## 深入浅出 - Android系统移植与平台开发（二） - 准备Android开发环境

编译Android源码

关于android系统的编译，Android的官方网站上也给出了详细的说明。

<http://source.android.com/source/building.html>

1.  初始化编译环境

切换到Android源码目录：

$ cd WORKING\_DIRECTORY

执行下面命令，加载编译过程中用到的命令、环境变量：

$ source build/envsetup.sh

2.  选择编译选项

执行下面的命令，从列表中选择一个编译项：

1. $ lunch
2. You're building on Linux
4. Lunch menu... pick a combo:
5. 1. full-eng
6. 2.full\_x86-eng
7. 3.vbox\_x86-eng
8. 4.full\_maguro-userdebug
9. 5.full\_tuna-userdebug
10. 6.full\_panda-eng

我们选择：1，也就是说，编译full-eng的目标，当然我们也可以直接指定编译项，如下：

$ lunch full-eng

其中，lunch命令是指打印或设置出当前系统中设置的编译项，full-eng这个编译项由两部分组成，其中前半部分full表示目标设备为Android的模拟器，官方解释为：fully configured with all languages,apps, input methods，全部的应用程序及语言，输入法等。后半部分eng表示带有调试功能的工程机。

lunch命令打印全部的信息，如下表所示：

| **Build name** | **Device** | **Notes** |
| --- | --- | --- |
| full | emulator | fully configured with all languages, apps, input methods |
| full\_maguro | maguro | full build running on Galaxy Nexus GSM/HSPA+ ("maguro") |
| full\_panda | panda | full build running on PandaBoard ("panda") |
| **Buildtype** | **Use** | |
| user | limited access; suited for production | |
| userdebug | like "user" but with root access and debuggability; preferred for debugging | |
| eng | development configuration with additional debugging tools | |

3.  编译前的准备

由于我们使用ubuntu12.04对Android进行编译，Android对Ubuntu12.04的编译平台的支持不是很推荐，有些库的兼容方面会有一些问题，在编译过程中会产生一些错误，我们要进行一些修正。

错误信息：

g++ selected multilib '32' not installed

或

1. <command-line>:0:0: error:"\_FORTIFY\_SOURCE" redefined [-Werror]
2. <built-in>:0:0: note: this is thelocation of the previous definition
3. cc1plus: all warnings being treated aserrors
4. make: \*\*\*[out/host/linux-x86/obj/EXECUTABLES/obbtool\_intermediates/Main.o] Error 1

原因：

在Android系统过程中，要使用gcc-4.4/g++-4.4的编译器，而Ubuntu12.04的gcc版本为4.6.3

解决方法：

安装gcc-4.4

1. $sudo apt-get install gcc-4.4
2. $sudo apt-get install g++-4.4

进入到/usr/bin目录下，删除gcc对gcc-4.6的链接，创建到新安装的gcc-4.4的链接：

1. $cd /usr/bin
2. $sudo rm -r gcc
3. $sudo ln -s gcc-4.4 gcc

验证结果：

1. $gcc -v
2. $g++ -v

打印其版本为gcc-4.4x、g++-4.4x即可。

3.1  编译源码

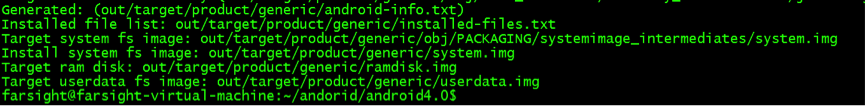
输入下面命令开始编译：

$ make -jn

其中，-jn表示，n个线程同时编译，一般n的值为CPU核的2倍，但是，也要和你的Ubuntu的内存有关系，每个线程在编译时最少需要1G内存，如果你没有很多内存，还是直接使用make命令吧，否则，编译到后面会出错。

这个过程，如果是虚拟机的话，至少要4个多小时，如果是实体机的话，要看配置，一般在1个小时以上。

编译成功结果如下图所示：



编译goldfish内核源码

编译Linux源码必然要先指定gcc交叉编译器，我们直接使用Android自带的arm-eabi-4.4.3编译器，它在WORKING\_DIRECTORY/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin路径下。

$ ls WORKING\_DIRECTORY/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin

http://img.my.csdn.net/uploads/201209/07/1346991605_1912.png

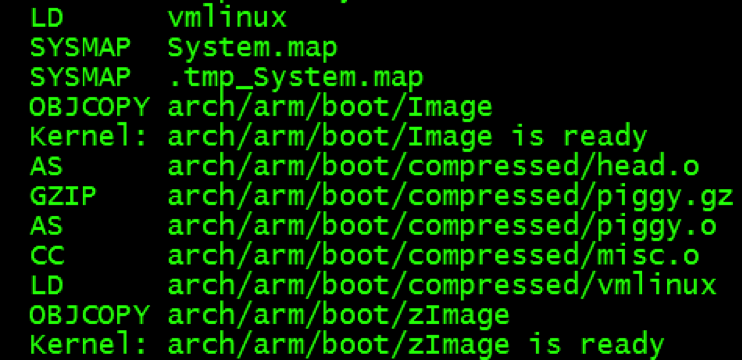
我们编译下面一个编译脚本make\_zImage.sh，让这个脚本去编译goldfish的内核：

1. #!/bin/bash
2. export PATH=/home/farsight/andorid/android4.0/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH
3. export ARCH=arm
4. export SUBARCH=arm
5. export CROSS\_COMPILE=arm-eabi-
6. make goldfish\_armv7\_defconfig
7. make

给make\_zImage.sh添加可执行权限，然后执行该编译脚本：

1. $ chmod a+x make\_zImage.sh
2. $ ./make\_zImage.sh

当我们看到下面的结果时，表示goldfish的内核编译出来了：



4.  Android编译过程分析

如果想要了解，Android的编译过程，可以参照下面三篇博文：

[Android编译系统详解（一）：http:/blog.csdn.net/mr\_raptor/article/details/7539978](http://blog.csdn.net/mr_raptor/article/details/7539978)

[Android编译系统详解（二）：http://blog.csdn.net/mr\_raptor/article/details/7540066](http://blog.csdn.net/mr_raptor/article/details/7540066)

[Android编译系统详解（三）：http:/blog.csdn.net/mr\_raptor/article/details/7540730](http://blog.csdn.net/mr_raptor/article/details/7540730)

按照google给出的编译步骤如下：

  1> source build/envsetup.sh:加载命令

  2> lunch：选择目标平台编译选项

  3> make：执行编译

我们按照编译步骤来分析编译过程的细节，最终添加自己的平台产品的编译选项。

4.1 source build/envsetup.sh

该命令是用来将envsetup.sh里的所有用到的命令加载到环境变量里去。主要几个命令如下：

1. function help() *#显示帮助信息*
2. function check\_product() *# 检查product*
3. function check\_variant() *# 检查变量*
4. function printconfig() *#打印配置*
5. function add\_lunch\_combo() *# 添加lunch项目*
6. function print\_lunch\_menu() *# 打印lunch列表*
7. function lunch() *#配置lunch*
8. function m() *#make from top*
9. function mm() *#make from current directory*
10. function mmm() *#make the supplied directories*

build/envsetup.sh其主要作用如下：

ü  加载了编译时使用到的函数命令，如：help，lunch，m，mm，mmm等

ü  添加了两个编译选项：generic-eng和simulator，这两个选项是系统默认选项

ü  查找vendor/\*/\*/vendorsetup.sh和device/\*/\*/vendorsetup.sh，也就是说从vendor/<厂商目录>/<平台设备目录>/目录和device/<厂商目录>/<平台设备目录>/目录下，查找vendorsetup.sh，如果找到的话，加载执行它，添加厂商自定义产品的编译选项。

如果想要添加自己的目标设备编译选项，我们就要在device或vendor目录下创建设备厂商，然后在设备厂商目录下，创建一个vendorsetup.sh文件，在里面添加上自己的编译项。

1. $mkdir -p device/farsight/fsdevice
2. $touch device/farsight/fsdevice/vendorsetup.sh
3. $echo "add\_lunch\_combo fs100-eng" > device/farsight/fsdevice/vendorsetup.sh

*注：*

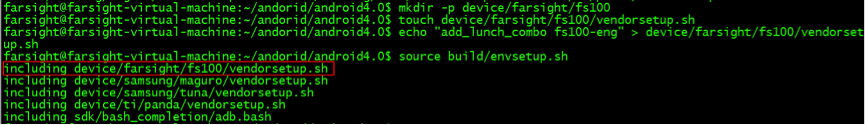
*farsight：厂商名*

fsdevice*：平台设备名*

*fs100-eng：自定义的目标产品名为fs100，eng为编译目标类型*

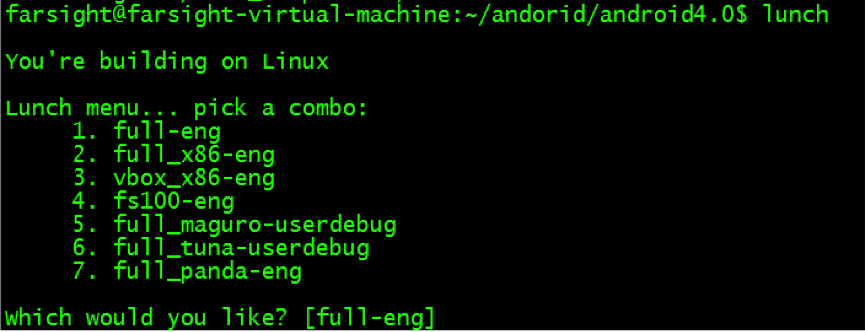
*所有的编译项的格式都是：设备名-编译目标类型，编译目标类型为：eng，user，userdebug，三种类型之一（详细见3.1节）。*

下面验证一下我们的结果，当在Android目录下，再次执行source build/envsetup.sh时，我们可以看到我们新添加的编译项：



4.2执行lunch full-eng

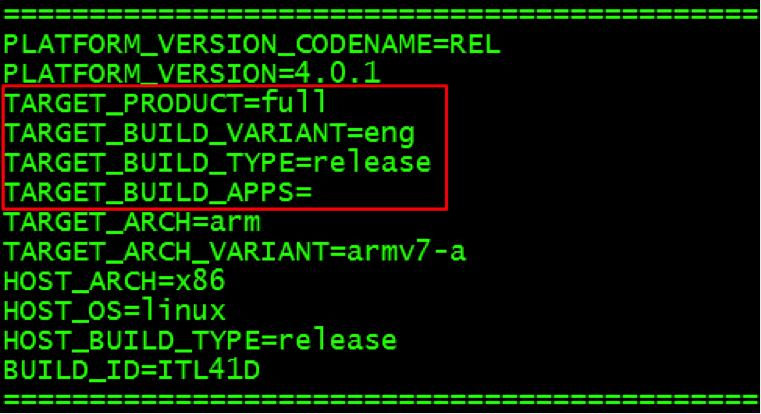
我们当然也可以直接执行lunch命令，然后在弹出菜单里面来选择我们的编译项，如下图所示：



*注：我们添加的编译项这时已经存在了，但是如果你选择了它，在编译时会出错，因为我们还没有对应编译项的配置文件。*

lunch命令可以带参数和不带参数，最终导出一些重要的环境变量，从而影响编译系统的编译结果。导出的变量如下（以lunch full-eng为例）：

1. TARGET\_PRODUCT=full
2. TARGET\_BUILD\_VARIANT=eng
3. TARGET\_BUILD\_TYPE=release



4.3执行make命令

当我们输入make命令后，就开始android系统的编译了，所有的Makefile都通过build/core/main.mk这个文件组织在一起，它定义了一个默认goals：droid，当我们在TOP目录下，敲make实际上就等同于我们执行make droid。

当make include所有的文件，完成对所有make文件的解析以后就会寻找生成droid的规则，依次生成它的依赖，直到所有满足的模块被编译好，然后使用相应的工具打包成相应的img。其中，config.mk，envsetup.mk，product\_config.mk文件是编译用户指定平台系统的关键文件。Android的编译系统比较复杂，详细的说明请参照前面给出博文内容。

下面列出在编译过程中经常使用的一些命令：

1. source build/envsetup.sh：加载编译命令，产生编译选项
2. lunch或lunch xxx-yyy：打印编译选项菜单或指定编译选项，xxx表示产品，yyy表示编译类型
3. make：根据lunch选项，编译Android系统，最后产出为：system.img，ramdisk.img，userdate.img
4. m：和make命令一样
5. mm：从当前目录下开始向下编译目标
6. mmm：指定一个目录，仅编译指定目录下的目标
7. make snod：只将out/target/product/XXX/system/目录下的内容打包生成system.img，不会检查依赖关系
8. make bootimage：只将out/target/product/XXX/root/目录下的内容打包生成ramdisk.img