[BOOTIMG使用教程](http://bbs.anzhi.com/thread-4719958-1-1.html)

**背景知识**  
一、Android手机的文件系统  
Android手机的文件系统有许多存储器组成，以下是在adb shell下面的输出：  
#

1. cat/proc/mtd

复制代码  
  [**下载**](http://bbs.隔壁的.com/attachment.php?aid=MTM4Njg0Nnw0NmQyOGE3MnwxMzE5NDc5MzM0fDg4Y2VtZ1VEV3hNSFFBWjJLYWZlTDBhdWhYQTd4Q05lSWcwcFFyM05oMnA1MnVR&nothumb=yes) (40.17 KB)   
2011-7-5 12:39  
  
  
注意，不同的手机在上述存储设备的顺序可能会各不相同!一定要检查您的手机，确定在以下的操作中选择正确的设备号（mtdX，这个X的序号一定要检查清楚）。  
**根目录以及分区的定义（在Android源代码的root.c文件中定义）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 根目录: | Linux 块设备 | /挂载点/ | 文件系统 | 大小 | 描述 |
| BOOT: | /dev/mtdblock[?] | / | (RAM) | Raw | 内核、内存盘和引导配置。 |
| DATA: | /dev/mtdblock5 | /data/ | yaffs2 | 91904kb | 用户、系统配置，[软件](http://bbs.anzhi.com/forum-2-1.html" \t "_blank)配置以及软件（没有a2sd的话） |
| CACHE: | /dev/mtdblock4 | /cache/ | yaffs2 | 30720kb | OTA缓存，recovery/更新配置及临时文件夹 |
| MISC: | /dev/mtdblock[?] | N/A |  | Raw,[?]kb | （等待添加） |
| PACKAGE: | （相对于刷机包） | N/A |  |  | 刷机包的伪文件系统。 |
| RECOVERY: | /dev/mtdblock[?] | / | (RAM) | Raw,[?]kb | recovery和更新环境的内核和内存盘。类似于BOOT:。 |
| SDCARD: | /dev/mmcblk0(p1) | /sdcard/ | fat32 | 32MB-32GB | TF卡。通常刷机包就放在这里。 |
| SYSTEM: | /dev/mtdblock3 | /system/ | yaffs2 | 92160kb | 系统分区，静态且是只读的。 |
| TMP: |  | /tmp/ | (RAM) | 标准的Linux临时文件夹 | 在关机/重启时清空。 |

您首先应该要做的事情是使用您的recovery对您的[ROM](http://bbs.anzhi.com/forum-335-1.html" \t "_blank)进行备份，以免操作失误照成数据的丢失！  
二、boot和recovery映像的文件结构  
boot和recovery映像并不是一个完整的文件系统，它们是一种android自定义的文件格式，该格式包括了2K的文件头，后面紧跟着是用gzip压缩过的内核，再后面是一个ramdisk内存盘，然后紧跟着第二阶段的载入器程序（这个载入器程序是可选的，在某些映像中或许没有这部分）。  
  
/\*  
**\*\* +-----------------+   
\*\* | boot header    | 1 page  
\*\* +-----------------+  
\*\* | kernel              | n pages    
\*\* +-----------------+  
\*\* | ramdisk           | m pages    
\*\* +-----------------+  
\*\* | second stage  | o pages  
\*\* +-----------------+**  
\*\*  
\*\* n = (kernel\_size + page\_size - 1) / page\_size  
\*\* m = (ramdisk\_size + page\_size - 1) / page\_size  
\*\* o = (second\_size + page\_size - 1) / page\_size  
\*\*  
\*\* 0. all entities are page\_size aligned in flash  
\*\* 1. kernel and ramdisk are required (size != 0)  
\*\* 2. second is optional (second\_size == 0 -> no second)  
\*\* 3. load each element (kernel, ramdisk, second) at  
\*\*    the specified physical address (kernel\_addr, etc)  
\*\* 4. prepare tags at tag\_addr.  kernel\_args[] is  
\*\*    appended to the kernel commandline in the tags.  
\*\* 5. r0 = 0, r1 = MACHINE\_TYPE, r2 = tags\_addr  
\*\* 6. if second\_size != 0: jump to second\_addr  
\*\*    else: jump to kernel\_addr  
\*/  
  
boot的ramdisk映像是一个最基础的小型文件系统，它包括了初始化系统所需要的全部核心文件，例如:初始化init进程以及init.rc（可以用于设置很多系统的参数）等文件。  
以下是一个典型的ramdisk中包含的文件目录列表：  
│  default.prop  
│  init  
│  init.goldfish.rc  
│  init.rc  
│  init.swift.rc  
│  initlogo.rle               //开机第二屏[图片](http://bbs.anzhi.com/forum-98-1.html" \t "_blank)  
│  ueventd.goldfish.rc  
│  ueventd.rc  
│  ueventd.swift.rc  
│  
├─data  
├─dev  
├─proc  
├─sbin  
│      adbd  
│  
├─sys  
└─system  
  
recovery的ramdisk映像包含了一些额外的文件，例如一个叫做recovery的二进制程序，以及一些对该程序支持性的资源图片文件（当您按下home+power组合键的时候就会运行这个recovery程序）。典型的文件列表如下：  
│  default.prop  
│  init  
│  init.rc  
│  initlogo.rle              //开机第二屏图片  
│  ueventd.goldfish.rc  
│  ueventd.rc  
│  ueventd.swift.rc  
│  
├─data  
├─dev  
├─etc  
│      recovery.fstab  
│  
├─proc  
├─res  
│  │  keys  
│  │  
│  └─images  
│          icon\_clockwork.png  
│          icon\_error.png  
│          icon\_installing.png  
│          indeterminate1.png  
│          indeterminate2.png  
│          indeterminate3.png  
│          indeterminate4.png  
│          indeterminate5.png  
│          indeterminate6.png  
│          progress\_empty.png  
│          progress\_fill.png  
│  
├─sbin  
│      adbd  
│      e2fsck  
│      fix\_permissions  
│      killrecovery.sh  
│      mke2fs  
│      nandroid-md5.sh  
│      parted  
│      recovery  
│      sdparted  
│      tune2fs  
│  
├─sys  
├─system  
│  └─bin  
└─tmp  
  
  
**使用方法**工具主要语言为python，分两处版本，源文件   bootimg.py.gz (7.11 KB) 及windows下可执行文件exe   bootimg.exe.gz (2.27 MB) ，内容及用法完全一致。  
  
**运行方法：**  
      在cmd下cd到文件所在目录，输入"bootimg.exe  功能 [参数]"（用源文件的话就是bootimg.py）  
  
**目前支持以下功能：**  
        --repack-ramdisk    生成 ramdisk  
        --unpack-ramdisk   解开 ramdisk  
        --repack-bootimg   生成 bootimg (包括boot.img及recovery.img)  
        --unpack-bootimg  解开 bootimg (包括boot.img及recovery.img)  
        --unpack-updata    解开 updata.app  
        --unpack-yafffs      解开 yafffs    (包括data.img及system.img)  
**下面一一说明功能中的参数。**  
--unpack-updata [文件]  
[文件]为空时，默认使用UPDATA.APP  
解开后，会有四个文件，boot.img, recovery.img, system.img, userdata.img  
这些都是刷机时可能需要的。  
  
--unpack-bootimg [文件]  
[文件]为空时，默认使用boot.img  
解开后，会有两个文件，kernel和ramdisk.gz  
同时，注意输出，比如base, cmdline, name等等  
  
--repack-bootimg [base] [cmdline]  
[base]为空时，使用0x200000   
[cmdline]为空时，使用mem=211 console=null androidboot.hardware=qcom   
生成bootimg时，会使用kernel和ramdisk.gz(如果存在ramdisk.cpio.gz，优先使用)，生成boot.img  
更新：  
--repack-bootimg [base] [cmdline] [page\_size] [padding\_header] [padding\_kernel] [padding\_ramdisk]  
具体的参数请在注意unpack时的原始参数。  
  
--unpack-ramdisk [文件] [目录]  
[文件]为空时，使用ramdisk.gz  
[目录]为空时，使用initrd, (请保证这个目录不存在)  
输出：cpiolist.txt, initrd目录下的文件  
  
--repack-ramdisk [cpiolist]  
[cpiolist]为空时，读取cpiolist.txt  
根据cpiolist.txt生成ramdisk.cpio.gz  
cpiolist.txt格式  
1). 文件  
file 目标系统路径 当前系统路径 目标系统权限  
2). 目录  
dir 目录系统路径 目录系统权限  
3). 软链接  
slink 目录系统路径 链接路径 目标系统权限  
  
--unpack-yafffs [文件] [目录]  
[文件]为空时，使用userdata.img (呃，这个比system.img小)  
[目录]为空时，使用文件除.img外的其它内容，比如userdata.img时使用userdata  
  
**ps：这些参数就算看不懂也没关系，可以不用加参数，直接用默认的就行，如解包updata.app的话就输入bootimg --unpack-updata就好。**  
  
  
**boot.img基址 base的计算方法（转）**       如果您看这个帖子，没有耐心的话，我只能对你说，出现任何问题都是你自己的错，有点耐心继续看吧，虽然在文章最后有点邪恶，大家原谅一下！我在工具包中给了一个参数，对于我的GT540肯定没有什么问题，但是对于其他手机我不敢保证，所以可能还需您自己计算，或者使用版区前辈得到的数值！如果您做完之后刷回到手机之后会卡在开机界面，屏幕一闪一闪啊黑屏啊之类的症状层出不穷，普遍有两种可能。  
       一、boot.img是加过密的。各位手机出产的公司，在做ROM的时候，以他们独自的加密算法，对整个文件进行CRC验算，然后将验算值添加到boot.img的最末尾。刷机的时候，手机的硬件BIOS就像一名尽职的士兵，对此进行验算，对不上口令的，那么对不起，请君离开，不离开我报警了~~呃，报不了警，那反正我不让你进门~~  
       二、boot.img还有一个重要的参数，基址 base,用于告诉手机从哪个地址开始，是准备给内存盘的入口，哪个地址是给kernel的入口。如果你对不上号，对不起，不能非法入室的。             
如果是第一种，那只能望风而逃，罢手了。   
     各种查阅，终于发现在ROM的boardconfig.h中存在地址偏移的define。虽然本人手机的ROM中不存在这文件，但各android系统这个偏移是通用的，一般没吃错药不会去改。  
偏移DEFINE如下：

1. #define PHYSICAL\_DRAM\_BASE   0x00200000<!--IWMS\_AD\_BEGIN-->   //基址，各不相同，我们的目的是为了获得这个基址，但由于没有ROM源代码，这貌似是不可能的。<!--IWMS\_AD\_END-->#define KERNEL\_ADDR          (PHYSICAL\_DRAM\_BASE + 0x00008000)#define RAMDISK\_ADDR         (PHYSICAL\_DRAM\_BASE + 0x01000000)#define TAGS\_ADDR            (PHYSICAL\_DRAM\_BASE + 0x00000100)#define NEWTAGS\_ADDR         (PHYSICAL\_DRAM\_BASE + 0x00004000)

复制代码  
还记得WIKI中有这句话吧？**For Nexus One** : Add *--base 0x20000000* to mkbootimg command-line.  翻译一下，如果是nexus one手机，那么需要在mkbootimg命令中加入 --base 0x20000000。。现在是不是很熟了？原来基址写在这里，mkbootimg会自动为你加入偏移量，并写入boot.img。。  
好吧，我们现在需要知道，这些值是在哪个位置的。  
为什么需要呢？在第上面，我们已经知道了偏移量了，那么如果我们再从官方ROM的boot.img中获取 kernel\_addr，再用kernel\_addr -偏移量0x00008000,不就可以得出基址了吗？  
     没辙，需要知道bootimg工具把它们写到哪了，只能查看mkbootimg的源代码(幸好bootimg作为一个典型的LINUX工具，是开源的。若换成WIN，哥就真悲情了)。  
     各种查找，找到bootimg.c文件，看吧。  
     header + padding + kernel + padding + ramdisk + padding + ...(padding是补全，还记得上面所说的，一个盒子装10张卡片，11张卡片需要几个盒子的事情吧？第二个盒子由于只放了一张卡片，所以需要9张空白卡片来填充位置，即padding）  
4 \* 2, magic，固定为"ANDROID!"  
4 \* 1, kernel长度，小端unsigned类型  
4 \* 1, kernel地址，应为base + 0x00008000   
4 \* 1, ramdisk长度，小端unsigned  
4 \* 1, ramdisk地址，应为base + 0x01000000  
4 \* 1, second stage长度，小端unsigned，为0  
4 \* 1, second stage地址，应为base + 0x00f00000  
4 \* 1, tags地址，应为base + 0x00000100  
4 \* 1, page大小，小端unsigned, 为2048或者4096  
4 \* 2, 未使用，固定为0x00  
4 \* 4, 板子名字，一般为空  
4 \* 128, 内核命令参数，一大串  
4 \* 8, id,不知道啥玩意，0x00