第3章 Android系统框架

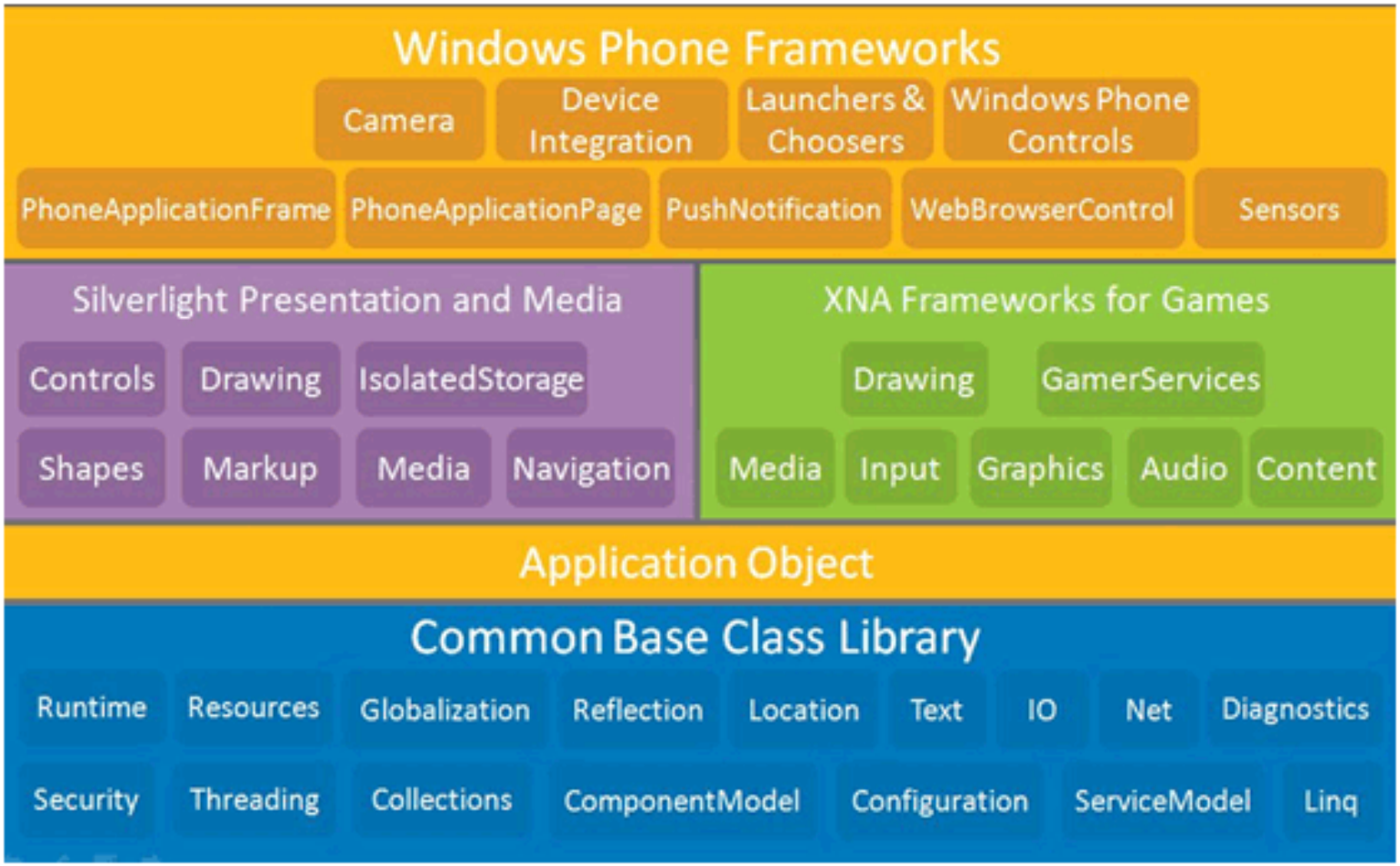
杨刚
中国人民大学

本章学习目标：

- 了解Android的软件系统框架
- 了解Android应用程序架构
- 认识软件架构的思想
- 延伸软件设计的层次模块设计

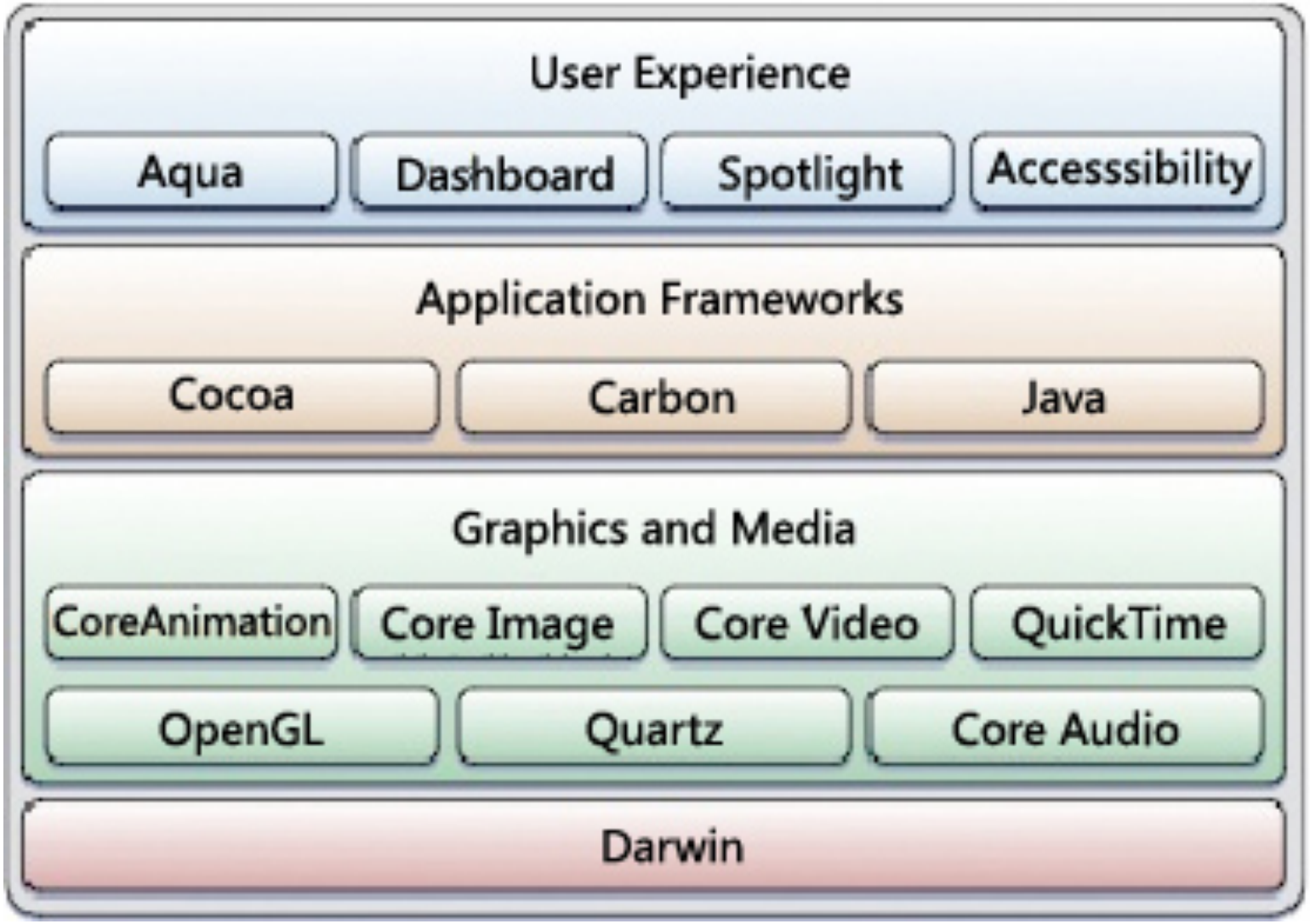
对比主流手机OS的架构

Win Phone 框架

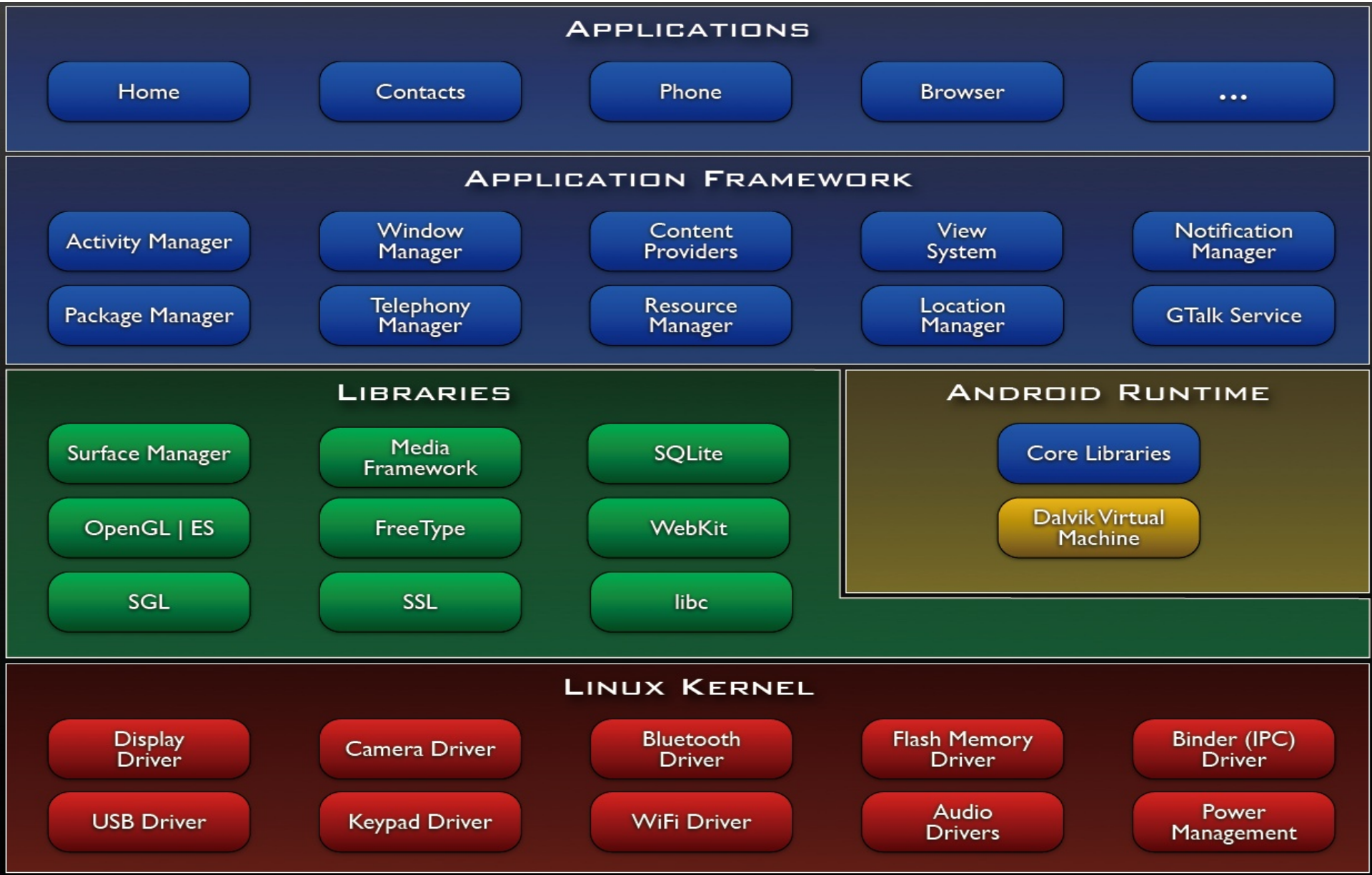


对比主流手机OS的架构

Mac OS X 框架



一、Google Android 软件架构



Android系统基础架构

- Android系统架构和其操作系统一样，采用了分层的架构。系统架构分为四个层，从高层到低层分别为
  - 应用程序层、
  - 应用程序框架层、
  - 系统运行库层
  - Linux核心层。

1. Linux内核

Android 的核心系统服务依赖于Linux内核，如安全性，内存管理，进程管理， 网络协议栈和驱动模型。 Linux 内核也同时作为硬件和软件栈之间的抽象层。

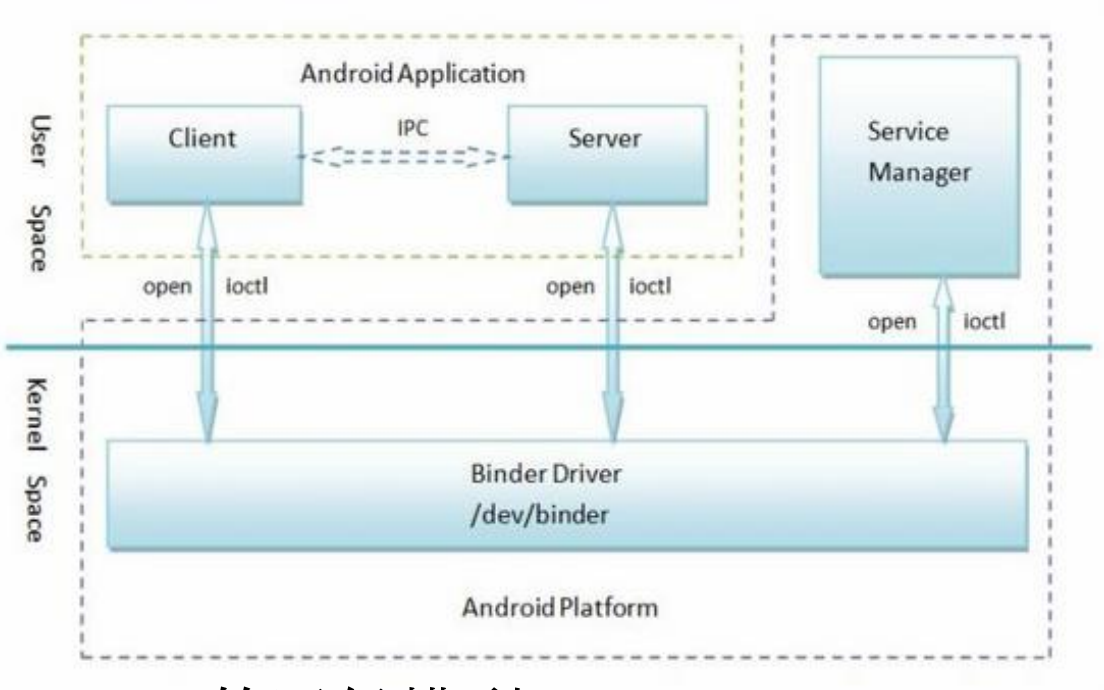
Android Linux内核与Linux的区别

- Android建立在linux内核之上，但是Android不是linux
- 没有支持本地窗口系统
- 没有支持glibc运行库
- 没有包含完整的linux内核

Binder：

- 驱动程序加快进程之间的通信
- 通过共享数据提高性能
- 对于处理请求，每个进程有单独的线程池
- 引用计数和对象引用进程间映射
- 进程间同步调用

Binder in Action



Binder的通信模型

- Client和Server是存在于用户空间
- Client与Server通信，是Binder驱动在内核空间上实现
- SM作为守护进程，处理client请求，管理全部服务项，并向Client提供查询Server接口的能力

电源管理

- 建立在标准的Linux Power Manager(PM)之上
- 更激进的能耗管理策略
- 组件通过“唤醒锁”发出保持开机的请求
- 支持多种类型的唤醒锁

电源管理

只要系统上有活动的**唤醒锁**，设备便无法进入挂起模式，除非释放唤醒锁。

谨慎使用：使用唤醒锁时，当使用结束时，必须将其正确释放，因为未释放的唤醒锁无法进入默认状态以节能，从而很快便会将设备的电池耗尽。

/proc/wakelocks 文件列出了定义使用唤醒锁的服务和驱动程序。 通过监控 /sys/power/wake_lock 文件（需要root权限），可以了解何时启用了唤醒锁，以及哪种服务启用了唤醒锁

目的：设备黑屏后，程序仍然要保持运行

电源管理

使用的应用	执行的操作	使用了唤醒锁的服务	运行状况
任意	按下 UI Widget（如点击按钮或 ListView 项）	PowerManagerService	启用并在 5 秒钟后释放锁定
地图/导航	启用地图或进入导航	gps-lock	启用锁定并使用 GPS
YouTube	观看流视频	PowerManagerService	在视频播放的整个过程中一直启用唤醒锁
Music	听音乐	PowerManagerService	在音乐播放的过程中一直启用唤醒锁

2. 系统运行库

- Android系统架构 包含一些C/C++库，这些库能被Android系统中不同的组件使用。它们通过 Android 应用程序框架为开发者提供服务。以下是一些核心库：
  - 系统 C 库：一个从 BSD 继承来的标准 C 系统函数库（ libc ）， 它是专门为基于 embedded linux 的设备定制的。
  - 媒体库：基于 PacketVideo OpenCORE；该库支持多种常用的音频、视频格式回放和录制，同时支持静态图像文件。
  - Surface Manager：对显示子系统的管理，并且为多个应用程序提 供了2D和3D图层的无缝融合。
  - LibWebCore：一个最新的web浏览器引擎用，支持Android浏览器和一个可嵌入的web视图。
  - SGL：底层的2D图形引擎
  - 3D libraries：基于OpenGL ES 1.0 APIs实现；该库可以使用硬件 3D加速（如果可用）或者使用高度优化的3D软加速。
  - FreeType -位图（bitmap）和矢量（vector）字体显示。
  - SQLite - 一个对于所有应用程序可用，功能强劲的轻型关系型数据库引擎。

1）本地库

- Bionic Libc
- 函数库(Funtion Libraries)
- 本地服务(Native Servers)
- 硬件抽象库(Hardware Abstraction Libraries)

Bionic Libc

- 定制库应用，优化嵌入式的应用
- BSD协议，使得GPL不出现在用户空间
- 体积小，代码路径短，会在每个进程中加载
- 自定义的线程实现，十分快速而简洁
- 与标准的GNU glibc库不兼容
- 所有的本地程序必须依照bionic库进行编译

2）Function libc

- **Webkit**
- **Media Framework**
- **SQLite**嵌入式数据库

WebKit

- 建立在开源的WebKit之上
- 网页渲染以桌面视图模式完整显示
- 完全的支持CSS, Javascipt, DOM, AJAX
- 支持单栏和自适应视图渲染

Media FrameWork

- 建立在PacketVideo OpenCORE平台之上
- 支持标准的视频，音频格式
- 支持硬件/软件解码插件

SQLite

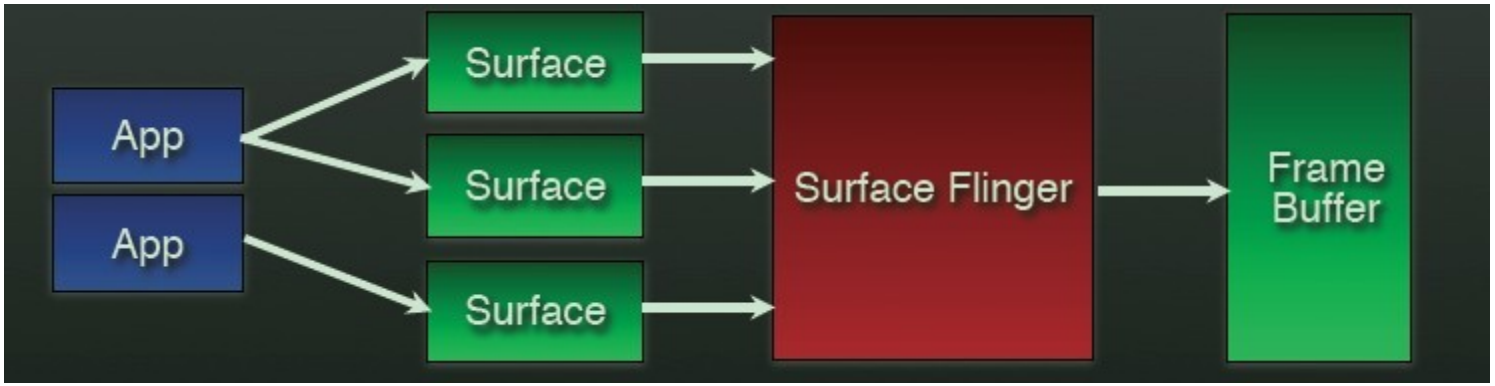
- 轻量级事务数据存储
- 关系型数据库
- 支持Windows/Linux/Unix等多款主流操作系统
- 多数嵌入式平台数据存储的后端

3）本地服务器

- 表面抛射器　　Surface Flinger
- 音频抛射器　　Audio Flinger

Surface Flinger

- 提供全系统的表面“设计器”，将所有表面渲染动作处理后传递到帧缓冲
- 可以结合二维和三维的表面，或者多个应用的表面

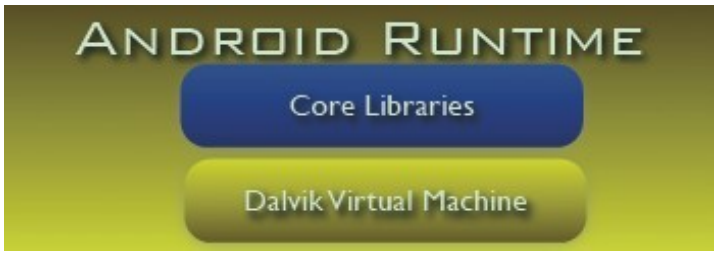


Audio Flinger

- 控制所有的音频设备
- 处理多音频流进行PCM音频输出路径
- 控制音频到不同的输出

4）Android Runtime

- Dalvik 虚拟机
- 核心库



Dalvik虚拟机

- 完全为Android定制的虚拟机
  - 提供应用程序可移植性和运行环境的一致性
  - 运行优化的dex格式文件和Dalvik字节码
  - Java .class/.jar 在创建的时候就被转换成为了.dex
- 为嵌入式环境设计
  - 支持每设备多个虚拟机进程
  - 高度CPU优化的字节码解释器
  - 高效内存使用

ART虚拟机

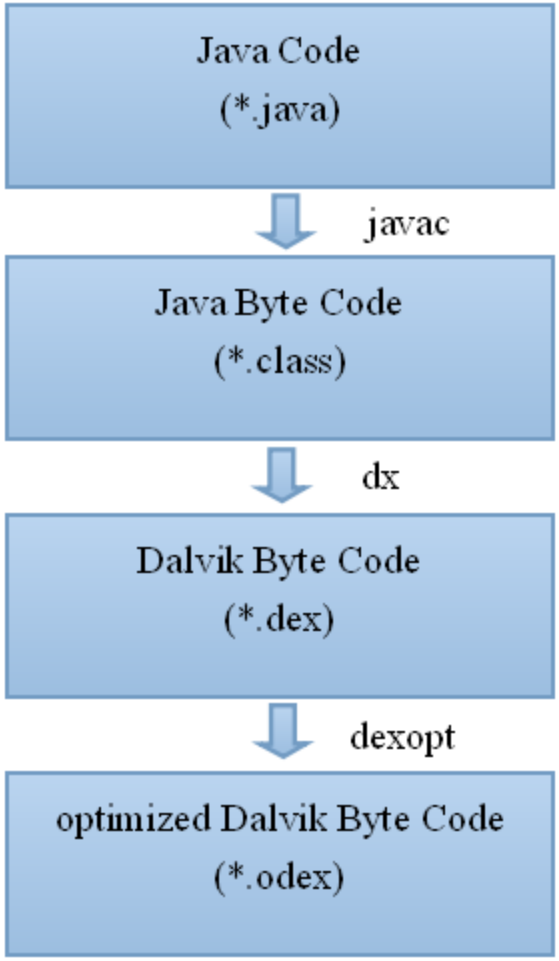
- ART
  - 使用AOT进行处理（Ahead-Of-Time）：在运行以前就把中间代码静态编译成本地代码，这就减去了JIT运行时的转换时间
- ART的编译器分两种模式：
  - 在开发机上编译预装应用；
    - C/C++开发应用程序的时候，编译器直接就把它们翻译成目标机器码。 Java to binary
  - 在设备上编译新安装的应用；
    - 在应用安装时将dex字节码翻译成本地机器码。 dex to binary

ART虚拟机

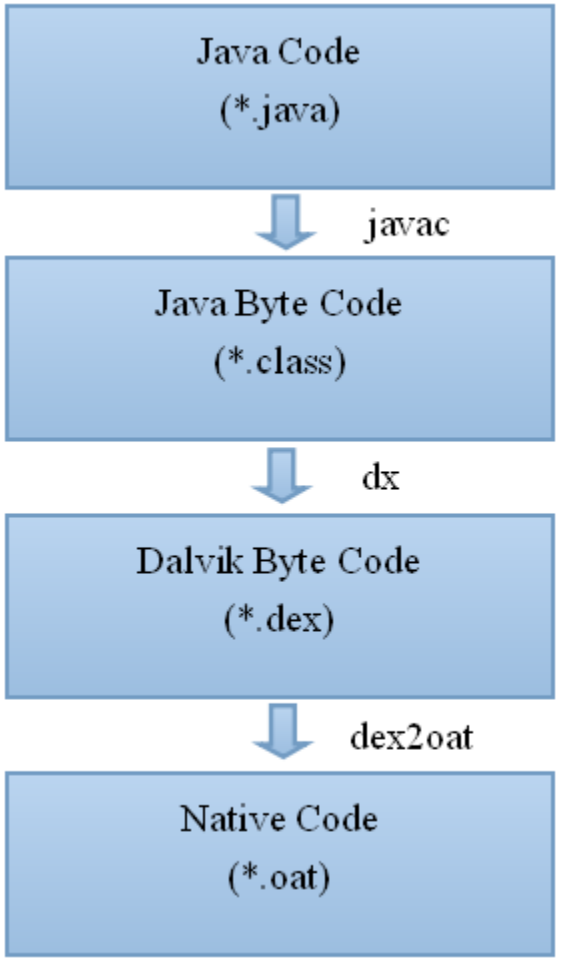
- ART优点
  - 系统性能的显著提升。
  - 应用启动更快、运行更快、体验更流畅、触感反馈更及时。
  - 更长的电池续航能力。
  - 支持更低的硬件。
- ART缺点
  - 更大的存储空间占用，可能会增加10%-20%
  - 更长的应用安装时间。

两种虚拟机下Java编译过程

- **Dalvik字节码生成过程**



ART本地码生成过程



对比

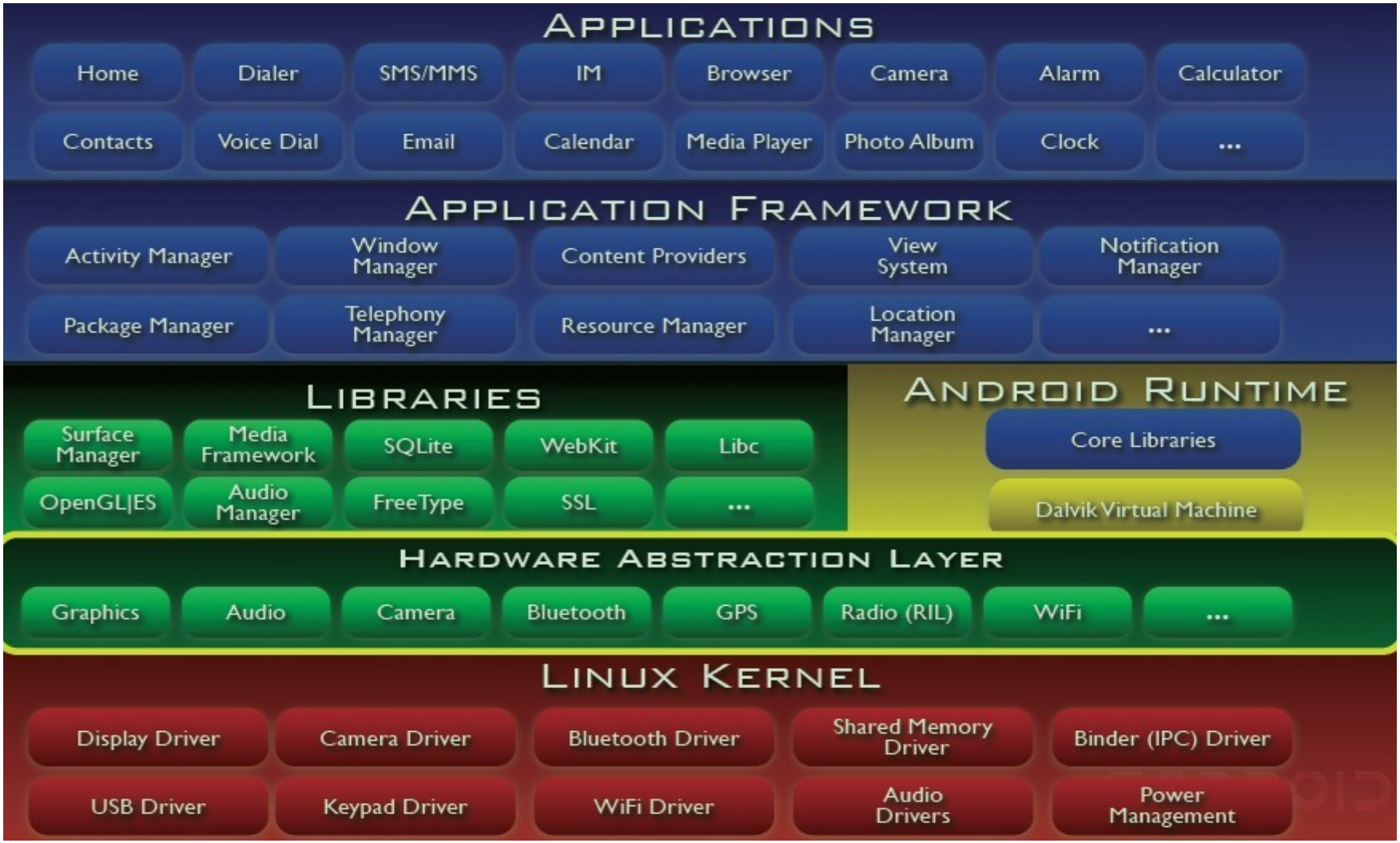
对比项目	CPU	RAM内存	ROM内存	流畅度	省电	APP加载速度	兼容性
ART模式	--	小	大	更佳	更佳	快	有待优化
Dalvik模式	--	大	小	普通	普通	慢	好

核心库

- 针对**Java**语言的**APIs**提供了一个强大，然而简单的开发平台
  - 数据结构
  - 实用类
  - 文件访问
  - ……

5）Hardware Abstraction Layer

- Google因应厂商「希望不公开源码」的要求下，所推出的新观念——HAL硬件抽象层。
- 并不是所有的硬件设备都有标准的linux kernel的接口。
- Kernel driver涉及到GPL的版权。某些设备制造商并不愿意公开硬件驱动，所以才去HAL方式绕过GPL。
- 针对某些硬件，Android有一些特殊的需求。


Applications
Home
Dialer
SMS/MMS
IM
Browser
Camera
Alarm
Calculator
Contacts
Voice Dial
Email
Calendar
Media Player
Photo Album
Clock
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
...
Libraries
Surface Manager
Media Framework
SQLite
WebKit
Libc
OpenGL|ES
Audio Manager
FreeType
SSL
...
Android Runtime
Core Libraries
Dalvik Virtual Machine
Hardware Abstraction Layer
Graphics
Audio
Camera
Bluetooth
GPS
Radio (RIL)
WiFi
...
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Shared Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

- C/C++库文件层
- 定义Android的驱动接口
- 将Android逻辑平台从硬件接口中分开

3. 应用程序框架

- 应用程序的架构设计简化了组件的重用；
- 任何一个应用程序都可以发布它的功能块并且任何其它的应用程序都可以使用其所发布的功能块（不过得遵循框架的安全性限制）；
- 帮助程序员快速的开发程序，并且该应用程序重用机制也使用户可以方便的替换程序组件

3. 应用程序框架

- 丰富而又可扩展的视图（Views），可以用来构建应用程序，它包括列表（lists），网格（grids），文本框（text boxes），按钮（buttons）， 甚至可嵌入的web浏览器。
- 内容提供器（Content Providers）使得应用程序可以访问另一个应用程序的数据（如联系人数据库）， 或者共享它们自己的数据。
- 资源管理器（Resource Manager）提供非代码资源的访问，如本地字符串，图形，和布局文件（layout files）。
- 通知管理器（Notification Manager）使得应用程序可以在状态栏中显示自定义的提示信息。
- 活动管理器（Activity Manager）用来管理应用程序生命周期并提供常用的导航回退功能。

4. 应用程序层

Android会同一系列核心应用程序包一起发布，该应用程序包包括email客户端，SMS短消息程序，日历，地图，浏览器，联系人管理程序等。所有的应用程序都是使用JAVA语言编写的。

Android 系统源代码目录结构

	Description
bionic	C runtime: libc, libm, libdl, dynamic linker Bionic含义为仿生，这里面是一些基础的库的源代码
bootloader/legacy	Bootloader reference code 启动引导相关代码
build	Build system build目录中的内容不是目标所用的代码，而是编译和配置所需要的脚本和工具
cts	Android兼容性测试套件标准
dalvik	Dalvik virtual machine JAVA虚拟机
development	High-level development and debugging tools 程序开发所需要的模板和工具
frameworks/base	Core Android app framework libraries 目标机器使用的一些库
frameworks/policies/base	Project
hardware/libhardware	Hardware abstraction library 与硬件相关的库
hardware/ril	Radio interface layer
out	编译完成后的代码输出与此目录

kernel	Linux kernel Linux2.6的源代码
prebuilt	Binaries to support Linux and Mac OS builds x86和arm架构下预编译的一些资源
packages	Android的各种应用程序
sdk	sdk及模拟器
recovery	System recovery environment 与目标的恢复功能相关
system	Android的底层的一些库
vendor	厂商定制代码

二、应用程序框架

- 进程
  - 当应用程序的第一个组件需要运行时，Android就创建一个只包含一个线程的Linux的进程
  - 默认情况下，应用程序的所有组件都在这个进程中的线程中执行
  - 每一个进程都被一个manifest file控制
  - 当内存资源很紧张的时候，Android会暂时中止掉一些优先级较低的进程
- 线程
  - 在一个进程的主线程中，所有的组件都将被初始化。
  - 可以通过Java传统的Thread类进行创建
  - Android操作系统会尽量长时间的保持线程

二、应用程序框架

- 一般情况Android应用程序是由以下四种组件所组成的：
  - 活动(Activity)
  - 服务(Service)
  - 广播接收器(Broadcast Receiver)
  - 内容提供器(Content Provider)

二、应用程序框架

- **活动(Activity)**
  - 一般所指的活动（Activity）是用户界面。一个应用程序可能有一个或以上的活动存在，每个活动也都会有自己的View。
  - 所有的活动在系统里由活动堆栈所管理，当一个新的活动被执行后，它将会被放置到堆栈的最顶端，并且变成“running activity”，而先前的活动原则上还是会存在于堆栈中，但它此时不会是在前景的情况，除非新加入的活动离开。

二、应用程序框架

- **服务(Service)**
  - 服务是在背景长时间运行的应用组件，不和用户直接进行互动。
  - 例如：某服务可能在后台播放音乐，而用于在执行其他的操作，或者它通过网络抓取资料或者执行某些计算，将结果提供给活动（Activity）。

二、应用程序框架

- **内容提供者(Content Provider)**
  - 内容提供者将应用程序数据组合成特定的集合供其它应用程序使用。数据可以是储存在文件、SQLite数据库，或是其它任何用户可以存取资料的地方。
  - 内容提供者继承於内容提供者基础类别，并实现一组标准的方法，使应用程序可以检索和储存它控制的数据。
  - 应用程序不是直接调用这些实现方法。而是通过内容解析器(ContentResolver)对象调用方法。内容解析器能够通知任何的内容提供者，并可以参与这些内容提供者进程间的管理。

二、应用程序框架

- **广播接收器(Broadcast Receiver)**
  - 广播接收器负责接受和响应通知，很多通知源自于系统所发送的，例如：发送时区变换的通知，电池电量不足，或用户改变语言设置。
  - 应用程序也可以发出广播通知，举例来说，通知其它应用程序，数据已下载完毕，可供使用。
  - 应用程序可以拥有任意数量的广播接收器来接收任何的通知。另外也可以启动活动（Activity）去响应接收到的通知，或利用通知管理器(Notification Manager)来通知使用者。

Activity的生命周期 (Life Cycle)

- 完全生命周期
  - 开始于onCreate(Bundle)
  - 结束于onDestroy()。
- 可见生命周期
  - 开始于onStart()
  - 结束于onStop()
- 前台生命周期
  - 开始于onResume()
  - 结束于onPause()