



## Chapter02

# Android底层开发平台

## 上章回顾

---

- ▶ Android与其他手机平台的联系与区别与优势
- ▶ Android 系统架构（四层）
- ▶ Android 是框架而非操作系统
- ▶ Android 可视化编辑环境搭建（eclipse+adt+avd+sdk）
- ▶ Android 几大组件的介绍（activity+service+intent+contentprovider）
- ▶ Android 附录（adb+ddms+ Emulator）以及与真机的区别



# questions

---

- ▶ 1、android到底和linux是个什么关系
- ▶ 2、网上开源下载的android2.2和手机里面的2.2系统是一样的吗？
- ▶ 3、刷机包里面到底是什么内容
- ▶ 4、底层硬件是如何工作的？



# 本章内容

---

- ▶ Android文件移植
- ▶ Android下文件目录
- ▶ Android的ADB工具使用





## 一、Android底层硬件

Linux内核编译、ARM编程、刷机

## APPLICATIONS

Java应用程序: java

Home

Contacts

Phone

Browser

...

## APPLICATION FRAMEWORK

Java框架: java

Activity Manager

Window  
Manager

Content  
Providers

View  
System

Package Manager

Telephony  
Manager

Resource  
Manager

Location  
Manager

Notification  
Manager

## LIBRARIES

Surface Manager

Media  
Framework

SQLite

OpenGL | ES

FreeType

WebKit

SGL

SSL

libc

## ANDROID RUNTIME

Core Libraries

Dalvik Virtual  
Machine

本地框架和Java运行环境: C/C++

## LINUX KERNEL

Linux操作系统以及驱动: C

Display  
Driver

Camera Driver

Flash Memory  
Driver

Binder (IPC)  
Driver

Keypad Driver

WiFi Driver

Audio  
Drivers

Power  
Management

# 一、一个手机是如何开机的？

---

- ▶ 1硬件（没有操作系统）--MDK
- ▶ 2操作系统的加载（linux）--boot、kernel(driver)、filesystem
- ▶ 3android图形化系统的启动
- ▶ 4刷机

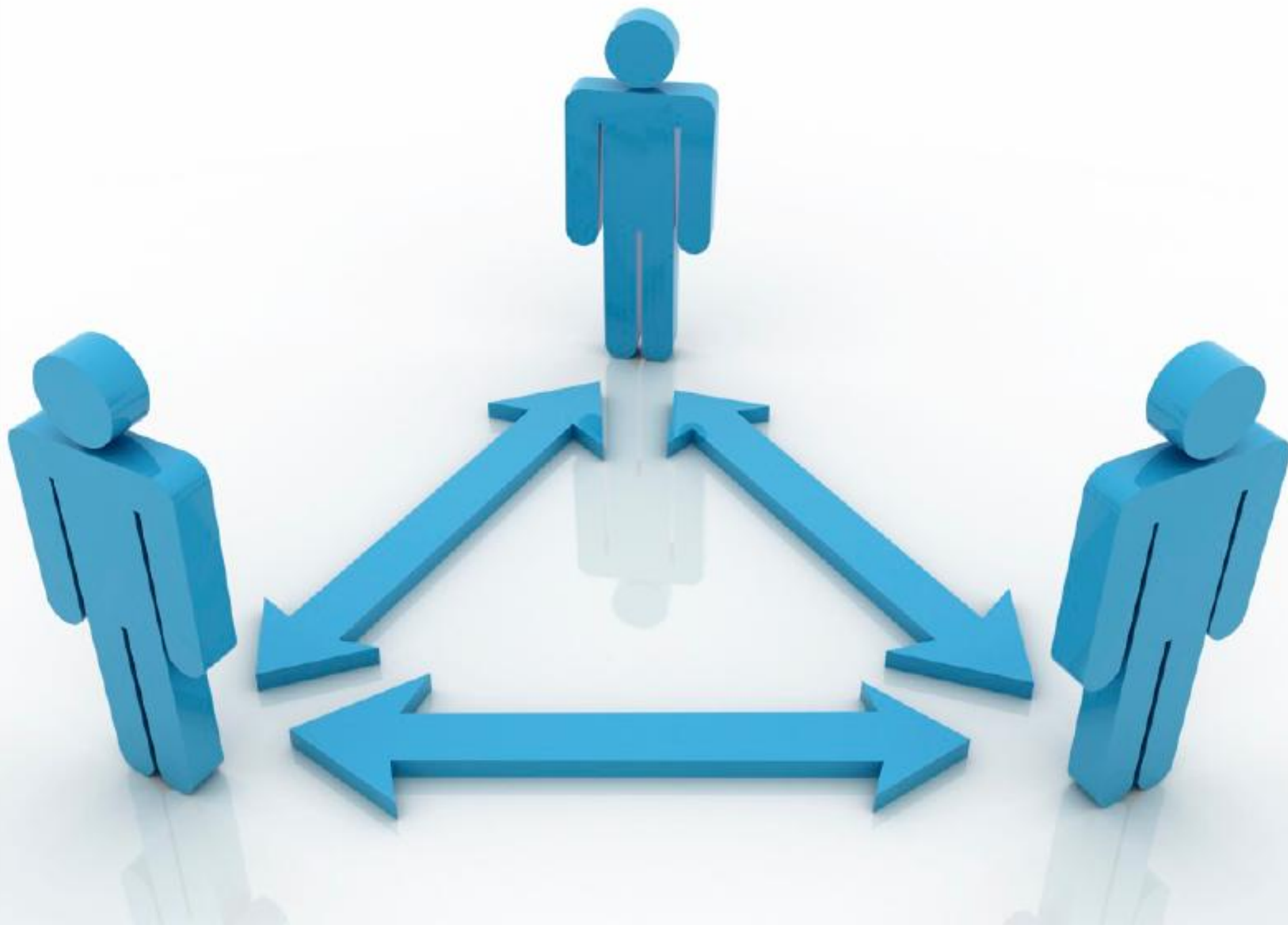




What does open mean?









Industry

- Software stack open-sourced under Apache 2.0 license
- Source available after first handsets ship
- Anyone will be able to build a system image



Industry

The diagram features three blue 3D human figures. One figure is at the top center, another at the bottom left, and a third at the bottom right. A central dark gray rectangular box contains three bullet points. Three large, dark gray, 3D arrows originate from the corners of this box and point towards the three human figures, suggesting a flow of information or control from the central concepts to the entities.

- Users have control of their experience
- They control what gets installed
- They choose the defaults

Users

Developers

Industry

- Don't need permission to ship an application
- No hidden or privileged framework APIs
- Can **integrate**, **extend**, and **replace** existing components

Users

# 1、1 硬件（没有操作系统裸机）--MDK

---

- ▶ 1.1.1嵌入式系统开发
- ▶ 1.1.2嵌入式ARM处理器
- ▶ 1.1.3RealView MDK开发：μVision 3集成开发环境、ULINK 2仿真器
- ▶ 1.1.4基于硬件的开发调试：汇编与C语言



## 1.1.1 嵌入式系统开发与应用

---

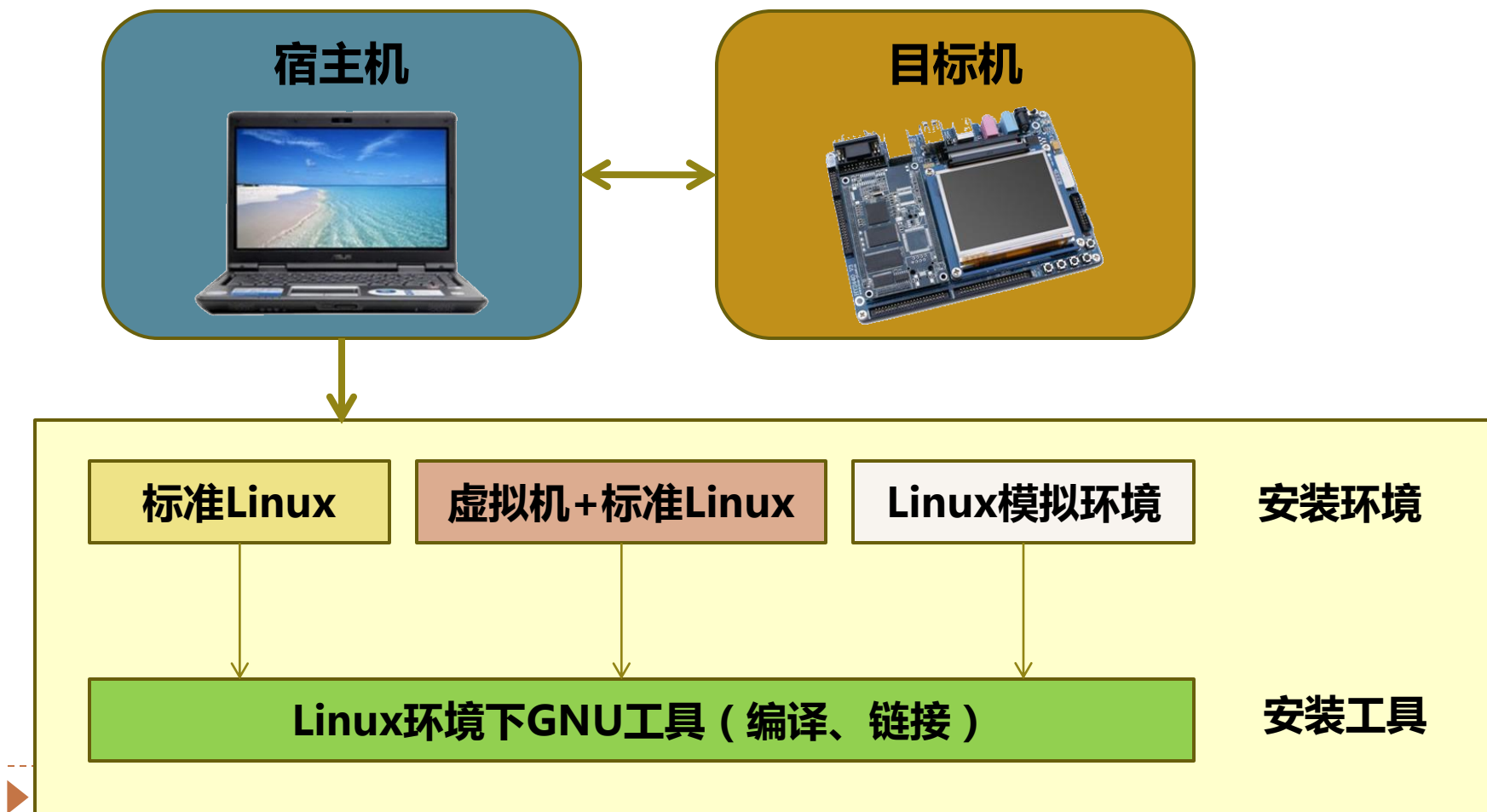
- ▶ 学习嵌入式系统的开发应用技术，应该是基于某种ARM核系统芯片应用平台基础上进行
- ▶ 作为嵌入式系统应用的ARM处理器，其应用软件的开发属于跨平台开发，因此需要一个交叉开发环境。交叉开发是指在一台通用计算机上进行软件的编辑编译，然后下载到嵌入式设备中进行运行调试的开发方式。





# 嵌入式Linux开发模型

- ▶ 嵌入式Linux开发在宿主机上进行：





## 1.1.2 ARM架构

---

- ▶ ARM架构，过去称作进阶精简指令集机器（Advanced RISC Machine，更早称作：Acorn RISC Machine），是一个32位元精简指令集（RISC）处理器架构，其广泛地使用在许多嵌入式系统设计。由于节能的特点，ARM处理器非常适用于行动通讯领域，符合其主要设计目标为低功耗的特性



# 嵌入式ARM处理器

- ▶ 1991年ARM公司成立于英国剑桥，主要出售芯片设计技术的授权。
- ▶ ARM公司是专门从事公司，作为知识产权供应商，靠转让设计许可，由世界各大半导体生产商ARM微处理器核，根当的外围电路，从而进入市场：三星、德



## 主流arm cpu

- ▶ Milestone 使用与iPhone3GS相同的600MHz OMAP3处理器
- ▶ HTC Hero采用的是高通MSM7200A处理器，主频达到了528MHz，而升级版的高通的处理器，不过型号变为600MHz
- ▶ A1++ 新版红钻王二代使用CPU，Marvell公司成为于中国上海设有研发中心，其处理器业务卖给Marvell。发布的PXA 3XX系列，300312已经成为了目前Windows最强大的CPU



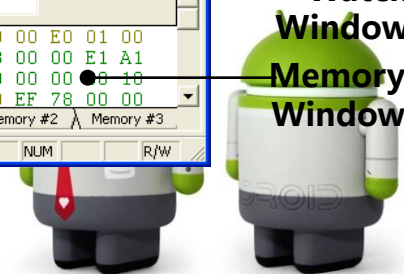
RD2

## 1.1.3MDK开发

---

- ▶ MDK ( Microcontroller Development Kit ) 是Keil公司 ( An ARM Company ) 开发的ARM开发工具，是用来开发基于ARM核的系列微控制器的嵌入式应用程序的开发工具。
- ▶ RealView MDK开发套件
  1.  $\mu$ Vision 3集成开发环境
  2. ULINK 2仿真器



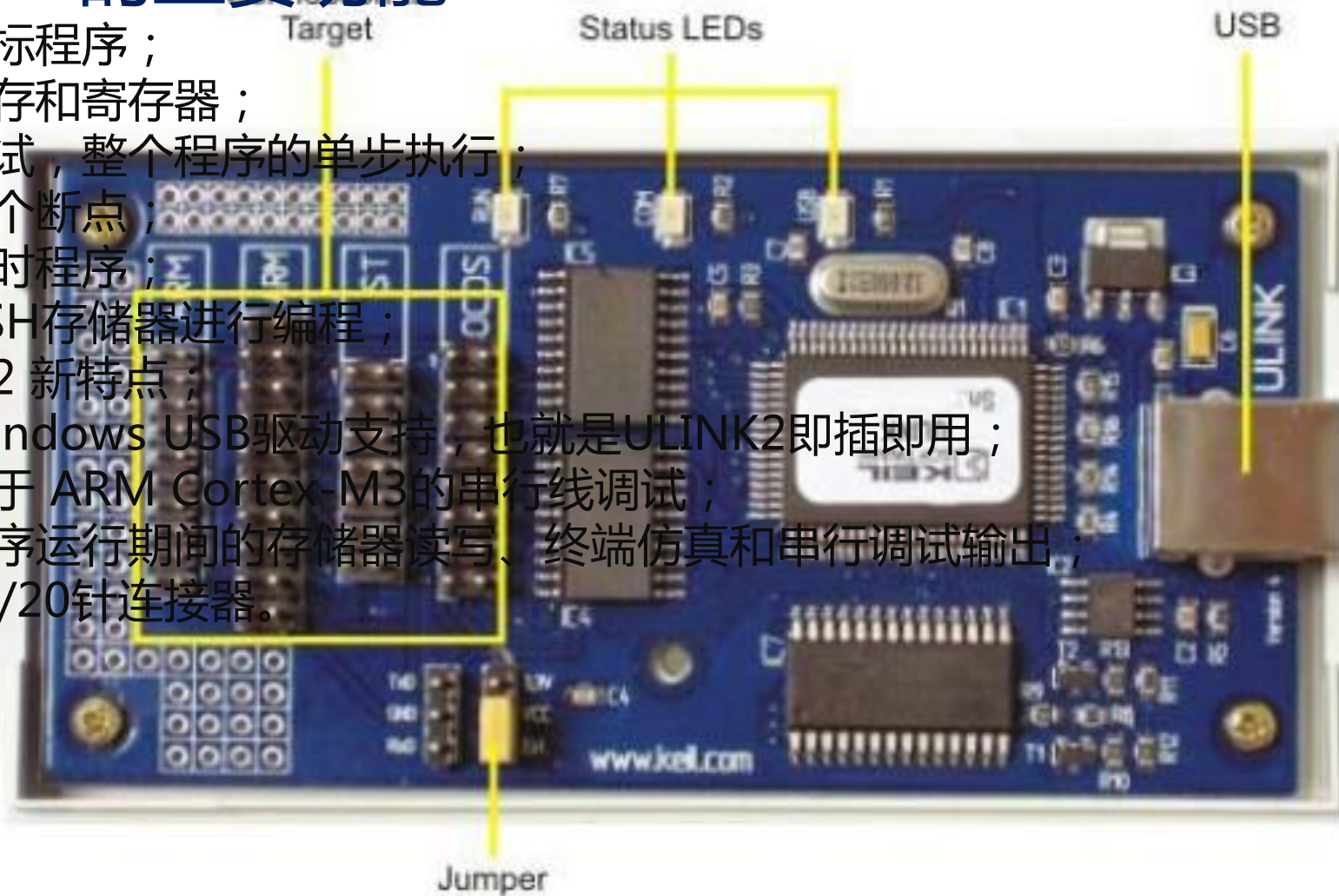




# ULINK是Keil公司提供的USB-JTAG接口仿真器

## ULINK 2的主要功能：

- 下载目标程序；
- 检查内存和寄存器；
- 片上调试，整个程序的单步执行；
- 插入多个断点；
- 运行实时程序；
- 对FLASH存储器进行编程；
- ULINK2 新特点；
- 标准Windows USB驱动支持，也就是ULINK2即插即用；
- 支持基于 ARM Cortex-M3的串行线调试；
- 支持程序运行期间的存储器读写、终端仿真和串行调试输出；
- 支持10/20针连接器。

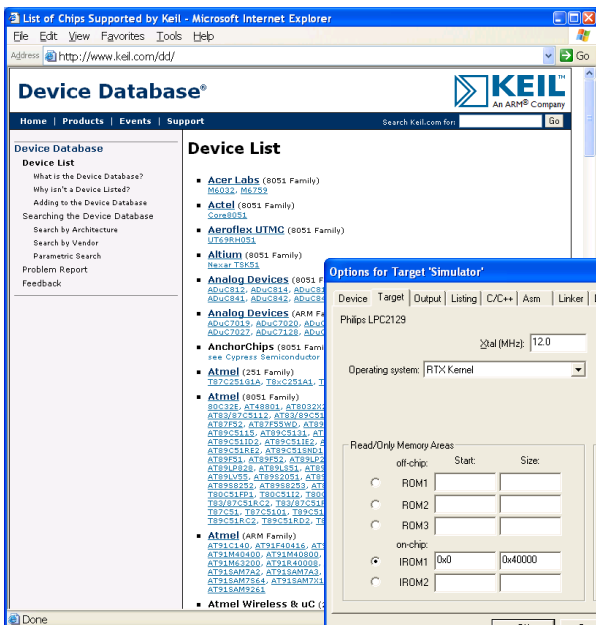


# MDK开发的四个步骤

## Step 1:选择设备和指定硬件对象



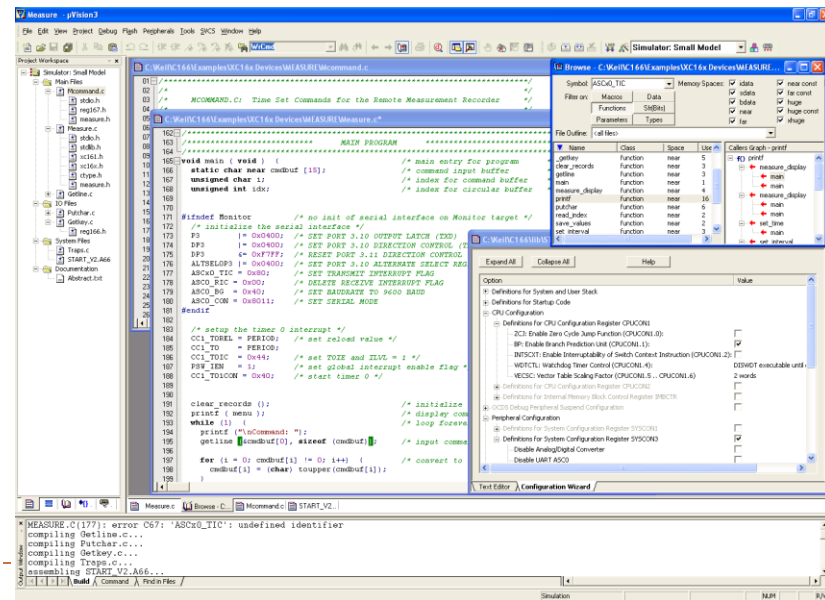
网上庞大设备数据库 &  $\mu$ Vision  
简单化地芯片选择与设置



## Step 2:配置设备和 创建应用程序代码



$\mu$ Vision 包含了工程管理器、编  
辑器和调试器



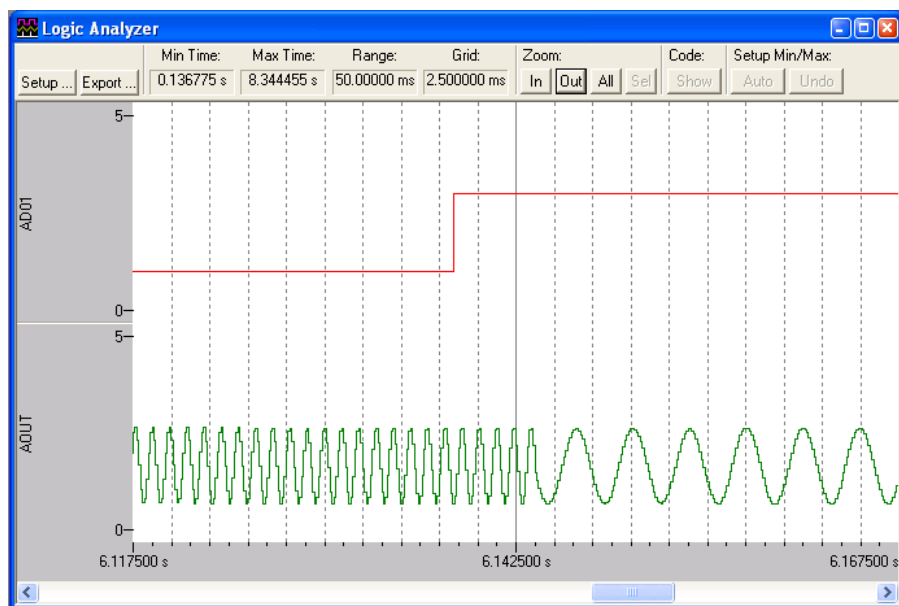


# MDK开发的四个步骤

Step 3: 用 $\mu$ Vision设备仿真器  
分析代码



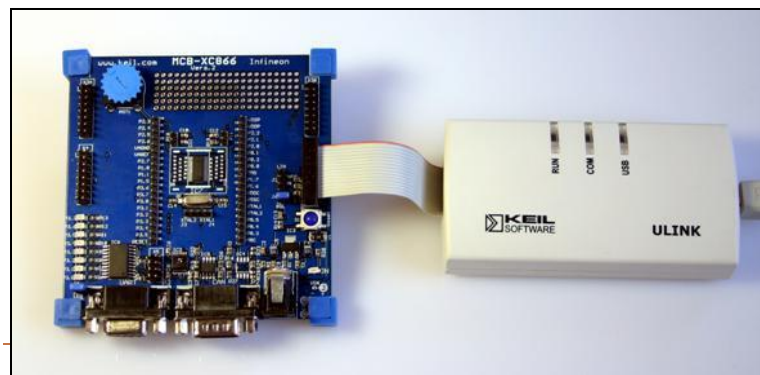
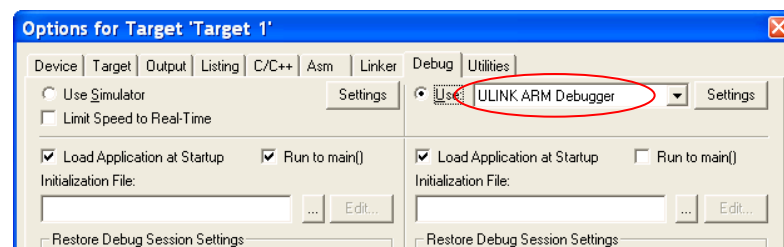
通过  $\mu$ Vision Debugger  
和 Device Simulator 调试



Step 4: Flash下载  
最后在目标硬件上测试



通过 ULINK 进行Flash Download  
和 Target Debugging



## 1.1.4 基于硬件的开发调试：汇编与C语言

---

- ▶ LED控制实验：利用S3C2410X芯片地址总线扩展的I/O来驱动LED显示
- ▶ 通过实验掌握触摸屏（TSP）的设计与控制方法：点击触摸屏任意位置，将触摸屏坐标转换为液晶对应坐标后显示坐标位置。掌握S3C2410X处理器的A/D转换功能
- ▶ IIS音频接口、USB接口相关控制寄存器的使用、矩阵LED的应用原理
- ▶ 使用汇编的原因：空间、时序



## 1.2 操作系统的加载（linux）

---

- ▶ 1.2.1搭建嵌入式Linux开发环境
- ▶ 1.2.2交叉编译
- ▶ 1.2.3Linux内核开发
- ▶ 1.2.4嵌入式Linux设备驱动开发



## 【知识点】内核是什么

---

- ▶ Linux不是一个操作系统，严格来讲，Linux只是一个操作系统中的内核。内核是什么？内核建立了计算机软件与硬件之间通讯的平台，内核提供系统服务，比如文件管理、虚拟内存、设备I/O等。可以把linux装在U盘或移动硬盘中（这一点是windows做不到的）。
- ▶ 既然Linux只是一个内核。然而，一个完整的操作系统不仅仅是内核而已。所以，许多个人、组织和企业，开发了基于GNU/Linux的 Linux发行版。



## 1.2.1 搭建嵌入式Linux开发环境

---

- ▶ 1) Linux内核的重要特点：
  - ▶ 可移植性（Portability），支持硬件平台广泛，在大多数体系结构上都可以运行；
  - ▶ 可量测性（Scalability），即可以运行在超级计算机上，也可以运行在很小的设备上（4MB RAM就能满足）；
  - ▶ 标准化和互用性（Interoperability），遵守标准化和互用性规范；
  - ▶ 完善的网络支持；
  - ▶ 安全性，开放源码使缺陷暴露无疑，它的代码也接受了许多专家的审查；
  - ▶ 稳定性（Stability）和可靠性（Reliability）；
  - ▶ 模块化（Modularity），运行时可以根据系统的需要加载程序；
  - ▶ 编程容易，可以学习现有的代码，还可以从网络上找到很多有用的资源。

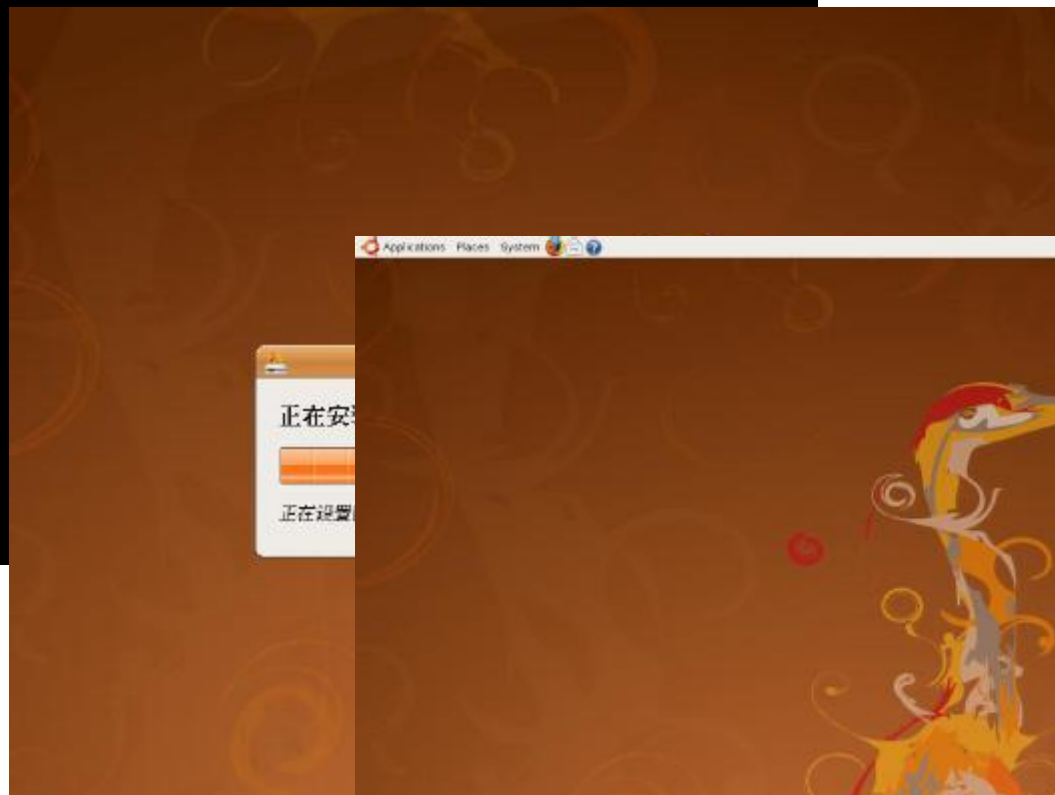


请选择要启动的操作系统:

Microsoft Windows XP Professional

Ubuntu

使用 ↑ 键和  
按 Enter 键做



# 认识Makefile

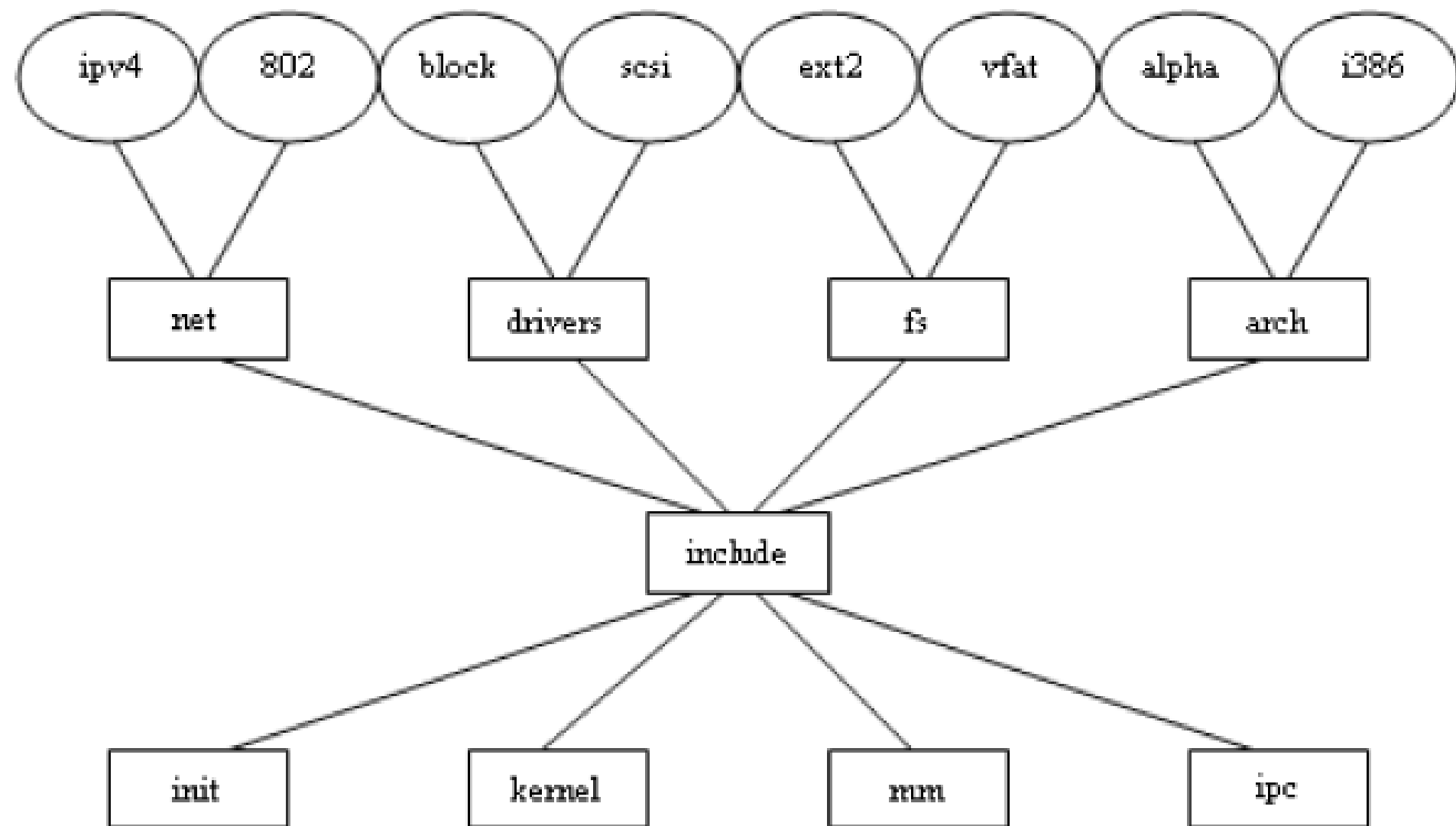
---

- ▶ make是Linux下的一款程序自动维护工具，配合Makefile的使用，就能够根据程序中模块的修改情况，自动判断应该对哪些模块重新编译，从而保证软件是由最新的模块构成





# 内核源代码目录结构



- 
- ▶ **Linux存储管理**：Linux操作系统采用了请求式分页存储管理方法。系统为每个进程提供了4GB的虚拟内存空间。各个进程的虚拟内存彼此独立。
  - ▶ **Linux进程管理**：Linux是一个多用户多任务的操作系统。多用户是指多个用户可以在同一时间使用计算机系统；多任务是指Linux可以同时执行几个任务，它可以在还未执行完一个任务时又执行另一项任务。



# Linux内核启动和初始化进程

---

- ▶ 引导程序Bootloader：系统上电后通过BIOS或者引导程序Bootloader加载系统内核
- ▶ 核心数据结构初始化---内核引导第一部分：  
start\_kernel ( ) 中调用了一系列初始化函数，以完成kernel本身的设置。
- ▶ 外设初始化---内核引导第二部分：init ( ) 函数作为核心线程
- ▶ rc启动脚本：激活交换分区，检查磁盘，加载硬件模块
- ▶ login：rc执行完毕后，返回init，这时基本系统环境已经配置好。各种守护进程也已经启动。接下来init会打开6个终端，以使用户登录系统。



# 嵌入式Linux

---

- ▶ 虽然大多数 Linux 系统运行在 PC 平台上，但 Linux 也可以作为嵌入式系统的操作系统
- ▶ 在嵌入式系统上运行 Linux 的一个缺点是 Linux 体系提供实时性能需要添加实时软件模块。而这些模块运行的内核空间正是操作系统实现调度策略、硬件中断异常和执行程序的部分。由于这些实时软件模块是在内核空间运行的，因此代码错误可能会破坏操作系统从而影响整个系统的可靠性，这对于实时应用将是一个非常严重的弱点



## 1.2.2交叉编译—1.2.3嵌入式Linux开发流程

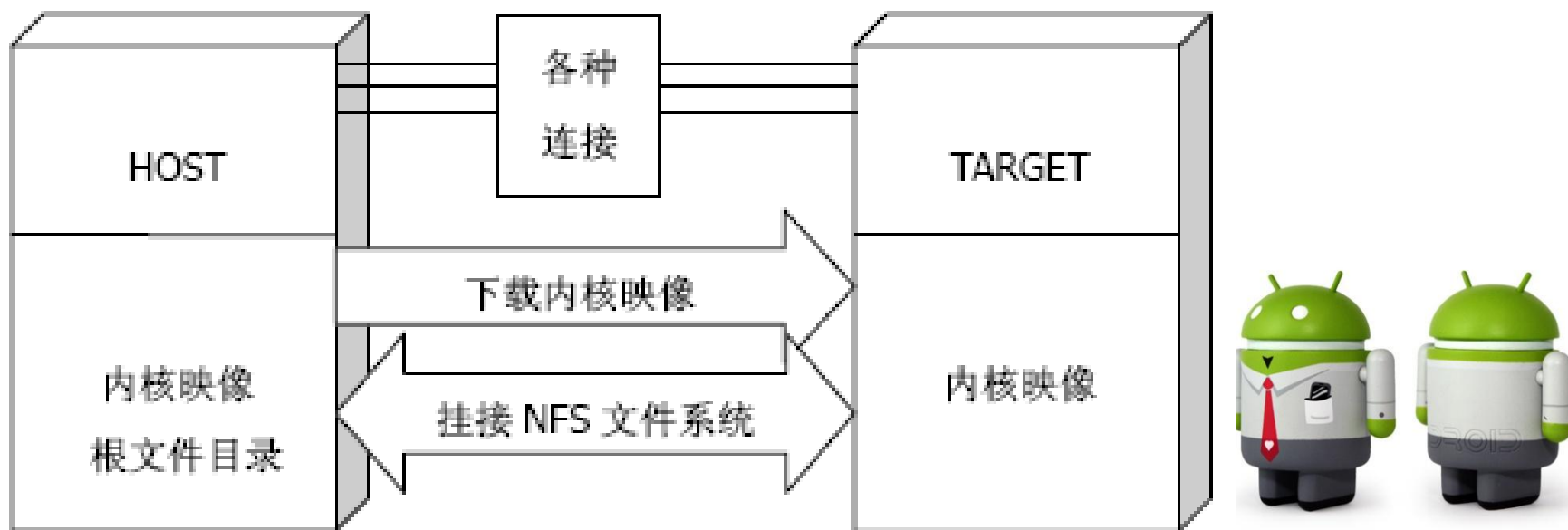
---

- ▶ 1 . 建立开发环境
- ▶ 2 . 配置开发主机
- ▶ 3 . 建立引导装载程序BOOTLOADER
- ▶ 4. 建立根文件系统
- ▶ 5 . 建立应用程序的Flash磁盘分区
- ▶ 6 . 开发应用程序
- ▶ 7 . 烧写内核、根文件系统、应用程序



# 交叉编译

- ▶ 交叉编译是嵌入式开发过程中的一项重要技术，简单地说，就是在一个平台上生成另一个平台上的可执行代码。
- ▶ 常用的计算机软件，都需要通过编译的方式，把使用高级计算机语言编写的代码（比如 C 代码）编译（compile）成计算机可以识别和执行的二进制代码。



# BOOTLOADER

---

- ▶ 简单地说，BootLoader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，
- ▶ Vivi：vivi是由韩国Mizi公司开发的一种Bootloader，适合于三星处理器
- ▶ 简单来说：先用vivi编辑软件将bootloader用汇编语言读入进来，然后执行驱动程序。





# Bootloader设计工作流程

## 阶段一：汇编

基本硬件设备初始化



为第二阶段准备RAM空间



复制Bootloader到RAM



设置堆栈



跳转到第二阶段C入口

## 阶段二：C语言

初始化本阶段所需硬件



检测系统内存映射



将内核文件系统映象读到RAM



设置内核启动参数



调用内核

提供SHELL

提供基本驱动：串口等  
支持固化内核和文件系统映象  
支持简单的命令操作



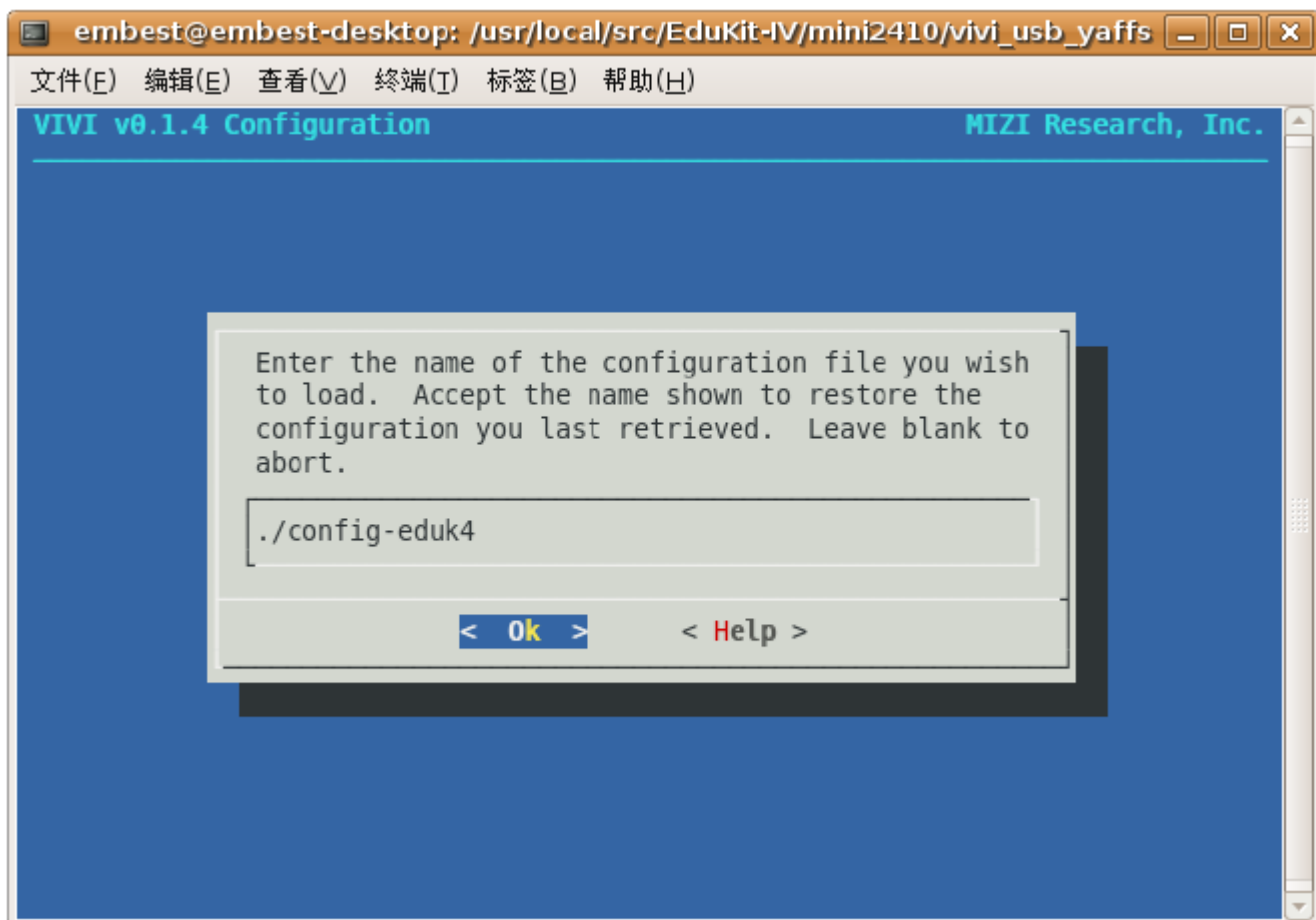
# Linux映像固化与运行

---

- ▶ 嵌入式Linux在宿主机上编译后会生成映像文件，这个映像文件一般来说需要固化到Flash中
- ▶ 一般BootLoader具有擦除Flash的功能，也具有从宿主主机接收映像文件的功能，因此我们一般通过BootLoader来完成映像的固化和更新。
- ▶ 刷机内容：映像——Linux的基本映像包含三个部分bootloader、内核、根文件系统
- ▶ 对于普通用户来说：刷机就是内核、根文件系统



# 编译vivi



```
$ make
```

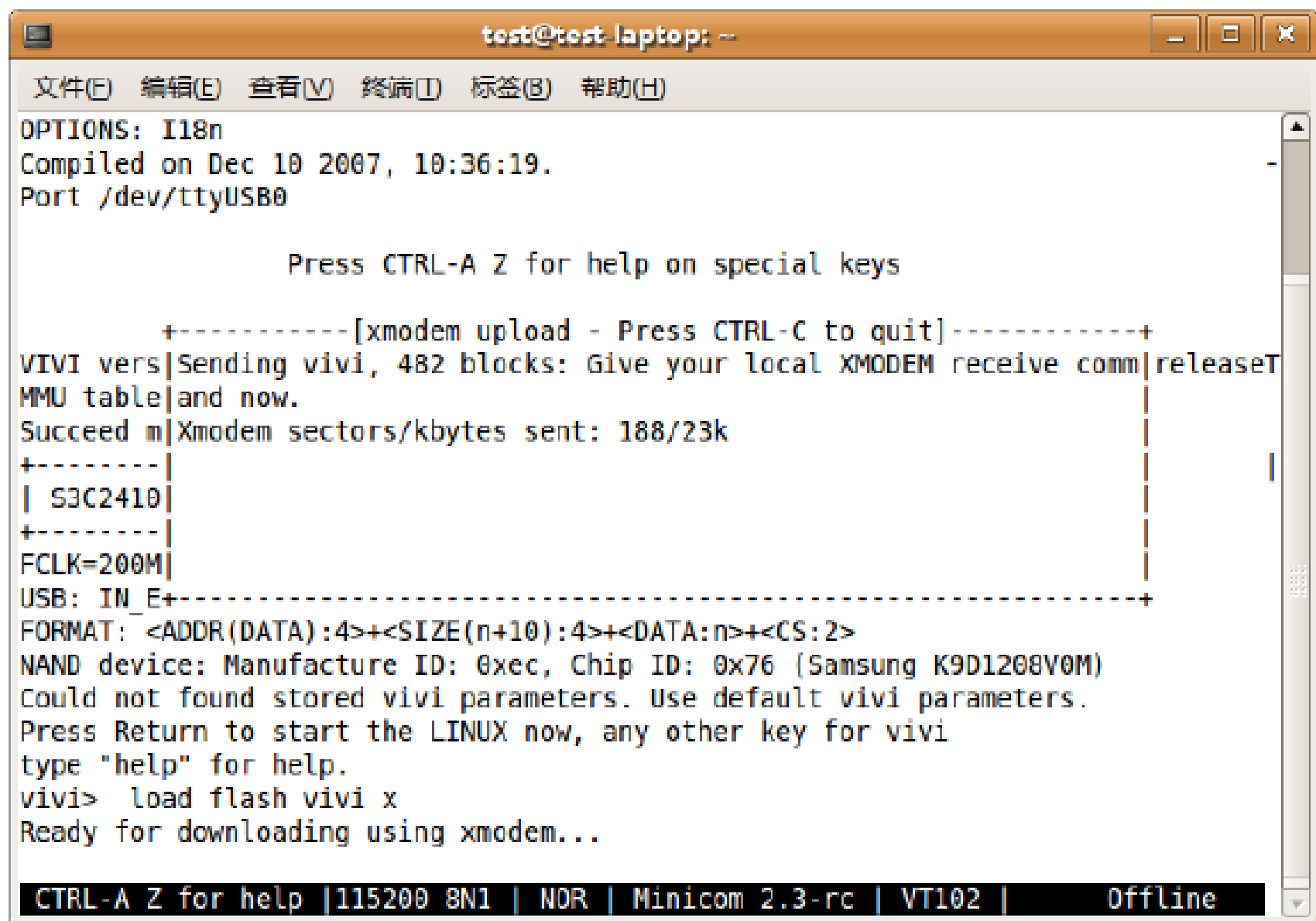
6) 拷贝文件:

```
$ make install
```

执行完命令后, 将自动拷贝 vivi 到/home/example/目录下。



# 固化vivi



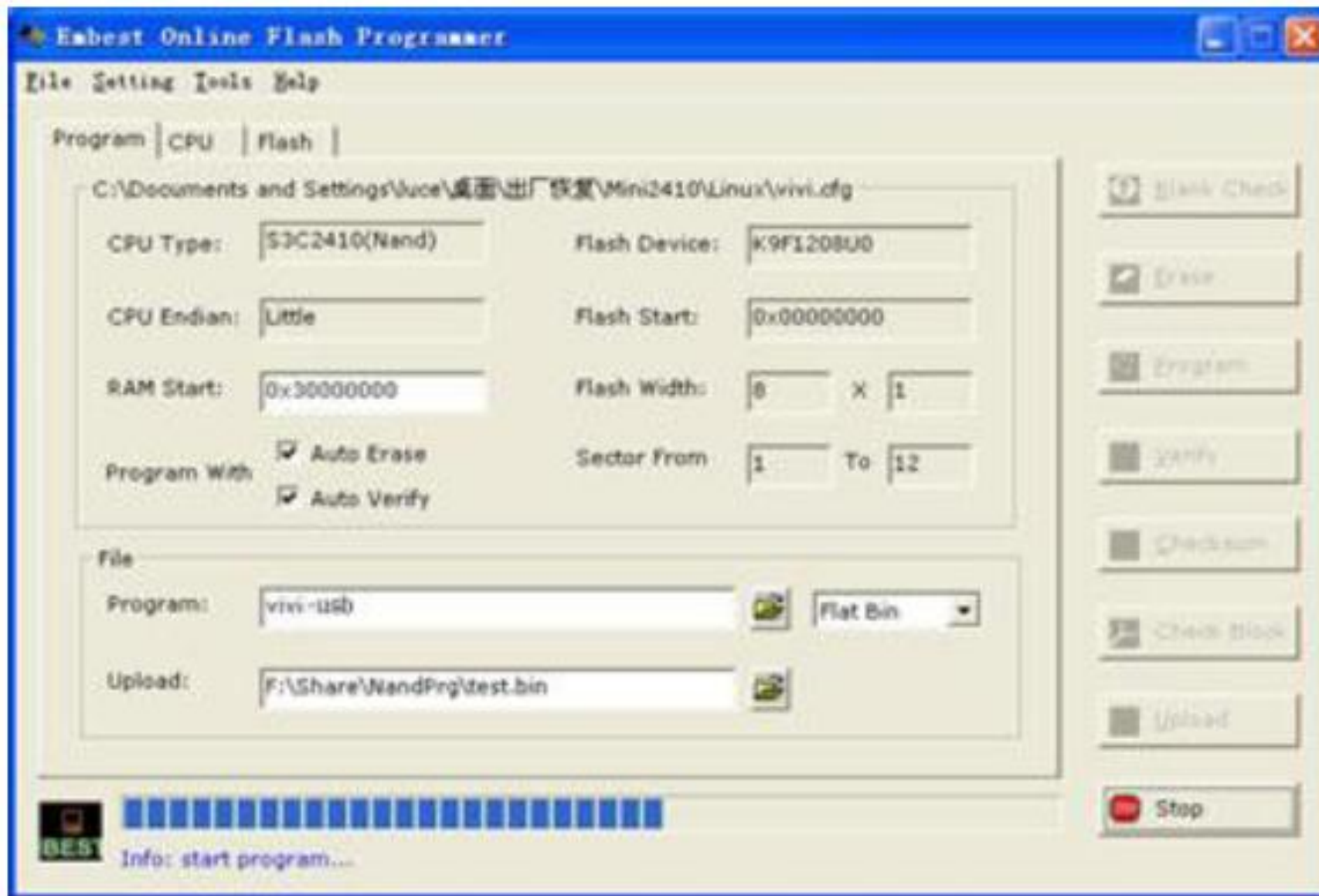
```
test@test-laptop: ~
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)
OPTIONS: I18n
Compiled on Dec 10 2007, 10:36:19.
Port /dev/ttyUSB0

Press CTRL-A Z for help on special keys

+-----[xmodem upload - Press CTRL-C to quit]-----+
VIVI vers|Sending vivi, 482 blocks: Give your local XMODEM receive comm|releaseT
MMU table|and now.
Succeed m|Xmodem sectors/kbytes sent: 188/23k
+-----|
| S3C2410|
+-----|
FCLK=200M|
USB: IN_E+-----+
FORMAT: <ADDR(DATA):4>+<SIZE(n+10):4>+<DATA:n>+<CS:2>
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
Could not found stored vivi parameters. Use default vivi parameters.
Press Return to start the LINUX now, any other key for vivi
type "help" for help.
vivi> load flash vivi x
Ready for downloading using xmodem...

CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.3-rc | VT102 | Offline
```

# 烧写vivi映像：固化启动映像Bootloader



# 内核编译与运行

---

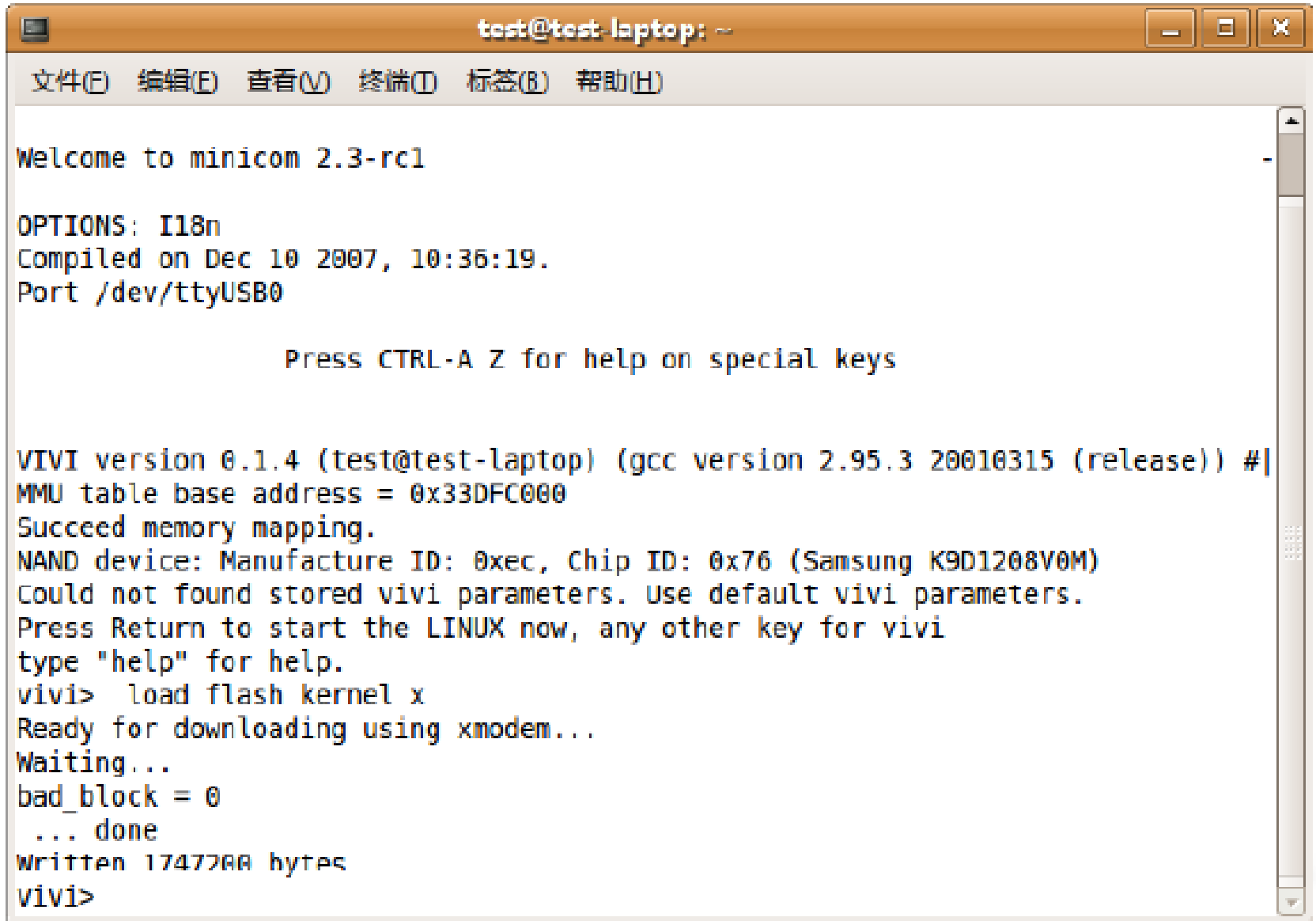
- Linux的基本映像包含三个部分bootloader、内核、根文件系统,下面处理内核

```
$ make zImage
```

编译完成后将在 arch/arm/boot 目录下生成 zImage 文件，并自动拷贝到/home/example 目录下。



# 在Ubuntu下采用minicom终端更新：新刚编译生成的zImage



```
test@test-laptop: ~
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)

Welcome to minicom 2.3-rc1

OPTIONS: I18n
Compiled on Dec 10 2007, 10:36:19.
Port /dev/ttyUSB0

        Press CTRL-A Z for help on special keys

VIVI version 0.1.4 (test@test-laptop) (gcc version 2.95.3 20010315 (release)) #|
MMU table base address = 0x33DFC000
Succeed memory mapping.
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
Could not found stored vivi parameters. Use default vivi parameters.
Press Return to start the LINUX now, any other key for vivi
type "help" for help.
vivi> load flash kernel x
Ready for downloading using xmodem...
Waiting...
bad_block = 0
... done
Written 1747200 bytes
vivi>
```



# 利用busybox制作一个文件系统

## 1. 编译 busybox

1) 单击菜单应用程序->附件->终端打开终端，设置环境变量：

```
$ source /usr/local/src/EduKit-IV/Mini2410/set_env_linux.sh  
$ source /usr/crosstool/gcc-3.4.5-glibc-2.3.6/arm-linux/path.sh
```

2) 执行命令切换到 busybox 实验目录下：

```
$ cd $SIMPLEDIR/6.3-busybox
```

3) 解压 busybox-1.1.2.tar.bz2:

```
$ tar jxvf busybox-1.1.2.tar.bz2
```

4) 进入目录，并打上补丁：

```
$ cd busybox-1.1.2  
$ patch -p1 <../1.1.2.patch
```

5) 清除早前可能存在的配置信息：

```
$ make distclean
```

6) 执行配置命令：

```
$ make menuconfig
```

1) 建立 root-mini 文件系统目录，并拷贝 busybox 生成的文件到该目录内：

```
$ cd $SIMPLEDIR/6.3-busybox  
$ mv busybox-1.1.2/_install root-mini
```

2) 创建文件系统目录树结构，并拷贝必要的文件到文件系统目录内：

```
$ cd root-mini  
$ mkdir -p dev etc/init.d home mnt proc root sys tmp var media  
$ cd ..  
$ cp rz root-mini/bin/  
$ cp flash_eraseall root-mini/sbin/
```

3) 拷贝 inittab 文件到 etc 目录下：

```
$ cp busybox-1.1.2/examples/inittab root-mini/etc/
```

4) 在 etc/init.d 目录下建立 rcS 文件，内容如下：

```
#!/bin/sh  
  
echo "running /etc/init.d/rcS"  
# mount the /proc file system  
/bin/mount -t proc proc /proc  
  
echo "mount tmpfs filesystem to /tmp"  
/bin/mount -t tmpfs none /tmp  
  
echo "mount ramfs filesystem to /var"  
/bin/mount -t ramfs none /var
```



# 建立根文件系统

- ▶ 利用ramdisk将已经完成的**bootloader**、**内核**、**文件系统**，制作一个根文件系统镜像。
- ▶ 制作**ramdisk根文件系统映像**

1) 单击菜单应用程序->附件->终端打开终端，设置环境变量：

```
$ source /usr/local/src/EduKit-IV/Mini2410/set_env_linux.sh  
$ source /usr/crosstool/gcc-3.4.5-glibc-2.3.6/arm-linux/path.sh
```

2) 执行命令切换到 ramdisk 实验目录下：

```
$cd $SIMPLEDIR/6.4-ramdisk
```

3) 运行脚本文件：

```
$ sudo sh ramdisk-install.sh
```



# 固化引导**ramdisk**文件系统

```
vivi> load flash ramdisk x
```

```
test@test-laptop: ~  
文件(F) 编辑(E) 查看(V) 终端(T) 标签(B) 帮助(H)  
  
Welcome to minicom 2.3-rc1  
  
OPTIONS: I18n  
Compiled on Dec 10 2007, 10:36:19.  
Port /dev/ttyUSB0  
  
Press CTRL-A Z for help on special keys  
  
VIVI version 0.1.4 (test@test-laptop) (gcc version 2.95.3 20010315 (release)) #|  
MMU table base address = 0x33DFC000  
Succeed memory mapping.  
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)  
Could not found stored vivi parameters. Use default vivi parameters.  
Press Return to start the LINUX now, any other key for vivi  
type "help" for help.  
vivi> load flash ramdisk x  
Ready for downloading using xmodem...  
Waiting...  
bad_block = 0  
... done  
Written 540416 bytes  
vivi>
```



## 1.2.4 驱动开发

- ▶ LCD显示驱动开发、触摸屏驱动、SD卡驱动、IIS音频驱动、USB通信、摄像头驱动、蓝牙协议和串口通信、GPS模块的控制、GPRS通信。每个生成手机的厂商对于购买的硬件需要正常工作编写硬件驱动



- ▶ 最下层编写的硬件驱动需要和编译的android源文件编译的系统相对应才能正常使用手机硬件



# 驱动编程思路

---

- ▶ 包含必要的linux头文件：import
- ▶ 打开驱动设备文件：初始化，其目的让硬件正常运行，例如灯泡不能狂闪，也让硬件为有序状态
- ▶ 调用驱动接口函数交换数据：注册设备、创建设备、许可协议、操作（open、read、write）+函数
- ▶ 关闭驱动设备：移除设备、注销设备



# 驱动分类

---

## ▶ 1、linux核心驱动

1. Android1.5 ( cupcake ) 使用linux2.6.27
2. Android1.6 ( donut ) 使用linux2.6.29
3. Android2.2 ( Froyo ) 使用linux2.6.32

Android对linux内核更改很少，增加下面私有的内容。

- ▶ 2、android专用驱动
- ▶ 3、android使用的设备驱动



### Get Android 2.2!

The Android 2.2 platform is now available for the Android SDK, along with new tools, documentation, and a new NDK. For information about new features and APIs, read the [version notes](#).

If you have an existing SDK, add Android 2.2 as an [SDK component](#). If you're new to Android, install the [SDK starter package](#).





## 【知识点】linux版本号

---

- 主版本号和次版本号标志着重要的功能变动；修正号表示较小的功能变动。以2.6.12版本为例，2代表主版本号，6代表次版本号，12代表修正号。其中次版本号还有特定的意义：如果次版本号是偶数，那么该内核就是稳定版的；若是奇数，则是开发版的。例如：1.2.0是发布版，而1.3.0则是开发版。头两个数字合在一齐可以描述内核系列。



# android专用驱动

---

- ▶ Ashmen：匿名共享内存驱动
- ▶ Logger：轻量级log驱动
- ▶ Binder：基于OpenBinder系统驱动，为android平台提供IPC支持
- ▶ Android Power Mangement：电源管理模块
- ▶ Low memory killer：缺少内存的情况下，杀死进程
- ▶ Andorid pmem：物理内存驱动



# android使用的设备驱动

作为主要为智能机定制的操作系统，android通常开可以使用为linux中一些标准的设备驱动程序

1. Framebuffer：显示驱动
2. Event输入设备驱动
3. V412摄像头视频驱动
4. OSS、ALSA音频驱动
5. MTD内存技术驱动，通常用于linux下flash驱动程序
6. 蓝牙、wlan驱动



# 【知识点】

## ▶ Linux设备驱动主要完成以下几个“动作”：

- ▶ 必备Linux头文件；
- ▶ 初始化与关停设备函数；
- ▶ 完成file\_operations结构函数。

```
#include <linux/module.h>
#include <linux/init.h>

static ssize_t my_read(struct file *file, char __user *buf, size_t count, loff_t *ppos){}
static ssize_t my_write(struct file *filp, const char __user *buf, size_t count, loff_t *f_pos){}
static int my_open(struct inode * inode, struct file * filp){}
static int my_release(struct inode * inode, struct file * filp){}

static struct file_operations my_fops = {
    .owner    = THIS_MODULE,
    .read     = my_read,
    .write    = my_write,
    .open     = my_open,
    .release  = my_release,
};

static int __init my_init(void){}
static void __exit my_exit(void){}

module_init(my_init);
module_exit(my_exit);

MODULE_ALIAS("mydev");
MODULE_DESCRIPTION("The first char driver");
MODULE_AUTHOR("luce");
MODULE_LICENSE("GPL");
```

# 初始化与关停

---

## ► 模块初始化：

```
static int __init initialization_function(void)
{
    /* Initialization code here */
}
module_init(initialization_function);
```

## ► 清除模块：

```
static void __exit cleanup_function(void)
{
    /* Cleanup code here */
}
module_exit(cleanup_function);
```

## ► 函数内容：

- 模块初始化函数主要是注册设备，初始化相关硬件；
- 清除模块函数主要是卸载设备，关闭硬件。



# 字符设备注册

## ▶ 注册/释放设备号：

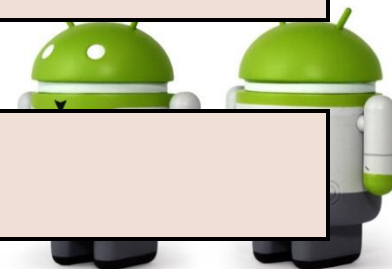
```
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor, unsigned int count, char *name);  
void unregister_chrdev_region(dev_t first, unsigned int count);
```

## ▶ 注册/注销驱动：

```
struct cdev *my_cdev = cdev_alloc(); // 分配内存  
void cdev_init( struct cdev *cdev, struct file_operations *fops); // 初始化  
my_cdev->ops = &chr_fops;  
int cdev_add( struct cdev *dev, dev_t num, unsigned int count); // 注册  
void cdev_del(struct cdev *dev); // 注销
```

## ▶ ``` int register_chrdev(unsigned int major, const char *name, struct file_operations *fops); int unregister_chrdev(unsigned int major, const char *name); ```

## ▶ ``` int devfs_mk_cdev(dev_t dev, umode_t mode, const char *fmt, ...); void devfs_remove(const char *fmt, ...); ```



# open&release

- ▶ open方法提供给驱动程序以初始化的能力，从而为以后的操作完成初始化做准备。Open一般完成以下工作：
  - ▶ 检查设备是否有特定的错误（诸如设备未就绪或类似的硬件问题）；
  - ▶ 如果设备是首次打开，则对其进行初始化；
  - ▶ 如有必要，更新f\_op指针；
  - ▶ 分配并填写置于filp->private\_data里的数据结构。

```
int scull_open(struct inode *inode, struct file *filp)
{
    filp->private_data = inode->i_cdev;

    if ( (filp->f_flags & O_ACCMODE) == O_WRONLY)
    {
        scull_trim(dev);
    }

    return 0;
}
```

- ▶ release方法的作用正好与open相反，一般应该完成以下工作：
  - ▶ 释放由open分配的，保存在filp->private\_data中的所有内容；
  - ▶ 在最后一次关闭操作时关闭设备。





# read&write

---

- ▶ read和write方法完成的任务类似，即拷贝数据到应用程序空间，或反过来从应用程序空间拷贝数据。函数原型为：

```
ssize_t read(struct file *filp, char __user *buff, size_t count, loff_t *offp);  
ssize_t write(struct file *filp, const char __user *buff, size_t count, loff_t *offp);
```

- ▶ 用户地址空间跟内核地址空间的数据交换：

```
unsigned long copy_to_user(void __user *to,const void *from,unsigned long count);  
unsigned long copy_from_user(void *to,const void __user *from,unsigned long count);
```



# 应用程序调用驱动接口

---

## ▶ 应用程序编程思路：

- ▶ 包含必要的linux头文件；
- ▶ 打开驱动设备文件；
- ▶ 调用驱动接口函数交换数据；
- ▶ 关闭驱动设备。

## ▶ 应用程序实例：

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char** argv){
    int fd, buf;
    float result;

    fd = open("/dev/adc",O_RDWR);
    if(fd < 0){
        printf("Can't open the s3c2410_adc drivers!\n");
        return fd;
    }
    while(1){
        read(fd,&buf,sizeof(int));
        result = buf*3.3000/0x3FF;
        printf("ADC Channel 0 data = %0.4f \n", result);
        usleep(1000*500);
    }
    close(fd);
    return 0;
}
```



```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/devfs_fs_kernel.h>
#include <linux/miscdevice.h>
#include <linux/delay.h>
```

```
#include <asm/irq.h>
#include <asm/io.h>
#include <asm/arch/regs-gpio.h>
#include <asm/hardware.h>

#define DEVICE_NAME "led"
#define LED_MAJOR 231
#define LED_BASE    (0xE1180000)
```

```
static int eduk4_led_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
    unsigned char status;
```

## 讲到这里你已经明白：

---

- ▶ 在没有系统的时候android手机中的硬件需要用到MDK进行编写驱动以及调试的，以及调试顺序。
- ▶ 刷机 and 烧入系统是一样的，需要通过boot刷入内核以及根文件系统
- ▶ Boot：BootLoader 就是在操作系统内核运行之前运行的一段小程序，是用汇编语言
- ▶ 内核：要了解linux、交叉编译、内核编译之后用minicom更新
- ▶ 用 busybox生成一个mini的文件内核，编译后固化到手机中
- ▶ 最后加载硬件驱动，手机就可以启动了。



## 【知识点】触摸屏

---

- ▶ 电容屏：容值的变化定位。因为静电感应，需要导体接触屏幕才会有反应，所以，不需要很用力，只要手指轻轻触摸屏幕即可被识别。那么，普通的手写笔就没法用于电容屏了，电容屏有专用手写笔，带静电的。支持多点触摸（wm不支持多点），灵敏度没有电阻屏高。
- ▶ 电阻屏：阻值变化定位。因为压力感应，需要用力才会有反应。电阻屏的优点是可以精确定位，适合编辑文档等商务应用（wm）。缺点是不支持多点触摸，灵敏度没有电容屏高。（山寨手机常用）



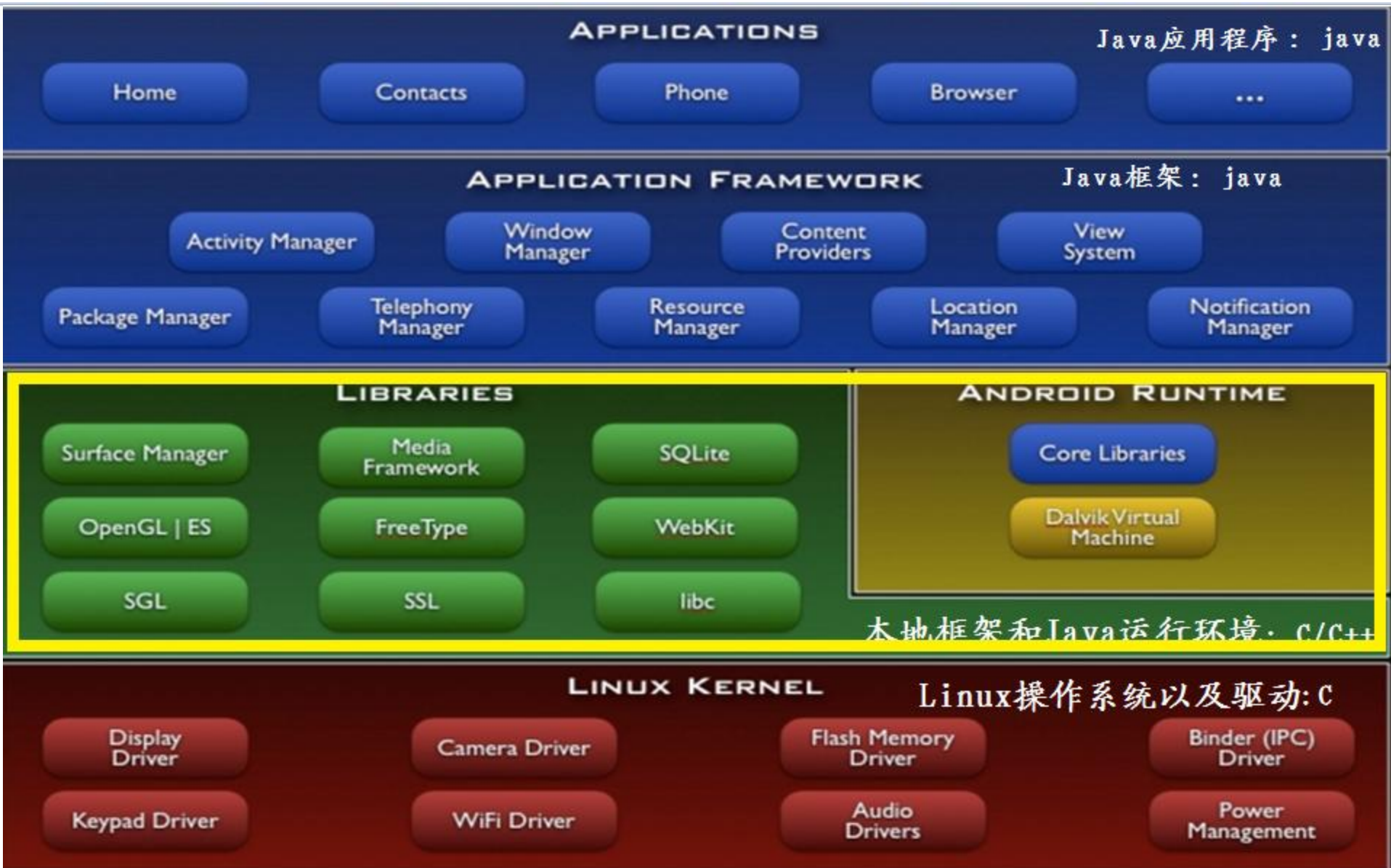


## 二、Android文件目录以及底层库

Android文件系统移植到嵌入式平台



从互联网上下载的android源代码是这样的（全）





【知识点】

---

# ▶ 如何获取Android源码



# Android源码管理

---

- ▶ Google Android源码是通过git管理和发布的，所以需要在ubuntu环境安装git工具。
  - ▶ Git是Linux Torvalds(Linux之父)为了帮助管理Linux内核开发而开发的一个开放源码的分布式版本控制软件；

```
$ sudo apt-get install git-core curl
```

- 因为Android是由Kernel、Dalvik、Bionic、Prebuilt、build等多个项目组成，如果我们分别使用Git来逐个获取显得很麻烦，所以Android项目编写了一个名为Repo的Python的脚本来统一管理这些项目的仓库，使得项目的获取更加简单。

```
$ curl http://android.git.kernel.org/repo >~/bin/repo
```

```
$ sudo cp repo /bin/
```

```
$ sudo chmod a+x /bin/repo
```



# 获取Android源码

- ▶ 确保网络正常，接下来就可以通过命令来获取Android了，整个过程比较长：

- ▶ 建立本地源码目录：

```
$ mkdir mydroid
```

```
$ cd mydroid
```

- ▶ 初始化本地项目库：

```
$ repo init -u git://android.git.kernel.org/platform/manifest.git
```

- ▶ 获取Android项目代码到本地

```
$ repo sync
```

## 小贴士：

如果只是想获取某个分之版本的代码可以使用以下命令来初始化本地项目库：

```
$ repo init -u git://android.git.kernel.org/platform/manifest.git -b cupcake
```



# 问题

---

- ▶ 网上下载源代码可否直接用？
- ▶ 这些源代码和linux kernel有什么关系？
- ▶ 手机里面的目录是怎么样的？
- ▶ 手机里面的文件系统和android开源代码之间存在一个什么关系？



# 本节概述

---

- ▶ Android源代码文件系统部分介绍
- ▶ Linux内核启动挂载android根文件系统的分析
- ▶ Android 文件系统初始化核心Init.c 文件分析
- ▶ 初始化核心的核心init.rc 文件分析



## 2.1 android 源代码文件系统部分介绍

- 从google 获得源代码后，在platform 目录下make 编译后我们可以看到生成了out 目录。

```
[root@monet android platform]# ls
```

```
Makefile
```

```
[root@monet android platform]# ll
```

```
total 60
```

```
drwxrwxrwx  2 root root 4096 2009-10-18 01:42
drwxrwxrwx  9 root root 4096 2009-10-18 04:03
drwxrwxrwx  5 root root 4096 2009-10-18 04:03
drwxrwxrwx  8 root root 4096 2009-10-18 04:03
drwxrwxrwx 18 root root 4096 2009-10-18 04:04
drwxrwxrwx 17 root root 4096 2009-10-18 04:05
drwxrwxrwx 75 root root 4096 2009-10-18 04:11
drwxrwxrwx  5 root root 4096 2009-10-18 04:12
drwxrwxrwx  6 root root 4096 2009-10-18 04:12
-rwxrwxrwx  1 root root   87 2009-10-18 04:03
drwxrwxrwx  5 root root 4096 2009-10-20 10:26
drwxrwxrwx  5 root root 4096 2009-10-18 04:12
drwxrwxrwx 11 root root 4096 2009-10-18 04:15
drwxrwxrwx  7 root root 4096 2009-10-18 04:15
drwxrwxrwx  7 root root 4096 2009-10-18 04:15
```

```
Makefile
```

```
[root@monet android platform]#
```

# 主要源代码目录介绍-内核移植

---

- ▶ Makefile ( 全局的Makefile )
- ▶ bionic ( Bionic 含义为仿生, 这里面是一些基础的库的源代码 )
- ▶ bootable ( 引导加载器, 在内核运行前执行 )
- ▶ build ( build 目录中的内容不是目标所用的代码, 而是编译和配置所需要的脚本和工具 )
- ▶ dalvik ( JAVA 虚拟机 )
- ▶ development ( 程序开发所需要的模板和工具 )
- ▶ external ( 目标机器使用的一些库 )
- ▶ frameworks ( 应用程序的框架层 )
- ▶ hardware ( 与硬件相关的库 )
- ▶ packages ( Android 的各种应用程序 )
- ▶ prebuilt ( Android 在各种平台下编译的预置脚本 )
- ▶ recovery ( 与目标的恢复功能相关 )
- ▶ system ( Android 的底层的一些库 )
- ▶ out (编译完成后产生的目录, 也就是我们移植文件系统需要的目录)



## out 目录

### ▶ 主要的两个目录为host 和target

1. 前者表示在主机 ( x86 ) 生成的工具
2. 后者表示目标机 ( 模认为 ARMv5 ) 运行的内容。

```
[root@monet out]# ls
casecheck.txt  CaseCheck.txt  host  target  zip
[root@monet out]# ll
total 20
-rwxrwxrwx 1 root root    2 2009-11-06 20:17 casecheck.txt
-rwxrwxrwx 1 root root    2 2009-11-06 20:17 CaseCheck.txt
drwxrwxrwx 4 root root 4096 2009-10-20 07:21 host
drwxrwxrwx 4 root root 4096 2009-10-20 07:22 target
drwxrwxrwx 3 root root 4096 2009-10-20 10:26 zip
```

```
out/
|-- CaseCheck.txt
|-- casecheck.txt
|-- host
|   |-- common
|   |-- linux-x86
|-- target
|   |-- common
|   |-- product
```





## host 目录的结构如下所示:

---

```
out/host/
|-- common
|   |-- obj                (JAVA 库)
|-- linux-x86
|   |-- bin                (二进制程序)
|   |-- framework          (JAVA 库, *.jar 文件)
|   |-- lib                (共享库*.so)
|   |-- obj                (中间生成的目标文件)
```



## target 目录的结构如下所示:

---

```
out/target/  
|-- common  
|   |-- R  
|   |-- docs  
|   |-- obj  
|-- product  
|   |-- generic
```

(资源文件)

(目标文件)

product 中则是针对产品的内容. 所以, 取文件系统主要是在 `/out/target/product/generic` 目录下



# Generic文件目录

```
[root@monet generic]# pwd
/home/android/android_platform/out/target/product/generic
[root@monet generic]# ll
total 49692
-rwxrwxrwx  1 root root          7 2009-10-20 10:36 android-info.txt
-rwxrwxrwx  1 root root        550 2009-11-06 20:17 clean_steps.mk
drwxrwxrwx  2 root root       4096 2009-10-20 07:44 data 数据文件
-rw-r--r--  1 root root      13241 2009-11-07 04:51 installed-files.txt
drwxrwxrwx 13 root root       4096 2009-10-20 10:36 root 目标根文件系统 root
-rwxrwxrwx  1 root root         40 2009-11-06 20:17 previous_build_config.mk
-rwxrwxrwx  1 root root     156417 2009-11-07 04:51 ramdisk.img
drwxrwxrwx  8 root root       4096 2009-10-20 08:03 root 目标根文件系统 root
drwxrwxrwx  4 root root       4096 2009-10-20 08:03 system 主文件系统
drwxrwxrwx 10 root root       4096 2009-10-20 10:13 system 主文件系统
-rwxrwxrwx  1 root root    50607744 2009-11-07 04:51 system.img
-rwxrwxrwx  1 root root       2112 2009-11-07 04:51 userdata.img
[root@monet generic]#
```



# 进入obj 目录里面是android 文件系统非常重要的内容

```
[root@monet obj]# ll
```

total 404							
drwxrwxrwx	48	root	root	4096	2009-10-20	10:28	APPS
drwxrwxrwx	3	root	root	4096	2009-10-20	10:28	DATA
drwxrwxrwx	11	root	root	4096	2009-10-20	10:28	ETC
drwxrwxrwx	106	root	root	4096	2009-10-20	10:30	EXECUTABLES
drwxrwxrwx	7	root	root	4096	2009-10-20	07:21	INCLUDE
drwxrwxrwx	5	root	root	4096	2009-10-20	10:13	KEYCHARS
drwxrwxrwx	2	root	root	4096	2009-10-20	10:36	LIC
drwxrwxrwx	4	root	root	4096	2009-11-07	04:06	NOTICE FILES
-rwxrwxrwx	1	root	root	293701	2009-11-07	04:07	NOTICE.html
-rwxrwxrwx	1	root	root	56410	2009-11-07	04:07	NOTICE.html.gz
drwxrwxrwx	3	root	root	4096	2009-10-20	10:36	PACKAGING
drwxrwxrwx	106	root	root	4096	2009-10-20	10:36	SHARED LIBRARIES
drwxrwxrwx	189	root	root	12288	2009-11-07	00:02	STATIC LIBRARIES



---

▶ /obj

APPS (文件系统下/system/apps 目录下的各种应用程序)

SHARED\_LIBRARIES ( 存放所有动态库 )

STATIC\_LIBRARIES ( 存放所有静态库 )

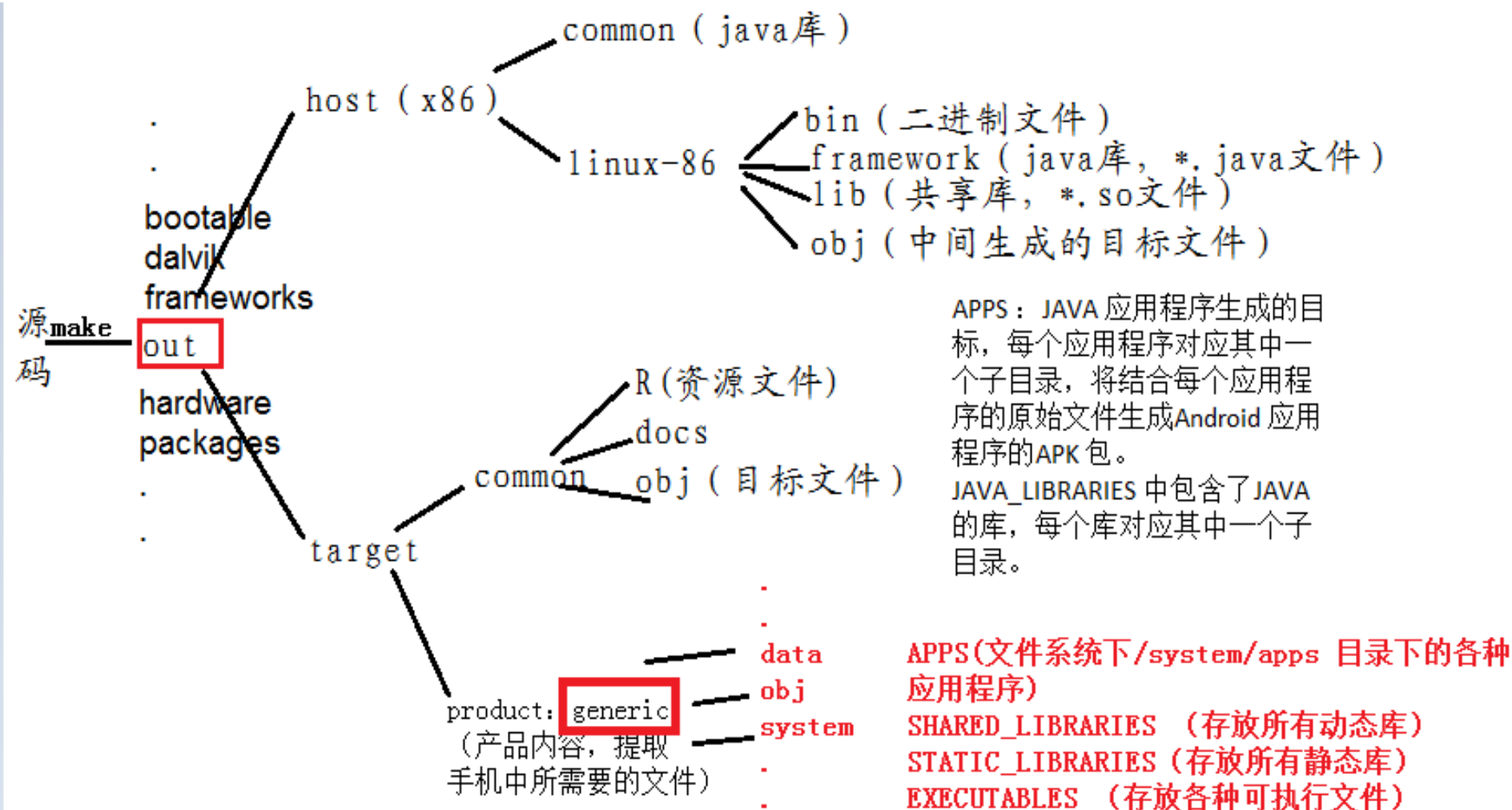
EXECUTABLES ( 存放各种可执行文件 )

还有其他需要的文件都是在

/out/target/product/generic 目录下



# 回顾一下



## 2.2 Linux 内核启动挂载android 根文件系统过程分析

---

- ▶ 分析linux 启动过程，一切要从内核  
/arch/arm/boot/compressed/head.S说起
- ▶ 顺便罗列一下内核启动流程





UBOOT

/arch/arm/boot/compressed/head.S:

Start:

Decompressed\_kernel() //在/arch/arm/boot/compressed/misc.c 中

Call\_kernel()

Stext:

/init/main.c

Start\_kernel()

Setup\_arch()

...

Rest\_init()

Init()

此段代码的亮点





## 2.3 Androindroid 文件系统初始化核心Init.c 文件

- ▶ 上面我们说的init 这个文件是由android 源代码编译来的，编译后在/out/target/product/generic/root/init
- ▶ Androindroid 文件系统初始化核心Init.c

其源码在 m/system/core/init/init.c

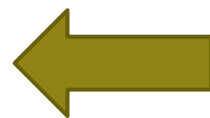
```
Android.mk  devices.h  keywords.h  property_service.c
bootchart.c grab-bootchart.sh logo.c  property_service.h
bootchart.h init.c  MODULE_LICENSE_APACHE2  README.BOOTCHART
builtins.c  init.c~  NOTICE  readme.txt
devices.c   init.h  parser.c  util.c

[root@monet init]# pwd
/home/android/android_platform/system/core/init
[root@monet init]#
```

- ▶ 至此整个android 文件系统已经起来了。



## 【知识点】 标准c/c++库bionic



- ▶ Bionic提供的C/c++标准库的功能，它是一个专为嵌入式系统设计的轻量级标准库实现。
- ▶ Bionic加入了android独有的一些功能，比如log的底层支持。另外，还实现了一套property系统，这是整个android全局变量的存储区域，bionic使用共享内存的方式来实现property系统。
- ▶ 此外android还提供libcutils c语言工具库，基本上android中所有的本地库和程序都连接了这个库。





### 三、android的java虚拟机和java环境

Dalvik、dex、JNI

# Dalvik虚拟机和核心库

- ▶ 目前机器运行一切正常
- ▶ 接下来android需要：
  1. java基本的运行环境：虚拟机Dalvik
  2. 与标准j2se兼容的类库：核心库



## 3.1 dex工具库和虚拟机的实现

---

- ▶ Core Libraries : 写java代码需要的语法。
- ▶ Dalvik虚拟机使用的字节码是dex格式，dex格式生成的最终文件的dex(java-class).
- ▶ 区别在于dex字节码将多个文件整个成一个，在通用性和可移植性上差一点，但是可以获得更好的性能。



## 3.2 核心库

- ▶ 核心库提供了基本java类库的功能，包含了基础数据结构、数学、IO、工具、数据库、网络等方面内容



### 3.3 第三层：AF/JAVA程序的运行环境

- ▶ 第三层：java程序的框架
- ▶ 第四层：java应用程序框架
- ▶ 第三层、第四层的接口是android系统的java API，包括有：java标准API（java包）、java扩展API（javax包）、企业和组织提供的java类库（org包）、android各种包（android包）





# API: 代码中的API和API描述文件相匹配

The screenshot shows the Android Developers website interface. At the top, there's a navigation bar with the Android logo and the word 'developers' in green. To the right, there's a language selector set to 'English' and a link to 'Android.com'. Below this is a search bar labeled 'search developer docs' with a 'Search' button. A secondary navigation bar contains links for 'Home', 'SDK', 'Dev Guide', 'Reference' (which is highlighted), 'Resources', 'Videos', and 'Blog'. On the far right of this bar is a 'Filter by API Level' dropdown set to '8'. The main content area is divided into a left sidebar and a main panel. The sidebar has links for 'Package Index' and 'Class Index', followed by a list of packages including 'android', 'android.accessibilityservice', 'android.accounts', 'android.app', 'android.app.admin', 'android.app.backup', 'android.appwidget', 'android.bluetooth', 'android.content', 'android.content.pm', 'android.content.res', 'android.database', 'android.database.sqlite', 'android.gesture', and 'android.graphics'. Below this is a 'Classes' section with a list of classes: 'Manifest', 'Manifest.permission', 'Manifest.permission\_group', 'R', 'R.anim', 'R.array', 'R.attr', and 'R.bool'. At the bottom of the sidebar is a 'Use Tree Navigation' link. The main panel displays the 'android' package page. It has a header 'package android' with 'Since: API Level 1' on the right. Below the header are links for 'Classes' and 'Description'. The 'Description' section contains the text: 'Contains the resource classes used by standard Android applications.' followed by a paragraph: 'This package contains the resource classes that Android defines to be used in Android applications. Third party developers can use many of them also for their applications. To learn more about how to use these classes, and what a resource is, see [Resources and Assets](#).' Below this is a license notice: 'Except as noted, this content is licensed under [Apache 2.0](#). For details and restrictions, see the [Content License](#).' and the version information 'Android 2.2 r1 - 02 Aug 2010 18:21'. At the bottom of the main panel are links for 'Site Terms of Service', 'Privacy Policy', and 'Brand Guidelines'.

Android developers

English Android.com

search developer docs Search

Home SDK Dev Guide **Reference** Resources Videos Blog

Filter by API Level: 8

[Package Index](#) | [Class Index](#)

**android**

android.accessibilityservice  
android.accounts  
android.app  
android.app.admin  
android.app.backup  
android.appwidget  
android.bluetooth  
android.content  
android.content.pm  
android.content.res  
android.database  
android.database.sqlite  
android.gesture  
android.graphics

**Classes**

Manifest  
Manifest.permission  
Manifest.permission\_group  
R  
R.anim  
R.array  
R.attr  
R.bool

Use Tree Navigation

package **android** Since: API Level 1

[Classes](#) | [Description](#)

Contains the resource classes used by standard Android applications.

This package contains the resource classes that Android defines to be used in Android applications. Third party developers can use many of them also for their applications. To learn more about how to use these classes, and what a resource is, see [Resources and Assets](#).

Except as noted, this content is licensed under [Apache 2.0](#). For details and restrictions, see the [Content License](#).

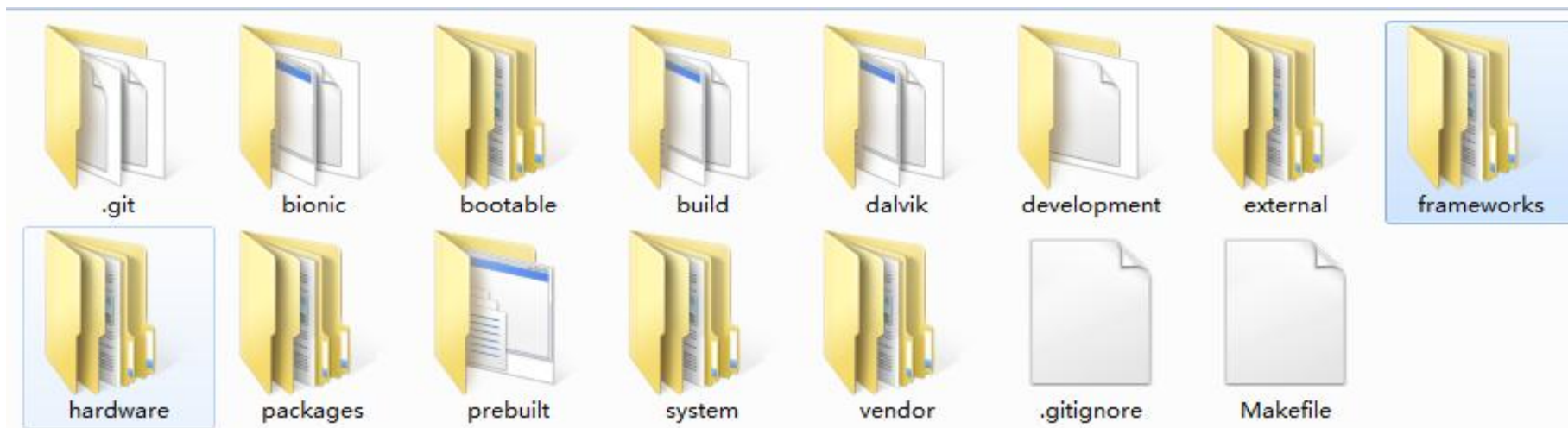
Android 2.2 r1 - 02 Aug 2010 18:21

[Site Terms of Service](#) - [Privacy Policy](#) - [Brand Guidelines](#)



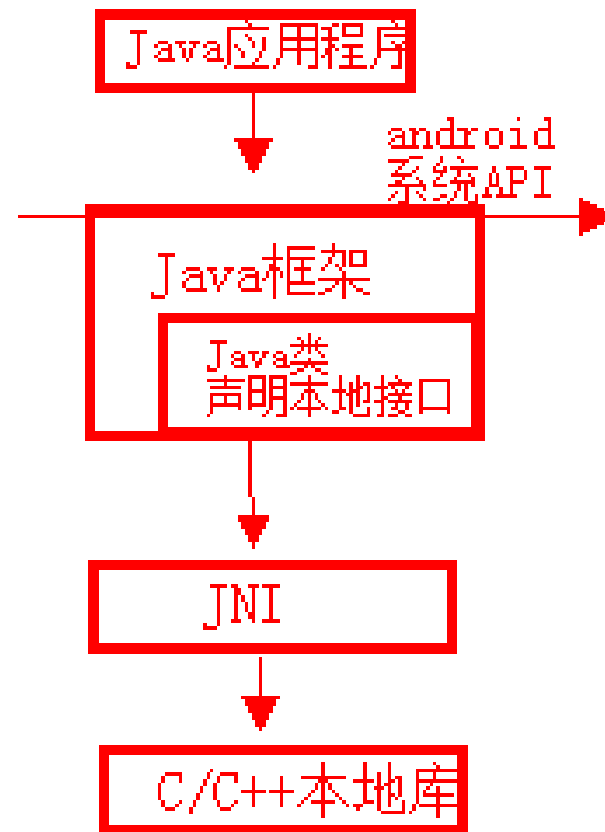
# framework

- ▶ 在framework除了android提供的核心java类，还包含了java框架图形、媒体、openGL、WiFi、电话短信类。
- ▶ 换句话说装好android之后，实现打电话的功能只需要调用函数即可。



## 3.4 JNI (java native interface) 的使用

- ▶ 由于Android的应用层级类别都是以Java撰写的，这些Java类别转译为Dex型式的Bytecode之后，必须仰赖Dalvik虚拟机(VM: Virtual Machine)来执行。
- ▶ 另外，当java需要调用c native组件时，VM就会去加载本地的c组件，让java函数能顺利的调用到C函数。此时，VM扮演着桥梁的角色，让java和c组件能通过透明的JNI接口相互沟通。





## 四、Android的ADB工具使用

Linux 命令、SD卡

# Android Debug Bridge

```
C:\Users\Administrator>adb version  
Android Debug Bridge version 1.0.26
```

```
C:\Users\Administrator>adb shell
```

```
# ls  
ls  
sqlite_stmt_journals  
config  
cache  
sdcard  
d  
etc  
system  
sys  
sbin  
proc  
init.rc  
init.goldfish.rc  
init  
default.prop  
data  
root  
dev  
#
```

运行



Windows 将根据您所输入的名称，为您打开相应的文件夹、文档或 Internet 资源。

打开(O):

cmd



使用管理权限创建此任务。

Windows 任务管理器

文件(F) 选项(O) 查看(V) 帮助(H)

应用程序 进程 服务 性能 联网 用户

映像名称	用户名	CPU	内存 (专用...
taskhost.exe	Admin...	00	1,348 K
WINWORD.EXE	Admin...	00	26,104 K
adb.exe	Admin...	00	720 K
explorer.exe	Admin...	00	18,992 K
POWERPNT.EXE	Admin...	00	33,940 K
emulator.exe	Admin...	03	143,512 K
taskmgr.exe	Admin...	01	3,012 K

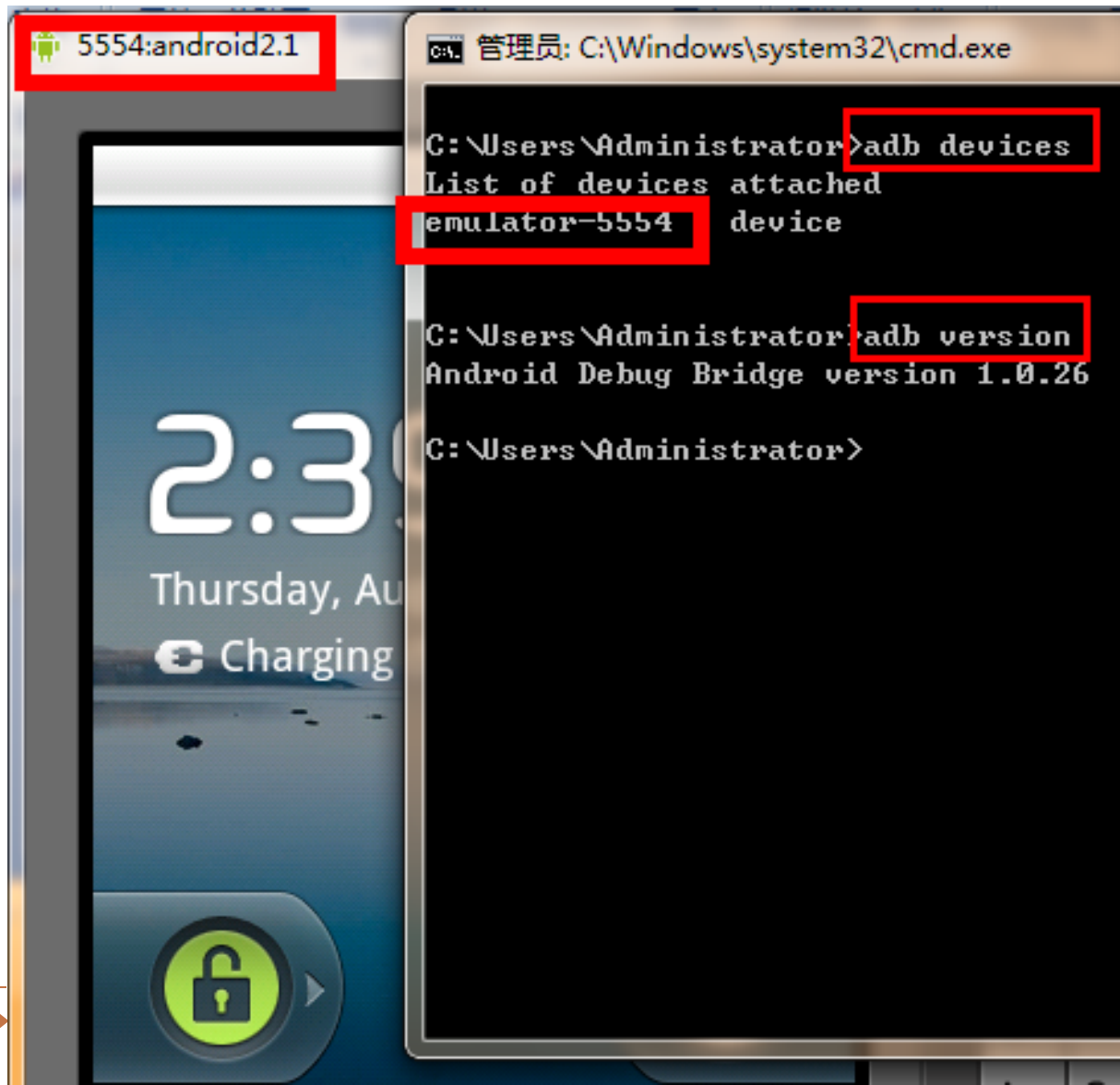
# ADB

---

- ▶ 在SDK的Tools文件夹下包含着Android模拟器操作的重要命令ADB
  - ( 1 ) 快速更新设备或手机模拟器中的代码，如应用或Android系统升级；
  - ( 2 ) 在设备上运行Shell命令；
  - ( 3 ) 管理设备或手机模拟器上的预定端口；
- ▶ ( 4 ) 在设备或手机模拟器上复制或粘贴文件。



- ▶ Android本来就是一个linux操作系统，所以大部分都是linux的命令，如mkdir,ls,netstat,mount,ps cp等，这里就不具体介绍了，主要介绍几个Android特有的。



## 本章小结

---

- ▶ 了解了android是如何的从无到有，从裸机硬件到下载linux内核，从内核到驱动，从驱动到java运行环境，到java的框架：android组件再到上层GUI界面层，最后到java应用程序。
- ▶ 本章最重要的、同时也是android最有趣的就是开源的android文档结构以及android带的命令行界面的调试工具adb。
- ▶ 下一讲步入android最上层的软件开发

