移动应用软件开发

本次课内容：Android基础

王继良
软件学院
时间：周三（2）

目录

移动应用软件开发基础（以安卓为例）

系统框架——Linux内核和HAL

系统框架——系统运行库层

系统框架——应用架构层

系统框架——应用层

Android 项目结构

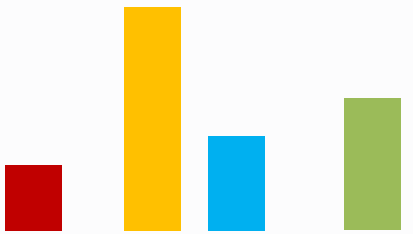
Android 应用构建

Android 是什么?

- 基于Linux 内核的移动操作系统
- 触摸屏用户界面
- 用于超过 80%的智能手机
- 为手表、电视和汽车等设备提供系统支持
- 具有高可定制性，可以满足不同设备和供应商的需求
- 开源

移动和嵌入式操作系统

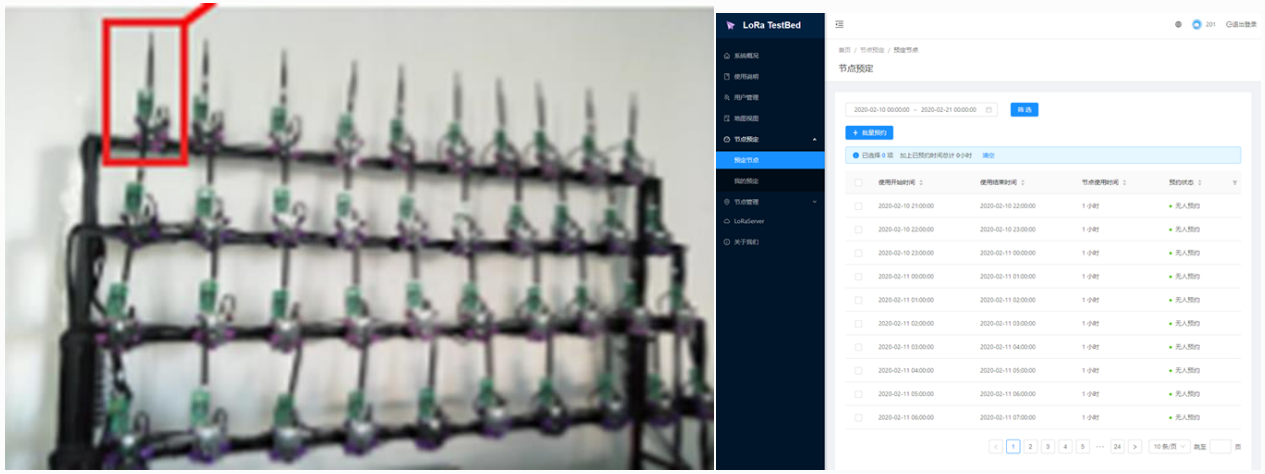
- 大家听过哪些？请大家弹幕说一说。



移动和嵌入式操作系统

- Windows CE
- OpenWrt
- Free RTOS 实时嵌入式操作系统
- RT-Thread
- TinyOS
- Contiki
- UC/OS
- 腾讯物联网操作系统（TencentOS tiny）
- AliOS Things
- Huawei LiteOS
- 不需要操作系统
- …

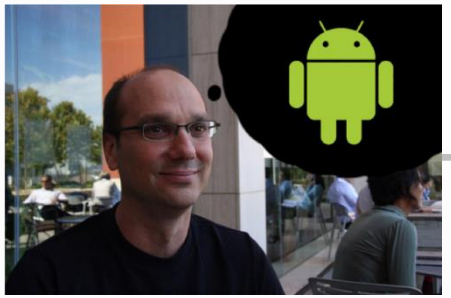
portable, scalable, preemptive, real-time, deterministic, multitasking/multi-threading,



从一个最简单的程序说起

- 你印象中的一个最简单的嵌入式系统的程序

Android发展历程



2003.10

Andy Rubin等人创建了与Android系统同名的Android公司，并组建了Android开发团队，最初的Android系统是一款针对数码相机开发的智能操作系统。

2005.08

Android被Google公司低调收购，并聘任Andy Rubin为Google公司工程部副总裁，继续负责Android项目。

2007.11

谷歌公司正式向外界展示了这款名为Android的操作系统，并且宣布建立一个全球性的联盟组织，该组织由34家手机制造商、软件开发商、电信运营商以及芯片制造商共同组成，并与84家硬件制造商、软件开发商及电信营运商组成开放手持设备联盟（Open Handset Alliance）来共同研发改良Android系统，这一联盟将支持谷歌发布的手机操作系统以及应用软件，Google以Apache免费开源许可证的授权方式，发布了Android的源代码。

2008.9

谷歌正式发布了Android 1.0系统，这是Android系统最早的版本。

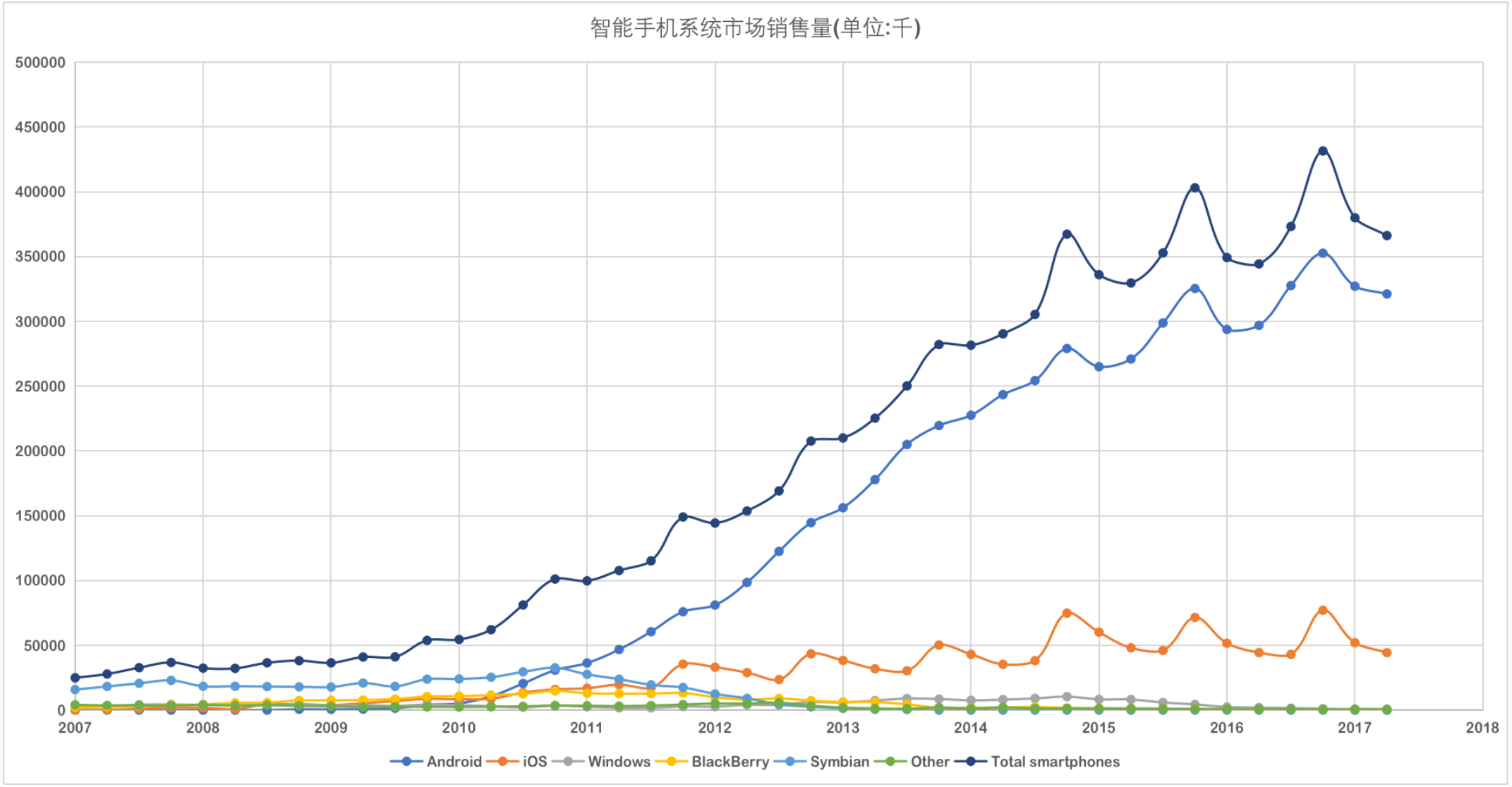
2009.04

谷歌发布了Android1.5系统，命名为Cupcake（纸杯蛋糕），该系统与Android1.0系统相比有了很大的改进，被外界认为是真正的智能操作系统。

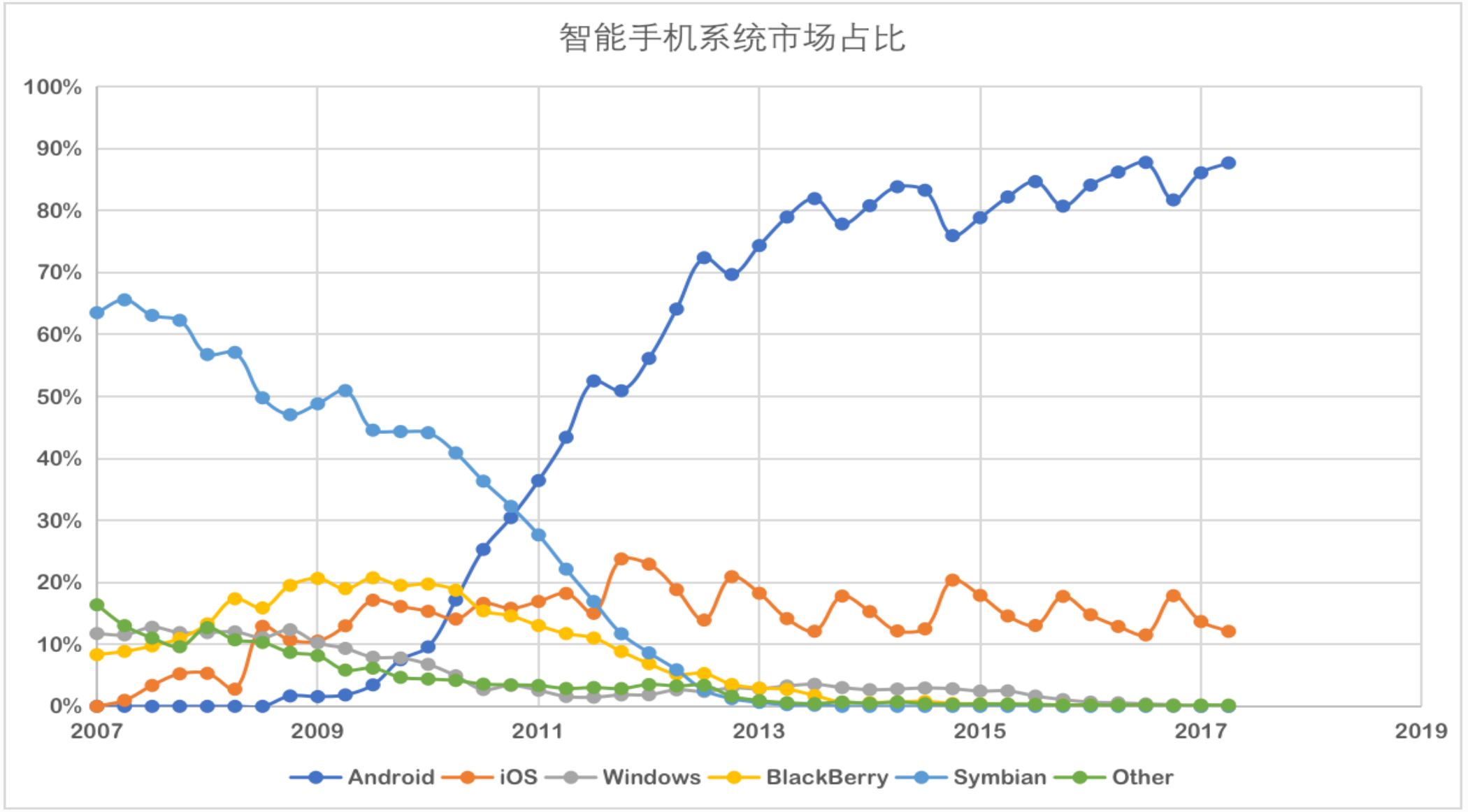
Android系统发布时间

Android版本	发布日期	代号
Android 1.0	2008年9月23日	无
Android 1.5	2009年4月30日	Cupcake（纸杯蛋糕，第一次采用甜点命名）
Android 2.0	2009年10月26日	Eclair（长松饼）
Android 2.2	2010年5月20日	Froyo（冻酸奶）
Android 3.0	2011年2月22日	Honeycomb（蜂巢）
Android 4.0	2011年10月19日	Ice Cream Sandwich（冰激凌三明治）
Android 4.2	2012年10月8日	Jelly Bean（果冻豆）
Android 5.0	2014年10月15日	Lollipop (棒棒糖)
Android 6.0	2015年5月28日	Marshmallow（棉花糖）
Android 7.0	2016年3月10日	Nougat（牛轧糖）
Android 8.0	2017年8月22日	Android Oreo（奥利奥）
Android 9.0	2018年8月7日	Pie（派）
Android 10.0	2019年9月3日	无（正式取消甜点命名）

智能手机系统销售量


智能手机系统市场销售量(单位:千)
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
Android
iOS
Windows
BlackBerry
Symbian
Other
Total smartphones

智能手机系统市场占比


智能手机系统市场占比
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
2007
2009
2011
2013
2015
2017
2019
Android
iOS
Windows
BlackBerry
Symbian
Other

Android系统迅速发展的原因

1. **安装软件方便**
   Android用户只需要下载一个apk包就可以安装软件。

2. **生态开放**
   安卓不仅仅开源了系统的源码，而且对开发者没有门槛要求。

3. **开发简单**
   安卓统一了自己的硬件风格，这大大减少了开发者的工作量。

目录

Android简介

系统框架——Linux内核和HAL

系统框架——系统运行库层

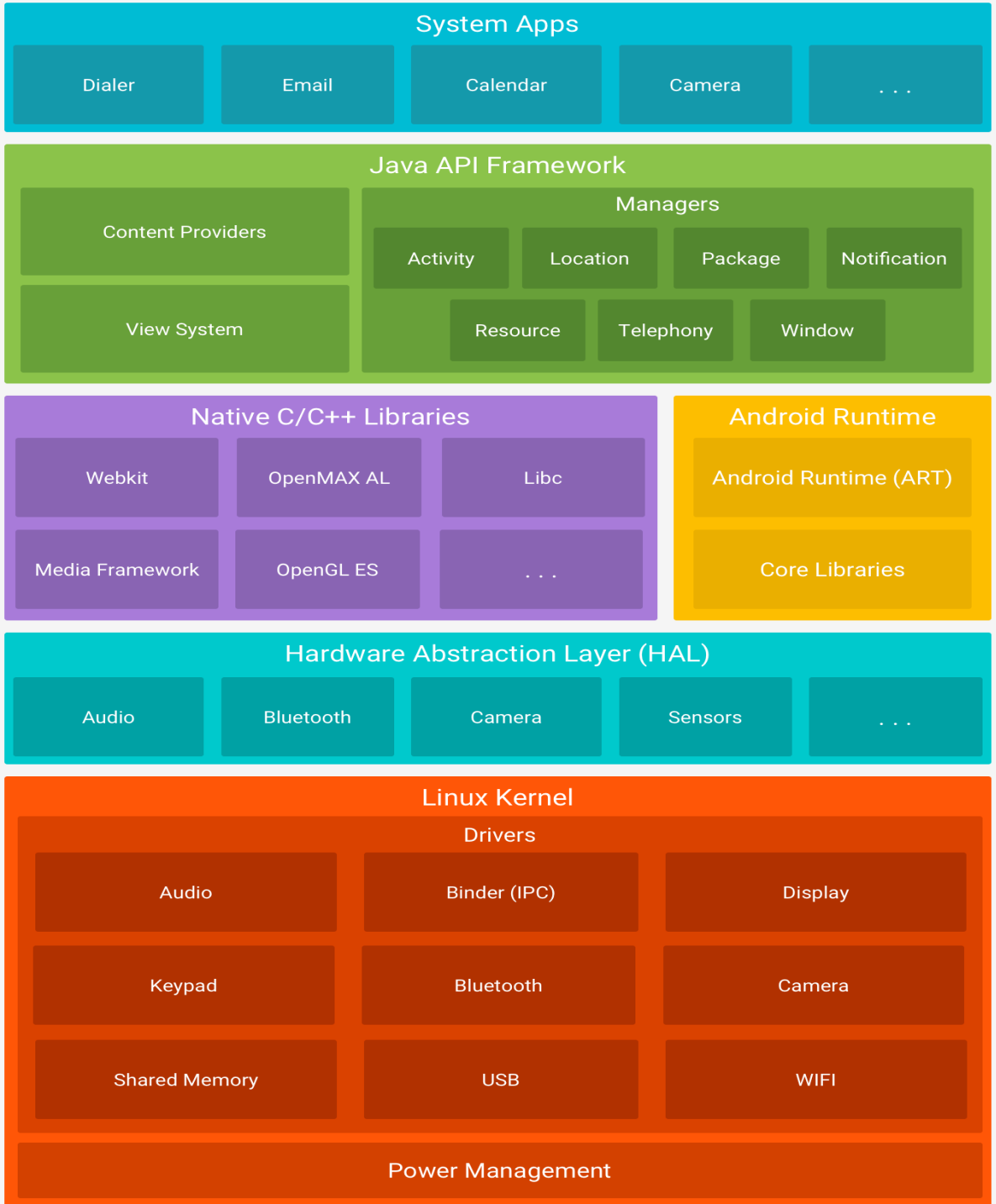
系统框架——应用架构层

系统框架——应用层

Android 项目结构

Android 应用构建

Android系统架构



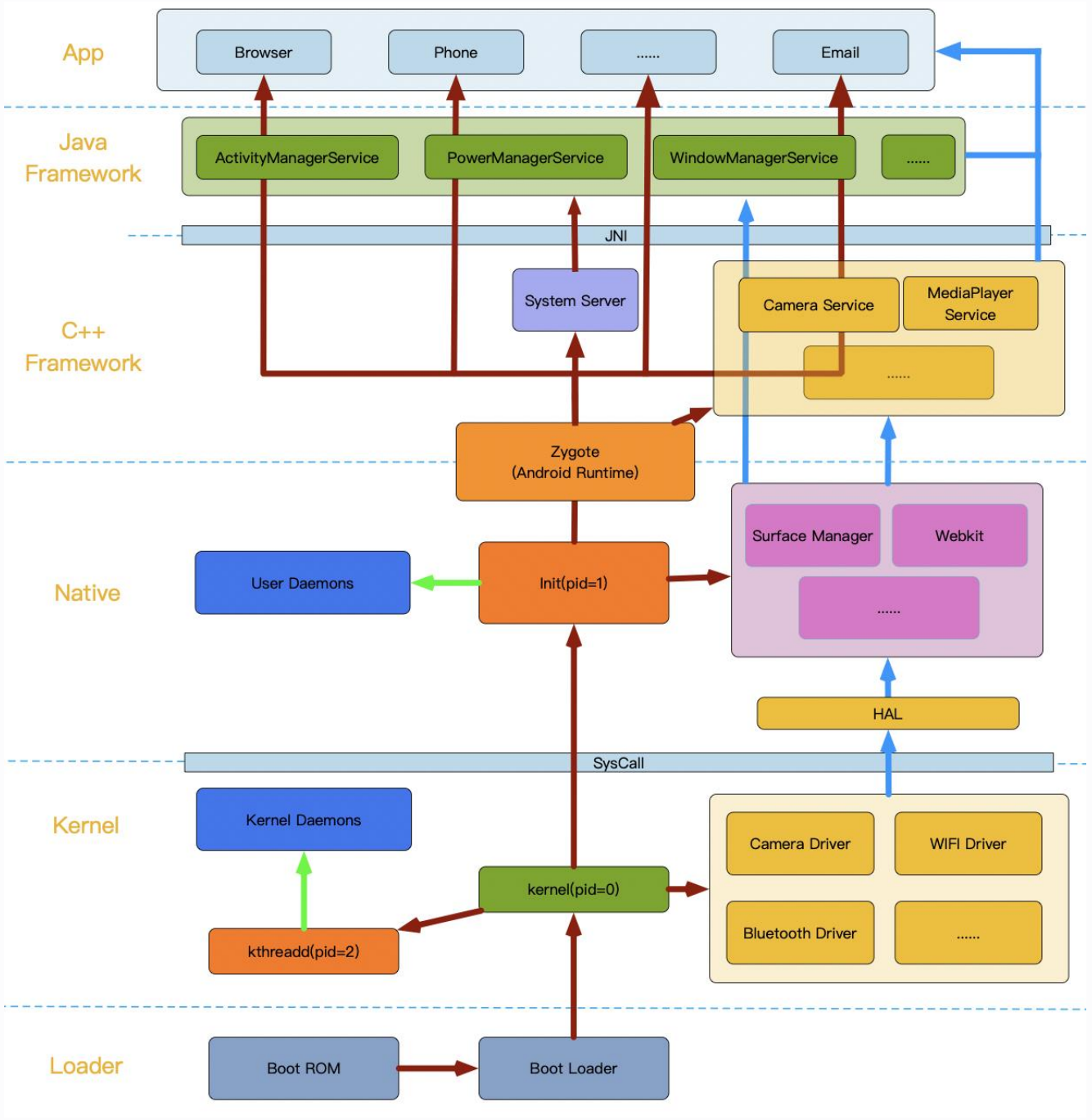
来思考一下如何实现?

系统启动过程

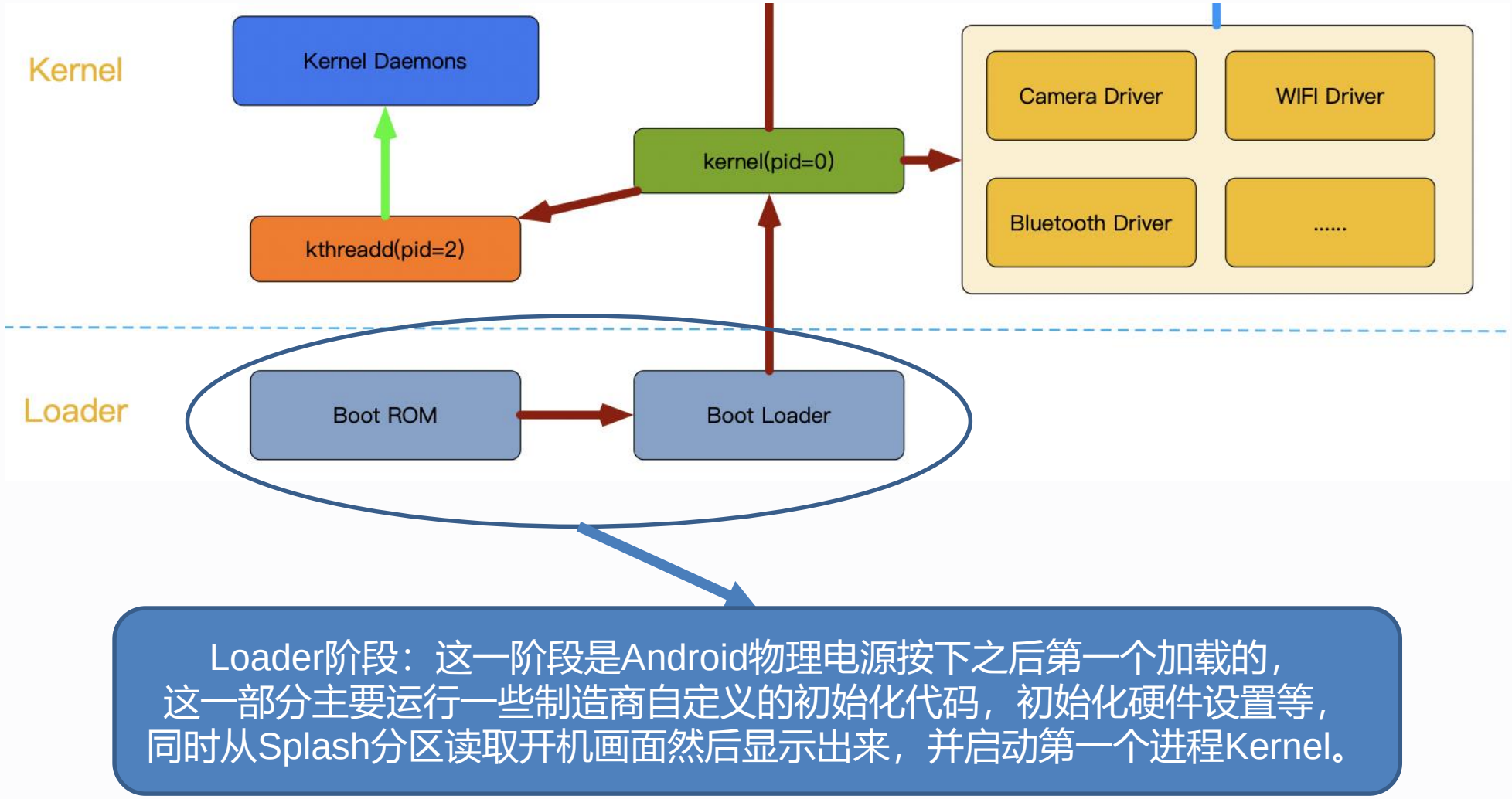
启动关系

管理关系

提供接口的关系



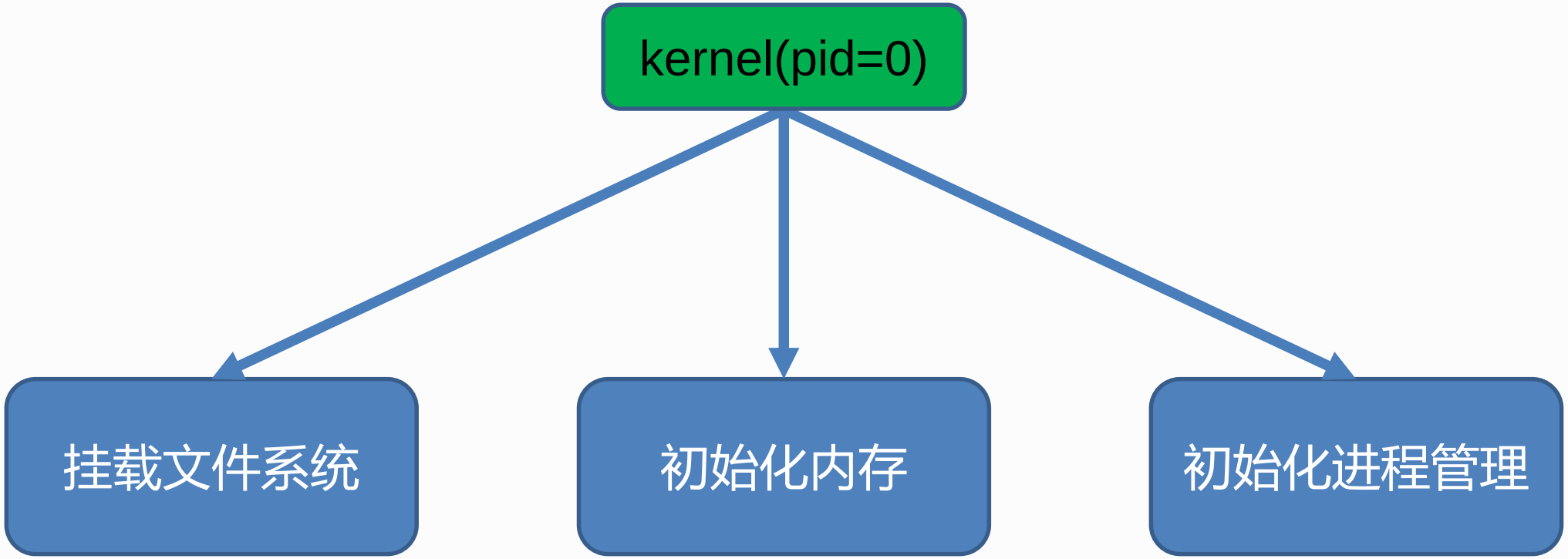
Linux内核层



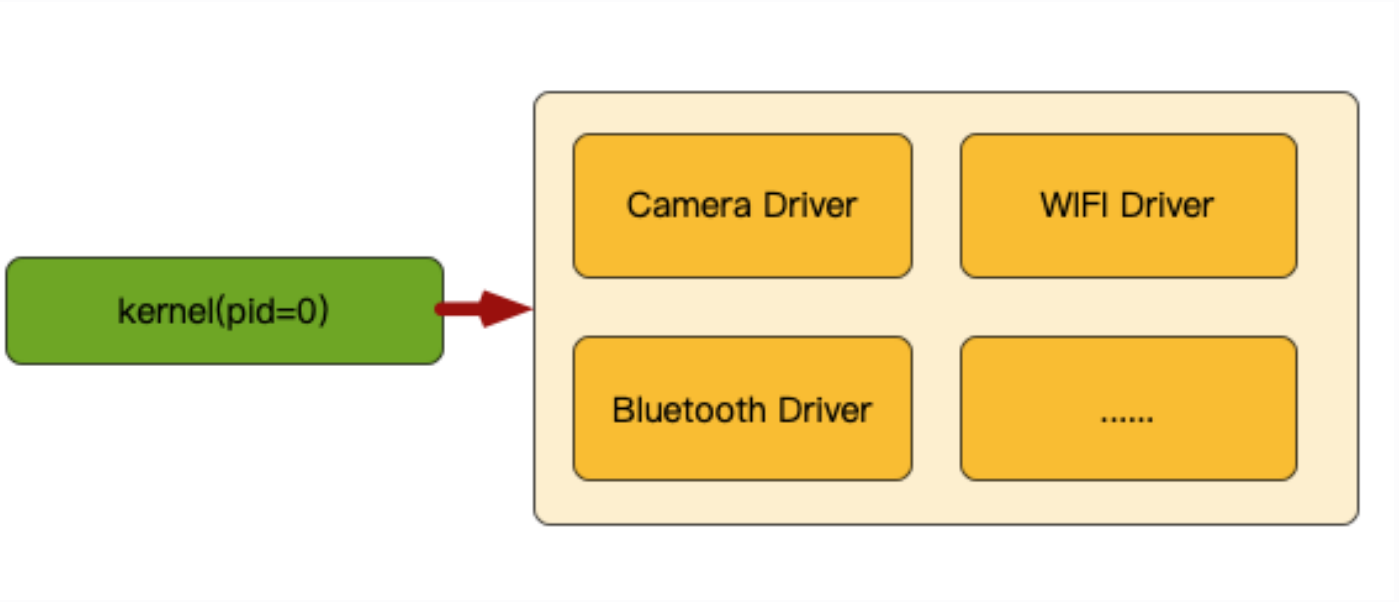
Loader阶段：这一阶段是Android物理电源按下之后第一个加载的，
这一部分主要运行一些制造商自定义的初始化代码，初始化硬件设置等，
同时从Splash分区读取开机画面然后显示出来，并启动第一个进程Kernel。

Linux内核层

系统第一个内核进程

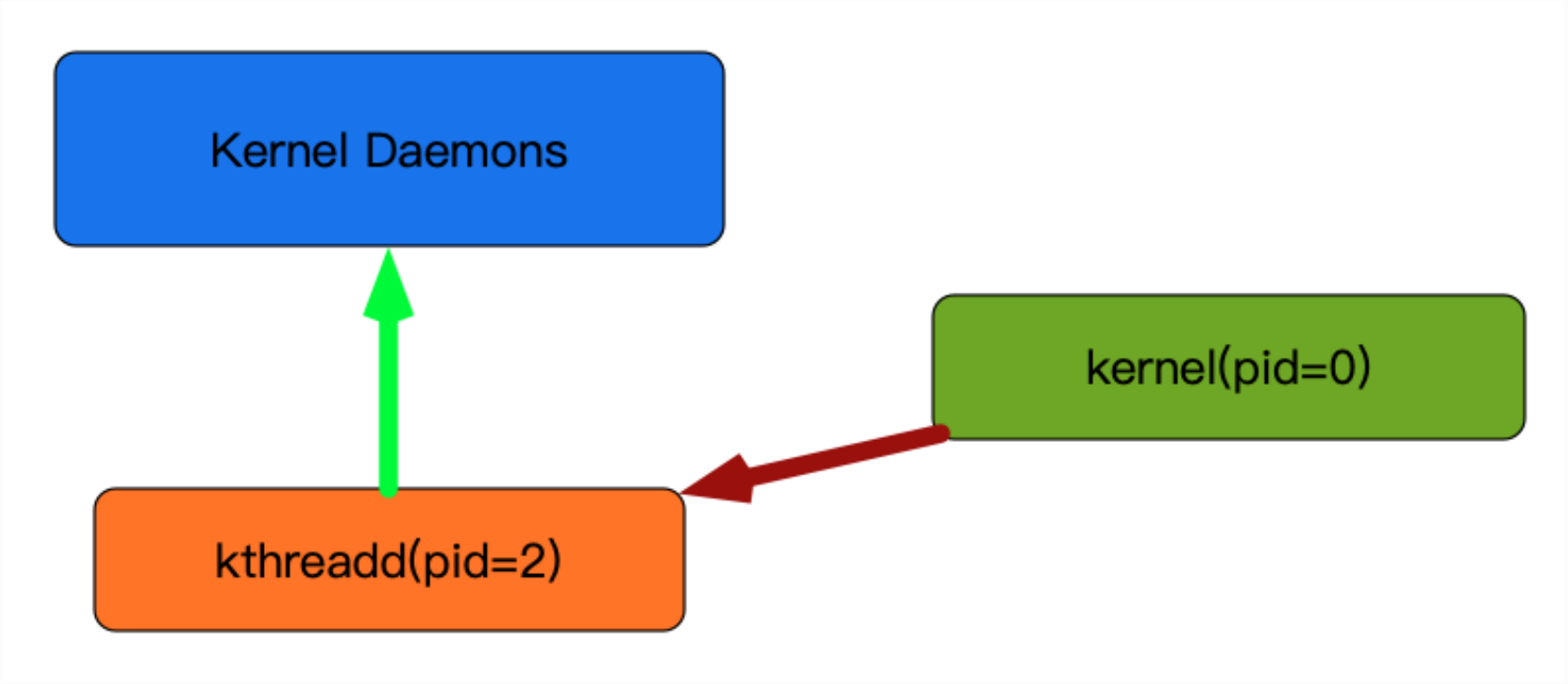


Linux内核层



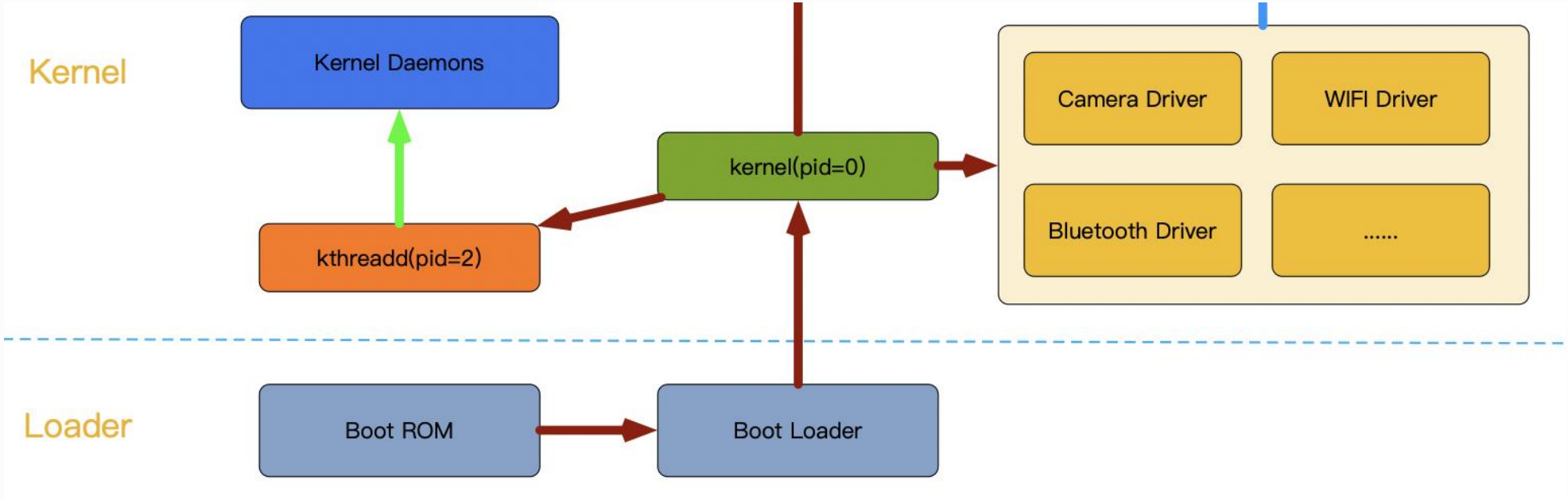
kernel还负责完成驱动的加载，这些驱动为硬件抽象层的实现提供了接口，让上层可以控制相机、wifi、蓝牙等硬件。

Linux内核层



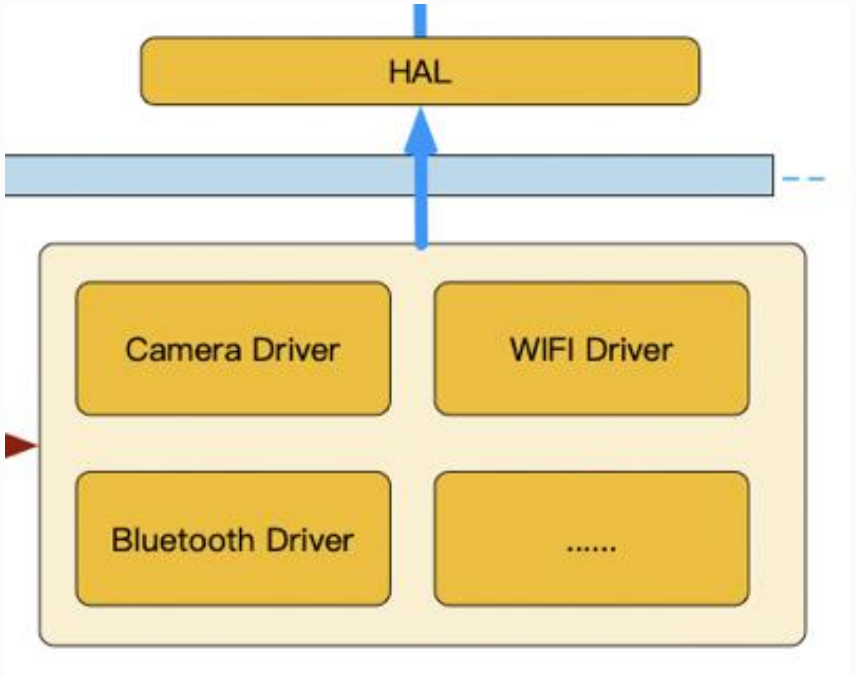
- kernel会在此时创建一个kthreadd进程，该进程的编号pid=2。
- Linux进程分为内核进程和用户进程两种，内核进程拥有系统最高权限，可以对Linux进行任意的操作。
- kthreadd进程负责管理Linux内核的所有内核进程，内核进程的创建、休眠、唤醒和销毁都由该进程完成。

Linux内核层



- Linux内核层为Android系统的所有运行提供了最底层的基础，Android系统的内存管理、文件访问、硬件控制、进程管理等最终都需要通过Linux内核来完成。
- 为了兼容移动设备，Android的Linux内核相对于桌面级Linux系统改动很大，删除了很多移动设备上用不到的内容。

硬件抽象层(HAL)



- 使用c语言实现，向上提供硬件访问的服务
- 向下屏蔽硬件驱动模块的实现细节

目录

Android简介

系统框架——Linux内核和HAL

系统框架——系统运行库层

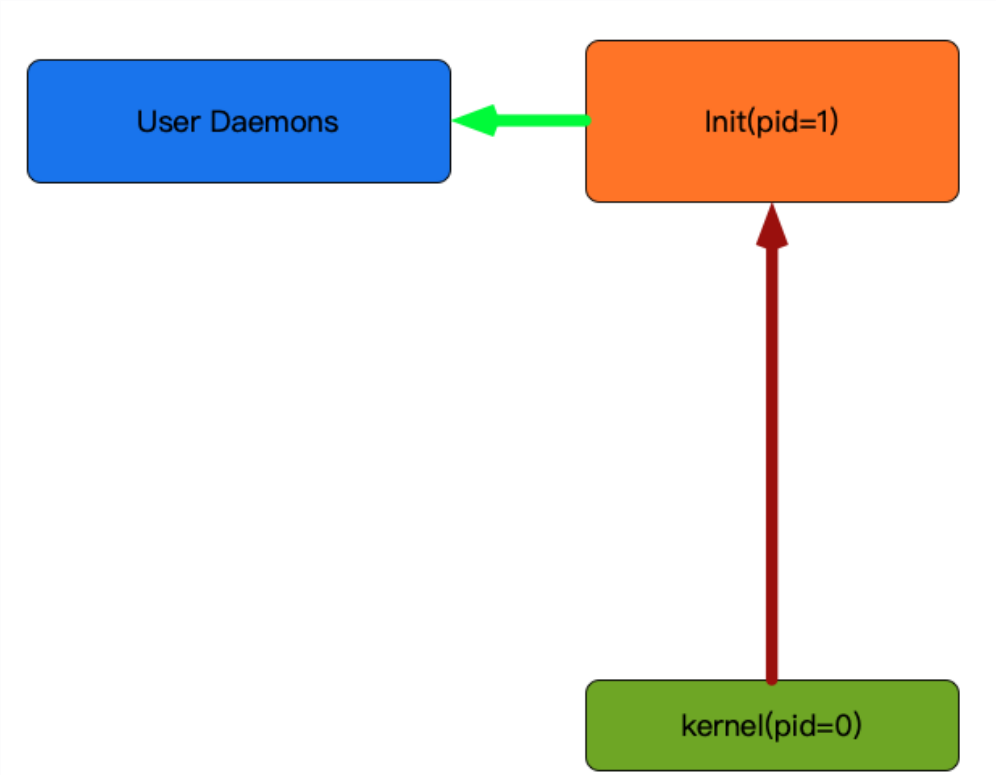
系统框架——应用架构层

系统框架——应用层

Android 项目结构

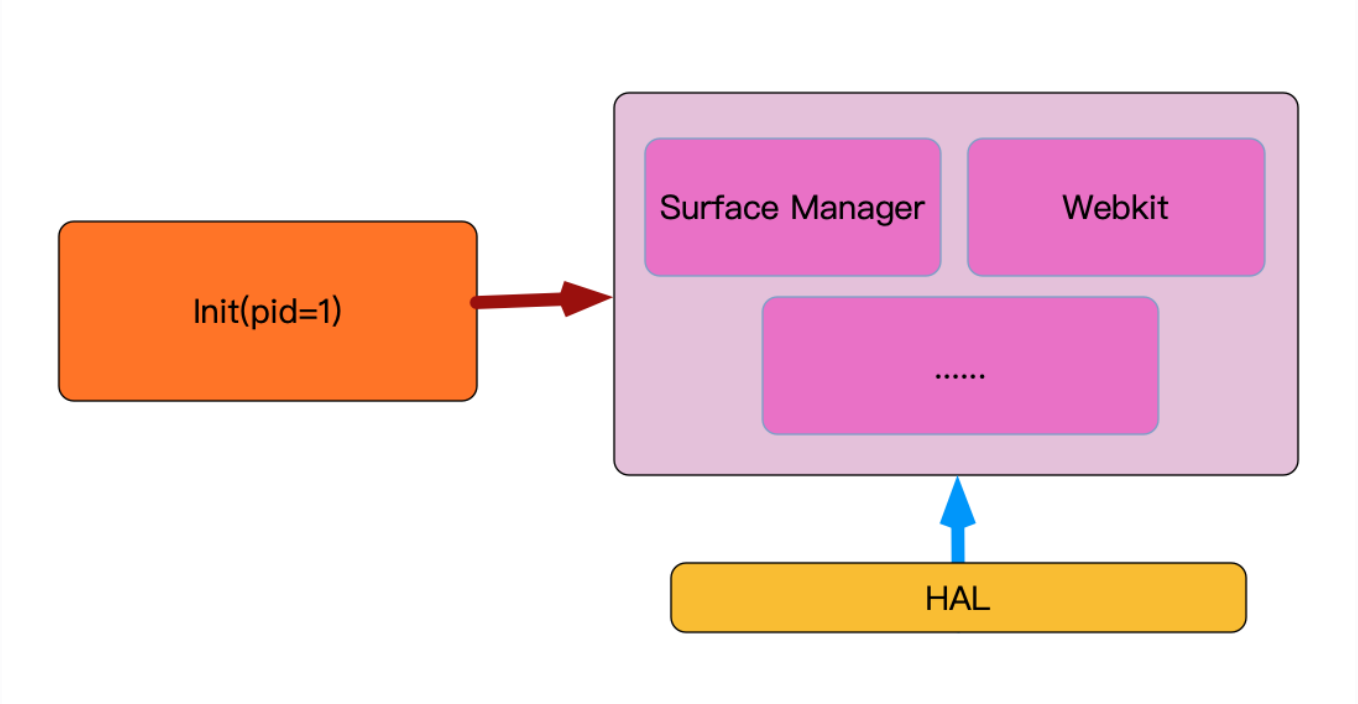
Android 应用构建

系统运行库层



kernel还会创建第一个用户进程init(pid=1)，该进程负责管理所有的用户进程。

系统运行库层



- init进程还会初始化Linux的本地库，这些库都基于C语言实现，并且使用了硬件抽象层提供的接口，这些本地库可以为更上层Android框架层的实现提供接口。
- 此外init还会初始化Android的系统。

比较主要的本地库

Libc库

Linux系统的c库。

Media FrameWork

提供音视频接口。

Surface Manage

主要负责管理显示系统。

WebKit

提供浏览器服务的支持。

SGL

一个内置的2D图形引擎。

SSL

安全套阶层，为数据通信提供支持。

OpenGL | ES

提供3D效果的支持。

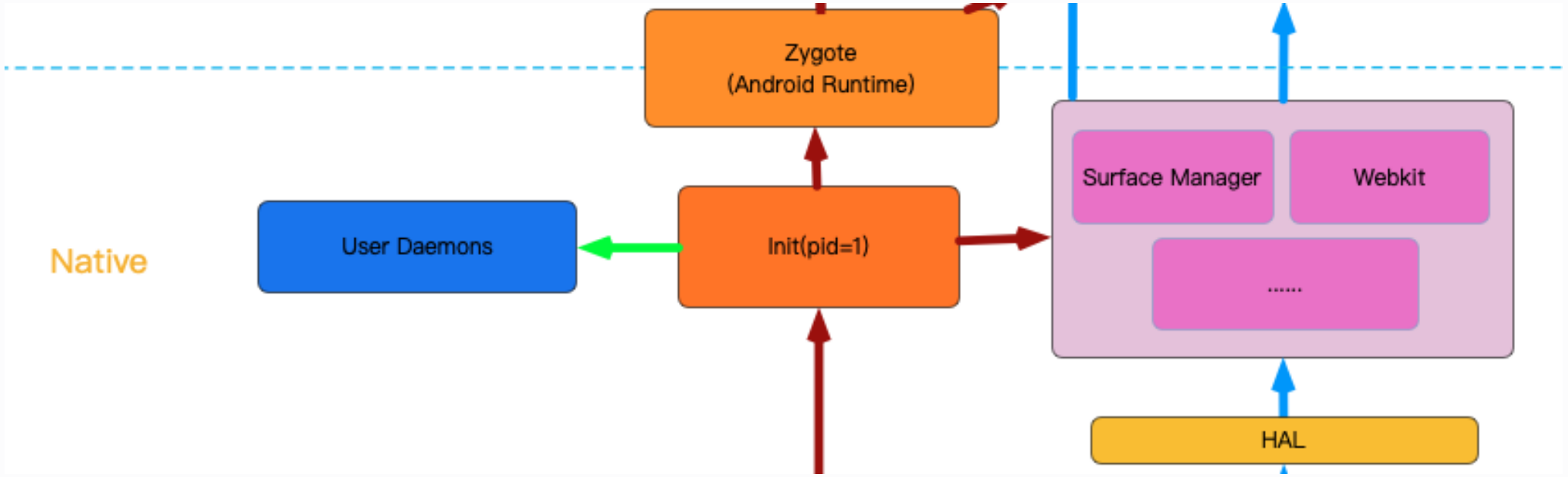
SQLite

轻量级数据库，用于本地存储数据。

FreeType

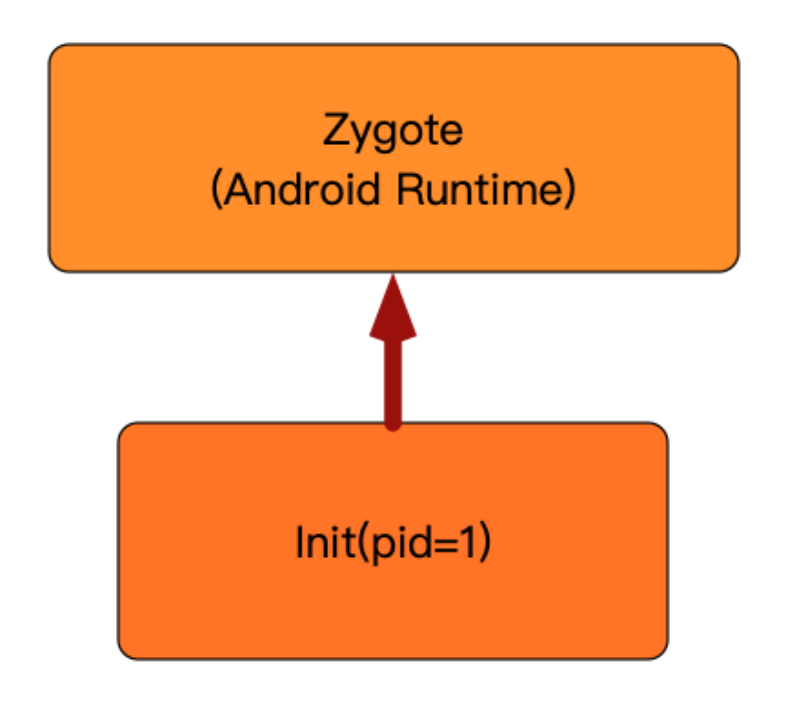
字体引擎，管理字体格式。

系统运行库层



系统运行库层运行在Linux内核上，主要是一些C语言实现的库。在这一层初始化完成以后，Android运行所需要的Linux层环境都已经具备，接下来会初始化Android底层依赖。

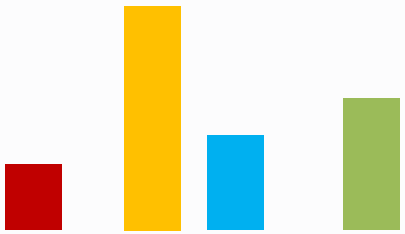
系统运行库层



init会创建一个Zygote进程

Zygote进程作用

- 初始化Android系统环境
- 启动Android Runtime虚拟机，给Java程序执行提供平台
- 启动System Server进程（安卓系统服务）
- 随时准备复制自己以产生新进程



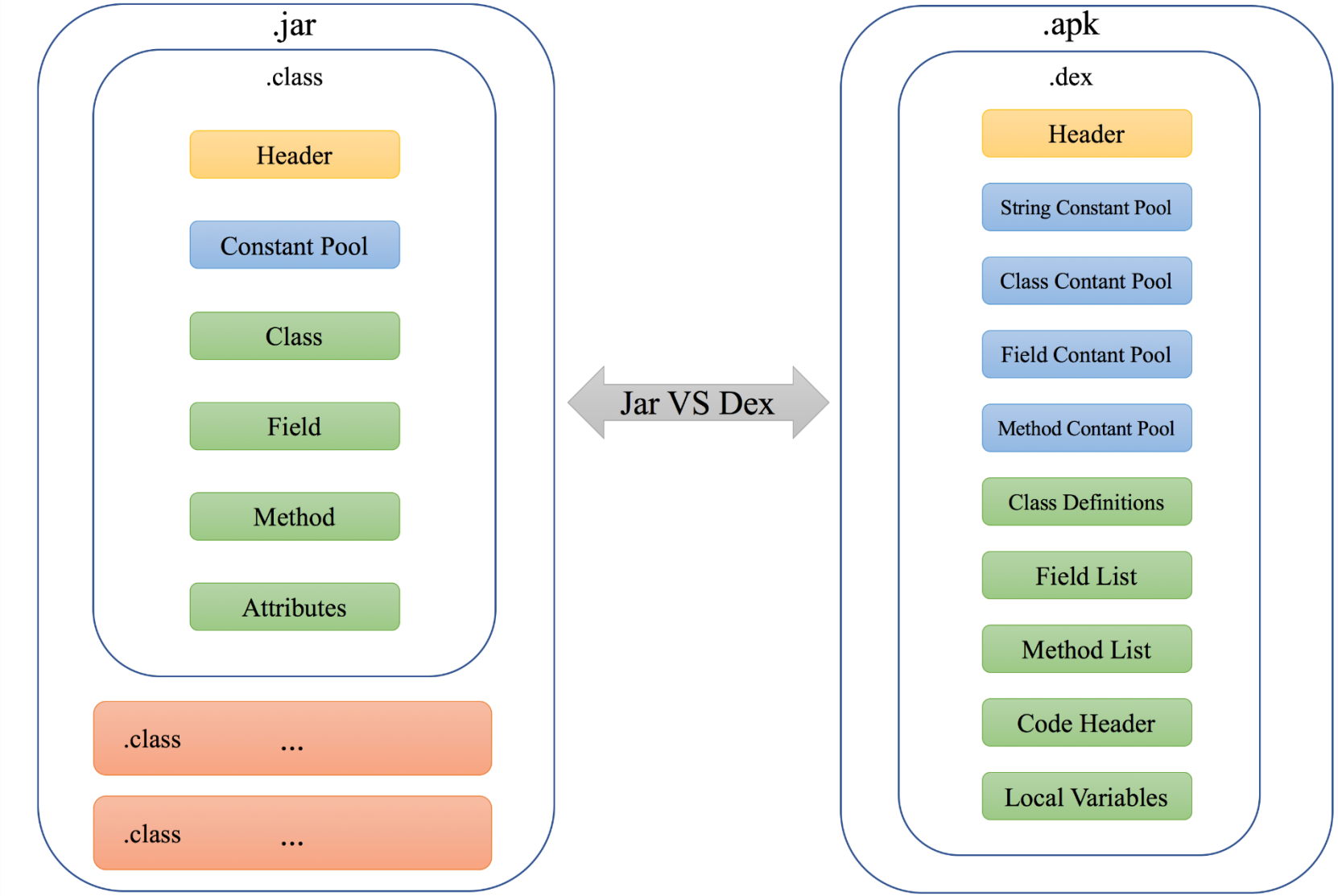
Android Runtime虚拟机简介

- Android Runtime由ART虚拟机和Android核心库组成。
- ART虚拟机是由Zygote进程创建的
- ART虚拟机类似于JAVA虚拟机，它为Android移动平台做了特殊的优化，同时给Android平台上的Java代码提供了运行环境。

ART虚拟机的主要功能

1. 进程管理: 每一个 Android应用在底层都会对应一个独立的Linux进程。
2. 类加载器：解析安卓文件（如dex文件）并加载（如dalvik字节码）
3. 解释器：根据自身的指令集解释字节码
4. 内存管理: 分配系统和应用程序需要的内存资源，并进行垃圾回收。
5. 本地方法调用(Java Native Interface, JNI)：提供接口让Java代码可以调用本地代码。
6. 调试支撑模块：提供开发者对程序的调试支持。

.dex文件详解



一个.class文件对应一个java文件，而一个.dex可能对应多个。

Android Runtime的其它作用

- 实现了Java语言大部分API，比如Java标准API（java包）、Java拓展API（javax包）、企业和组织提供的Java类库（org包），但是也有部分API没有被支持，比如GUI系统的swing等。
- 提供了Android的一些核心API，如android.OS、android.net、android.media等。
- 设计了Android NDK，即Android原生库。Android原生库为开发者提供了直接使用Linux系统资源的接口，即Android开发者可以通过实现JNI，而直接使用C++利用Linux系统资源开发应用。

目录

Android简介

系统框架——Linux内核和HAL

系统框架——系统运行库层

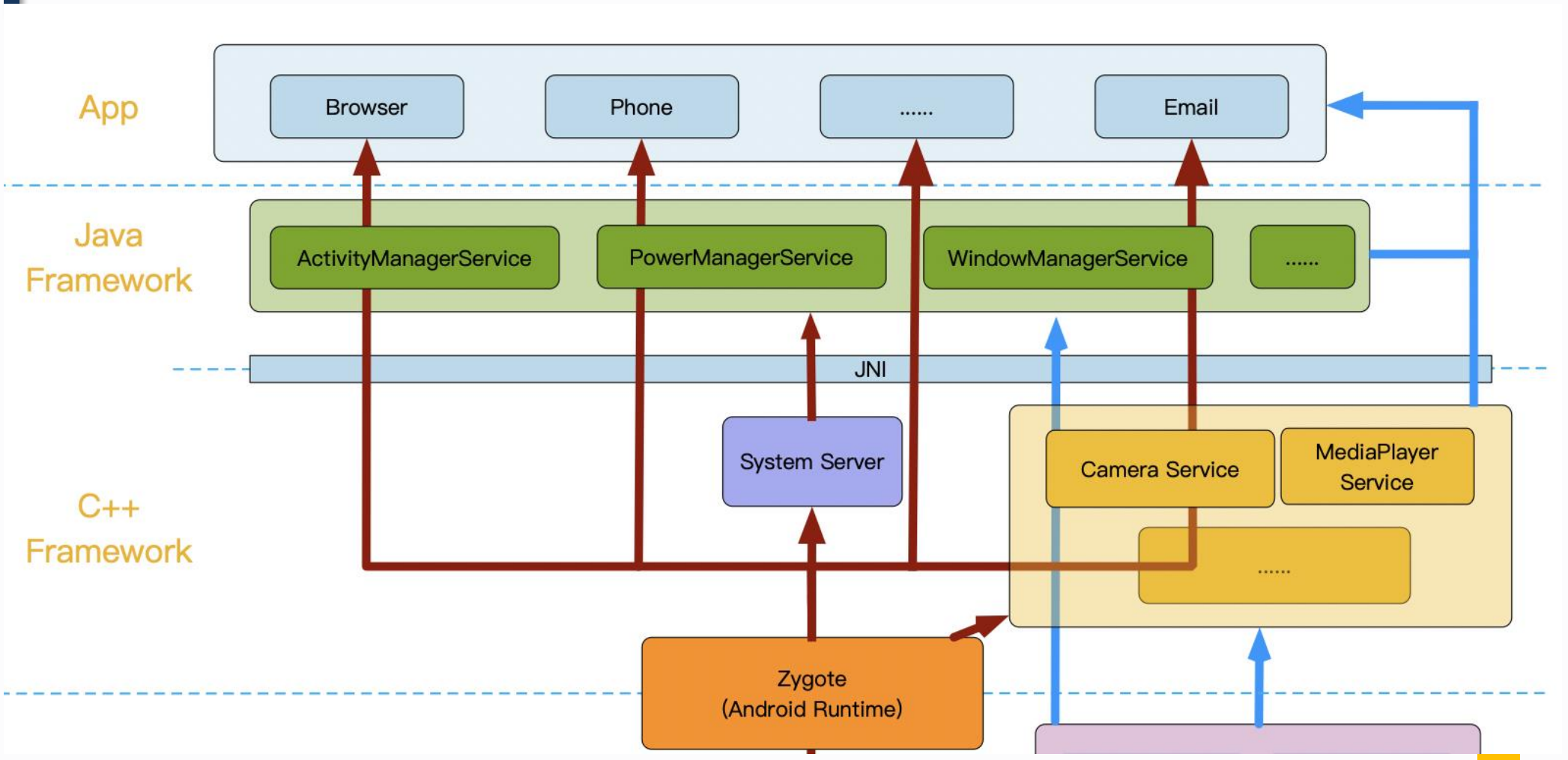
系统框架——应用架构层

系统框架——应用层

Android 项目结构

Android 应用构建

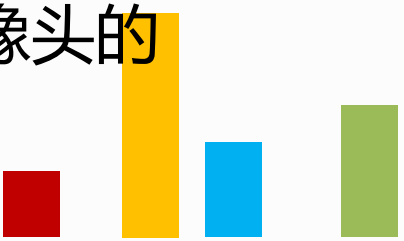
应用架构层



架构层初始化由Zygote完成，分Java Framework和C++ Framework两种。
为app提供了接口，Java层想要调用C++需要通过JNI

应用架构层说明

- 分C++框架和Java框架两种
- 给Android系统和所有App的运行提供了支持
- Java框架运行于ART虚拟机上，C++框架运行于Linux内核上
- C++框架基于系统运行库层中的C库实现
- Java框架都运行在System Server进程中，主要提供系统管理功能，如ActivityMangerService负责管理所有Activity
- C++框架主要提供功能性的支持，如Camera Service负责提供摄像头的功能性支持



应用架构层部分架构说明

Activity Manager

管理Activity的生命周期，并且提供常用的导航回退功能。

Window Manager

提供访问手机屏幕的方法，可以让程序修改屏幕的透明度、亮度、背

Content Providers

让应用程序之间共享数据，使得一个应用程序可以访问另一个应用程

这些就构成了应用的基础，是你写应用中要面对的东西。

提供

种各样的视图控件，如List、Button、Text View等。

装、卸载应用以及控制应用权限，获取应用信息等。

ger

接口，比如获取手机电量信息、sim卡信息等等。

Resource Manager

提供程序对非代码资源的访问，比如本地字符串、图片、布局文件等。

Location Manager

提供对设备进行定位的功能。

Notification Manager

提供消息通知功能。

目录

Android简介

系统框架——Linux内核和HAL

系统框架——系统运行库层

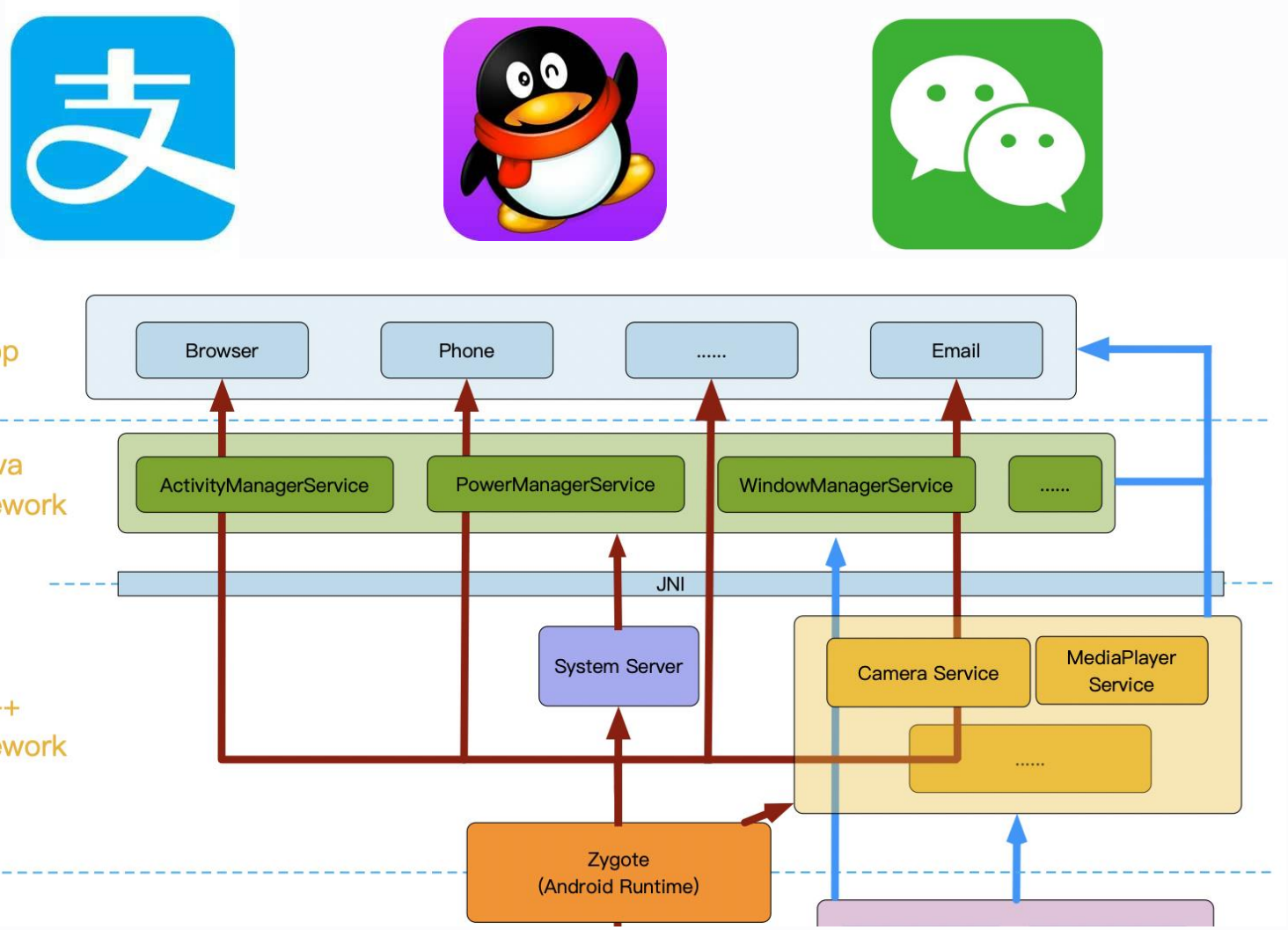
系统框架——应用架构层

系统框架——应用层

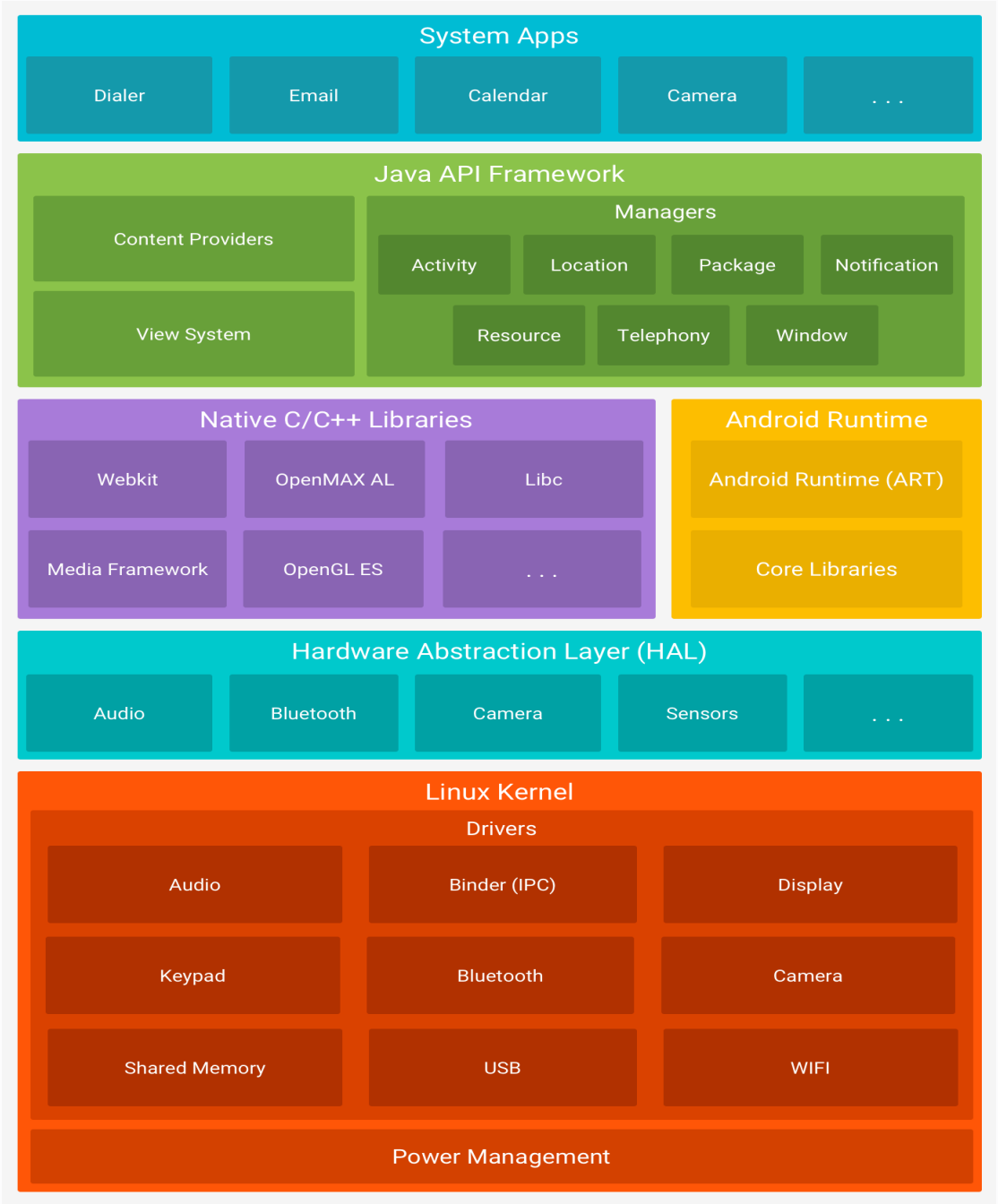
Android 项目结构

Android 应用构建

应用层


App
Browser
Phone
.....
Email
Java
Framework
ActivityManagerService
PowerManagerService
WindowManagerService
.....
JNI
C++
Framework
System Server
Camera Service
MediaPlayer
Service
.....
Zygote
(Android Runtime)

回顾



Android从系统最底层开始，层层分工，层层优化，设计了五层架构，五层架构的分工明确，每一层都有自己独到的作用，是一个良好的系统设计的典范。

目录

Android简介

系统框架——Linux内核和HAL

系统框架——系统运行库层

系统框架——应用架构层

系统框架——应用层

Android 项目结构

Android 应用构建

Android app是什么?

- 一个或多个可交互的屏幕
- 使用 Java 编程语言和 XML 编写
- 使用Android Libraries 和 Android 应用框架
- 由Android Runtime 虚拟机执行（ART）

大家可以想想Java项目一般由哪些文件构成?

Android项目结构

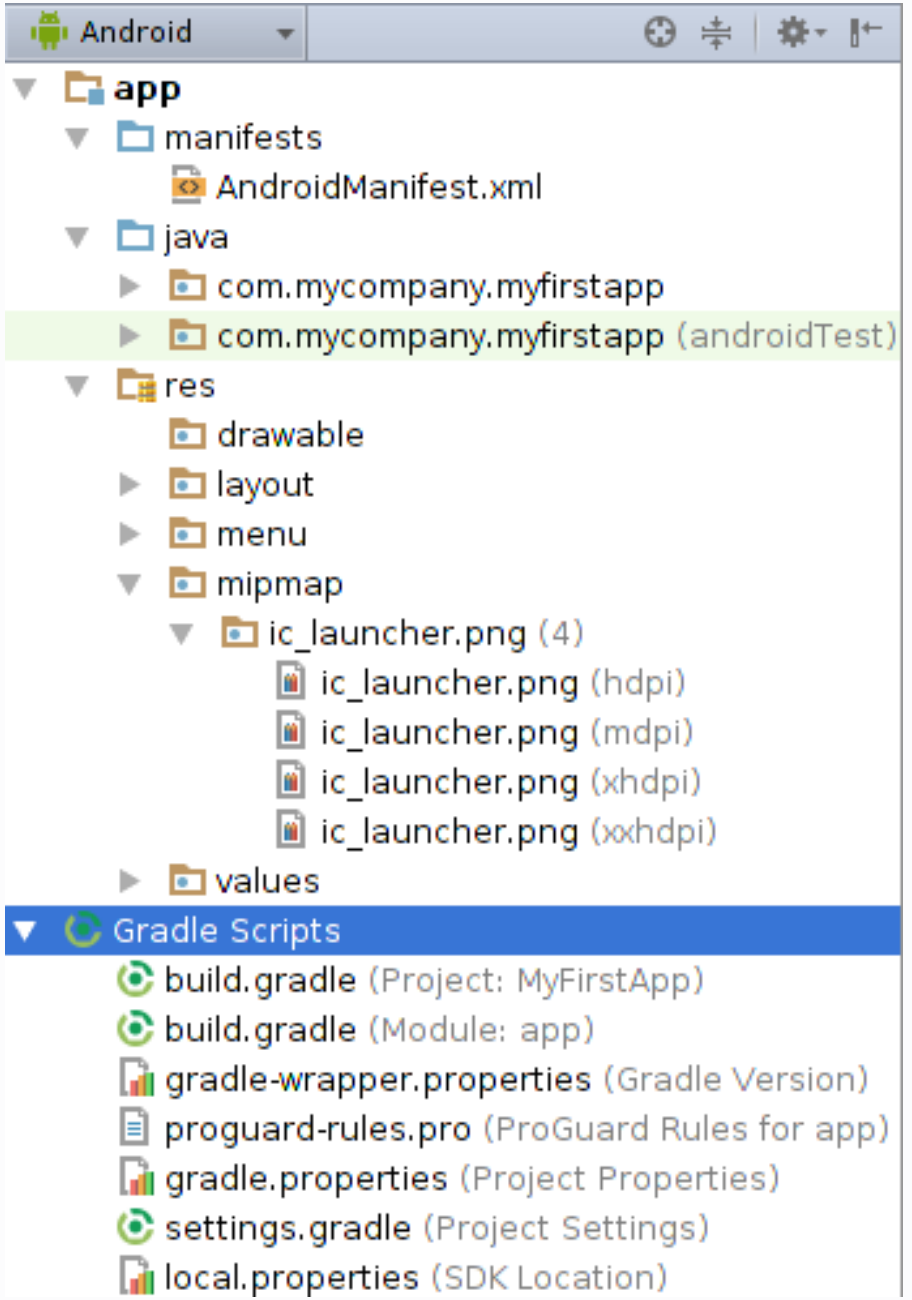
- Android 项目包含应用程序工作空间中的所有内容，包括源代码和资源文件，以及测试代码和构建配置。

- 以Android Studio为例，项目结构展示视图有: Android视图，Project 视图等。

Android项目结构

Android 视图

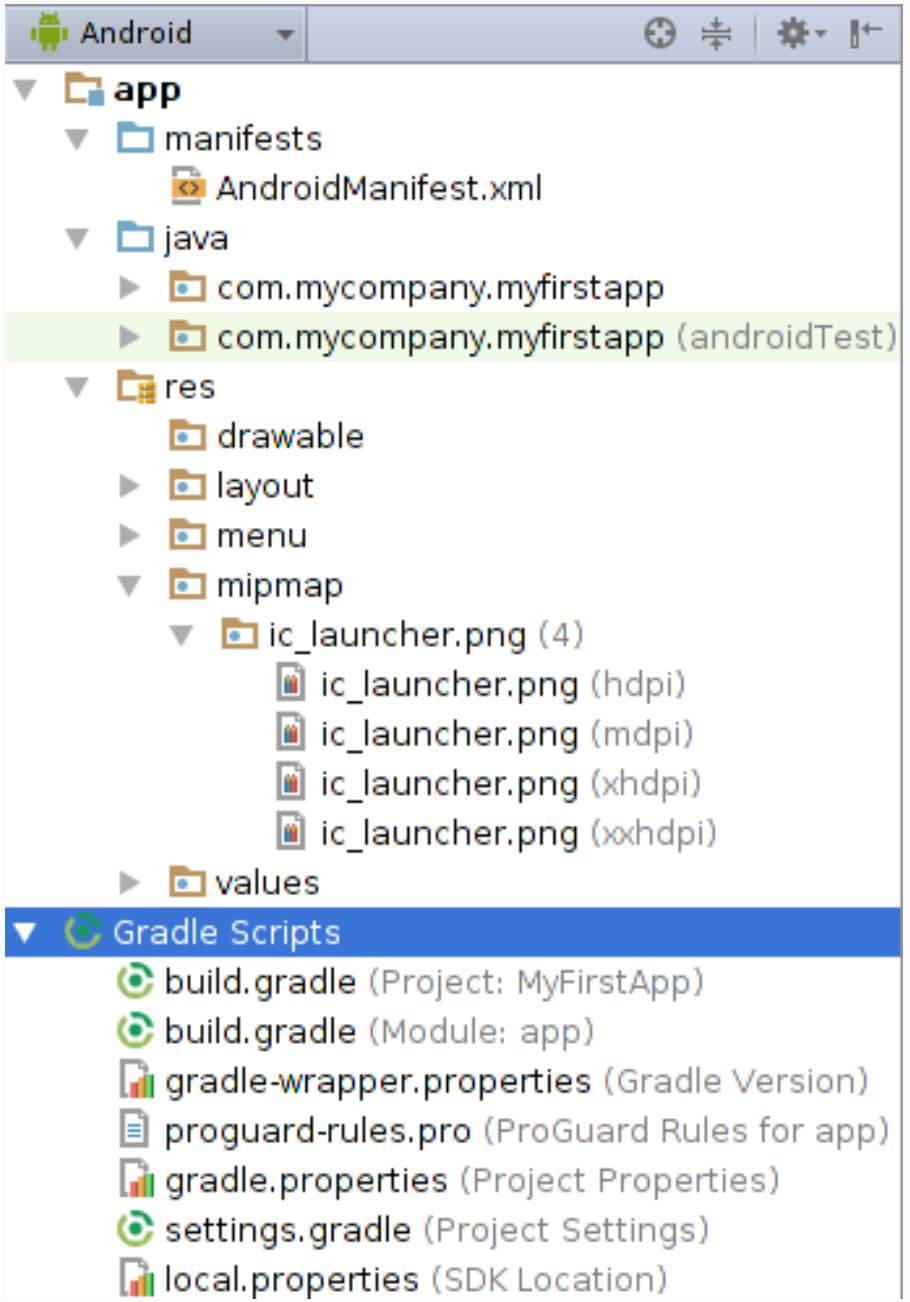
- Android Studio 默认显示方式，此视图不反映磁盘上的实际文件层次结构，而是按模块和文件类型进行组织，以简化项目源文件之间的导航方式，并隐藏一些不常用的文件或目录。



Android项目结构

Android 视图

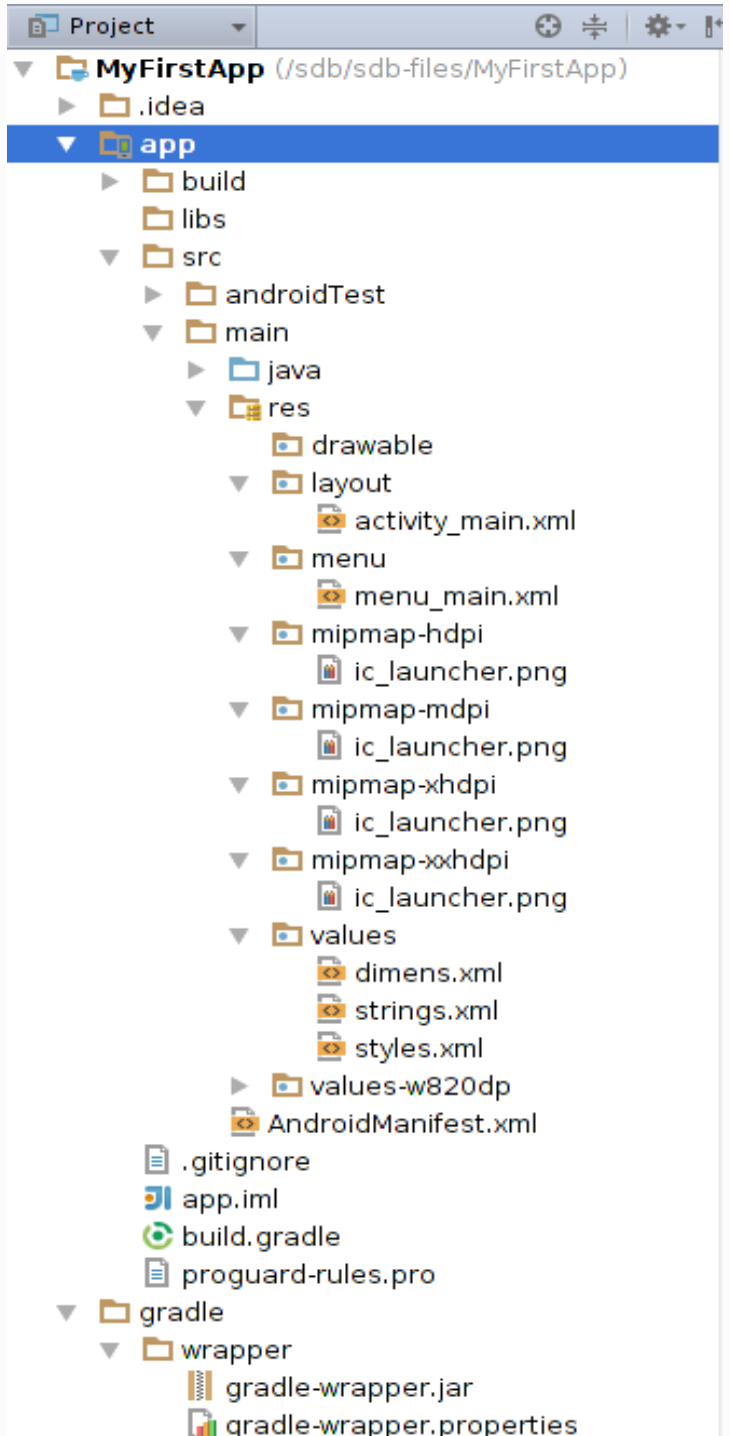
- manifests：AndroidManifest.xml 文件，包含了应用的基本信息。
- java: Java 源代码文件，包括测试代码。
- res：所有非代码资源，例如 XML 布局、界面字符串和位图图像。
- Gradle Script：项目的所有与编译相关的配置文件。



Android项目结构

Project 视图

- Project 视图将展示出项目实际的文件结构，相对于Android视图更加详细、清晰。



Android项目结构

module-name/

build/: 包含编译输出。

libs/: 包含私有库。

src/: 包含该模块在以下子目录中的所有代码和资源文件：

androidTest/: 包含在 Android 设备上运行的所有测试代码。

main/: 包含主要的源文件，如所有的 Android 代码和资源文件。

AndroidManifest.xml: 描述应用及其各个组件的性质。

java/: 包含 Java源代码。

jni/: 包含使用 Java Native Interface (JNI)的原生代码。

gen/: 包含 Android Studio 生成的 Java 文件，例如 R.java 。

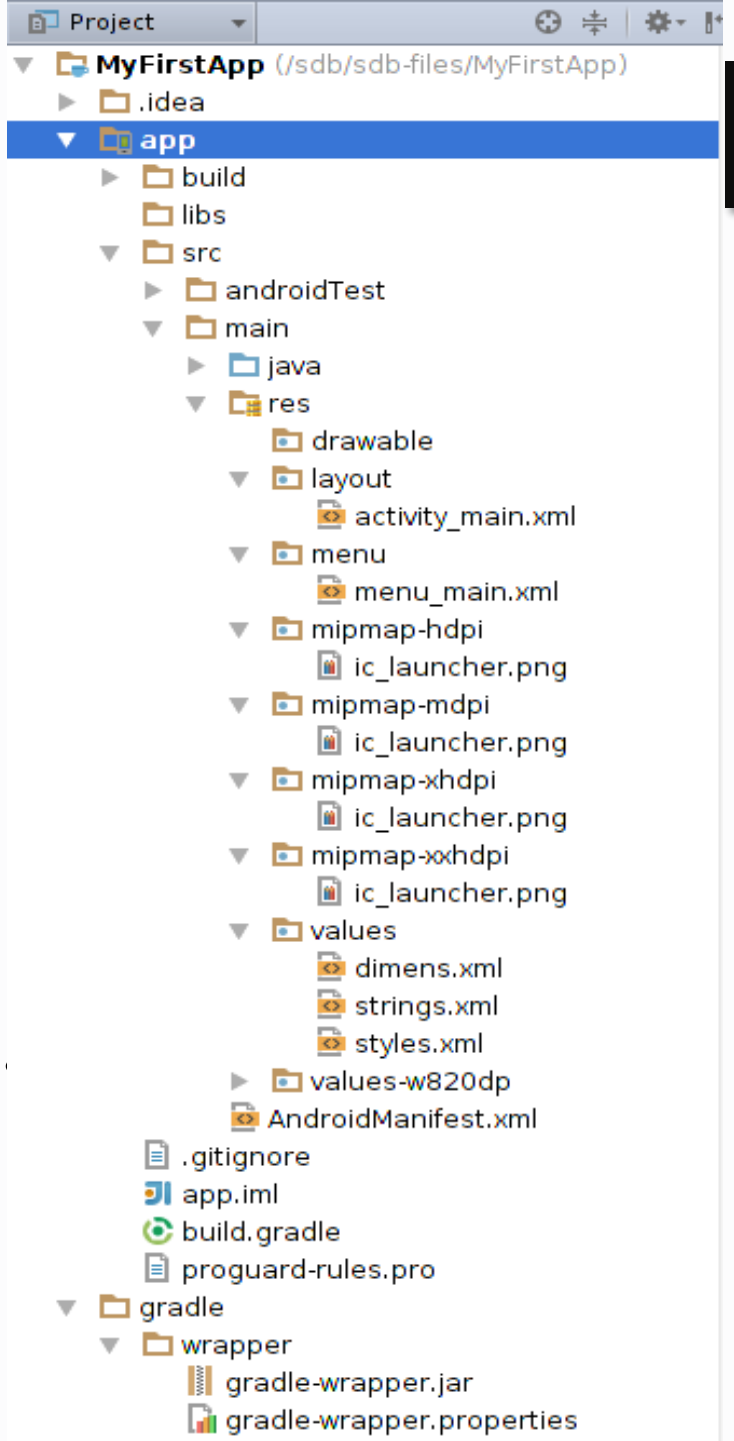
res/: 包含应用资源，例如，可绘制文件，布局文件和界面字符串

assets/:用于存放需要打包到应用程序的静态文件。

test/: 包含在 本地 JVM 上运行的本地测试代码。

build.gradle(module): 定义了该模块的编译配置。

build.gradle(Project): 定义了适用于所有模块的编译配置。



AndroidManifest

- 每个Android应用项目必须包含AndroidManifest.xml文件。该文件描述了应用及其各个组件的基本信息，并将这些信息提供给Android构建工具、Android 操作系统、权限要求、设备要求等。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.SEND_SMS"/>
    ...
    <uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true" />
    ...
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity" ...>
            ...
        </activity>
    </application>
</manifest>
```

AndroidManifest

- AndroidManifest.xml需声明以下内容：
  - **应用的软件包名称，**通常与代码的命名空间相匹配，如com.example.myapp。构建项目时，Android 构建工具会使用此信息来确定代码实体的位置。
  - **应用的组件**，包括所有 Activity、BroadcastReceiver、ContentProvider和Service 。 每个组件都必须定义基本属性，如Java类的名称。
  - 应用**所需的权限**，应用可能需要访问系统或其他应用的受保护部分，如需要访问短信或使用相机。
  - 应用需要的**硬件和软件特性**，从而说明应用程序可以兼容的设备类型。

AndroidManifest

package 属性

- AndroidManifest.xml必须在根元素<manifest>中包含**package**属性，即app的包名。 Android构建工具将利用该属性完成两件事：
  - 将该属性值作为应用生成 R.java 类的命名空间
  - 基于该属性值解析该文件中声明的其他类的名称。如，将声明为 <activity android:name= ".MainActivity" > 的 Activity 解析为 com.example.myapp.MainActivity

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
</manifest>
```

AndroidManifest

package 属性

注意：

- package 属性值应**始终**与项目中保存 Activity 和其他应用代码的 package name相匹配。

AndroidManifest

应用组件

- 在app中创建的每一个组件，都必须在Manifest文件中进行对应的声明，否则系统无法启动该组件。
- 各组件对应的xml元素：
  - Activity的子类：<activity>
  - Service的子类：<service>
  - BroadcastReceiver的子类：<receiver>
  - ContentProvider的子类：<provider>

AndroidManifest

应用组件

- 在组件元素中必须使用name属性指定子类的名称。
  - name可以是带包名的完整名称
  - name也可以以‘.’为起始，表示该子类名称以package属性值为前缀
  - 如图为com.example.myapp.MainActivity类的两种表示方式

```
<manifest ... >
    <application ... >
        <activity android:name="com.example.myapp.MainActivity" ...>
            ...
        </activity>
    </application>
</manifest>
```

```
<manifest package="com.example.myapp" ... >
    <application ... >
        <activity android:name=".MainActivity" ...>
            ...
        </activity>
    </application>
</manifest>
```

AndroidManifest

权 限

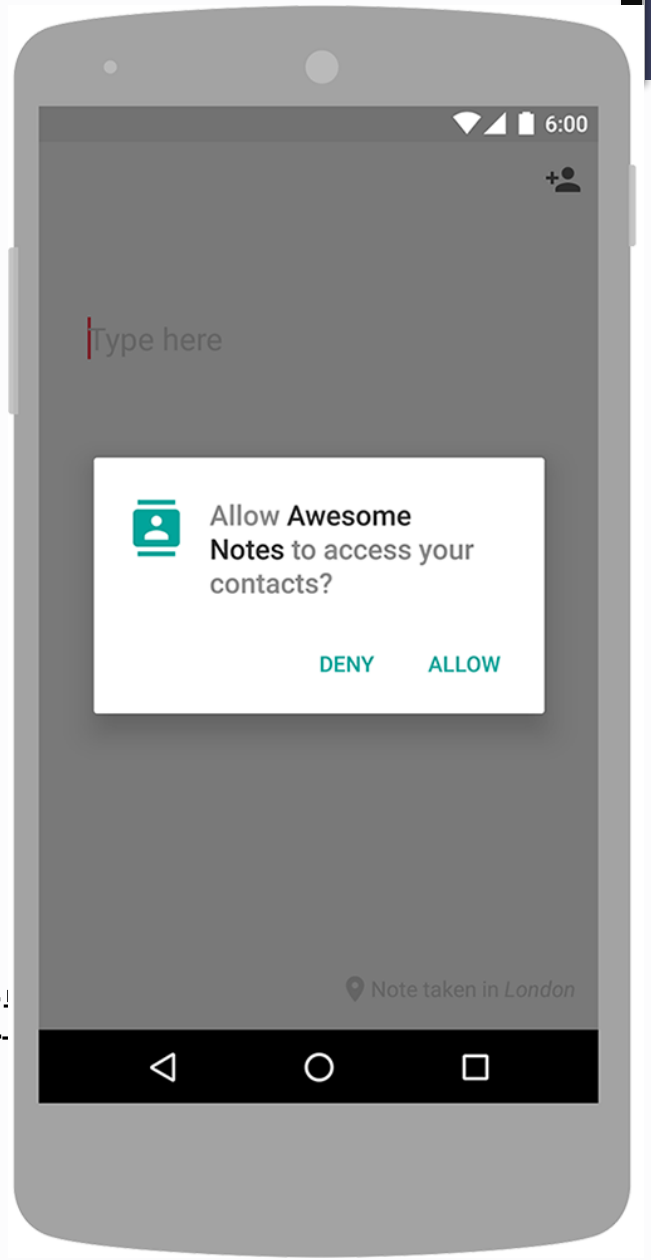
- 如果应用需要访问敏感用户数据（如联系人和短信）或某些系统功能（如相机和互联网访问），那么应用必须请求相关权限。在AndroidManifest.xml中，通过增加<uses-permission>元素声明需要的权限。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>

    <application ...>
        ...
    </application>
</manifest>
```

AndroidManifest

权 限

- 每种权限均有唯一标识，如：
  - android.permission.INTERNET：访问网络
  - android.permission.CAMERA：使用摄像头
  - android.permission.RECORD_AUDIO：录音
  - android.permission.READ_CONTACTS：读取联系人
  - android.permission.READ_EXTERNAL_STORAGE：读SD卡
  - android.permission.WRITE_EXTERNAL_STORAG：写SD卡
  - android.permission.ACCESS_FINE_LOCATION：请求精准位
  - .....



AndroidManifest

设备兼容性

- 为了说明应用程序可以兼容的设备类型，需要在AndroidManifest中声明应用运行依赖的硬件或软件特性。
- 利用<uses-feature>元素进行声明
  - android:required 属性：是否必须满足
- 一旦声明，Google Play store将不允许在未提供应用所需特性的设备上安装该应用。

```
<manifest ... >
    <uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true" />
    ...
</manifest>
```

AndroidManifest

设备兼容性

- uses-feature举例：
  - 硬件：
    - android.hardware.camera：摄像头
    - android.hardware.microphone：麦克风
    - android.hardware.location：位置
    - android.hardware.sensor.accelerometer：加速度传感器
  - 软件：
    - android.software.live_wallpaper：动态壁纸
    - android.software.input_methods：自定义输入法

AndroidManifest

uses-feature v.s. uses-permission

- 从定义上：
  - uses-feature 负责提供应用依赖的软硬件信息。
  - uses-permission 负责请求应用所需要的权限。
- 从功能上：
  - uses-feature 相当于是一个过滤器，Google Play store 会根据该标签来过滤设备。
  - uses-permission 相当于是一个权限助手，帮助应用向用户请求应用需要使用的权限。
- 两者经常会搭配在一起使用，应根据自己的实际需求来使用。

什么是Gradle

- Gradle是一个开源的项目自动化构建工具。
- Grade引入了DSL来声明项目设置，抛弃了传统的XML声明项目配置的形式。
  - DSL(domain specific language)，即特定领域语言。如:xml、html。Gradle中利用Groovy或Kotlin。
- Gradle主要支持Java、Groovy、Scala和C++的开发和部署，计划在将来支持更多的编程语言。
- Gradle能够自动化地完成依赖管理、打包、发布、部署等。

Why Gradle

- 免费、开源；
- 具有丰富的 API 和插件；
- 支持依赖管理；
- 可定制性强，能够创建自定义的构建逻辑；
- Gradle集成了大量工具，拥有最佳的构建特性。

Android官方构建工具

Gradle in Android

- Android Studio中的构建子系统
- 3个主要的配置文件：
  - ★ **settings.gradle**
  - ★ **Project bulid.gradle**
  - ★ **Module build.gradle**
- 不需要了解Gradle的底层细节
- 关于Gradle更多的信息，可参考： https://gradle.org/

Gradle配置文件

settings.gradle

- Gradle设置文件，位于项目的根目录，是项目的结构声明与入口，用于指示 Gradle 在编译应用时应该将哪些模块包含在内。
  - include ‘PROJECT_NAME’ 声明包含的模块

```
include ':app'
```

Gradle配置文件

Project build.gradle

- 定义适用于项目中所有模块的编译配置，文件位于项目根目录。
  - buildscript block：为Gradle本身配置需要的仓库和依赖项，不应在这里包含模块的依赖项。
    - repositories block：Gradle搜索、下载依赖项的仓库。如JCenter、Maven Central、Google's Maven
    - dependencies：Gradle构建该项目需要的依赖，如Gradle Android Plugin

```
buildscript {

    repositories {
        google()
        jcenter()
    }


    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.2'
    }
}


allprojects {
    repositories {
        google()
        jcenter()
    }
}
```

Gradle配置文件

Project build.gradle

- allprojects block：项目中所有模块需要的仓库和依赖项。
  - repositories block：Gradle搜索、下载依赖项的仓库。如JCenter、Google’s Maven
  - 不要在该区块中配置特定模块相关的依赖

```
buildscript {

    repositories {
        google()
        jcenter()
    }


    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.2'
    }
}


allprojects {
    repositories {
        google()
        jcenter()
    }
}
```

Gradle配置文件

Module build.gradle

- 为其所在的特定模块配置编译设置，可以通过配置这些编译设置来自定义打包选项。
  - apply plugin：使得Android的Gradle插件可以用于构建这个模块。
  - dependencies：该模块的依赖
  - android block：设置针对andorid的配置
    - compileSdkVersion：编译使用的SDK版本
    - buildToolsVersion：SDK build-tools版本

```
apply plugin: 'com.android.application'

android {

    compileSdkVersion 25

    buildToolsVersion "25.0.2"

    defaultConfig {...}

    buildTypes {...}

}


dependencies {...}
```

Gradle配置文件

Module build.gradle

- android block：设置针对andorid的配置
  - defaultConfig block：封装了所有构建变量的默认设置，并且可以在构建系统时动态地覆盖main/AndroidManifest.xml中的一些属性。
    - applicationId：唯一的应用标识符，Google Play store和设备根据该Id识别不同的应用
    - minSdkVersion：运行app需要的最低版本SDK
    - targetSdkVersion：与前向兼容有关的目标SDK版本
    - versionCode：app版本号，Int类型，用于判断是否需要升级
    - versionName：app版本名，String类型，显示给用户的版本信息

```
android {

    compileSdkVersion 25

    buildToolsVersion "25.0.2"

    defaultConfig {

        applicationId 'com.example.myapp'

        minSdkVersion 15

        targetSdkVersion 25

        versionCode 1

        versionName "1.0"
    }

    buildTypes {...}

}
```

Gradle配置文件

Module build.gradle

- android block：设置针对andorid的配置
  - buildTypes block：用于配置多个构建类型。默认情况下，将定义两种构建类型，debug和release。
    - debug block: 默认设置没有显式显示在配置文件中，如需修改可手动添加
    - release block：release版本的配置
      - » minifyEnabled：是否进行代码压缩
      - » proguardFiles：混淆配置文件

```
android {
    ...

    defaultConfig {...}

    buildTypes {

        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

}
```

Gradle配置文件

compileSdkVersion

- compileSdkVersion: 指定了Gradle要采用哪个版本的Android SDK编译应用。应用中使用的API级别不得高于compileSdkVersion 。如果新增了高级别的API调用，那么compileSdkVersion也要随之升高。
  - **建议总是使用最新的SDK进行编译。**虽然有可能更改后在编译时可能会出现新的编译警告/错误，但这样可以对现有代码进行新的编译检查，能够避免使用最新弃用的API，也可以为后续更新的API调用做准备。
  - **compileSdkVersion的改变不会改变应用运行时行为。**因为它只在编译阶段起作用，不会被包含到APK中。

Gradle配置文件

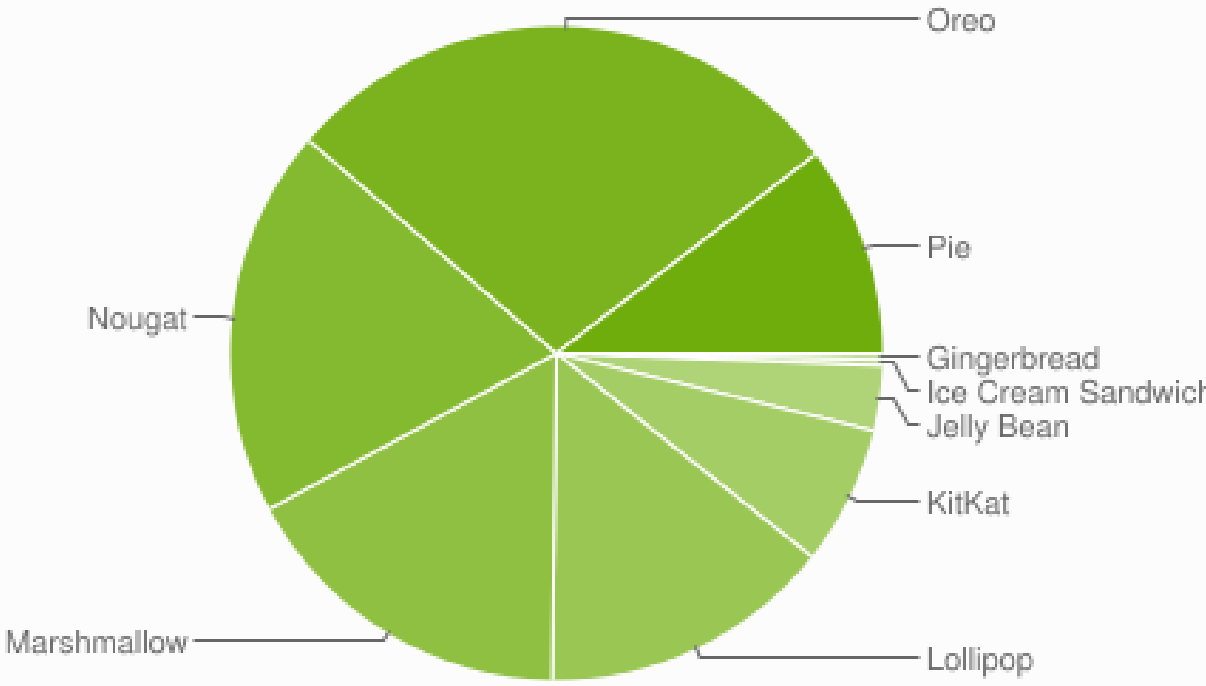
minSdkVersion

- minSdkVersion：应用可以运行的最低要求，即应用兼容的最低 SDK 版本。它是Google Play Store用来确定用户设备上是否可以安装某应用的信号之一。
  - 要注意对高于minSdkVersion APIs的处理，以免在某些设备上运行时因尝试调用不存在的APIs而出错。
  - minSdkVersion的确定需要综合考虑产品定位和应用的用户群体特点，需要在兼容性和应用的新特性、新功能之间做trade-off。高级APIs是否整个应用程序的关键？兼顾更多的设备更重要？

Gradle配置文件

minSdkVersion

- 在确定minSdkVersion时，应参考dashboards 上的统计信息。它包含7天内访问过Google Play Store的所有设备的整体信息，可以帮助决定是否值得为少部分的设备进行额外的开发和测试，以确保获得最佳体验。



不同Android版本的设备占比

Gradle配置文件

targetSdkVersion

- targetSdkVersion: 是 Android 系统提供前向兼容的主要手段。
  - 随着 Android 系统的升级，某些 APIs 或者模块的行为可能会发生改变，但是为了保证老 APK 的行为还是和以前兼容，只要 APK 的 targetSdkVersion 不变，即使这个 APK 安装在新 Android 系统上，其行为还是保持老的系统上的行为，这样就保证了系统对老应用的前向兼容性。
  - 建议将targetSdkVersion更新到最新版的SDK，以便获得系统的最佳特性。

Gradle配置文件

targetSdkVersion

- targetSdkVersion应用举例：

在 Android 4.4 (API 19) 以后，AlarmManager 的 *set()* 和 *setRepeat()* 这两个 API 的行为发生了变化。在 Android 4.4 以前，这两个 API 设置的都是精确的时间，系统能保证在 API 设置的时间点上唤醒 Alarm。因为省电原因 Android 4.4 系统中的 AlarmManager 将这两个 API 设置唤醒的时间都处理为不精确的时间，系统只能保证在设置的时间点之后某个时间唤醒。

这时，**虽然 API 没有任何变化，但是实际上 API 的行为却发生了变化。**如果老的 APK 中使用了此 API，并且在应用中的行为非常依赖 AlarmManager 在精确的时间唤醒，例如闹钟应用，一旦Android 系统不能保证兼容，老的 APK 安装在新的系统上，就会出现问题。

Gradle配置文件

targetSdkVersion

Android 系统是怎么保证这种兼容性的呢?

- 利用 **targetSdkVersion**!
  - APK 在调用系统 AlarmManager 的 *set()* 或者 *setRepeat()* 的时候，系统首先会检查一下 APK 的 targetSdkVersion 信息，如果小于 19，则还是按照老的行为，即精确设置唤醒时间，否者执行新的行为特性。

Gradle配置文件

- xxxSdkVersion的设置小结
  - 这三个值之间的关系:

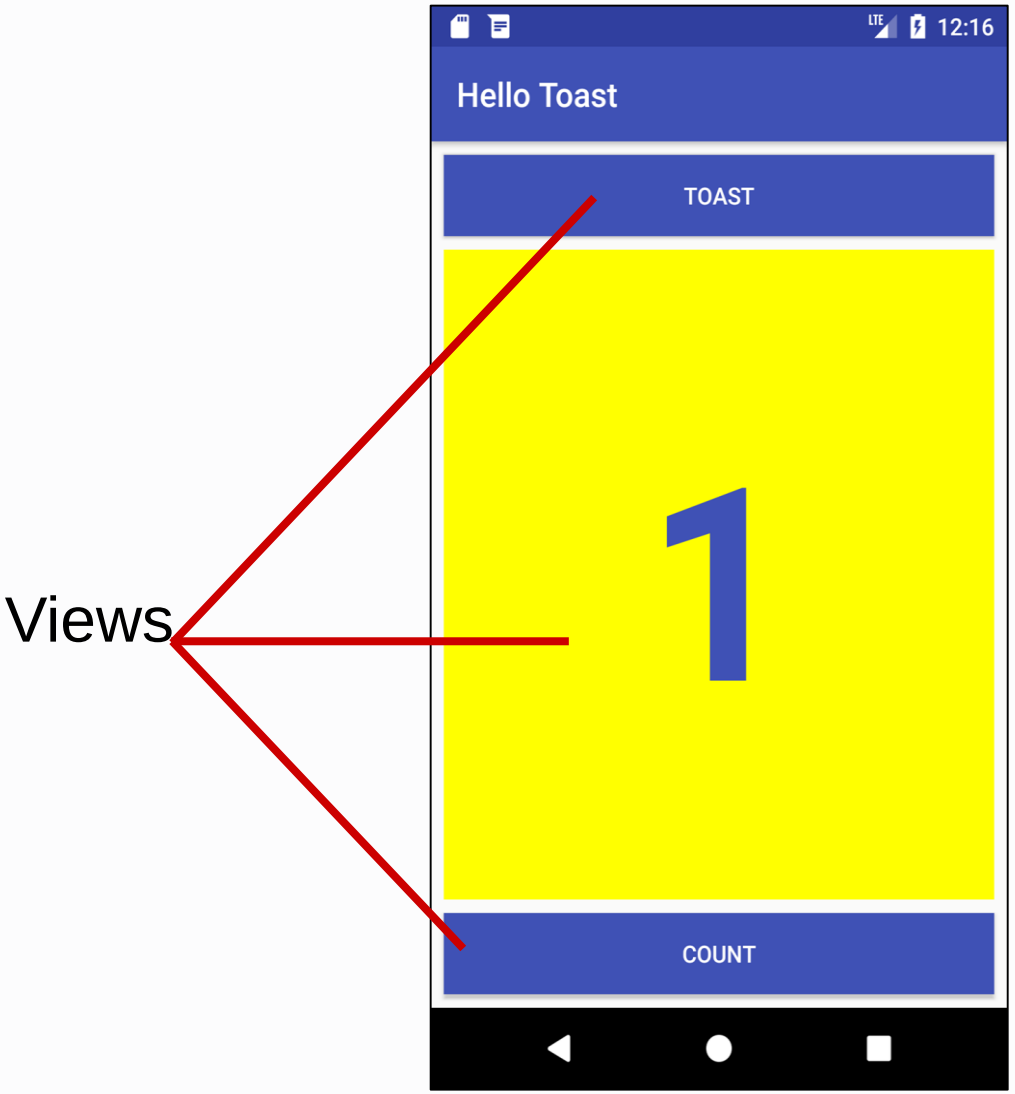
    minSdkVersion <= targetSdkVersion <= compileSdkVersion
  - 理想情况下，稳定状态下的关系应该是:

    minSdkVersion <= targetSdkVersion == compileSdkVersion
    - 以一个较低的minSdkVersion来覆盖较大的人群，并通过使用最新的SDK进行编译和选择生效的特性，来获得最佳的应用外观和性能。

资源文件

- 资源文件主要包括视图和其它静态资源

- 如果查看移动设备，你看到的每个用户界面元素都是一个视图。



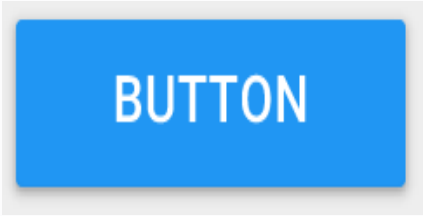
什么是视图

视图(View)子类是用户界面基本的构建模块

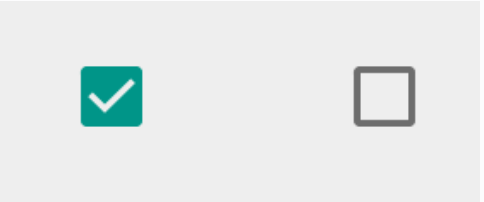
- 显示文本（TextView 类)，编辑文本（EditText 类)
- 按钮（Button 类)，菜单(menus)，其他控件
- 可滚动视图(ScrollView, RecyclerView)
- 显示图像（ImageView ）
- 视图组（ConstraintLayout 和 LinearLayout ）

视图子类示例

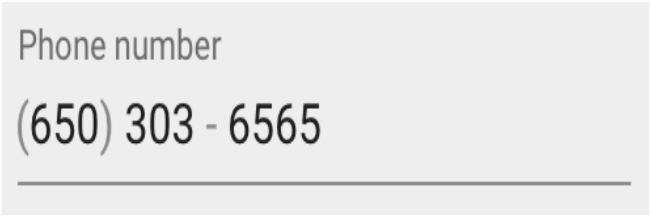
按钮
(Buttion)



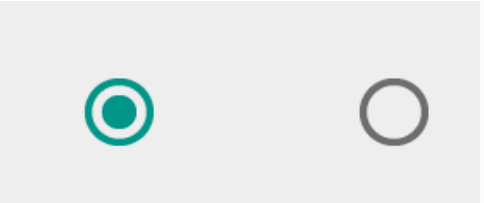
复选框
(CheckBox)



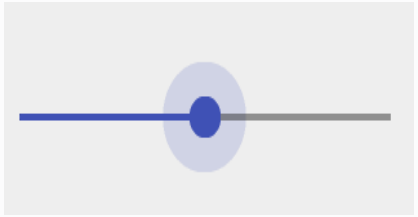
可编辑文本框
(EditText)



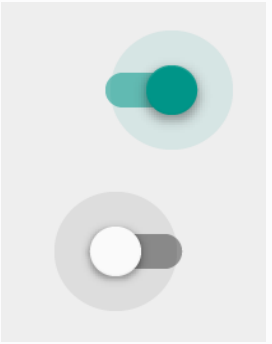
单选按钮
(RadioButton)



滑块
(Slider)



开关
(Switch)



通过设置视图属性来控制视图

- 颜色、尺寸、位置
- 焦点（例如，选择接收用户输入）
- 可能是交互式的（响应用户点击）
- 是否可见
- 与其他视图的关系

资源

- 将布局中的静态数据与代码分开
- 字符串，尺寸，图像，菜单文本，颜色，样式
- 有助于定位各种资源

代码中使用资源的示例

- Layout:

  `R.layout.activity_main`

  ```
  setContentView(R.layout.activity_main);
  ```

- View:

  `R.id.recyclerview`

  ```
  rv = (RecyclerView) findViewById(R.id.recyclerview);
  ```

- String:

  In Java: `R.string.title`

  In XML: `android:text="@string/title"`

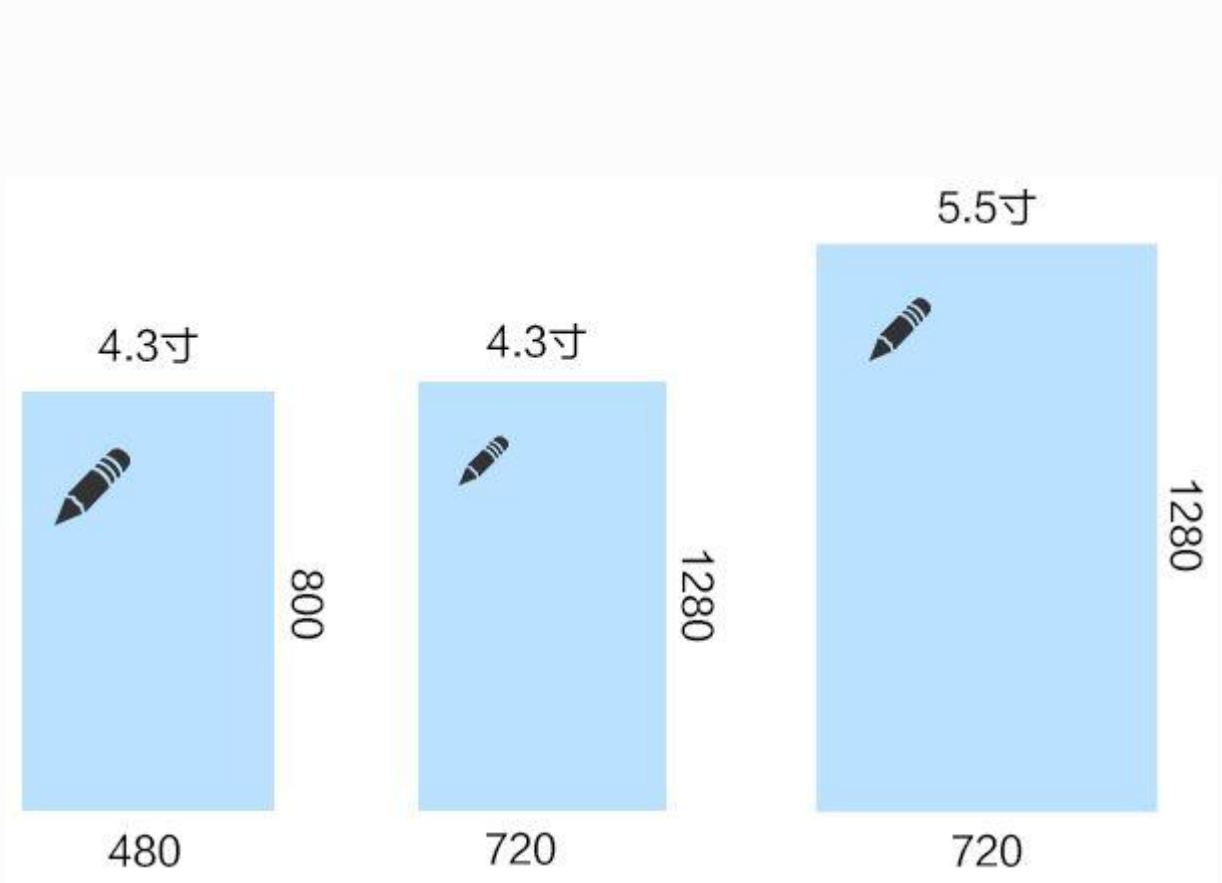
尺寸

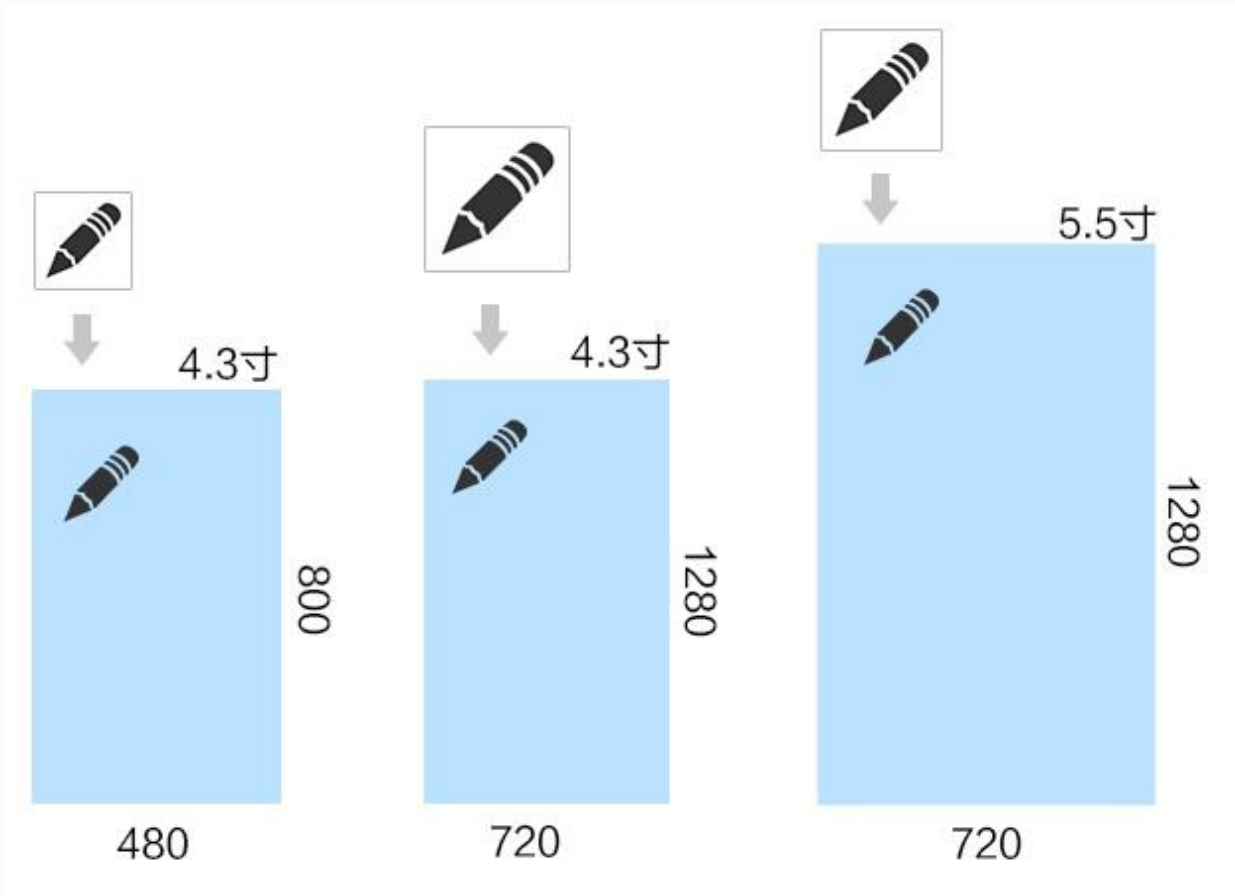
- 视图：Density-independent Pixels (dp)
- 文本：Scale-independent Pixels (sp)

不要使用 device-dependent 或 density-dependent 的尺寸单位:

- Actual Pixels (px)
- Actual Measurement (in, mm)
- Points - typography 1/72 inch (pt)

为什么使用dp

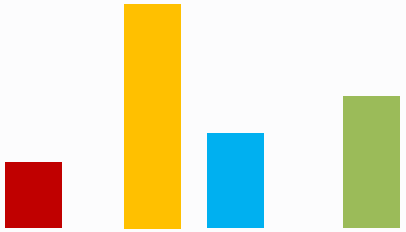

4.3寸
800
480
4.3寸
1280
720
5.5寸
1280
720


4.3寸
800
480
4.3寸
1280
720
5.5寸
1280
720

组件

组件是即使用代码实现的功能，其可以控制用户和UI的交互，加载资源等，目前Android中主要有四大组件：

- Activity：每个Activity一般都至少对应一个视图布局，Activity一般根据用户和视图的交互结果来进行不同的操作，如点击按钮打开购物车等，Android的开发一般都以Activity为单位进行，点击桌面App图标，将进入App的主Activity中
- Service：不对应视图布局，主要用于后台运行某些任务，如后台音乐播放等
- Content Provider：主要用于数据的存储和共享
- Broadcast Receiver：主要用于系统和App或者不同的App之间进行通信



目录

Android简介

系统框架——Linux内核和HAL

系统框架——系统运行库层

系统框架——应用架构层

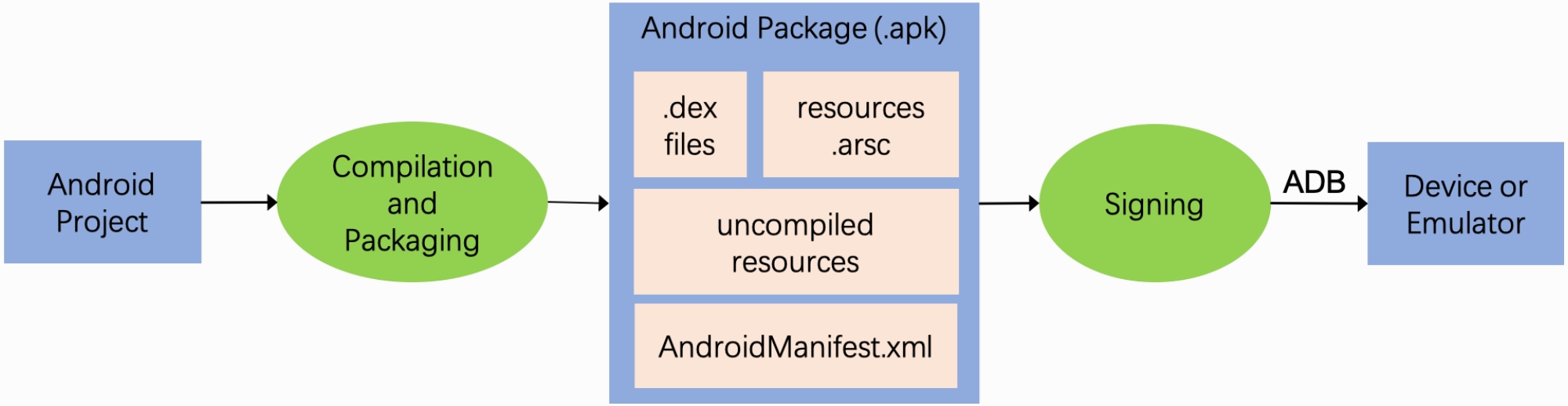
系统框架——应用层

Android 项目结构

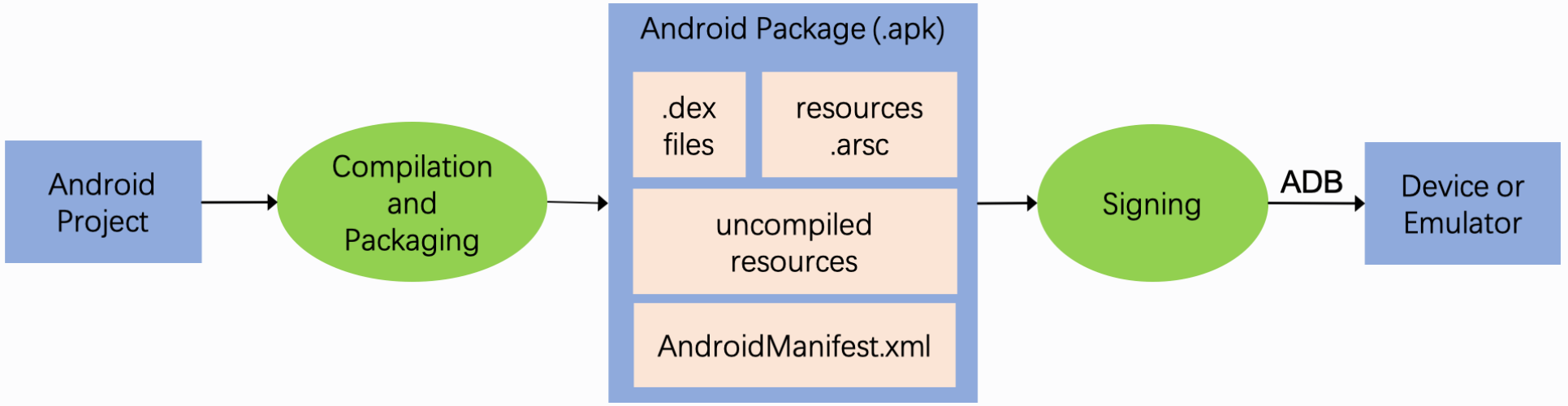
Android 应用构建

Android 应用构建

- Android 编译系统会编译应用资源和源代码，并将它们打包成可供测试、部署、签署和分发的apk文件。对于每一个步骤，Android都提供了相应的工具来执行相应的功能。



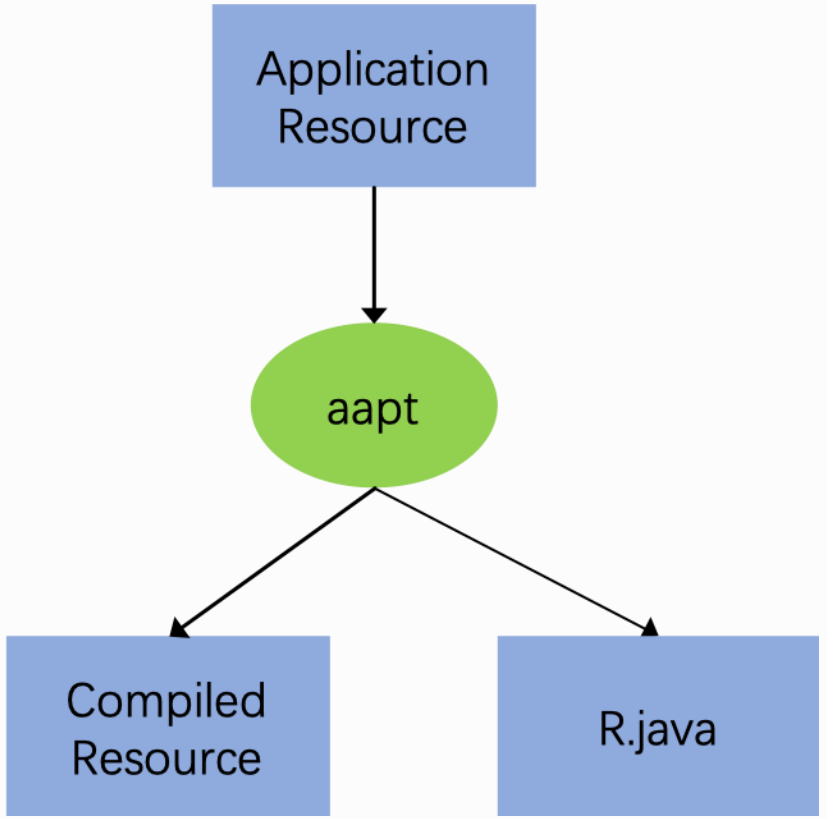
Android 应用构建



- 将Android 项目文件编译和打包，得到.apk包
- .apk中包含:
  - .dex文件: 从.java文件编译成 .class文件后再编译为Dalvik所需要的.dex文件
  - resources.arsc: 编译后的二进制资源文件
  - uncompiled resources: 不需要进行编译的资源文件，如 .png
  - AndroidManifest.xml: 工程清单文件
- 在部署到设备上或应用商店上架前需要对.apk签名，确保唯一性和安全性
- 通过ADB(Android Debug Bridge)安装到手机或虚拟机

Android 应用构建

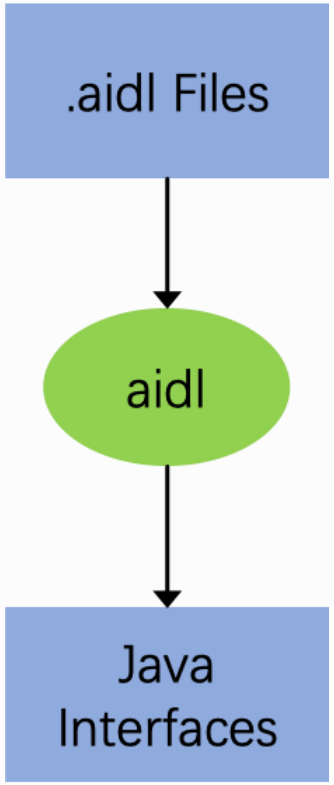
1. aapt(Android Asset Packaging Tool)资源打包工具会将应用的资源文件(AndroidManifest清单文件、Acttivity所需的xml文件等)进行编译，生成对应资源ID文件R.java和二进制资源文件。



Android 应用构建

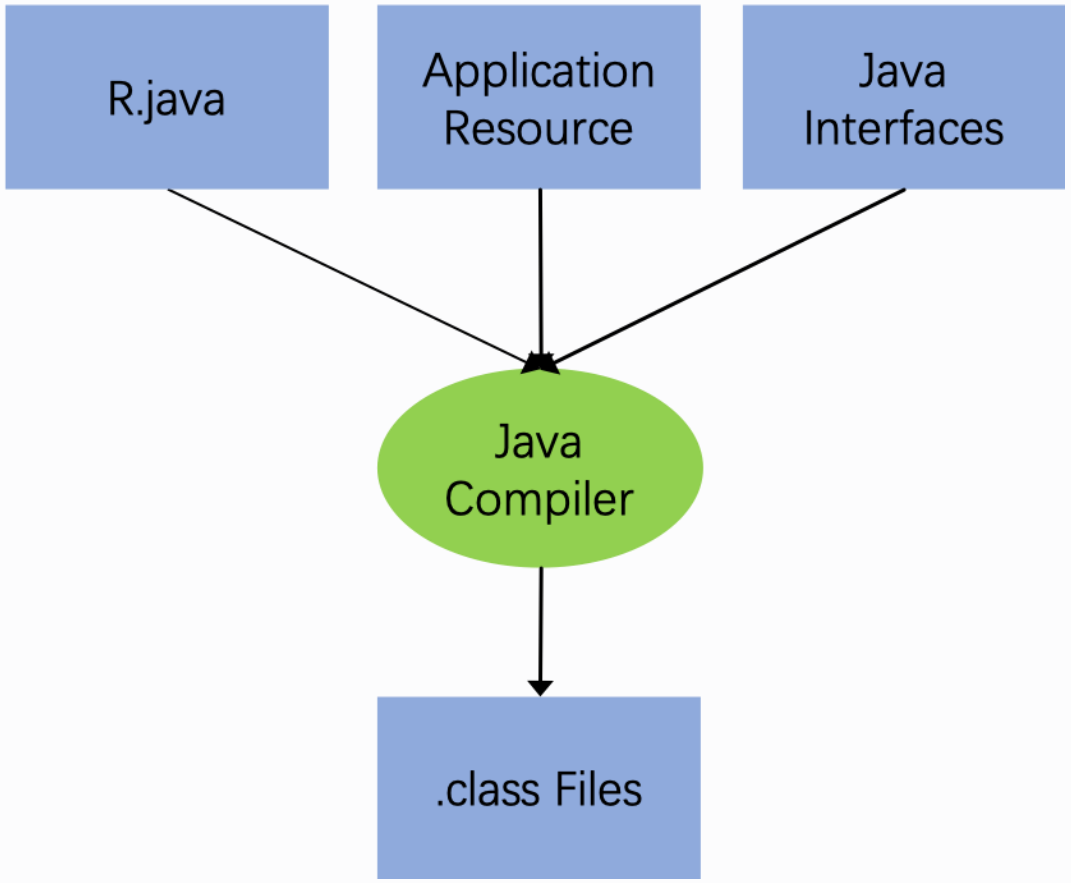
2. 如果存在.aidl文件，aidl(Android Interface Definition Language)工具会将其中的.aidl接口转化成Java的接口。

- .aidl是编写进程间通信代码时定义的接口。 该工具的输入是aidl后缀的文件，输出是可用于进程通信的C/S端Java代码。



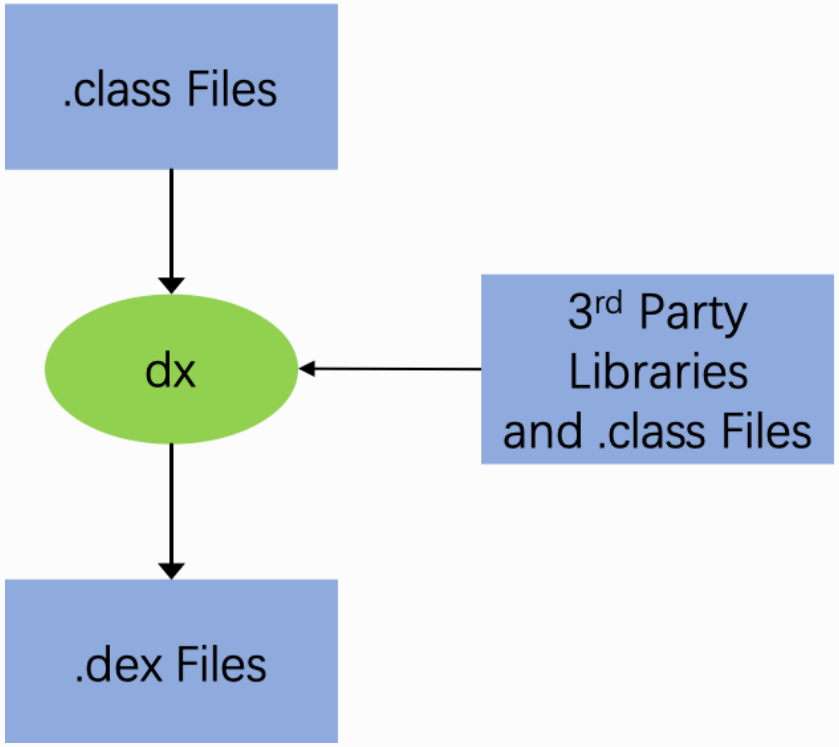
Android 应用构建

3. 利用javac进行java编译，将Java文件(包含R文件、Java源代码、aidl转化来的Java接口)向class文件的转化，统一转化成.class文件。



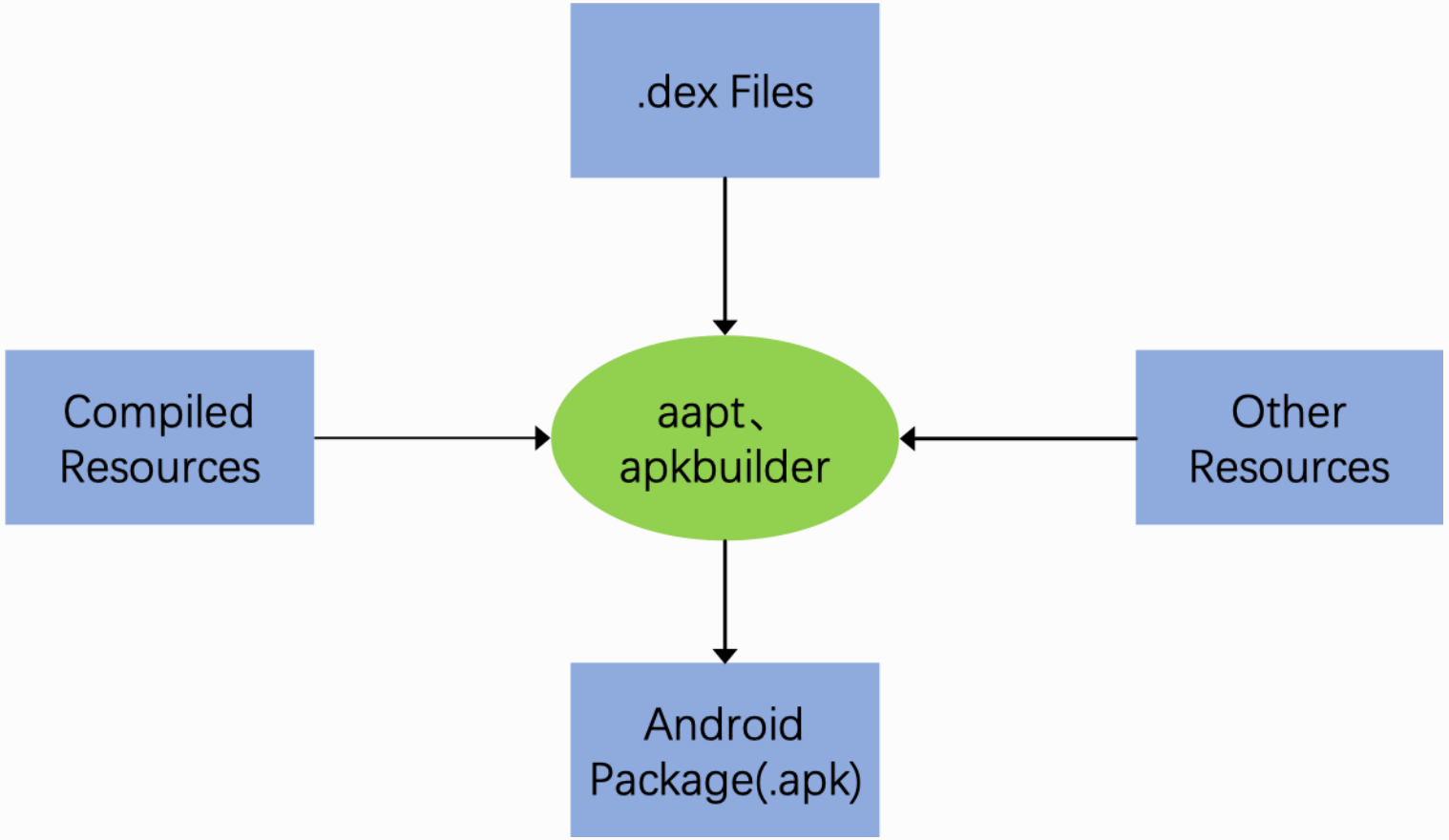
Android 应用构建

4. dx工具将.class文件转化为.dex文件。
   - .dex文件是一种专门为Android设计的字节码格式，Android虚拟机可以执行.dex文件
   - 将生成常量池，消除冗余数据等。转化后各个类能够共享数据，在一定程度上降低了冗余，同时也使得文件结构更加紧凑。



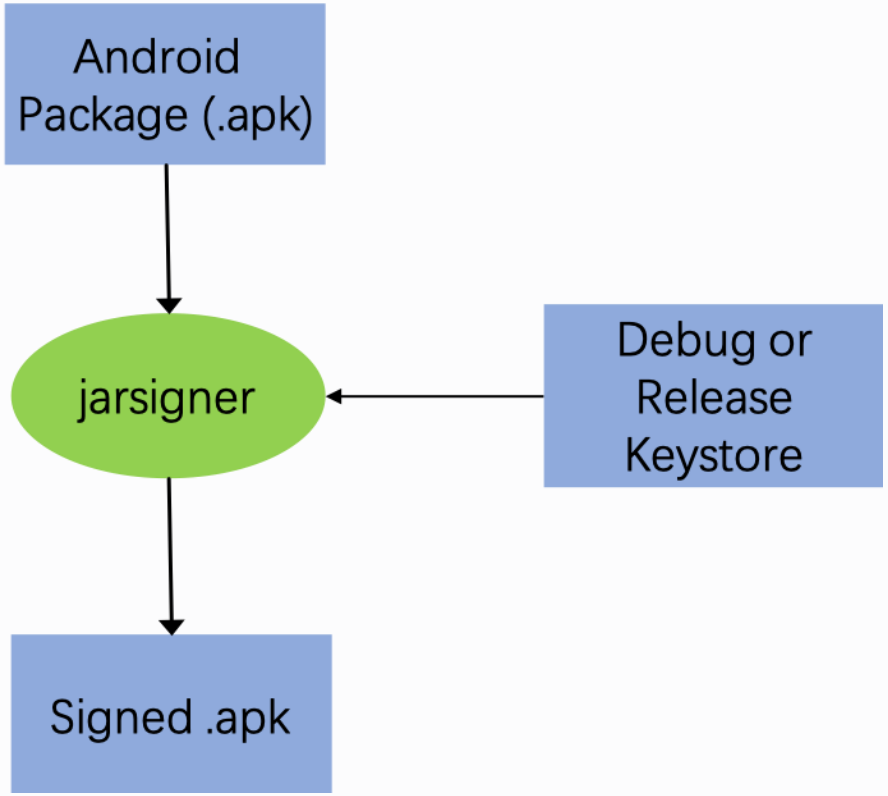
Android 应用构建

5. aapt、apkbuilder将资源文件和.dex文件，打包成.apk文件。



Android 应用构建

6. 利用jarsigner签名工具对apk签名。
   - 必要步骤。 Android系统在安装APK的时候，首先会检验APK的签名，如果发现签名文件不存在或者校验签名失败，则会拒绝安装。同时，签名信息中包含有开发者信息，在一定程度上可以防止应用被伪造。



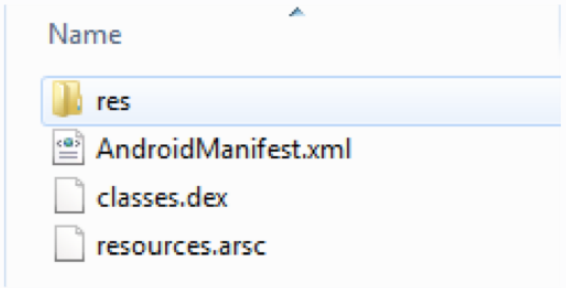
Android 应用构建

应用签名解析

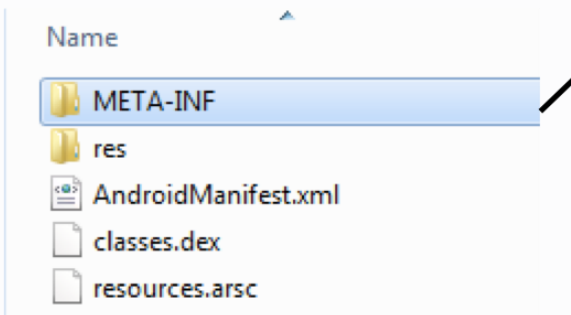
1. 对编译后生成的所有的文件进行扫描，为每个文件生成一个数字摘要，保存在 MANIFEST.MF 文件中

2. 对 MANIFEST.MF 文件生成数字摘要，并写入 CERT.SF文件

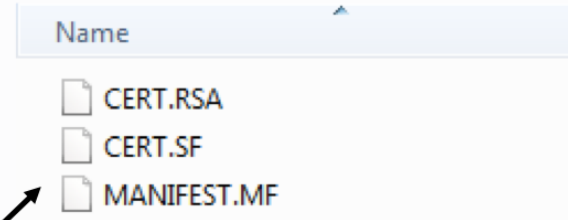
- 除了对整个文件生成摘要，还将文件分成多块, 生成对应的摘要，保存在 CERT.SF 文件中



(a) 签名前的apk文件



(b) 签名后的apk文件



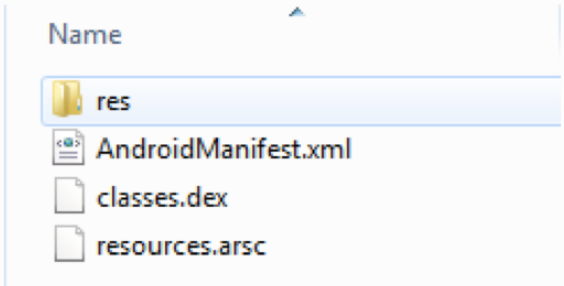
(c) META-INF文件夹

Android 应用构建

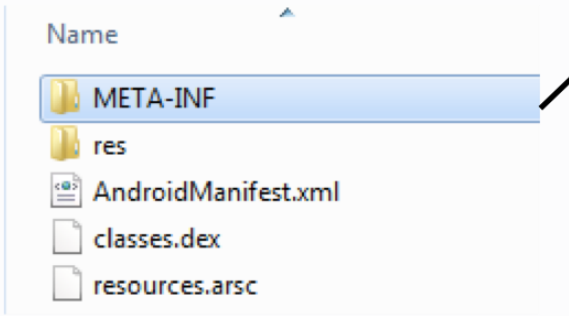
应用签名解析

3. 计算 CERT.SF 的数字摘要，并使用 私钥进行加密，生成签名

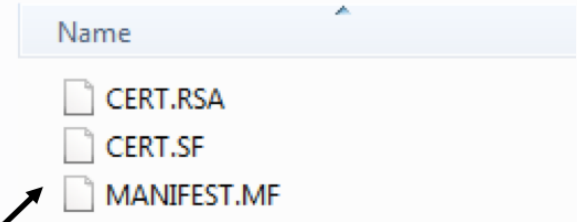
4. 将签名、公钥、哈希算法信息写入 CERT.RSA 文件，并将这些文件添加到 APK 压缩包 META-INF 目录中



(a) 签名前的apk文件



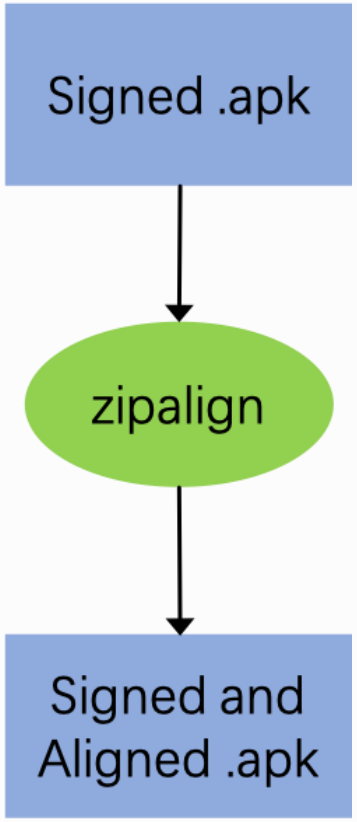
(b) 签名后的apk文件



(c) META-INF文件夹

Android 应用构建

7. 对.apk包进行优化，减小应用在设备上的内存消耗，提升应用运行效率。
   - 通过对签名后的.apk文件做字节对齐处理，能够使得内存映射访问APK文件时更高效。




Thank
you