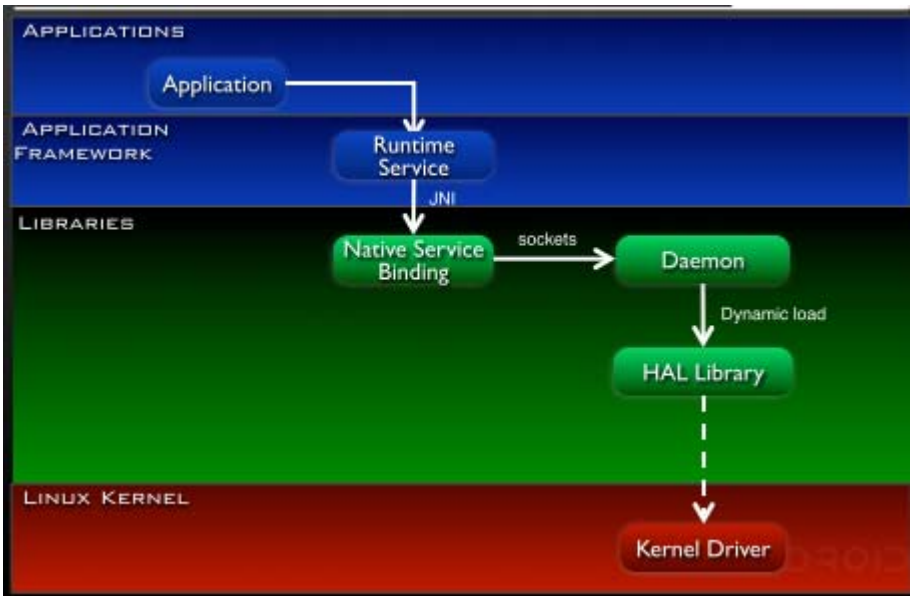


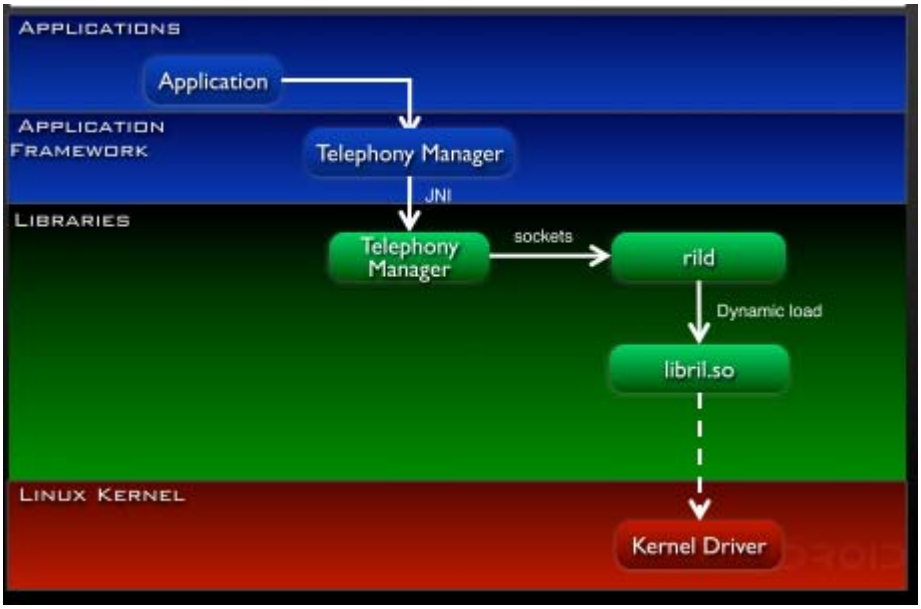
· App Runtime Service Native Service lib (Android Native Services)





第三种情形∶ App Runtime Service Native Daemon lib


APPLICATIONS
Application
APPLICATION FRAMEWORK
Runtime Service
JNI
LIBRARIES
Native Service Binding
sockets
Daemon
Dynamic load
HAL Library
LINUX KERNEL
Kernel Driver


APPLICATIONS
Application
APPLICATION FRAMEWORK
Telephony Manager
JNI
LIBRARIES
Telephony Manager
sockets
rild
Dynamic load
libril.so
LINUX KERNEL
Kernel Driver