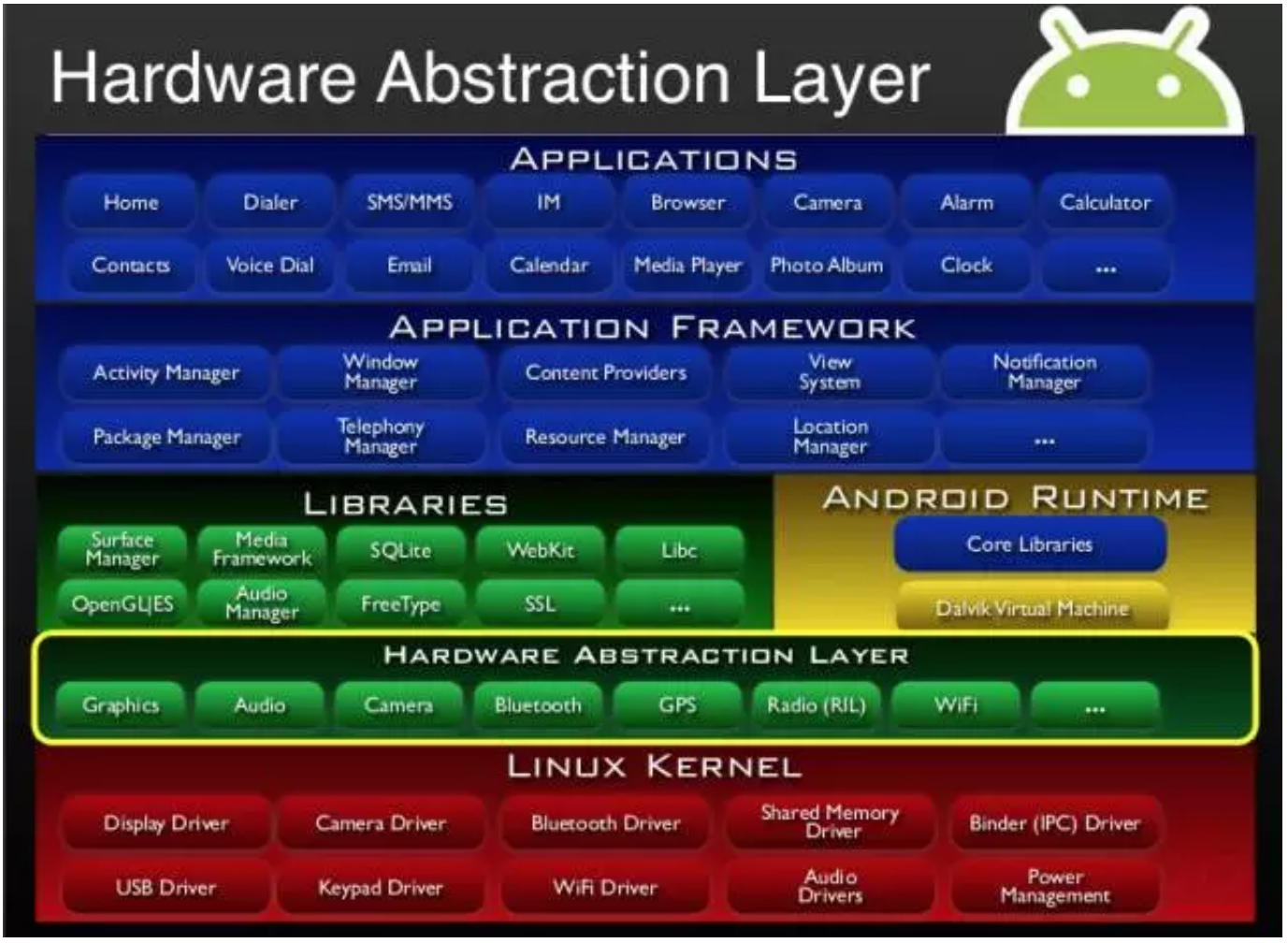
Android 随想录之 Android 系统架构

2015-09-25 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：肥肥鱼
> 链接：http://codingfish.top/2015/08/27/android-systen-framwork-md/



应用层（Application）

Android 的应用层由运行在 Android 设备上的所有应用程序共同构成（系统预装程序以及第三方应用程序）。

系统预装应用程序包含拨号软件、短信、联系人、邮件客户端、日历、地图以及浏览器等提供基础功能的应用程序构成。第三方应用程序则是基于 Android SDK（Android Software Development Kit）进行开发，并受到框架层 SDK 接口约束的应用程序。两者的区别在于，系统预装应用比第三方应用拥有更高的系统使用权限：系统预装应用可以使用 SDK 中隐藏

的、不对第三方应用开放的系统 API。

通常情况下，Android 应用程序是使用 Java 作为主要开发语言的。如果需求中涉及到性能要求较高的运算、图形处理或者需要使用已经存在的 C/C++类库，通过 Java 实现的话，则可能带来运行效率低下、C/C++类库移植成本过高等问题。因此，在开发过程中，我们可以使用 C/C++实现底层模块，并通过 JNI（Java Native Interface）接口实现底层模块与 Java 层的互相调用。

Android 提供的 NDK（Native Development Kit）中包含的交叉编译工具用来支持 Java 与 C/C++的相互调用。通过 NDK 甚至可以绕过框架层 SDK 接口的约束，直接使用 C++调用特定的系统功能。自 Android 2.3 （Api Level 9）之后，新增 android.app.NativeActivity 类（通过调用预定义的 JNI 接口实现），这就意味着可以使用 C/C++完成整个应用程序的开发。

由于 Android 中大多数的控件并没有提供 Native 的实现，如果需要进行大量的 UI 开发工作（使用游戏引擎开发游戏除外），则大大降低了开发效率。

一种理想情况下的设计思想是，使用 Java 作为上层 UI 的实现，使用 C/C++作为业务层面的实现，通过 Android 的交叉编译工具完成整个应用程序的开发。

由此带来的好处是，执行效率的提升以及对核心业务的保护。效率的提升是由 C/C++ 本身的执行效率决定的，而对核心业务的保护则是由 C/C++ 反编译的难度较高、反编译后文件难于阅读的特性决定的。

但是由此带来的开发效率降低、维护和调试难度增大，甚至会造成兼容性下降等因素也是不得不考虑的。

框架层（Application Framework）

框架层是 Android 系统中最为核心的部分，也是 Android 系统设计思想集中体现的部分。

框架层提供了大量的 API 供开发者调用，而弄清楚这些 API 的具体功能和用法则是 Android 应用程序开发过程中最为重要的环节。

框架层不只是应用程序开发的基础，也是软件复用的重要手段，任何一个应用程序都可以发布它的功能模块——只要发布时遵守了框架层的约定，那么其他的应用程序也可以使用这个模块。

框架层由多个系统服务（System Service）组成，包括组件管理服务、窗口管理服务、地理

信息服务、电源管理服务、通话管理服务等。所有的系统服务都寄宿在系统核心进程（System Core Process）中，在运行时，每个服务都占据一个独立的进程，彼此之间通过进程间通信机制（IPC，Inter-Process Communication）发送消息以及传输数据。

对于开发者而言，最为直观的框架层体现就是 Android SDK，它通过一系列的 Java 功能模块来实现应用所需要的可复用的组件，提供了应用开发的规范，屏蔽了应用层与底层交互的复杂性。

框架层的设计决定了上层应用程序的开发模式、开发效率以及能够实现的功能范畴。

从系统运行的角度来看，Android 期望框架层是所有应用程序运行的核心，参与到应用层的每一次操作中，并进行全局统筹。Android 应用的最大特征是基于组件的设计方式，每个应用程序都由若干个组件构成，组件与组件之间通过框架层的系统服务集中的调度和传递信息。

框架层主要是使用 Java 和 JNI 实现的，位于该层的主要组件如下：

视图系统（View System）提供了可扩展的用于构建应用程序 UI 的控件，包括 ListView、GridView、TextView、Button、WebView 等。

内容提供器（Content Provider）提供应用程序之间数据共享的接口。

资源管理器（Resource Manager）提供非代码资源的访问接口，如字符串、布局文件等。

通知管理器（Notification Manager）用来在状态栏显示自定义的 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通用的导航回退功能。

窗口管理器（Window Manager）管理所有的窗口程序。

包管理器（Package Manager）提供 Android 系统内的程序管理。

框架层是基于 Android 核心类库实现的 Android 系统框架层的 API，也就是应用开发人员所使用的 Android 开发包。Android 开发包以 android.开头，其实现位于framework\base\目录中，其中framework \base\core\java包含了大部分 API，还有另外一部分 API 属于Android 系统库中的扩展库部分，分别位于：

framework\base\graphics\java\
framework\base\media\java\

framework\base\opengl\java\
framework\base\wifi\java\
framework\base\location\java\

Android 运行时（Android Runtime）

Android 运行时有两部分组成：Android 核心类库和 Dalvik 虚拟机。其中核心类库提供了 Java 语言核心库所能使用的绝大部分功能，包括Java 对象库、文件管理库、网络通信等。Dalvik 虚拟机则提供了 Android 应用程序的运行环境，负责动态解析执行应用、分配空间、管理对象生命周期等工作。

Android 运行时使得 Android 设备从本质上与一个移动 Linux 实现区分开来。

Dalvik 虚拟机

与传统的 Java 虚拟机不同的是，一方面，Dalvik 虚拟机使用新的二进制码格式文件.dex，而不再使用传统虚拟机的二进制码作为其编译的中间文件（.class）。在 Android 应用程序的编译过程中，Android 会对部分.class文件中的指令做转义处理，使用 Dalvik 虚拟机特有的指令集 OpCodes 替换，并将所有的.class文件统一转换成.dex文件。.dex文件会整合.class文件中的重复信息，并对冗余部分做全局的优化，合并重复的常量定义，节约常量池的资源开销，另一方面，Dalvik虚拟机放弃了传统虚拟机基于栈的指令架构，采用基于寄存器的指令架构设计，以确保更为高效的执行效率和硬件能力。

基于堆栈的指令架构相对于基于寄存器的架构，其指令更为紧凑。Java 虚拟机使用的指令只占据一个字节，因而被称作字节码。基于寄存器的指令架构由于需要指定源地址和目标地址，因而需要更多的指令空间（Dalvik 虚拟机的某些指令只要占据两个字节码）。一般来说，执行同样的代码，前者需要更多的指令，而后者则需要更多的指令空间。需要更多的指令意味着需要占用更多的 CPU 时间，而需要更大的指令空间则意味着数据缓冲（d-cache)更容易失效。

此外，基于寄存器的设计则更有利于进行 AOT（AHead-Of-Time）优化。AOT 优化就是在解释型语言运行之前，就先将它编译成本地机器语言。

每个 Android 应用都运行在单独的 Dalvik 虚拟机中（每个 Android 进程对应一个虚拟机进程），每个 Dalvik 虚拟机则运行在独立的 Linux 进程空间中。每个 Android 进程之间通过进程间通信机制（IPC，Binder 的实现机制）进行通信，而虚拟机的线程管理、内存管理则是依赖于 Linux 内核实现的。

不同的应用程序最后在不同的 Linux 进程空间中运行，即使其中一个虚拟机进程意外关闭或

终止，不会对其他的虚拟机进程造成影响，从而最大限度的保证了应用程序的独立运行和隔离。

Android 核心类库(Core Libraries)

通常情况下，Android 应用程序使用 Java 语言编写，其大部分 Java 语言基础功能都由 Android 核心库提供，比如基础数据结构、数学、I/O、工具、数据库、网络等库。其中大部分实现来源于 ApacheHarmony 项目，核心库的具体实现位于libcore目录中，Java 部分最终会被打包为core.jar包，经过安装，最终将被放置在目标文件系统的system\framework\目录中，当桌面启动时首先加载，作为 Java 程序的一个基础包。

libcore中的 C/C++ 代码被编译为libjavacore.a静态库，是 Java 核心库的本地代码。

另外，libcore目录中还包括部分测试用例，用来测试 Java 核心库的基本接口功能实现，在移植 Android 或者其虚拟机时，也可以使用它们来测试 Java 核心库的功能。

核心库主要实现了以下 Java 基础包：

Java 标准API（java.*）
Java 扩展 API（javax.*）
企业和组织提供的 Java 类库（org.*）
以及以下 Android 的核心 API：
android.os
android.net
android.media

类库(Libraries)

核心类库有一系列的二进制动态链接库组成，通常使用 C/C++进行开发。与框架层服务相比，核心类库不能独立运行于线程中，而是通过系统服务将其加载到其所在的进程空间中，并通过类库提供的 JNI 接口进行调用。

下面是常见的系统类库：

- Surface Manager 执行多个应用程序的时候，负责管理显示与存取操作间的互动，另外也负责2D 与3D 图层的无缝整合。
- Media Framework 多媒体支持库，基于 PacketVideo 的 OpenCore，支持多种常见的音视频格式的录制和播放，支持的编码格式包括 MPEG4、MP3、H.246、AAC、ARM。

- SQLite 轻量级的关系型数据库。
- OpenGL ES 基于OpenGL ES 的3D 绘图函数库。
- Free Type 位图与向量字体的显示。
- WebKit 网页浏览器的解析引擎。
- SGL底层的2D图形渲染引擎。
- SSL 在 Android 通信过程中实现握手。
- Libc 从 BSD 派生出来的标准 C 函数库。

硬件抽象层（HAL）

硬件抽象层（HAL，Hardware Abstraction Layer）是介于内核和 Libraries 层中间的，抽象出来的一层结构。Android 的硬件抽象层可以以闭源源码的形式提供硬件驱动模块。HAL 存在的目的是为了把 Framework 层与 Linux 内核隔离开，使得 Android 不会过度依赖 Linux 内核，以达到“内核独立”的目的。它是对 Linux 驱动的封装，对上层服务提供统一的接口，屏蔽了底层的实现细节。

台湾著名架构师高焕堂老师对 HAL 的的理解则是“为了一些硬件提供商提出的‘专利保护’的硬件驱动程序而产生的，简而言之，就是为了避开 Linux内核的 GPL 协议的束缚”。

关于 Android 增加 HAL 层的更多内容请参考 Android ，在争议中逃离 Linux 内核的 GPL 约束。

Linux 内核

Android 最早的版本是基于 Linux2.6 内核提供核心系统系统服务的，例如安全、内存管理、进程管理、网络堆栈、驱动模型等。Linux 内核作为硬件和软件之间的抽象层，屏蔽掉硬件的差异性并为上层提供统一服务。

参考文献

《疯狂 Android 讲义》
《Android 开发精要》
《Android 的设计与实现》
《Android 4高级编程（第三版）》
麦子学院 Android 架构之 HAL 的商业意义(高焕堂)
RednaxelaFX 的 虚拟机随谈（一）：解释器，树遍历解释器，基于栈与基于寄存器，大杂烩
罗升阳的 Dalvik虚拟机简要介绍和学习计划

安卓应用频道

微信号：**AndroidPD**

打造东半球最好的 安卓技术 微信号

商务合作QQ：2302462408

投稿网址：top.jobbole.com

微信扫一扫
关注该公众号