Android源码目录及编译系统分析

作者:程姚根 联系方式:chengyaogen123@163.com

一、Android 5.0源码目录

目录名或文件名	大小	用途
abi	120k	(application binary interface)应用程序二进制接口
art	19M	Android runtime
bionic	23M	针对android系统设计的C语言函数库
bootable	5.9M	Android启动引导相关的代码
build	15M	存放Android编译系统
cts	277M	google提供的Compatibility Test Suit(CTS)兼容性测试组，用于测试android系统的兼容性以及稳定性
dalvik	19M	dalvik Java虚拟机代码存放目录
developers	104M	开发者相关目录
development	255M	开发者需要的一些例子程序及相关的开发工具
device	971M	产品相关代码，按照厂家分类
docs	16M	Android相关的介绍文档
external	3.3G	来自开源社区(第三方)的源代码，如:webkit、sqlite...
frameworks	1.4G	android的运行框架集合，包含系统运行的各种服务框架，向app层提供api
hardware	118M	硬件抽象层，描述对Linux Kernel中的相关驱动模块的具体操作，而在kernel中的驱动模块只拥有通用操作接口，比如设置寄存器值，IO拉高拉低的函数接口。但具体设置什么值，拉高拉低的时序都写在hardware层对应的module中，这就是google对于硬件驱动的商业保护。
libcore	52M	核心库
libnativehelper	208K	用于Android本地开发的的函数库
Makefile	4.0k	
modules	34M	一些编译好的驱动模块
ndk	89M	Native Delelopment Kit 本地开发包，提供android本地开发工具及相关代码

out	17G	Android编译后的目标代码输出目录
packages	379M	系统级android app应用程序
PDK	1.5M	Platform Development Kit 合作伙伴开发套接，主要用来加快OEM厂商的update速度，他包含一些平台的少量Code,以及一些bingary文件，能够使系统工程师尽快的进行平台的一些驱动的开发、调试、以及一些平台的新功能开发。
prebuilts	7.3G	x86和ARM架构下预编译的一些资源
SDK	33M	Software Development Kit ,为应用程序开发人员提供开发工具，主要是所有公开API的集合，应用程序开发人员可以借助SDK中的API快速进行应用程序开发
system	15M	android系统底层的文件系统，应用组件，包含一些系统库，以及启动的配置文件

二、Android编译流程分析

当我们拿到一个android源码的时候，我们需要做的是就是根据我们自己的硬件平台来编译android源码。具体如何做呢,下面我们就来分析一下具体的步骤。

1. source build/envsetup.sh

```
a@b:~/workdir/androidL$ source build/envsetup.sh
including device/softwinner/fspad-733/vendorsetup.sh
including device/softwinner/common/vendorsetup.sh
including sdk/bash_completion/adb.bash
```

(1)source 命令用来执行一个shell脚本，大家可能会问了，这和用bash或sh命令来执行一个shell脚本有什么区别呀?
<1>bash 或sh来执行一个shell脚本的时候，是在一个子shell（创建的子进程）中运行shell脚本，这样shell脚本中设置的变量和函数不会影响到它的父shell(就是你当前终端中运行的bash)
<2>source命令执行的shell脚本是在父shell中运行的，此时shell脚本设置的变量和函数会影响到父shell
问题:什么是影响父shell呢?
回答:就是shell脚本运行结束后，shell脚本中设置的变量和函数可以直接在父shell中调用。

(2)build/envsetup.sh 是一个在编译android系统前需要运行的一个shell脚本文件，这个脚本文件就是设置一下当前环境下android系统编译需要的一些环境参数。
我们可以通过hmm来看一下build/envsetup.sh运行后，给我们提供的一些命令(这些命令在envsetup.sh中都是以函数的形式呈现的)

```
a@b:~/workdir/androidL$ hmm
Invoke ". build/envsetup.sh" from your shell to add the following functions to your environment:
- lunch:   lunch <product_name>-<build_variant>
- tapas:   tapas [<App1> <App2> ...] [arm|x86|mips|armv5|arm64|x86_64|mips64] [eng|userdebug|user]
- croot:   Changes directory to the top of the tree.
- m:       Makes from the top of the tree.
- mm:      Builds all of the modules in the current directory, but not their dependencies.
- mmm:     Builds all of the modules in the supplied directories, but not their dependencies.
```

```
            To limit the modules being built use the syntax: mmm dir/:target1,target2.
- mma:      Builds all of the modules in the current directory, and their dependencies.
- mmma:     Builds all of the modules in the supplied directories, and their dependencies.
- cgrep:    Greps on all local C/C++ files.
- ggrep:    Greps on all local Gradle files.
- jgrep:    Greps on all local Java files.
- resgrep:  Greps on all local res/*.xml files.
- sgrep:    Greps on all local source files.
- godir:    Go to the directory containing a file.
```

它的功能可以总结如下:

(1)加载编译参数

(2)加载平台信息

2.lunch(配置)

```
a@b:~/workdir/androidL$ lunch

You're building on Linux

Lunch menu... pick a combo:
     1. aosp_arm-eng
     2. aosp_arm64-eng
     3. aosp_mips-eng
     4. aosp_mips64-eng
     5. aosp_x86-eng
     6. aosp_x86_64-eng
     7. fspad_733-eng
     8. fspad_733-user

Which would you like? [aosp_arm-eng] 7
```

注意:默认情况下选择的是"aosp_arm-eng",我们可以输入自己需要的选择对应的"产品类型-编译类型",例如我们的开源平板我们可以选择"fspad_733-eng"或者7

(1)功能

<1> 选择要编译的产品和编译类型（"产品型号-编译类型"）。

产品型号就是你的android系统所对应的平台，例如我们选择的是"fspad-733"。

产品的编译类型有以下三种:

eng	工程机版本，开发阶段的版本,能够看到所有的log信息输出，便于调试发现bug
userdebug	用户试用机版本，用户试用阶段的版本，此版本的系统有root访问权限
user	用户最终版本，用户最终使用的版本,关闭root权限，取消log信息输出

<2>检查选择的产品是否存在， 存在，获取产品信息，根据产品信息，检查产品的设备信息是否存在，存在，获取设备信息(选择跟硬件相关的软件模块)

<3>检查选择的编译类型(选择软件模块)是否正确

<4>打印选择的产品信息及设备信息

(2)结果

```
============================================
PLATFORM_VERSION_CODENAME=REL     代码名称，“REL”表示发布版
PLATFORM_VERSION=5.0 android源码版本
TARGET_PRODUCT=fspad_733 产品型号
TARGET_BUILD_VARIANT=eng 编译类型
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=  当前编译的应用app
TARGET_ARCH=arm   cpu类型
TARGET_ARCH_VARIANT=armv7-a-neon  cpu指令集
TARGET_CPU_VARIANT=cortex-a7  cpu名称
TARGET_2ND_ARCH=
TARGET_2ND_ARCH_VARIANT=
TARGET_2ND_CPU_VARIANT=
HOST_ARCH=x86_64   编译主机架构
HOST_OS=linux   编译主机操作系统
HOST_OS_EXTRA=Linux-3.13.0-32-generic-x86_64-with-Ubuntu-14.04-trusty 编译主机内核版本
HOST_BUILD_TYPE=release
BUILD_ID=LRX21V   Android版本名称
OUT_DIR=out   编译结果输出目录
============================================
```

3.make(编译)

在其他的开源工程中，我们一般通过make调用的是Makefile文件就行编译。Android源码目录下的Makefile特别阶段，只有一句话。

```
### DO NOT EDIT THIS FILE ###
include build/core/main.mk
### DO NOT EDIT THIS FILE ###
```

在android源码的编译的系统更模块化，每个目录下都有一个xxx.mk，每个目录都可以单独编译。android源码就是将需要编译的目录下的xxx.mk文件包含到一个Makefile文件中，然后对相关的目录进行编译。

3.编译结果

最终编译的结果都输出到了android源码目录下的out目录。我们来看看，编译后out目录下内容:

```
out
├── host                                SDK 中的各种工具(emulator, adb...)
│   ├── common                          所有主机都用的通用库(java库)
│   │   └── obj
│   │       └── JAVA_LIBRARIES
│   └── linux-x86                       用linux-x86主机上工具程序
└── target                              目标机上运行的各种程序
    ├── common                          所有产品都用的通用程序(java程序或库)
    │   ├── docs                        文档目录
    │   ├── obj                         中间文件
    │   │   ├── APPS                    应用程序的中间文件
    │   │   └── JAVA_LIBRARIES          java库的中间文件
    │   └── R                           资源文件(java代码形式)
    │       ├── android
    │       ├── com
    │       ├── jp
    │       └── org
    ├── product                         产品(特定平台)专用程序
    │   └── fspad-733                   fspad-733型号产品(全志A33平台)程序
    │       ├── data                    用户数据, 该目录中的内容被挂载到/data目录下
    │       └── obj
```

```
├──── recovery                恢复版的根文件系统
├──── root                    根文件系统，装有最基本的命令，该目录中的内容被挂载到/目录下
├──── symbols
├──── system                  系统文件系统，该目录中的内容被挂载到/system目录下
└──── ramdisk.img             root目录打包
      ramdisk-recovery.img    recovery目录打包
      boot.img                kernel + ramdisk.img
      system.img              system目录打包
      userdata.img            data目录打包
      recovery.img            kernel + ramdisk-recovery.img
```

为什么root目录打包后叫ramdisk.img?
(1) 什么是ramdisk?
用内存模拟磁盘存放文件系统，内存称为ramdisk

(2) 为什么root目录打包后叫ramdisk.img?
root目录下的内容在运行时存放在内存
注意：root目录下存放的是根文件系统

Android比较重要的三个img文件：

- **make systemimage ---> system.img**
- **make userdataimage ---> userdata.img**
- **make bootimage ---> boot.img**
- **make ramdisk ---> ramdisk.img**
- **make snod - 快速打包system.img (with this command, it will build a new system.img very quickly. well, you cannot use “make snod” for all the situations. it would not check the dependences. if you change some code in the framework which will effect other applications)**
- **make ramdisk-nodeps**
- **make bootimage-nodeps**