华清远见 FARSIGHT 始于2004 | 嵌入式培训专家

华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

《Android 系统移植和驱动开发》

作者：华清远见

专业始于专注 卓识源于远见

第 4 章 Android 移植环境搭建

本章目标

本章讲解 Android 移植的第一步，主要学习如何搭建 Android 移植的环境。因为 Android 底层是基于 Linux 内核的，所以本章从交叉编译环境等嵌入式开发环境的搭建开始，介绍了 Bootloader 的概念及 U-Boot 的编译和移植方法；然后介绍了 Linux 内核的相关知识，以及内核编译和移植的方法。本章主要内容：

- 构建 Android 移植环境。
- Bootloader 介绍。
- Android 内核与移植。

4.1 构建 Android 移植交叉开发环境

构建开发环境是任何开发工作的基础，对于软、硬件非常丰富的嵌入式系统来说，构建高效、稳定的环境是能否开展工作的重要因素之一。本节将介绍如何构建一套 Android 移植环境。在构建开发环境以前，有必要了解相关开发流程。因为 Android 移植往往会涉及多个层面，这与 Android 应用程序开发有很大不同，流程如下：

（1）下载 Android 内核。

（2）熟悉开发环境和工具。交叉开发环境是 Android 系统移植开发的基本模型。Linux 环境配置、GNU 工具链、测试工具甚至集成开发环境都是 Android 系统移植的利器。

（3）熟悉 Linux 内核。因为 Android 系统移植开发一般需要重新定制 Linux 内核，所以熟悉内核配置、编译和移植很重要。

（4）熟悉目标板引导方式。开发板的 Bootloader 负责硬件平台最基本的初始化，并且具备引导 Linux 内核启动的功能。由于硬件平台是专门定制的，一般需要修改编译 Bootloader。

4.1.1 嵌入式交叉编译环境搭建

搭建交叉编译环境是 Android 移植的第一步，也是关键的一步。不同的体系结构、不同的操作内容甚至是不同版本的内核，都会用到不同的交叉编译器。选择交叉编译器非常重要，有些交叉编译器经常会有部分的 BUG，都会导致最后的代码无法正常运行。

交叉编译器完整的安装一般涉及多个软件的安装（读者可以从 ftp://gcc.gnu.org/pub/下载），包括 binutils、gcc、glibc、glibc-linuxthreads 等软件。其中，binutils 主要用于生成一些辅助工具，如 readelf、objcopy、objdump、as、ld 等；gcc 是用来生成交叉编译器的，主要生成 arm-linux-gcc 交叉编译工具（应该说，生成此工具后已经搭建起了交叉编译环境，可以编译 Linux 内核了，但由于没有提供标准用户函数库，用户程序还无法编译）；glibc 主要提供用户程序所使用的一些基本的函数库，glibc-linuxthreads 是线程相关函数库。这样，交叉编译环境就完全搭建起来了。

上述搭建交叉编译环境比较复杂，很多步骤都涉及对硬件平台的选择。因此，现在嵌入式平台社区或厂商一般会提供在各种平台上测试通过的交叉编译器，而且也有很多把以上安装步骤全部写入脚本文件或者以发行包的形式提供，这样就大大方便了用户的使用。例如，crosstool 是美国人 Dan Kegel 开发的一套可以自动编译不同版本的交叉编译器。

本书采用广泛使用的 cross-4.3.2 交叉编译器工具链，使用非常简单。

```
$ mkdir -p /usr/local/arm  /* 这是交叉编译器安装目录*/
$ cp cross-4.3.2.bar.bz2 /usr/local/arm
$ cd /usr/local/arm
$ tar jxvf cross-4.3.2.tar.gz
```

此时在/usr/local/arm/4.3.2/bin/下已经出现了很多交叉编译工具。显示如下：

```
 arm-none-linux-gnueabi-addr2line    arm-none-linux-gnueabi-gfortran
arm-none-linux-gnueabi-ar          arm-none-linux-gnueabi-gprof
arm-none-linux-gnueabi-as          arm-none-linux-gnueabi-ld
arm-none-linux-gnueabi-c++         arm-none-linux-gnueabi-ldd
arm-none-linux-gnueabi-cc          arm-none-linux-gnueabi-nm
arm-none-linux-gnueabi-c++filt      arm-none-linux-gnueabi-objcopy
arm-none-linux-gnueabi-cpp         arm-none-linux-gnueabi-objdump
arm-none-linux-gnueabi-ct-ng.config  arm-none-linux-gnueabi-populate
arm-none-linux-gnueabi-g++         arm-none-linux-gnueabi-ranlib
arm-none-linux-gnueabi-gcc         arm-none-linux-gnueabi-readelf
arm-none-linux-gnueabi-gcc-4.3.2    arm-none-linux-gnueabi-run
arm-none-linux-gnueabi-gccbug      arm-none-linux-gnueabi-size
arm-none-linux-gnueabi-gcov        arm-none-linux-gnueabi-sstrip
arm-none-linux-gnueabi-gdb         arm-none-linux-gnueabi-strings
arm-none-linux-gnueabi-gdbtui      arm-none-linux-gnueabi-stri
```

华清远见

可以看到，这个交叉编译工具确实集成了 binutils、gcc、glibc 这几个软件，而每个软件也都有比较复杂的配置信息。

接下来，在环境变量 PATH 中添加路径，就可以直接使用 arm-none-linux—gnueabi-gcc 命令了。

```
$ export PATH=$PATH:/usr/local/arm/4.3.2/bin
```

把交叉开发工具链的路径添加到环境变量 PATH 中，这样可以方便地在 Bash 或者 Makefile 中使用这些工具。通常可以在环境变量的配置文件有两个：profile 类文件和 bashrc 类文件。

- profile 类文件：用户登录时第一次仅运行一次，profile 类文件包括每个用户主目录下的.profile 文件和/etc/profile 等。哪个用户登录就会运行主目录下的.profile 文件的脚本。
- bashrc 类文件：每当打开 bash shell 时（例如，当打开一个虚拟终端时）运行该脚本文件。bash 类文件包括每个用户主目录下的.bashrc 文件和/etc/bash.bashrc 等。

把环境变量配置的命令添加到其中一个文件中即可。

```
$ arm-linux-gcc -v /*查看交叉编译器的版本信息*/
arm-none-linux-gnueabi-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /home/linux/crosstooll/toolchain_build/targets/src/gcc-4.3.2/configure
--build=i686-build_pc-linux-gnu --host=i686-build_pc-linux-gnu --target=arm-none-linux-gnueabi
--prefix=/usr/local/arm/4.3.2 --with-sysroot=/home/linux/toolchain/arm-none-linux-gnueabi//sys-root
--enable-languages=c,c++,fortran --disable-multilib --with-arch=armv4t --with-cpu=arm9tdmi
--with-tune=arm920t --with-float=soft --with-pkgversion=crosstool-NG-1.8.1-farsight
--disable-sjlj-exceptions --enable-__cxa_atexit --disable-libmudflap
--with-gmp=/home/linux/crosstooll/toolchain_build/targets/arm-none-linux-gnueabi/build/static--with-mpfr=/h
ome/linux/crosstooll/toolchain_build/targets/arm-none-linux-gnueabi/build/static--enable-threads=posix--ena
ble-target-optspace--with-local-prefix=/home/linux/toolchain/arm-none-linux-gnueabi//sys-root --disable-nls
--enable-symvers=gnu--enable-c99 --enable-long-long
Thread model: posix
gcc version 4.3.2
```

从上面打印的版本信息中可以看到“--prefix=/usr/local/arm/ 4.3.2”，这就是交叉编译器安装的路径。它是在编译前通过 prefix 选项配置的。所以，这个工具链安装的路径必须是/usr/local/arm/4.3.2。

4.1.2 主机交叉开发环境配置

1．配置控制台程序

要查看目标板的输出，可以使用控制台程序。在各种操作系统上一般都有现成的控制台程序可以使用。例如，Windows 操作系统中有超级终端（Hyper Terminal）工具；Linux/UNIX 操作系统有 minicom（使用“minicom”命令启动该软件）等工具。无论什么操作系统和通信工具，都可以作为串口控制台。如果在 Windows 平台上运行 Linux 虚拟机，这个串口通信软件可以任选一种。配置一个超级终端，如图 4-1 所示，配置 minicom，如图 4-2 所示（使用“minicom –s”命令进入配置界面），配置参数包括串口号、通信速率、数据位数、停止位数、奇偶校验、数据流控制等设置。一次配置可以保存下来，以供以后使用。

图 4-1 配置串口控制台

图 4-2 minicom 配置

2. 配置 TFTP 服务

TFTP 是一个传输文件的简单协议，它是基于 UDP 实现的。此协议设计时是进行小文件传输的，因此它不具备通常的 FTP 的许多功能，它只能从文件服务器上获得或写入文件，不能列出目录，不进行认证，传输 8 位数据。

TFTP 分为客户端和服务器端两种。通常，首先在宿主机上开启 TFTP 服务器端服务，设置好 TFTP 的根目录内容（也就是供客户端下载的文件），接着，在目标板上开启 TFTP 的客户端程序（TFTP 客户端主要在 Bootloader 交互环境下运行，几乎所有 Bootloader 都提供该服务，用于下载操作系统内核和文件系统）。这样，把目标板和宿主机用直连线相连之后，就可以通过 TFTP 传输可执行文件了。下面分别讲述在 Linux 下和 Windows 下的配置方法。

1）Linux 下 TFTP 服务配置

Linux 下 TFTP 的服务是由 xinetd（还有 openbsd-inetd 等其他服务）所设定的，默认情况下是处于关闭状态的。

首先，要修改 TFTP 的配置文件，开启 TFTP 服务，代码如下：

```
$ vim /etc/xinetd.d/TFTP
service TFTP
{
          socket_type= dgram
          protocol = udp
          wait = yes
          user = root
          server = /usr/sbin/in.TFTPd
          server_args = -s /TFTPboot
          disable = no
          per_source = 11
          cps = 100 2
          flags = IPv4
}
```

在这里，主要要将“disable=yes”改为“no”，另外，由“server_args”可以看出，TFTP 服务器端的默认根目录为“/TFTPboot”，用户若需要，可以更改为其他目录。

接下来，重启 xinetd 服务，使刚才的更改生效，代码如下：

```
$ /etc/init.d/xinetd restart
```

接着，使用命令“netstat -au”以确认 TFTP 服务是否已经开启，代码如下：

```
$ netstat -au | grep TFTP
Proto Recv-Q Send-Q  Local Address      Foreign Address          State
udp        0      0  *:TFTP             *:*
```

这时，用户就可以把所需要的传输文件放到“/TFTPboot”目录下，这样，主机上的 TFTP 服务就可以建立起来了。用网络交叉线把目标板和宿主机连起来，并且将其配置成一个网段的地址，再在目标板上启动 TFTP 客户端程序（注意：不同的 Bootloader 所使用的命令会有所不同，读者可查看帮助来获得确切的命令名及格式，本书以 U-Boot 为例讲解），代码如下：

```
# TFTP 0x30008000 zImage
TFTP from server 192.168.1.112; our IP address is 192.168.1.120
Filename 'zImage'.
Load address: 0x33000000
Loading:#################################################################
#################################################################
#################################################################
done
Bytes transferred = 881988 (d7544 hex)
```

可以看到，此处目标板使用的 IP 为“192.168.1.120”，宿主机使用的 IP 为“192.168.1.112”，下载到目标板的地址为 0x33000000，文件名为“zImage”。

2）Windows 下 TFTP 服务配置

集团官网：www.hqyj.com 嵌入式学院：www.embedu.org 企业学院：www.farsight.com.cn 北京|上海|深圳|成都|南京|武汉|西安|广州
移动互联网学院：www.3g-edu.org 物联网学院：www.topsight.cn 咨询电话：400-706-1880，15010390966

在 Windows 下配置 TFTP 服务需要安装使用 TFTP 服务器软件，常见的可使用 TFTPd32，网上有很多下载该软件的地方，读者可以自行下载。要注意的是，该软件是 TFTP 的服务器端，而目标板上则是 TFTP 的客户端。打开该软件，如图 4.3 所示。

接下来，用户可以在 setting 中配置服务器端的各个选项，如 IP 地址等，如图 4.4 所示。

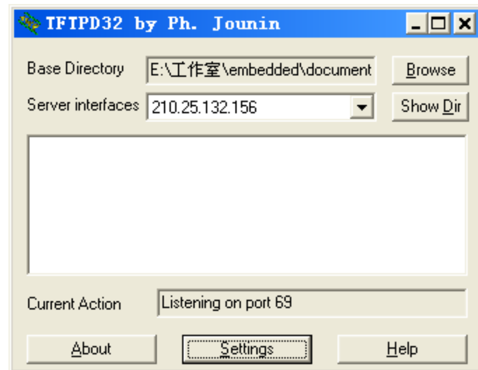


图 4.3 TFTPd32 软件

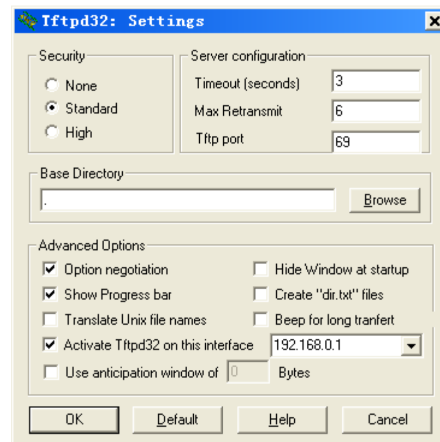


图 4.4 TFTPd32 的配置界面

另外，还需要在 Browse 中选择 TFTP 的服务器端根目录。这时，TFTPd 会提示用户重启该软件，使修改的参数生效。至此，TFTP 的服务就配置完毕了。此时可以用直连线连接目标机和宿主机，且在目标机上开启 TFTP 服务进行文件传输。

2．NFS 文件系统

NFS 是 Network File System 的简称，最早是由 Sun 公司提出发展起来的，其目的就是让不同的机器、不同的操作系统之间可以彼此共享文件。

NFS 可以让不同的主机通过网络将远端的 NFS 服务器共享的文件安装到自己的系统中，从客户端看来，使用 NFS 的远端文件就像是使用本地文件一样。在嵌入式中使用 NFS 会使应用程序的开发变得十分方便，并且不用反复地进行烧写镜像文件。

NFS 的使用分为服务器端和客户端，其中服务器端提供要共享的文件，而客户端则通过挂载“mount”这一动作来实现对共享文件的访问操作。在嵌入式开发中，通常 NFS 服务端在宿主机上运行，而客户端在目标板上运行。

NFS 服务器端是通过读入它的配置文件“/etc/ exports”来决定所共享的文件目录的。在这个配置文件中，每一行都代表一项要共享的文件目录，以及所指定的客户端对其的操作权限。客户端可以根据相应的权限，对该目录下的所有目录文件进行访问。

配置文件中每一行的格式如下：

```
[共享的目录] [客户端主机名称或 IP]([参数 1，参数 2…])
```

在这里，主机名或 IP 是可供共享的客户端主机名或 IP，若对所有的 IP 都可以访问，则可用“*”表示。这里的参数有很多中组合方式，表 4.1 列出了常见的参数。

表 4.1 NFS 配置文件的常见参数

选　　项	参 数 含 义
Rw	可读写的权限
Ro	只读的权限
no_root_squash	NFS 客户端分享目录使用者的权限，即如果客户端使用的是 root 用户，那么对于这个共享的目录而言，该客户端就具有 root 的权限
Sync	资料同步写入到内存与硬盘当中
Async	资料会先暂存于内存中，而非直接写入硬盘

下面是配置文件“/etc/exports”的一个示例：

```
$ cat /etc/exports
/home/david/project    *(rw,sync,no_root_squash)
```

在设定完配置文件之后，需要启动 NFS 服务和 portmap 服务，这里的 portmap 服务允许 NFS 客户端查看 NFS 服务所用的端口，在它被激活之后，就会出现一个端口号为 111 的 sun RPC（远端过程调用）的服务。这是 NFS 服务中必须实现的一项，因此，也必须把它开启，代码如下：

```
$ /etc/init.d/portmap restart
启动 portmap:                    [确定]
$ /etc/init.d/nfs restart(在 Ubuntu 中应为/etc/init.d/nfs-kernel-server)
启动 NFS 服务:                   [确定]
关掉 NFS 配额:                   [确定]
启动 NFS 守护进程:               [确定]
启动 NFS mountd:                 [确定]
```

可以看到，在启动 NFS 服务时启动了 mountd 进程，它是 NFS 挂载服务，用于处理 NFSD 递交过来的客户端请求。另外，还会激活至少两个以上的系统守护进程，然后开始监听客户端的请求，用 dmesg 命令（或者 cat /var/log/messages）可以看到操作是否成功。与 NFS 相关的还有两个命令，可以方便 NFS 的使用。

其中一个是 showmount，它可以显示 NFS 服务器的挂载信息，其格式为：

```
showmount [选项]
```

表 4.2 列出了 showmount 的常见选项。

表 4.2　showmount 的常见选项

选　项	参 数 含 义
-a	列出客户端主机名或 IP 地址，和挂载在主机的目录
-e	显示 NFS 服务器的导出列表

另一个是 exportfs，它可以重新扫描“/etc/exports”，使用户在修改“/etc/exports”配置文件时不需要每次重启 NFS 服务，其格式为：

```
exportfs [选项]
```

表 4.3 列出了 exportfs 的常见选项。

表 4.3　exportfs 常见选项

选　项	参 数 含 义
-a	全部挂载（或卸载）/etc/exports 中的设定文件目录
-r	重新挂载/etc/exports 中的设定文件目录
-u	卸载某一目录
-v	在 export 时，将共享的目录显示到屏幕上

用户若希望 NFS 服务在每次系统引导时自动开启，可使用以下命令：

```
# /sbin/chkconfig nfs on
(在 Ubuntu 中应该输入 /sbin/chkconfig nfs-kernel-server on)
```

4.2　Bootloader

Bootloader 是在操作系统运行之前执行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映像表，从而建立适当的系统软硬件环境，为最终调用操作系统内核做好准备。

对于嵌入式系统，Bootloader 是基于特定硬件平台来实现的。因此，几乎不可能为所有的嵌入式系统建立一个通用的 Bootloader，不同的处理器架构都有不同的 Bootloader。Bootloader 不但依赖于 CPU 的体系结构，而且依赖于嵌入式系统板级设备的配置。对于两块不同的嵌入式板而言，即使它们使用同一种处

集团官网：www.hqyj.com　嵌入式学院：www.embedu.org　企业学院：www.farsight.com.cn　北京 | 上海 | 深圳 | 成都 | 南京 | 武汉 | 西安 | 广州
移动互联网学院：www.3g-edu.org　物联网学院：www.topsight.cn　咨询电话：400-706-1880，15010390966

理器，要想让运行在一块板子上的 Bootloader 程序也能运行在另一块板子上，一般也都需要修改 Bootloader 的源程序。

反过来，大部分 Bootloader 仍然具有很多共性，某些 Bootloader 也能够支持多种体系结构的嵌入式系统。例如，U-Boot 就同时支持 PowerPC、ARM、MIPS 和 X86 等体系结构，支持的板子有上百种。通常，它们都能够自动从存储介质上启动，都能够引导操作系统启动，并且大部分都支持串口和以太网接口。

4.2.1 Bootloader 的种类

嵌入式系统世界已经有各种各样的 Bootloader，种类划分也有多种方式。除了按照处理器体系结构不同划分以外，还有功能复杂程度的不同。

首先区分一下“Bootloader”和“Monitor”的概念。严格来说，“Bootloader”只是引导设备并且执行主程序的固件；而“Monitor”还提供了更多的命令行接口，可以进行调试、读写内存、烧写 Flash、配置环境变量等。“Monitor”在嵌入式系统开发过程中可以提供很好的调试功能，开发完成以后，就完全设置成了一个“Bootloader”。所以，习惯上大家把它们统称为 Bootloader。

表 4.4 列出了 Linux 的开放源码引导程序及其支持的体系结构，给出了 X86、ARM、PowerPC 体系结构的常用引导程序，并且注明了每一种引导程序是否为“Monitor”。

表 4.4 开放源码的 Linux 引导程序

Bootloader	Monitor	描述	X86	ARM	PowerPC
LILO	否	Linux 磁盘引导程序	是	否	否
GRUB	否	GNU 的 LILO 替代程序	是	否	否
Loadlin	否	从 DOS 引导 Linux	是	否	否
ROLO	否	从 ROM 引导 Linux 而不需要 BIOS	是	否	否
Etherboot	否	通过以太网卡启动 Linux 系统的固件	是	否	否
LinuxBIOS	否	完全替代 BIOS 的 Linux 引导程序	是	否	否
BLOB	否	LART 等硬件平台的引导程序	否	是	否
Vivi	是	主要为 S3C2410 等三星处理器引导 Linux	否	是	否
U-Boot	是	通用引导程序	是	是	是
RedBoot	是	基于 eCos 的引导程序	是	是	是

华清远见

1．X86

X86 的工作站和服务器上一般使用 LILO 和 GRUB。LILO 曾经是 Linux 发行版主流的 Bootloader。不过现在几乎所有的发行版已经都使用了 GRUB，GRUB 比 LILO 有更有好的显示接口，使用配置也更加灵活方便。

在某些 X86 嵌入式单板机或者特殊设备上，会采用其他的 Bootloader，如 ROLO。这些 Bootloader 可以取代 BIOS 的功能，能够从 Flash 中直接引导 Linux 启动。现在 ROLO 支持的开发板已经并入 U-Boot，所以 U-Boot 也可以支持 X86 平台。

2．ARM

ARM 处理器的芯片商很多，所以每种芯片的开发板都有自己的 Bootloader。结果 ARM Bootloader 也变得多种多样。最早的有为 ARM720 处理器开发的 Bootloader，又有了 armboot，StrongARM 平台的 BLOB，还有 S3C2410 处理器开发板上的 vivi 等。现在 armboot 已经并入了 U-Boot，所以 U-Boot 也支持 ARM/XSCALE 平台。U-Boot 已经成为 ARM 平台事实上的标准 Bootloader。

3．PowerPC

PowerPC 平台的处理器有标准的 Bootloader，即 PPCBOOT。PPCBOOT 在合并 armboot 等之后，创建

了 U-Boot，成为各种体系结构开发板的通用引导程序。U-Boot 仍然是 PowerPC 平台的主要 Bootloader。

4．MIPS

MIPS 公司开发的 YAMON 是标准的 Bootloader，也有许多 MIPS 芯片商为自己的开发板写了 Bootloader。现在，U-Boot 也已经支持 MIPS 平台。

5．SH

SH 平台的标准 Bootloader 是 sh-boot。RedBoot 在这种平台上也很好用。

6．M68K

M68K 平台没有标准的 Bootloader。RedBoot 能够支持 M68K 系列的系统。

需要说明的是 RedBoot，它几乎能够支持所有的体系结构，包括 MIPS、SH、M68K 等。RedBoot 是以 eCos 为基础，采用 GPL 许可的开源软件工程。现在由 core eCos 的开发人员维护，源码下载网址是 http://www.ecoscentric.com/snapshots。RedBoot 的文档也相当完善，有详细的使用手册 *RedBoot User's Guide*。

4.2.2 U-Boot 编译与使用

最早，DENX 软件工程中心的 Wolfgang Denk 基于 8xxrom 的源码创建了 PPCBOOT 工程，并且不断添加处理器的支持。后来，Sysgo Gmbh 把 PPCBOOT 移植到 ARM 平台上，创建了 ARMBOOT 工程。然后以 PPCBOOT 工程和 ARMBOOT 工程为基础，创建了 U-Boot 工程。

现在，U-Boot 已经能够支持 PowerPC、ARM、X86、MIPS 体系结构的上百种开发板，已经成为功能最多、灵活性最强并且开发最积极的开放源码 Bootloader。U-Boot 的源码包可以从 sourceforge 网站下载，还可以订阅该网站活跃的 U-Boot Users 邮件论坛，这个邮件论坛对于 U-Boot 的开发和使用都很有帮助。

U-Boot 软件包下载网站：http://sourceforge.net/project/U-Boot。

U-Boot 邮件列表网站：http://lists.sourceforge.net/lists/listinfo/U-Boot-users/。

DENX 相关的网站：http://www.denx.de。

解压 u-boot-2010.03.tar.bz2 就可以得到全部 U-Boot 源程序。在顶层目录下有 29 个子目录，分别存放和管理不同的源程序。这些目录中所要存放的文件有其规则，可以分为如下 3 类：

- 与处理器体系结构或者开发板硬件直接相关。
- 一些通用的函数或者驱动程序。
- U-Boot 的应用程序、工具或者文件。

表 4.5 列出了 U-Boot 顶层目录下各级目录的存放原则。

表 4.5 U-Boot 的源码顶层目录说明

目　录	特　性	解 释 说 明
board	平台依赖	存放电路板相关的目录文件，如 RPXlite(mpc8xx)、smdk2410(arm920t)、sc520_cdp(x86) 等目录
Cpu	平台依赖	存放 CPU 相关的目录文件，如 mpc8xx、ppc4xx、arm720t、arm920t、xscale、i386 等目录
lib_ppc	平台依赖	存放对 PowerPC 体系结构通用的文件，主要用于实现 PowerPC 平台通用的函数
lib_arm	平台依赖	存放对 ARM 体系结构通用的文件，主要用于实现 ARM 平台通用的函数
lib_i386	平台依赖	存放对 X86 体系结构通用的文件，主要用于实现 X86 平台通用的函数
lib_avr32	平台依赖	存放对 AVR32 体系结构通用的文件，主要用于实现 AVR32 平台通用的函数
lib_blackfin	平台依赖	存放对 BLACKFIN 体系结构通用的文件，主要用于实现 BLACKFIN 平台通用的函数
lib_m68k	平台依赖	存放对 m68k 体系结构通用的文件，主要用于实现 m68k 平台通用的函数

lib_microblaze	平台依赖	存放对 microblaze 体系结构通用的文件，主要用于实现 microblaze 平台通用的函数
lib_mips	平台依赖	存放对 MIPS 体系结构通用的文件，主要用于实现 MIPS 平台通用的函数
lib_nios	平台依赖	存放对 NIOS 体系结构通用的文件，主要用于实现 NIOS 平台通用的函数
lib_nios2	平台依赖	存放对 NIOS 体系结构通用的文件，主要用于实现 NIOS2 平台通用的函数
lib_sh	平台依赖	存放对 SH 体系结构通用的文件，主要用于实现 SH 平台通用的函数
lib_sparc	平台依赖	存放对 SPARC 体系结构通用的文件，主要用于实现 SPARC 平台通用的函数
libfdt	通用	支持设备树的库文件
api	通用	存放 uboot 提供的接口函数
common	通用	通用的代码，涵盖各个方面，已命令行处理为主
disk	通用	磁盘分区相关代码

续表

目　录	特　性	解 释 说 明
Nand_spl	通用	NAND 存储器相关代码
include	通用	头文件和开发板配置文件，所有开发板的配置文件都在 configs 目录下
common	通用	通用的多功能函数实现
lib_generic	通用	通用库函数的实现
Net	通用	存放网络相关程序
Fs	通用	存放文件系统相关程序
post	通用	存放上电自检程序
drivers	通用	通用的设备驱动程序，主要有以太网接口的驱动
Disk	通用	硬盘接口程序
examples	应用例程	一些独立运行的应用程序的例子，如 helloworld
tools	工具	存放制作 S-Record 或者 U-Boot 格式的镜像等工具，如 mkimage
Doc	文档	开发使用文档
Rtc	通用	RTC 的驱动程序

华清远见

U-Boot 的源代码包含对几十种处理器、数百种开发板的支持。不过对于特定的开发板，配置编译过程只需要其中的部分程序。这里以 S3C2410 处理器为例，具体分析 S3C2410 处理器和开发板所依赖的程序，以及 U-Boot 的通用函数和工具。

U-Boot 的源代码是通过 gcc 和 Makefile 组织编译的。顶层目录下的 Makefile 首先可以设置开发板的定义，然后递归地调用各级子目录下的 Makefile，最后把编译过的程序链接成 U-Boot 映像。

它负责 U-Boot 整体配置编译。按照配置的顺序阅读其中关键的几行。

每种开发板在 Makefile 都需要有板子配置的定义。例如，smdk2410 开发板的定义如下。

```
smdk2410_config :   unconfig
 @$(MKCONFIG) $(@:_config=) arm arm920t smdk2410 samsung s3c24x0
```

执行配置 U-Boot 的命令 make smdk2410_config，通过 mkconfig 脚本生成 include/config.mk 的配置文件。文件内容正是根据 Makefile 对开发板的配置生成的。

```
ARCH = arm
CPU = arm920t
BOARD = smdk2410
```

```
VENDOR = samsung
SoC = s3c24x0
```

上面的 include/config.mk 文件定义了 ARCH、CPU、BOARD、VENDOR、SoC 这些变量。这样，硬件平台依赖的目录文件可以根据这些定义来确定。SMDK2410 平台相关目录如下：

```
board/Samsung/smdk2410
cpu/arm920t/
cpu/arm920t/s3c24x0/
lib_arm/
include/configs/smdk2410.h
```

再回到顶层目录的 Makefile 文件开始的部分，其中，下列几行包含了这些变量的定义。

```
 # load ARCH, BOARD, and CPU configuration
  include $(obj)include/config.mk
  export  ARCH CPU BOARD VENDOR SOC
```

Makefile 的编译选项和规则在顶层目录的 config.mk 文件中定义。各种体系结构通用的规则直接在这个文件中定义。通过 ARCH、CPU、BOARD、VENDOR、SoC 等变量为不同硬件平台定义不同选项。不同体系结构的规则分别包含在 ppc_config.mk、arm_config.mk、mips_config.mk 等文件中。

顶层目录的 Makefile 中还要定义交叉编译器，以及编译 U-Boot 所依赖的目标文件。

```
ifeq (arm,$(ARCH))
    CROSS_COMPILE ?=arm-none-linux-gnueabi-
# 交叉编译器的前缀
endif
# load other configuration
include $(TOPDIR)/config.mk
# U-Boot objects....order is important (i.e. start must be first)
OBJS  = cpu/$(CPU)/start.o    # 处理器相关的目标文件
   …
LIBS  = lib_generic/libgeneric.a  #定义依赖的目录，每个目录下先把目标文件链接成*.a 文件
LIBS += lib_generic/lzma/liblzma.a
LIBS += lib_generic/lzo/liblzo.a
LIBS += $(shell if [ -f board/$(VENDOR)/common/Makefile ];
Then echo      "board/$(VENDOR)/common/lib$(VENDOR).a"; fi)
LIBS += cpu/$(CPU)/lib$(CPU).a
ifdef SOC
LIBS += cpu/$(CPU)/$(SOC)/lib$(SOC).a
endif
ifeq ($(CPU),ixp)
LIBS += cpu/ixp/npe/libnpe.a
endif
LIBS += lib_$(ARCH)/lib$(ARCH).a
   …
```

然后还有 U-Boot 镜像编译的依赖关系。

```
ALL += $(obj)u-boot.srec $(obj)u-boot.bin $(obj)System.map $(U_BOOT_NAND) $(U_BOOT_ONENAND)
all:          $(ALL)
$(obj)u-boot.hex:      $(obj)u-boot
              $(OBJCOPY) ${OBJCFLAGS} -O ihex $< $@
$(obj)u-boot.srec:     $(obj)u-boot
    $(OBJCOPY) -O srec $< $@
$(obj)u-boot.bin:      $(obj)u-boot
  $(OBJCOPY) ${OBJCFLAGS} -O binary $< $@
$(obj)u-boot.ldr:      $(obj)u-boot
              $(CREATE_LDR_ENV)
             $(LDR) -T $(CONFIG_BFIN_CPU) -c $@ $< $(LDR_FLAGS)
$(obj)u-boot.ldr.hex:  $(obj)u-boot.ldr
            $(OBJCOPY) ${OBJCFLAGS} -O ihex $< $@ -I binary
$(obj)u-boot.ldr.srec: $(obj)u-boot.ldr
             $(OBJCOPY) ${OBJCFLAGS} -O srec $< $@ -I binary
$(obj)u-boot.img:      $(obj)u-boot.bin
          ./tools/mkimage -A $(ARCH) -T firmware -C none \
          -a $(TEXT_BASE) -e 0 \
```

```
            -n $(shell sed -n -e 's/.*U_BOOT_VERSION//p' $(VERSION_FILE) | \
                    sed -e 's/"[     ]*$$/ for $(BOARD) board"/') \
            -d $< $@
$(obj)u-boot.imx:      $(obj)u-boot.bin
             $(obj)tools/mkimage -n $(IMX_CONFIG) -T imximage \
          -e $(TEXT_BASE) -d $< $@
$(obj)u-boot.kwb:      $(obj)u-boot.bin
        $(obj)tools/mkimage -n $(KWD_CONFIG) -T kwbimage \
       -a $(TEXT_BASE) -e $(TEXT_BASE) -d $< $@
$(obj)u-boot.sha1:     $(obj)u-boot.bin
            $(obj)tools/ubsha1 $(obj)u-boot.bin
$(obj)u-boot.dis:      $(obj)u-boot
          $(OBJDUMP) -d $< > $@
```

Makefile 默认的编译目标为 all，包括 U-Boot.srec、U-Boot.bin、System.map。U-Boot.srec 和 U-Boot.binU-Boot 就是通过 ld 命令按照 U-Boot.map 地址表把目标文件组装成 U-Boot 的。其他 Makefile 内容就不再详细分析了，上述代码分析应该可以为阅读代码提供了一个线索。

除了编译过程 Makefile 以外，还要在程序中为开发板定义配置选项或者参数。这个头文件是 include/configs/<board_name>.h。<board_name>用相应的 BOARD 定义代替。

这个头文件中主要定义了两类形式的参数。

参数用来选择处理器、设备接口、命令、属性等，用来定义总线频率、串口波特率、Flash 地址等参数。

大部分参数前缀是 CONFIG_，例如：

```
#define CONFIG_ARM920T        1
#define CONFIG_KGDB_BAUDRATE   115200
#define CONFIG_CS8900
#define CONFIG_KGDB_BAUDRATE   115200
```

另一类形式的参数如下：

```
#define PHYS_FLASH_SIZE        0x00100000
#define  USE_920T_MMU          1
```

根据对 Makefile 的分析，编译分为两步。第 1 步是配置，如 make smdk2410_config；第 2 步是编译，执行 make 就可以了。

编译完成后，可以得到 U-Boot 各种格式的映像文件和符号表，如表 4.6 所示。

表 4.6 U-Boot 编译生成的镜像文件

文件名称	说明	文件名称	说明
System.map	U-Boot 映像的符号表	U-Boot.bin	U-Boot 映像原始的二进制格式
U-Boot	U-Boot 映像的 ELF 格式	U-Boot.srec	U-Boot 映像的 S-Record 格式

U-Boot 的 3 种映像格式都可以烧写到 Flash 中，但需要看加载器能否识别这些格式。一般 U-Boot.bin 最为常用，直接按照二进制格式下载，并且按照绝对地址烧写到 Flash 中就可以了。U-Boot 和 U-Boot.srec 格式映像都自带定位信息。

U-Boot 上电启动后，按任意键可以退出自动启动状态，进入命令行。

```
U-Boot 2010.03 (Sep 25 2011 - 16:18:50)

DRAM:  64 MB
Flash:  2 MB
NAND:  64 MiB
In:    serial
Out:   serial
Err:   serial
Net:   CS8900-0
Hit any key to stop autoboot:  1
```

在命令行提示符下，可以输入 U-Boot 的命令并执行。U-Boot 可以支持几十个常用命令，通过这些命令，可以对开发板进行调试，可以引导 Linux 内核，还可以擦写 Flash 完成系统部署等功能。掌握这些命

令，才能顺利地进行嵌入式系统的开发。

输入 help 命令，可以得到当前 U-Boot 的所有命令列表。每一条命令后面是简单的命令说明。

U-Boot 还提供了更加详细的命令帮助，通过 help 命令还可以查看每个命令的参数说明。由于开发过程的需要，有必要先把 U-Boot 命令的语法弄清楚。接下来，根据每一条命令的帮助信息，解释一下这些命令的功能和参数。

1．bootm 命令

bootm 命令可以引导启动存储在内存中的程序映像。这些内存包括 RAM 和可以永久保存的 Flash。

第 1 个参数 addr 是程序映像的地址，这个程序映像必须转换成 U-Boot 的格式。

第 2 个参数对于引导 Linux 内核有用，通常作为 U-Boot 格式的 RAMDISK 映像存储地址；也可以是传递给 Linux 内核的参数（默认情况下传递 bootargs 环境变量给内核）。

```
# help bootm
bootm - boot application image from memory

Usage:
bootm [addr [arg ...]]
    - boot application image stored in memory
        passing arguments 'arg ...'; when booting a Linux kernel,
        'arg' can be the address of an initrd image
Sub-commands to do part of the bootm sequence.  The sub-commands must be
issued in the order below (it's ok to not issue all sub-commands):
        start [addr [arg ...]]
        loados  - load OS image
        cmdline - OS specific command line processing/setup
        bdt     - OS specific bd_t processing
        prep    - OS specific prep before relocation or go
        go      - start OS
```

2．bootp 命令

bootp 命令通过 bootp 请求，要求 DHCP 服务器分配 IP 地址，然后通过 TFTP 下载指定的文件到内存。

第 1 个参数是下载文件存放的内存地址。

第 2 个参数是要下载的文件名称，这个文件应该在开发主机上准备好。

```
# help bootp
bootp - boot image via network using BOOTP/TFTP protocol

Usage:
bootp [loadAddress] [[hostIPaddr:]bootfilename]
```

3．cmp 命令

cmp 命令可以比较两块内存中的内容。.b 以字节为单位；.w 以字为单位；.l 以长字为单位。注意：cmp.b 中间不能保留空格，需要连续输入命令。

第 1 个参数 addr1 是第一块内存的起始地址。

第 2 个参数 addr2 是第二块内存的起始地址。

第 3 个参数 count 是要比较的数目，单位是字节、字或者长字。

```
# help cmp
cmp - memory compare

Usage:
cmp [.b, .w, .l] addr1 addr2 count
```

4．cp 命令

cp 命令可以在内存中复制数据块，包括对 Flash 的读/写操作。

第 1 个参数 source 是要复制的数据块起始地址。

第 2 个参数 target 是数据块要复制到的地址。这个地址如果在 Flash 中，那么会直接调用写 Flash 的函数操作。所以，U-Boot 写 Flash 就使用这个命令，当然需要先把对应 Flash 区域擦除干净。

集团官网：www.hqyj.com 嵌入式学院：www.embedu.org 企业学院：www.farsight.com.cn 北京 | 上海 | 深圳 | 成都 | 南京 | 武汉 | 西安 | 广州
移动互联网学院：www.3g-edu.org 物联网学院：www.topsight.cn 咨询电话：400-706-1880，15010390966

第 3 个参数 count 是要复制的数目，根据 cp.b、cp.w、cp.l 分别以字节、字、长字为单位。

```
# help cp
cp - memory copy

Usage:
cp [.b, .w, .l] source target count
```

5. crc32 命令

crc32 命令可以计算存储数据的校验和。

第 1 个参数 address 是需要校验的数据起始地址。

第 2 个参数 count 是要校验的数据字节数。

第 3 个参数 addr 用来指定保存结果的地址。

```
 # help crc32
crc32 - checksum calculation

Usage:
crc32 address count [addr]
    - compute CRC32 checksum [save at addr]
```

6. echo 命令

echo 命令回显参数。

```
# help echo
echo - echo args to console

Usage:
echo [args..]
    - echo args to console; \c suppresses newline
```

7. erase 命令

erase 命令可以擦除 Flash。参数必须指定 Flash 擦除的范围。

按照起始地址和结束地址，start 必须是擦除块的起始地址；end 必须是擦除末尾块的结束地址，这种方式最常用。举例说明：擦除 0x20000～0x3ffff 区域命令为 erase 20000 3ffff。

按照组和扇区，N 表示 Flash 的组号，SF 表示擦除起始扇区号，SL 表示擦除结束扇区号。另外，还可以擦除整个组，擦除组号为 N 的整个 Flash 组。擦除全部 Flash 只要给出一个 all 的参数即可。

```
# help erase
erase - erase FLASH memory

Usage:
erase start end
    - erase FLASH from addr 'start' to addr 'end'
erase start +len
    - erase FLASH from addr 'start' to the end of sect w/addr 'start'+'len'-1
erase N:SF[-SL]
    - erase sectors SF-SL in FLASH bank # N
erase bank N
    - erase FLASH bank # N
erase all
    - erase all FLASH banks
```

8. nand 命令

nand 命令可以通过不同的参数实现对 Nand Flash 的擦除、读、写操作。

常见的几种命令的含义如下（具体格式见 help nand）。

- nand erase：擦除 Nand Flash。
- nand read：读取 Nand Flash，遇到 Flash 坏块时会出错。
- nand write：写 Nand Flash，nand write 命令遇到 Flash 坏块时会出错。

```
# help nand
```

```
nand - NAND sub-system

Usage:
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
   read/write 'size' bytes starting at offset 'off'
   to/from memory address 'addr', skipping bad blocks.
nand erase [clean] [off size] - erase 'size' bytes from
   offset 'off' (entire device if not specified)
nand bad - show bad blocks
nand dump[.oob] off - dump page
nand scrub - really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset (UNSAFE)
```

9. flinfo 命令

flinfo 命令打印全部 Flash 组的信息，也可以只打印其中某个组。一般嵌入式系统的 Flash 只有一个组。

```
# help flinfo
flinfo - print FLASH memory information

Usage:
flinfo
   - print information for all FLASH memory banks
flinfo N
   - print information for FLASH memory bank # N
```

10. go 命令

go 命令可以执行应用程序。

第 1 个参数是要执行程序的入口地址。

第 2 个可选参数是传递给程序的参数，可以不用。

```
# help go
go - start application at address 'addr'

Usage:
go addr [arg ...]
   - start application at address 'addr'
     passing 'arg' as arguments
```

11. iminfo 命令

iminfo 可以打印程序映像的开头信息，包含映像内容的校验（序列号、头和校验和）。第 1 个参数指定映像的起始地址，可选的参数是指定更多的映像地址。

```
# help iminfo
iminfo - print header information for application image

Usage:
iminfo addr [addr ...]
   - print header information for application image starting at
     address 'addr' in memory; this includes verification of the
     image contents (magic number, header and payload checksums)
```

12. loadb 命令

loadb 命令可以通过串口线下载二进制格式文件。

```
# help loadb
loadb - load binary file over serial line (kermit mode)

Usage:
loadb [ off ] [ baud ]
   - load binary file over serial line with offset 'off' and baudrate 'baud'
```

13．loads 命令

loads 命令可以通过串口线下载 S-Record 格式文件。

```
# help loads
loads - load S-Record file over serial line

Usage:
loads [ off ]
    - load S-Record file over serial line with offset 'off'
```

14．mw 命令

mw 命令可以按照字节、字、长字写内存，.b、.w、.l 的用法与 cp 命令相同。
第 1 个参数 address 是要写的内存地址。
第 2 个参数 value 是要写的值。
第 3 个可选参数 count 是要写单位值的数目。

```
# help mw
mw - memory write (fill)

Usage:
mw [.b, .w, .l] address value [count]
```

15．nfs 命令

nfs 命令可以使用 NFS 网络协议通过网络启动映像。

```
# help nfs
nfs - boot image via network using NFS protocol

Usage:
nfs [loadAddress] [[hostIPaddr:]bootfilename]
```

16．printenv 命令

printenv 命令打印环境变量。可以打印全部环境变量，也可以只打印参数中列出的环境变量。

```
# help printenv
printenv - print environment variables

Usage:
printenv
    - print values of all environment variables
printenv name ...
    - print value of environment variable 'name'
```

17．protect 命令

protect 命令是对 Flash 写保护的操作，可以使能和解除写保护。
第 1 个参数 on 代表使能写保护；off 代表解除写保护。
第 2、第 3 个参数是指定 Flash 写保护操作范围，跟擦除的方式相同。

```
help protect
protect - enable or disable FLASH write protection

Usage:
protect on  start end
    - protect FLASH from addr 'start' to addr 'end'
protect on start +len
    - protect FLASH from addr 'start' to end of sect w/addr 'start'+'len'-1
protect on  N:SF[-SL]
    - protect sectors SF-SL in FLASH bank # N
protect on  bank N
    - protect FLASH bank # N
protect on  all
    - protect all FLASH banks
protect off start end
```

```
    - make FLASH from addr 'start' to addr 'end' writable
protect off start +len
    - make FLASH from addr 'start' to end of sect w/addr 'start'+'len'-1 wrtable
protect off N:SF[-SL]
    - make sectors SF-SL writable in FLASH bank # N
protect off bank N
    - make FLASH bank # N writable
protect off all
    - make all FLASH banks writable
```

18. rarpboot 命令

rarpboot 命令可以使用 TFTP 通过网络启动映像。也就是把指定的文件下载到指定地址，然后执行。

第 1 个参数是映像文件下载到的内存地址。

第 2 个参数是要下载执行的镜像文件。

```
# help  rarpboot
rarpboot - boot image via network using RARP/TFTP protocol

Usage:
rarpboot [loadAddress] [[hostIPaddr:]bootfilename]
```

19. run 命令

run 命令可以执行环境变量中的命令，后面可以跟几个环境变量名。

```
# help run
run - run commands in an environment variable

Usage:
run var [...]
    - run the commands in the environment variable(s) 'var'
```

20. setenv 命令

setenv 命令可以设置环境变量。

第 1 个参数是环境变量的名称。

第 2 个参数是要设置的值，如果没有第 2 个参数，表示删除这个环境变量。

```
# help setenv
setenv - set environment variables

Usage:
setenv name value ...
    - set environment variable 'name' to 'value ...'
setenv name
    - delete environment variable 'name'
```

21. sleep 命令

sleep 命令可以使用 TFTP 通过网络下载文件。按照二进制文件格式下载。另外使用这个命令，必须配置好相关的环境变量，如 serverip 和 ipaddr。

第 1 个参数 loadAddress 是下载到的内存地址。

第 2 个参数是要下载的文件名称，必须放在 TFTP 服务器相应的目录下。

```
help sleep
sleep - delay execution for some time

Usage:
sleep N
    - delay execution for N seconds (N is _decimal_ !!!)
```

sleep 命令可以延迟 *N* 秒执行，*N* 为十进制数。

22. nm 命令

nm 命令可以修改内存，可以按照字节、字、长字操作。

参数 address 是要读出并且修改的内存地址。

```
# help nm
nm - memory modify (constant address)

Usage:
nm [.b, .w, .l] address
```

23．TFTPboot 命令

TFTPboot 命令可以通过使用 TFTP 通过网络下载二进制格式的文件。

```
TFTPboot - boot image via network using TFTP protocol

Usage:
TFTPboot [loadAddress] [[hostIPaddr:]bootfilename]
```

24．saveenv 命令

saveenv 命令可以保存环境变量到存储设备。

```
# help saveenv
saveenv - save environment variables to persistent storage

Usage:
saveenv
```

这些 U-Boot 命令为嵌入式系统提供了丰富的开发和调试功能。在 Linux 内核启动和调试过程中，都可以用到 U-Boot 的命令。但是一般情况下，不需要使用全部命令。例如，已经支持以太网接口，可以通过 TFTPboot 命令来下载文件，那么就没必要使用串口下载的 loadb。反过来，如果开发板需要特殊的调试功能，也可以添加新的命令。

4.2.3 U-Boot 移植

U-Boot 能够支持多种体系结构的处理器，支持的开发板也越来越多。因为 Bootloader 是完全依赖硬件平台的，所以在新电路板上需要移植 U-Boot 程序。

开始移植 U-Boot 之前，要先熟悉硬件电路板和处理器。确认 U-Boot 是否已经支持新开发板的处理器和 I/O 设备。假如 U-Boot 已经支持一块非常相似的电路板，那么移植的过程将非常简单。移植 U-Boot 工作就是添加开发板硬件相关的文件、配置选项，然后配置编译。开始移植之前，需要先分析一下 U-Boot 已经支持的开发板，比较硬件配置最接近的开发板。选择的原则是，首先处理器相同，其次处理器体系结构相同，然后是以太网接口等外围接口相同。还要验证一下这个参考开发板的 U-Boot，至少能够配置编译通过。

以 S3C2410 处理器的 FS2410 开发板为例，U-Boot 的高版本已经支持 SMDK2410 开发板。可以基于 SMDK2410 移植，移植 U-Boot 的基本步骤如下。

（1）在顶层 Makefile 中为开发板添加新的配置选项，使用已有的配置项目为例。

```
smdk2410_config   :     unconfig
        @./mkconfig $(@:_config=) arm arm920t smdk2410 NULL s3c24x0
```

参考上面两行，添加下面两行：

```
fs2410_config   :     unconfig
    @./mkconfig $(@:_config=) arm arm920t EduKit2410 NULL s3c24x0
```

（2）创建一个新目录存放开发板相关的代码，并且添加新文件。

① board/fs2410/config.mk。

② board/ fs2410/flash.c。

③ board/ fs2410/EduKit2410.c。

④ board/ fs2410/Makefile。

⑤ board/ fs2410/memsetup.S。

⑥ board/ fs2410/U-Boot.lds。

华清远见

（3）为开发板添加新的配置文件。可以先复制参考开发板的配置文件，再修改。例如：

```
$cp include/configs/smdk2410.h  include/configs/fs2410.h
```

如果是为一颗新的 CPU 移植，还要创建一个新的目录存放 CPU 相关的代码。

（4）配置开发板。

```
$ make fs2410_config
```

（5）编译 U-Boot。执行 make 命令，编译成功可以得到 U-Boot 映像。有些错误是与配置选项有关系的，通常打开某些功能选项会带来一些错误，一开始可以尽量与参考板配置相同。

（6）添加驱动或者功能选项。在能够编译通过的基础上，还要实现 U-Boot 的以太网接口、Flash 擦写等功能。对于 FS2410 开发板的以太网驱动和 smdk2410 完全相同，所以可以直接使用。CS8900 驱动程序代码包括：

```
drivers/cs8900.c
drivers/cs8900.h
```

对于 Flash 的选择就麻烦多了，Flash 芯片价格或者采购方面的因素都有影响。多数开发板大小、型号不都相同。所以还需要移植 Flash 的驱动。每种开发板目录下一般都有 flash.c 文件，需要根据具体的 Flash 类型修改。例如：

```
board/fs2410/flash.c
```

（7）调试 U-Boot 源代码，直到 U-Boot 在开发板上能够正常启动。调试的过程可能是很艰难的，需要借助工具，并且有些问题可能会困扰很长时间。

4.3　Android 内核与移植

华清远见

Linux 内核是 Android 操作系统的核心，是用 C 语言编写的，符合 POSIX 标准。Linux 最早是由芬兰黑客 Linus Torvalds 为尝试在英特尔 X86 架构上提供自由免费的类 UNIX 操作系统而开发的。该计划开始于 1991 年，这里有一份 Linus Torvalds 当时在 Usenet 新闻组 comp.os.minix 所登载的帖子，这份著名的帖子标志着 Linux 计划的正式开始。在计划的早期有一些 Minix 黑客提供了协助，而今天全球无数程序员正在为该计划无偿提供帮助。

Android 基于 Linux 操作系统，由硬件、系统内核、系统服务和应用程序四大部分嘣组成。其中，内核（Kernel）是最核心的部分，其主要作用在于与计算机硬件进行交互，实现对硬件的编程控制和接口操作，调度访问硬件资源，同时向应用程序提供一个高级的执行环境和对硬件的虚拟接口。主要功能包括：中断服务程序、进程调度程序、进程地址空间的内存管理、进程间通信。内核与普通应用程序不同。其拥有所有硬件设备的访问权限，以及启动时即划分的受保护的内存空间.

Android 内核和标准的 Linux 内核一样，主要实现内存管理、进程调度、进程间通信等功能。它是在标准 Linux 内核的基础上修改而成的，为了适应嵌入式硬件环境和移动应用程序的开发，Android 对标准 Linux 内核进行了一定的修改。

经过与标准 Linux 内核源代码进行详细对比可以发现，Android 内核与标准 Linux 内核在文件系统、进程间通信机制、内存管理等方面存在不同。

（1）文件系统不同于桌面系统与服务器。移动设备大多采用的不是硬盘而是采用 Flash 作为存储介质，因此，Android 内核中增加了标准 Linux 内核中没有采纳的 YAFFS2 文件系统 YAFFS2（Yet Another Flash File Sy-steIn. 2nd edition）是专用于 Flash 的文件系统，对 NAND. Flash 芯片有着良好的支持。YAFFS2 是日志结构的文件系统，提供了损耗平衡和掉电保护，可以有效地避免意外断电对文件系统一致性和完整性的影响。YAFFS2 按层次结构设计，分为文件管理接口、内部实现层和 NAND，简化了其本身与系统的接 EI 设计，能更方便地集成到系统当中。

（2）进程间通信机制。Android 增加了一种进程间的通信机制 IPC Bin—der，在内核源代码中，驱动程序文件为 COredroid/inelude/linux/binder.h 和 coredroid/drivers/android/binder。c.Binder 通过守护进程

Service Manager 管理系统中的服务，负责进程间的数据交换。各进程通过 Binder 访问同一块共享内存，以达到数据通信的机制，从应用层的角度看，进程通过访问数据守护进程获取用于数据交换的程序框架接口，调用并通过接口共享数据。而其他进程要访问数据，也只需与程序框架接口进行交互，方便了程序员开发需要交互数据的应用程序。

（3）内存管理。在内存管理模块，Android 内核采用了一种不用于标准 Linux 内核的低内存管理策略，在标准 Linux 内核当中，使用一种叫做 OOM（Out Of Memory）的低内存管理策略；当内存不足时，系统检查所有的进程，并对进程进行限制评分，获得最高分的进程将被关闭（内核进程除外）。Android 系统采用的则是一种叫作 LMK（Low Memory Killer）的机制，这种机制将进程按照重要性进行分级、分组，内存不足时，将处于最低级别组的进程关闭。例如，在移动设备当中，UI 界面处于最高级别，所以该进程永远不会被中止，这样，在终 端用户看来，系统是稳定运行的。在 Andorid 内核源码中，LMK 的位置是 coredroid / drivers / mise / lowme—morykiller.c。与此同时，Android 新增加了一种内存共享的处理方式 Ashmem（Anonymous Shared Memory，匿名共享内存）。通过 Ashmem，进程间可以匿名自由共享具名的内存块，这种共享方式在标准 Linux 中不被支持。

（4）电源管理。由于 Android 主要用于移动设备，电源管理就显得尤为重要。因此，在 Android 内核当中，增加了一种新的电源管理策略。目前，Android 采用的是一种较为简单的电源管理策略，通过开关屏幕、开关屏幕背光、开关键盘背光、开关按钮背光和调整屏幕亮度来实现电源管理，并设有休眠和待机功能。有 3 种途径判断调整电源管理策略：RPC 调用、电池状态改变和电源设置它通过广播 Intent 或直接调用 API 的方式来与其他模块进行联系，电源管理策略同时还有自动关机机制，当电力低于最低可接受程度时，系统将自动关机。Android 的电源管理模块还会根据用户行为，自动调整屏幕亮度。

（5）驱动及其他。相对于标准内核，Android 内核还添加了字符输出设备、图像显示设备、键盘输入设备、RTC 设备、USBDevice 设备等相关设备驱动。增加了日志（Logger）系统，使应用程序可以访问日志消息。

华清远见

4.3.1 Android 移植简介

所谓移植，就是把程序代码从一种运行环境转移到另一种运行环境。对于内核移植来说，主要是从一种硬件平台转移到另外一种硬件平台上运行。

在一个目标板上 Android 内核的移植包括 3 个层次，分别为体系结构级别的移植、SoC 级别的移植和板子级别的移植。

体系结构级别的移植是指在不同体系结构平台上 Android 内核的移植，例如，在 ARM、MIPS、PPC 等不同体系结构上分别都要对每个体系结构进行特定的移植工作。一个新的体系结构出现就需要进行这个层次上的移植。

SoC 级别的移植是指在具体的 SoC 处理器平台上 Android 内核的移植， 板子级别的移植是指在具体的目标板子上 Android 内核的移植。在这里讨论板子级别的移植，主要是添加开发板初始化和驱动程序的代码，不同的开发板可以使用不同的 SDRAM、Flash、以太网接口芯片等。这就需要根据硬件修改或者开发驱动程序。

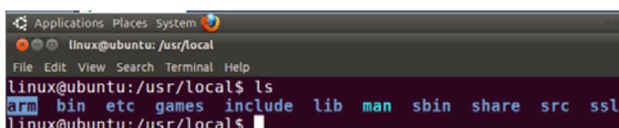
4.3.2 FS_S5PC100 开发平台移植环境搭建

在 Linux 中建立交叉开发环境，步骤如下：

1．安装交叉编译工具链：

（1）解压“Linux-Android\toolchain”目录下的“arm-none-eabi-4.2.2.tgz”到根“/usr/local”目录下，在“/usr/local”目录下会生成“arm”目录。

执行命令：#tar zxvf arm-none-eabi-4.2.2.tgz

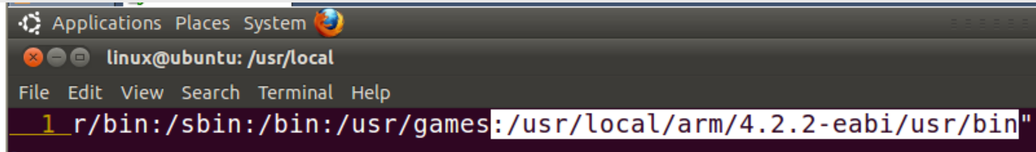


（2）修改环境变量“PATH”：

```
~$ sudo vim /etc/environment
```

将路径添加到 PATH 变量的最后面，省略号代表原来 PATH 的值：

```
PATH=……:/usr/local/arm/4.2.2-eabi/usr/bin”
```

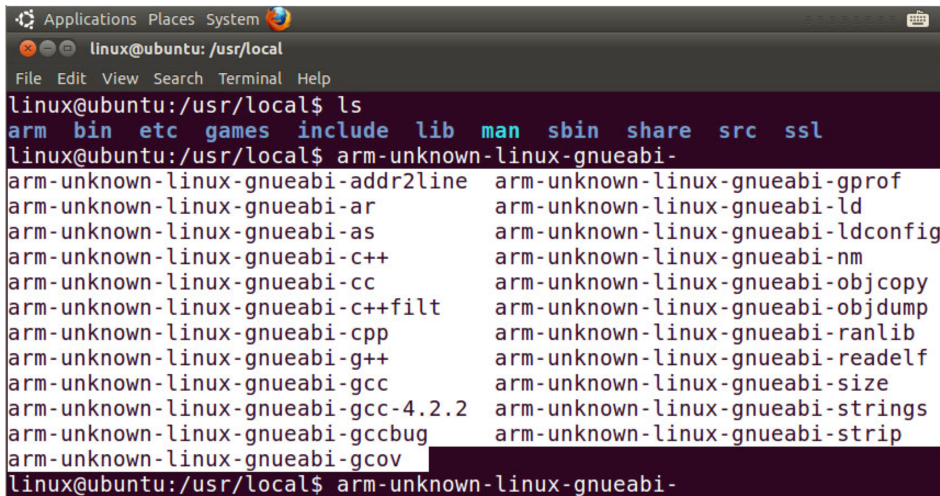


保存退出后执行。

```
source /etc/ environment
```

这样修改的环境变量会立即生效。

于是我们就得到交叉编译工具：



2. 安装 JDK

（1）安装“Linux-Android\toolchain”目录下的“jdk.bin”，jdk.bin 是 Jdk1.5。Jdk1.5 是编译 Android 2.1 必需的工具，而且只能是 1.5，其他版本都不行。

把“jdk.bin”复制到“/usr”目录下，然后执行，按照提示安装即可。

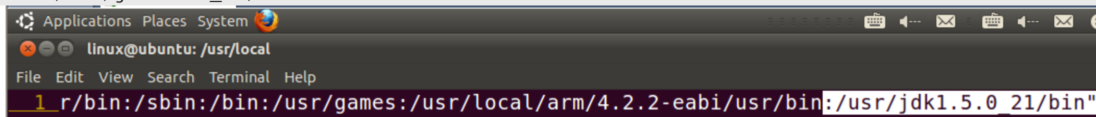
```
#sudo ./jdk.bin
```

（2）安装结束后还要配置一下环境变量，代码如下：

```
~$ sudo vim /etc/environment
```

将路径添加到 PATH 变量的最后面：

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/arm/4.2.2-eabi
/usr/bin:/usr/jdk1.5.0_21/bin
```



保存退出后执行：

```
source /etc/environment
```

4.4 U-Boot、内核、文件系统编译

4.4.1 U-Boot 的编译

复制“Linux-Android\源码”目录下的“uboot-s5pc100.tar.bz2”到目标目录下，然后执行如下命令：

```
# tar jxvf uboot-s5pc100.tar.bz2
# make smdkc100_config
# make
```

在源码根目录（uboot-samsung）下会生成“u-boot.bin”。

4.4.2 内核镜像的编译

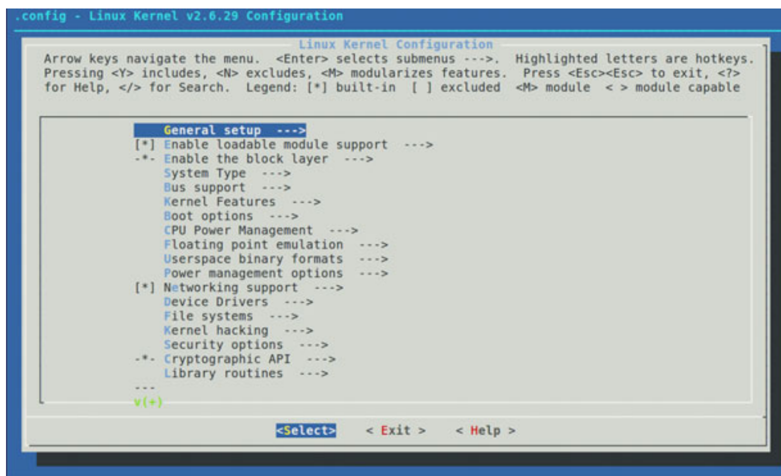
（1）复制“Linux-Android\源码”目录下的“android-2.6.29-samsung.tar.bz2”到目标目录下，然后解压。

```
tar jxvf android-2.6.29-samsung.tar.bz2
```

（2）执行如下命令：

```
make menuconfig
```

进入配置界面。



配置内核编译选项时需要注意按照实际情况选择 Framebuffer 显示设备的类型，选项位于：

```
  -> Device Drivers
    -> Graphics support
       -> Support for frame buffer devices (FB [=y])
         -> S3C Framebuffer support (FB_S3C [=y])
           -> Select LCD Type (<choice> [=y])
```

如果开发板接 480×272 的 LCD 屏，应该选择

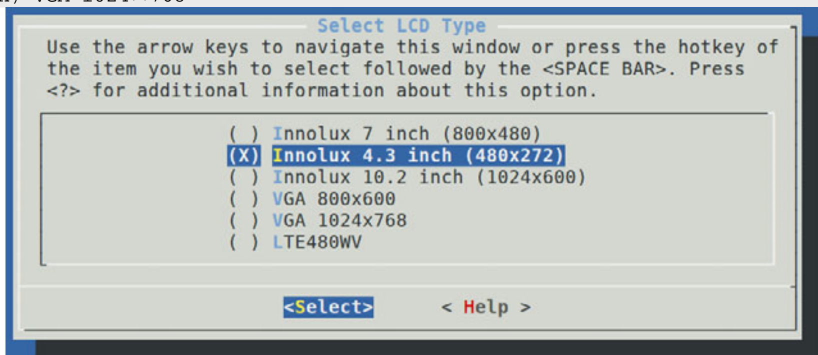
```
(X)Innolux 4.3 inch (480x272)
```

如果开发板接 800×600 的 VGA 显示器，应该选择

```
(X) VGA 800x600
```

如果开发板接 1024×768 的 VGA 显示器，应该选择

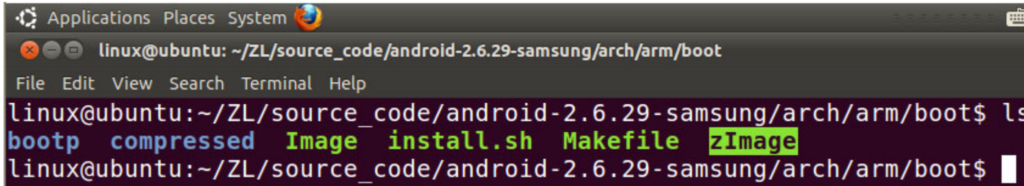
```
(X) VGA 1024×768
```



（3）执行如下命令：

```
make zImage
```

编译内核，生成的 zImage 位于 arch/arm/boot/下。



4.4.3 Android 文件系统的编译

（1）复制“Linux-Android\源码”目录下的“eclair_2.1_farsight.tar.gz”到目标目录下，然后解压。

```
tar zxvf eclair_2.1_farsight.tar.gz
```

（2）Android_2.1 编译。

①初始化 Android 构建子系统（导出几个命令到环境变量）：

```
~/eclair_2.1_farsight/$ . ./build/envsetup.sh
```

注意，这里两个“.”之间有一个空格，第一个“.”指定用当前 Shell 解析这个脚本，否则不能执行。

②配置板级信息：

```
~/eclair_2.1_farsight/$ tapas
```

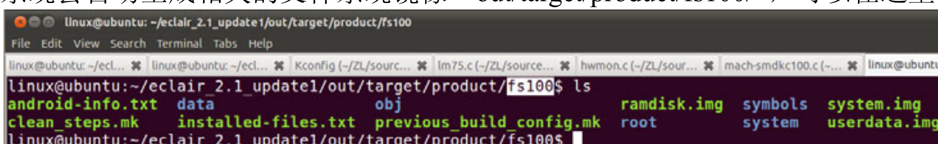
tapas 就是上一步执行结束之后导出到环境变量中的命令，专门用来配置板级信息的。具体配置选项如下：

```
Build for the simulator or the device?
    1. Device
    2. Simulator
Which would you like? [1] 1
Build type choices are:
    1. release
    2. debug
Which would you like? [1] 1
Which product would you like? [fs100] fs100
Variant choices are:
    1. user
    2. userdebug
    3. eng
Which would you like? [eng] eng
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=2.1-update1
TARGET_PRODUCT=fs_s5pc100
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=false
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID= ERE27
```

③开始编译（如果不能找到 mm，则执行“source build/envsetup.sh”）：

```
~/eclair_2.1_farsight/$ mm
```

系统会自动生成相关的文件系统镜像“out/target/product/fs100/”，可以在这里找到各个部分。



（3）制作 yaffs2 文件系统镜像。执行如下命令：

```
~/eclair_2.1_farsight/$ ./make_fs100_yaffs2_image.sh
```

会在 Android 源码根目录下生成目录“fs100_root”，这个目录就是编译生成的 Android 文件系统，调试时可以直接把这个目录作为 NFS-Server 的目录。还会生成一个“fs100_root.img”文件，这个文件就是 Android 的 yaffs2 格式的镜像，可以使用 dnw 工具将这个镜像烧写到 Nand Flash 上。

4.5 小结

本章主要讲解搭建 Android 系统移植开发环境的整个流程。首先讲解如何搭建嵌入式交叉开发环境，包括交叉编译环境、各种服务程序和应用程序的安装、配置和使用。

为了驱动目标板，必须先做好 Bootloader、操作系统内核及文件系统。介绍 Bootloader 的概念及 U-Boot 的编译和移植的方法；接下来讲解了 Android 系统下 Linux 内核的相关知识，然后讲解了内核编译和移植的方法。

基于嵌入式系统的特点，它的开发与 PC 上开发相比有很多复杂的前提工作，这正是嵌入式开发的难点之一，希望读者熟悉开发环境搭建的每个环节。

4.6 思考题

1. 练习在主机上搭建交叉编译环境，并用交叉编译器编译 hello.c 程序。
2. 练习在主机上安装和配置 minicom、TFTP、NFS 等应用程序和服务器。
3. Bootloader 的种类有几种？它和 Android 什么关系？

华清远见

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

广州地址：广州市天河区中山大道 268 号天河广场 3 层，电话：020-28916067