

Rockchip

recovery 开发指南

发布版本:**1.0.2**

日期:**2018.10**

免责声明

本文档按“现状”提供，福州瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自所有者所有。

版权所有 © 2018 福州瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

福州瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd.

地址：福建省福州市铜盘路软件园 A 区 18 号

网址：www.rock-chips.com

客户服务电话：+86-591-83991906

客户服务传真：+86-591-83951833

客户服务邮箱：www.rock-chips.com

前言

概述

本文档主要介绍 Rockchip 处理器 OTA 升级时的 recovery 开发流程以及技术细节。
本文中详细介绍了该方案的开发过程以及注意事项。

产品版本

芯片名称	内核版本
RK3308	4.4

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

日期	版本	作者	修改说明
2018.09.18	1.0.0	Chad.ma	初始版本
2018.10.16	1.0.1	Chad.ma	增加恢复出厂模式一节
2018.11.06	1.0.2	Chad.ma	增加 SD 卡启动盘升级一章 附录章节增加常见问题汇总

目录

1 OTA 升级.....	6
1.1 概述.....	6
1.2 编译.....	6
1.3 升级流程.....	7
1.4 恢复出厂模式.....	8
1.5 注意事项.....	9
2 运行调试.....	10
2.1 Recovery 模式中 log 的查看.....	10
3 SD 卡制作启动盘升级.....	11
3.1 Recovery 支持 SD 卡启动升级.....	11
3.2 制作 SD 卡启动盘.....	11
4 附录.....	14
4.1 misc 分区说明.....	14
4.2 Recovery 不同场景下的使用.....	15
4.3 常见问题汇总.....	17

插图目录

图 1 - 1 recovery 升级流程图..... 8

图 2 - 1 recovery 中创建隐藏文件..... 10

图 3 - 1 制作 SD 卡启动盘.....11

图 4 - 1 misc 分区结构内容.....14

图 4 - 2 misc.img 文件内容.....15

图 4 - 3 Recovery Makefile 相关.....18

表格目录

1 OTA 升级

1.1 概述

OTA (Over-the-Air) 即空间下载技术。OTA 升级是 Android 系统提供的标准软件升级方式。它功能强大, 可以无损失升级系统, 主要通过网络, 例如 WIFI、3G/4G 自动下载 OTA 升级包、自动升级, 也支持通过下载 OTA 升级包到 SD 卡/U 盘升级, OTA 的升级包非常的小, 一般几 M 到十几 M。

本文主要介绍了使用 OTA 技术升级时, 本地升级程序 recovery 执行升级的流程及技术细节, 方便用户在开发过程中了解升级的过程及注意事项。

1.2 编译

rootfs 主系统

rootfs 要打开 recoverySystem 的支持, configs 文件中把 BR2_PACKAGE_RECOVERYSYSTEM=y 选上。

或者配置 BR2_PACKAGE_UPDATE=y。

注意:

目前主系统中实现调用升级功能的有两套代码, recoverySystem 与 update, 二者使用相同的参数, 均可实现进入 recovery 模式, 进行 OTA 的升级。

后续将统一使用 update。

Recovery

系统根目录下执行

```
$ ./build.sh recovery
```

会生成文件 buildroot/output/rockchip_rk3308_recovery/images/recovery.img。

```
$ ./mkfirmware.sh
```

会将生成的固件拷贝至 rockdev/目录下。

与 recovery 相关的主要源码路径:

external/recovery/ : 主要生成 recovery 二进制 bin 程序, recovery 模式下的关键程序。

external/rkupdate/: 主要生成 rkupdate 二进制 bin 程序, 解析 update.img 固件中各个分区数据, 并执行对各分区执行升级的关键程序。

若有修改以上两个目录中的源码文件之后的编译方法:

1. Source envsetup.sh
2. 选择某一平台的 recovery 配置
3. make recovery-rebuild / make rkupdate-rebuild
4. ./build.sh recovery
5. ./mkfirmware.sh
6. 烧写 recovery.img

1.3 升级流程

升级固件准备

使用 rockchip 固件打包工具生成的 update.img。

固件的打包可参考[《Rockchip Linux 升级固件打包指南》](#)。

升级过程

- 将升级固件 update.img 放在 SD 卡或 U 盘根目录或者设备的/userdata 目录下。
- Normal 系统下执行升级程序 **recoverySystem ota /xxx/update.img**，设备会进入 recovery 模式，并进行升级。

可使用的路径如下：

U 盘的挂载路径：/udisk

sdcard 的挂载路径：/mnt/sdcard/ 或/sdcard

flash 的挂载路径：/userdata/

- 升级成功后会 reboot 到正常的 normal 系统。

升级流程图

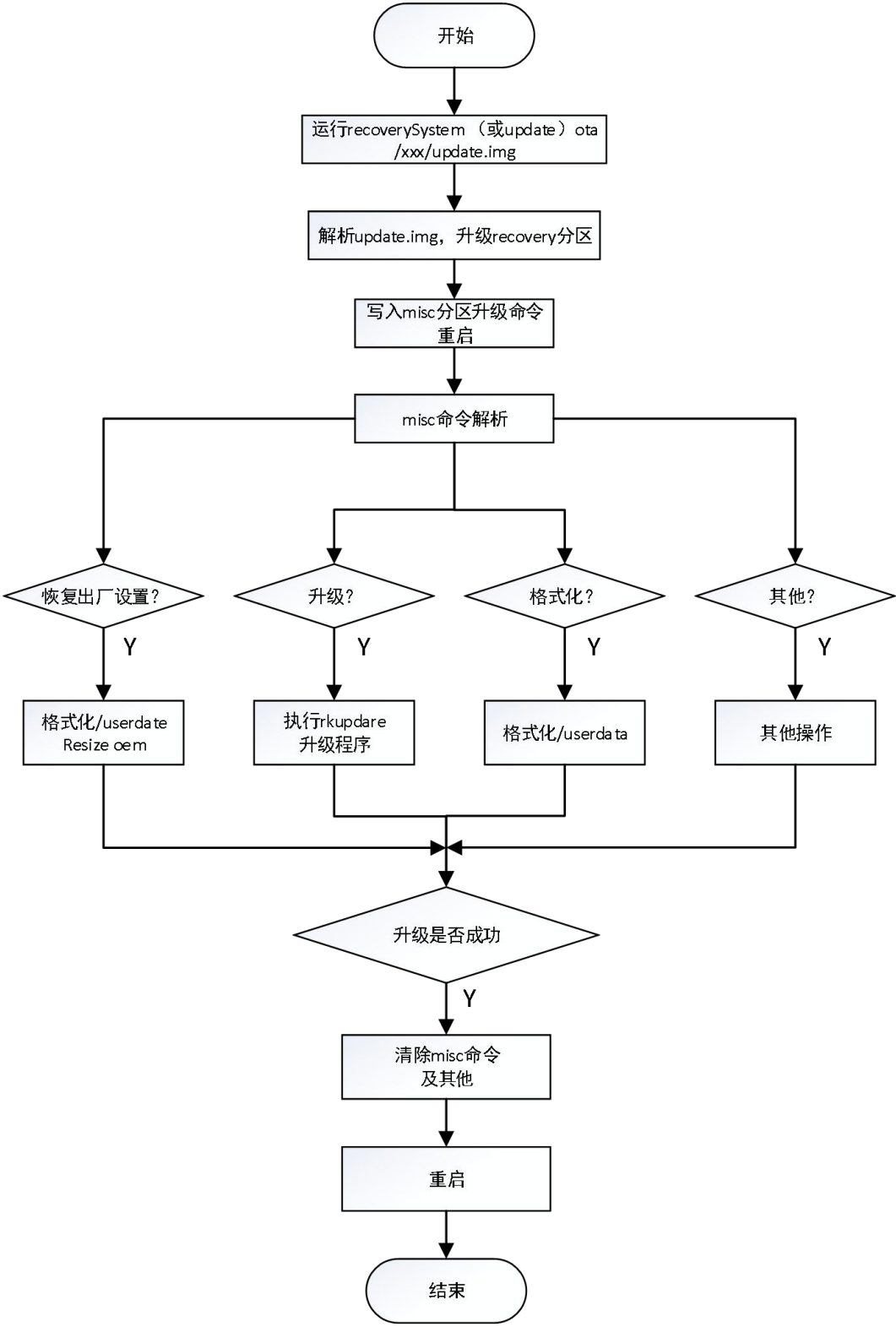


图 1 - 1 recovery 升级流程图

升级的详细流程，开发者可根据流程图，阅读升级方案的源码，这里不做进一步的展开说明。

1.4 恢复出厂模式

我们把可以读写的配置文件保存在 **userdata** 分区，出厂固件会默认一些配置参数，用户使用一段时间后会生成或修改配置文件，有时用户需要清除这些数据，我们就需要恢复到出厂配置。

- 方案：打包固件时，生成 userdata.img，用户请求恢复出厂配置时，将 userdata 分区格式化。
- SDK 实现：
功能键 RECOVERY + VOLUMEUP 触发恢复出厂配置，代码请参考：
buildroot/board/rockchip/rk3308/fs-overlay/etc/input-event-daemon.conf
board/rockchip/rk3308/fs-overlay/usr/sbin/factory_reset_cfg

直接运行 **recoverySystem**（或 **update**）后面不加任何参数或者加 **factory/reset** 参数均可进入 **recovery** 后恢复出厂配置。

1.5 注意事项

- 打包 update.img 固件时需要注意，升级固件不一定要全分区升级，可修改 package-file 文件，将不要升级的分区去掉，这样可以减少升级包（update.img）的大小。
- package-file 中 **recovery.img** 如果打包进去的话，不会在 **recovery** 模式中升级，为了预防升级 recovery.img 过程中掉电导致后面其他分区无法正常升级的问题，该分区升级放在 normal 系统下升级，即，执行 recoverySystem 命令时会先检测 update.img 升级包中是否有打包 recovery.img，若有则升级 recovery 分区，再进入 recovery 模式升级其他分区固件。
- **misc** 分区不建议打包进 **update.img** 中，即使有打包进去，也会在升级程序中加载判断到而忽略该分区，原因是：即使升级了 misc 分区，升级成功后 recovery 程序仍会清空 misc 分区中所有的命令及参数，从而导致预想的结果达不到。
- 如果将 **update.img** 升级包放置在 **flash** 中的 **userdata** 分区，则需要保证 **package-file** 中 **不包括 userdata.img 被打包进去**，原因是可能会导致文件系统的损坏，升级成功后可能使 oem 或 userdata 分区 mount 不成功。若从 SD 卡或 U 盘升级时，可以打包 userdata.img，从而对 userdata 分区进行升级。升级完成后会对 userdata 分区重新 resize 操作。

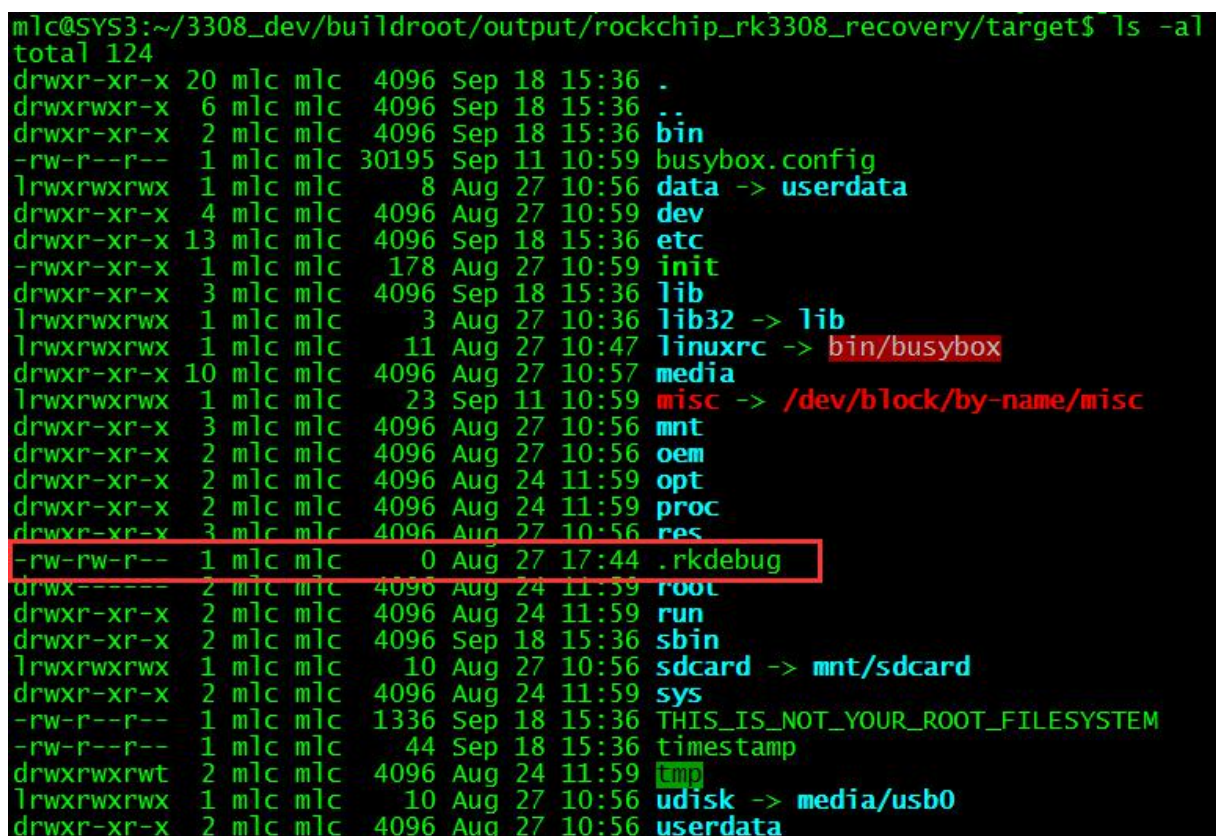
2 运行调试

2.1 Recovery 模式中 log 的查看

- buildroot/output/rockchip_rk3308_recovery/target 目录下

```
$ touch .rkdebug
```

创建这个隐藏文件，可将 recovery 模式中升级的 log 在串口中打印出来。



```
m1c@SYS3:~/3308_dev/buildroot/output/rockchip_rk3308_recovery/target$ ls -al
total 124
drwxr-xr-x 20 m1c m1c 4096 Sep 18 15:36 .
drwxrwxr-x 6 m1c m1c 4096 Sep 18 15:36 ..
drwxr-xr-x 2 m1c m1c 4096 Sep 18 15:36 bin
-rw-r--r-- 1 m1c m1c 30195 Sep 11 10:59 busybox.config
lrwxrwxrwx 1 m1c m1c 8 Aug 27 10:56 data -> userdata
drwxr-xr-x 4 m1c m1c 4096 Aug 27 10:59 dev
drwxr-xr-x 13 m1c m1c 4096 Sep 18 15:36 etc
-rwxr-xr-x 1 m1c m1c 178 Aug 27 10:59 init
drwxr-xr-x 3 m1c m1c 4096 Sep 18 15:36 lib
lrwxrwxrwx 1 m1c m1c 3 Aug 27 10:36 lib32 -> lib
lrwxrwxrwx 1 m1c m1c 11 Aug 27 10:47 linuxrc -> bin/busybox
drwxr-xr-x 10 m1c m1c 4096 Aug 27 10:57 media
lrwxrwxrwx 1 m1c m1c 23 Sep 11 10:59 misc -> /dev/block/by-name/misc
drwxr-xr-x 3 m1c m1c 4096 Aug 27 10:56 mnt
drwxr-xr-x 2 m1c m1c 4096 Aug 27 10:56 oem
drwxr-xr-x 2 m1c m1c 4096 Aug 24 11:59 opt
drwxr-xr-x 2 m1c m1c 4096 Aug 24 11:59 proc
drwxr-xr-x 3 m1c m1c 4096 Aug 27 10:56 res
-rw-rw-r-- 1 m1c m1c 0 Aug 27 17:44 .rkdebug
drwx----- 2 m1c m1c 4096 Aug 24 11:59 root
drwxr-xr-x 2 m1c m1c 4096 Aug 24 11:59 run
drwxr-xr-x 2 m1c m1c 4096 Sep 18 15:36 sbin
lrwxrwxrwx 1 m1c m1c 10 Aug 27 10:56 sdcard -> mnt/sdcard
drwxr-xr-x 2 m1c m1c 4096 Aug 24 11:59 sys
-rw-r--r-- 1 m1c m1c 1336 Sep 18 15:36 THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
-rw-r--r-- 1 m1c m1c 44 Sep 18 15:36 timestamp
drwxrwxrwt 2 m1c m1c 4096 Aug 24 11:59 tmp
lrwxrwxrwx 1 m1c m1c 10 Aug 27 10:56 udisk -> media/usb0
drwxr-xr-x 2 m1c m1c 4096 Aug 27 10:56 userdata
```

图 2-1 recovery 中创建隐藏文件

- 通过查看 userdata/recovery/Log 文件查看

升级之后，在设备 userdata/recovery 目录中查看 log 文件。

```
$ cat userdata/recovery/Log
```

3 SD 卡制作启动盘升级

本章节主要为了解决使用 SD 卡启动，进行裸片升级的需求，详细描述 SD 卡启动盘的制作及相关升级的问题。

3.1 Recovery 支持 SD 卡启动升级

1. 首先确保 external/recovery 及 external/rkupdate 目录中代码已经添加了支持 SD 卡启动升级的代码。

如果没有添加这部分代码，需要打上以下补丁：

external/recovery 目录下：

```
0001-recovery-add-support-sdcard-boot-update.patch
```

external/rkupdate 目录下：

```
0001-rkupdate-add-support-sdcard-boot-update.patch
```

2. 重新编译 recovery 与 rkupdate

```
$ make recovery-rebuild
```

```
$ make rkupdate-rebuild
```

3. 编译 recovery

```
$ ./build.sh recovery
```

4. 生成固件

```
$ ./build.sh
```

3.2 制作 SD 卡启动盘

如图 3-1 所示，使用工程目录中 tools\windows\SDDiskTool 中的 SD 卡启动盘升级制作工具制作 SD 卡启动盘。

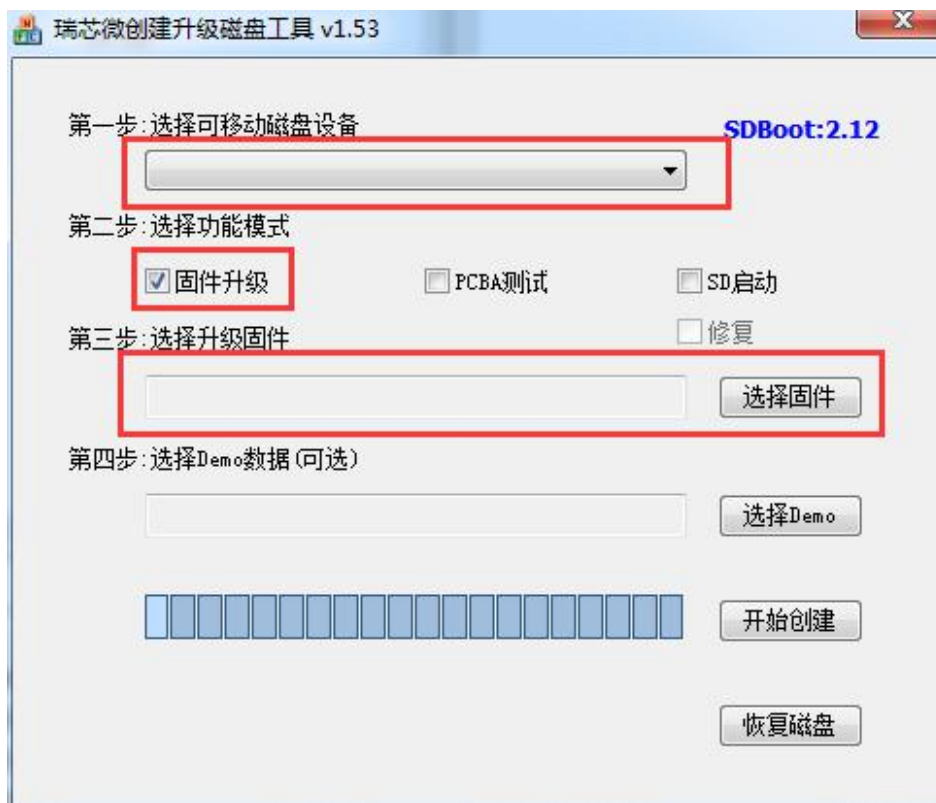


图 3-1 制作 SD 卡启动盘

选择固件中选择打包好的 **update.img** 文件。

所有准备工作完成后，点击开始创建按钮，如果创建成功，会弹窗提示。

此时 SD 卡中根目录会存在两个文件，其中选择升级的固件 **update.img**，会被命名为 **sdupdate.img**。

所有准备工作做好后，设备中插入 SD 卡，并重新上电。

Log 中如果出现下面内容，说明 SD 卡启动设备成功。

```
U-Boot 2017.09-g1bee468 (Oct 11 2018 - 16:53:06 +0800) V1.000
Model: USM-110 a102-1
Board:Advantech usm110_rk3288 Board,HW version:0
DRAM:  2 GiB
Relocation Offset is: 7ff5a000
PMIC:  RK808
vdd_arm 1100000 uV
vdd_gpu 1100000 uV
vcc_io 3300000 uV
regulator(LDO_REG2) init 3300000 uV
regulator(LDO_REG3) init 1100000 uV
regulator(LDO_REG4) init 1800000 uV
regulator(LDO_REG5) init 3300000 uV
regulator(LDO_REG6) init 1100000 uV
regulator(LDO_REG7) init 1800000 uV
regulator(LDO_REG8) init 1800000 uV
MMC:  dwmmc@ff0c0000: 1, dwmmc@ff0f0000: 0
SF: Detected w25q32bv with page size 256 Bytes, erase size 4 KiB, total 4 MiB
*** Warning - bad CRC, using default environment
In:  serial
Out: serial
Err: serial
switch to partitions #0, OK
mmc1 is current device
do_rking_test found IDB in SDcard
Boot from SDcard
enter Recovery mode!
SF: Detected w25q32bv with page size 256 Bytes, erase size 4 KiB, total 4 MiB
Skipped ethaddr assignment due to invalid,using default!
Net:  No ethernet found.
Hit any key to stop autoboot:  0
ANDROID: reboot reason: "recovery"
FDT load addr 0x10f00000 size 263 KiB
Booting kernel at 0x3575c70 with fdt at 42cf470...
```

若串口 log 中打印如下的 log，说明 SD 卡启动进入了 **recovery** 固件对裸片设备的升级过程。

firmware update will from SDCARD.

```
is_sdcard_update out
sdupdate_package = /mnt/sdcard/sdupdate.img
Command: "/usr/bin/recovery"
>>>sdboot update will update from /mnt/sdcard/sdupdate.img
start with main.
librkupdate_Start to upgrade firmware...
librkupdate_INFO:is emmc devices...
librkupdate_INFO:CRKUsbComm-->is emmc.
librkupdate_INFO:CRKUsbComm-->/dev/vendor_storage=24
librkupdate_INFO:CRKUsbComm-->/dev/mmcblk2=26
librkupdate_uid: 52 4F 43 4B 43 48 49 50 45 EB C3 5B 6C 87 6F 6D
87 DA 4E 76 A1 FB 38 28 57 98 5C 8F B3 23
librkupdate_Get FlashInfo...
librkupdate_INFO:FlashInfo: 00 F8 E8 00 01 00 00 00 9C 51 E6
GetFlashInfo: 266 info.uiFlashSize = 15267840 total uiBlockNum = 30535680
GetFlashInfo: 267 FlashSize = 14910 MB
##### update bootloader start#####
librkupdate_IDBlock Preparing...
##### IDBlock Preparing...
librkupdate_ERROR:PrepareIDB-->New IDblock offset=0 1 2 3 4 .
librkupdate_IDBlock Writing...
##### IDBlock Writing...
librkupdate_INFO:MakeIDBlockData in
librkupdate_INFO:MakeIDBlockData out
librkupdate_INFO:WriteIDBlock in
librkupdate_INFO:-----
librkupdate_dwSectorNum=180
librkupdate_uiTotal=92160
librkupdate_INFO:WriteIDBlock out
##### update bootloader Suceess#####
librkupdate_##### RKA_Gpt_Download #####
librkupdate_##### Download trust ... #####
librkupdate_INFO:Start to download trust,offset=0x6000,size=4194304
librkupdate_##### Download uboot ... #####
librkupdate_INFO:Start to download uboot,offset=0x4000,size=4194304
librkupdate_##### Download misc ... #####
librkupdate_INFO:Start to download misc,offset=0x8000,size=49152
librkupdate_##### Download boot ... #####
librkupdate_INFO:Start to download boot,offset=0xa000,size=7774208
librkupdate_##### Download rootfs ... #####
librkupdate_INFO:Start to download rootfs,offset=0x5a000,size=2548436992
.....
```


4 附录

4.1 misc 分区说明

misc 其实是英文 **miscellaneous** 的前四个字母，杂项、混合体、大杂烩的意思。

misc 分区概念来源于 **Android** 系统，**Linux** 系统中常用来作为系统升级时或者恢复出厂设置时使用。

misc 分区的读写：misc 分区在以下情况下会被读写。

1) **Uboot**: 设备加电启动时，首先启动 **uboot**，在 **uboot** 中会读取 **misc** 分区的内容。根据 **misc** 分区中 **command** 命令内容决定是进入正常系统还是 **recovery** 模式。

Command 为 **boot-recovery**，则进入 **recovery** 模式。

Command 为空，则进入正常系统。

2) **Recovery**: 在设备进入 **recovery** 模式中，可以读取 **misc** 分区中 **recovery** 部分的内容，从而执行不同的动作。或升级或擦除用户数据等等。

Misc 分区的结构及内容：

Misc 分区的结构组成详见下图。

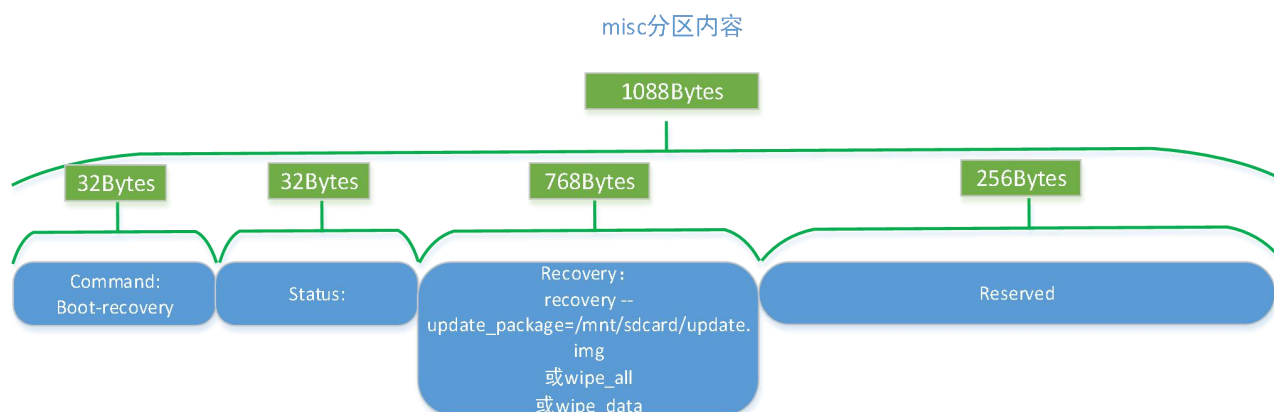


图 4 - 1 misc 分区结构内容

下面以 3308 平台使用的 **misc** 分区为例，使用 **winhex** 或 **ultraEdit** 等工具，以二进制形式打开 **misc.img** 文件，在距文件开始位置偏移 16K（16384 Byte）字节位置处开始，存放 **BootLoader Msg** 结构体的内容。

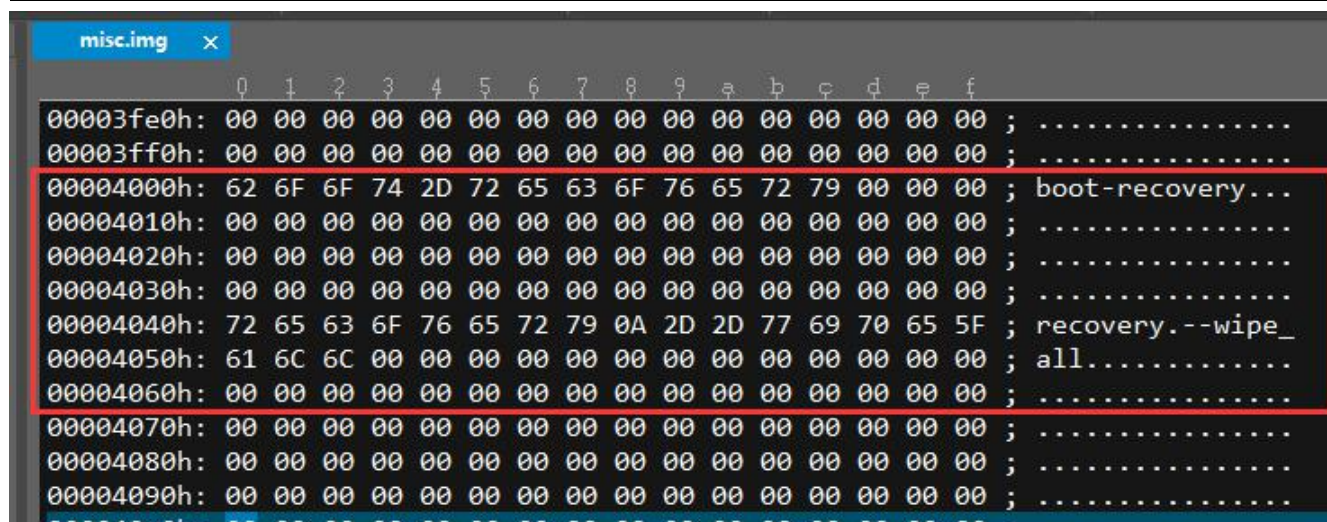


图 4 - 2 misc.img 文件内容

Recovery 中支持的命令部分，可参考 external/recovery/recovery.c 中 OPTIONS 结构中内容。

4.2 Recovery 不同场景下的使用

- 第一次开机

烧写过 misc.img、recovery.img 的机器会进入第一次开机流程。

串口有如下 log 打印：

```
I:Boot command: boot-recovery
I:Got arguments from boot message
Command: "recovery" "--wipe_all"
format '/dev/block/by-name/userdata' to ext2 filesystem
executing '/sbin/mke2fs'
executed '/sbin/mke2fs' done
executed '/sbin/mke2fs' return 0
executing '/sbin/e2fsck'
e2fsck 1.43.9 (8-Feb-2018)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/block/by-name/userdata: 11/2304 files (0.0% non-contiguous), 82/2299 blocks
executed '/sbin/e2fsck' done
executed '/sbin/e2fsck' return 0
executing '/usr/sbin/e2fsck'
e2fsck 1.43.9 (8-Feb-2018)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
```



```

/dev/block/by-name/oem: 18/2448 files (0.0% non-contiguous), 513/16384 blocks
executed '/usr/sbin/e2fsck' done
executed '/usr/sbin/e2fsck' return 1
executing '/usr/sbin/resize2fs'
resize2fs 1.43.9 (8-Feb-2018)
The filesystem is already 16384 (1k) blocks long. Nothing to do!
executed '/usr/sbin/resize2fs' done
executed '/usr/sbin/resize2fs' return 0

```

- 恢复出厂设置

命令行运行 recoverySystem 程序，机器会进入 recovery，并进行格式化，格式化完成之后会自动进入 normal system。

```
$ recoverySystem
```

串口会有如下 log 打印：

```

I:Boot command: boot-recovery
I:Got arguments from boot message
Command: "recovery" "--wipe_data"
format '/dev/block/by-name/userdata' to ext2 filesystem
executing '/sbin/mke2fs'
[ 4.692437] vendor storage:20160801 ret = -1
[ 6.030842] phy phy-ff008000.syscon:usb2-phy@100.0: charger =
USB_SDP_CHARGER
[ 10.891460] random: nonblocking pool is initialized
executed '/sbin/mke2fs' done
executed '/sbin/mke2fs' return 0
executing '/sbin/e2fsck'
e2fsck 1.43.9 (8-Feb-2018)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/block/by-name/userdata: 11/2304 files (0.0% non-contiguous), 82/2299 blocks
executed '/sbin/e2fsck' done
executed '/sbin/e2fsck' return 0
[ 11.141033] cpu0 limit freq=816000 min=816000 max=816000
[ 11.141773] rkndand_shutdown...
[ 11.142139] nand th quited
[ 11.142484] rk_ftl_de_init 0
[ 11.150824] rkndand_shutdown:OK
[ 11.152054] reboot: Restarting system

```

- 升级

命令行运行 recoverySystem ota /xxx/update.img，机器会进入 recovery，并进行升级。

```
$ recoverySystem ota /udisk/update.img
```

这里以从 U 盘升级为例。

串口可能打印的 log 如下：

```
I:Boot command: boot-recovery
I:Got arguments from boot message
Command: "recovery" "--update_package=/udisk/update.img"
. . . .
librkupdate_ui_print = parameter writing....
##### RKA_Gpt_Download #####
librkupdate_##### Download trust ... #####
ui_print = trust writing....
librkupdate_##### Download uboot ... #####
ui_print = uboot writing....
librkupdate_##### Download boot ... #####
librkupdate_ui_print = boot writing....
librkupdate_##### Download rootfs ... #####
librkupdate_ui_print = rootfs writing....
librkupdate_##### Ignore recovery download #####
librkupdate_##### Download oem ... #####
ui_print = oem writing....
librkupdate_##### Download userdata:grow ... #####
ui_print = parameter checking....
ui_print = trust checking....
ui_print = uboot checking....
ui_print = boot checking....
ui_print = rootfs checking....
ui_print = oem checking....
ui_print = userdata:grow checking....
librkupdate_##### Ignore recovery Check #####
librkupdate_Finish to upgrade firmware.
[ 38.655387] EXT2-fs (rk NAND0p8): warning: mounting unchecked fs, running
e2fsck is recommended
[ 38.663735] cpu0 limit freq=816000 min=816000 max=816000
[ 38.664552] rk NAND shutdown...
[ 38.664865] NAND th quited
[ 38.665127] rk_ftl_deinit 0
[ 38.676436] rk NAND shutdown:OK
[ 38.678078] reboot: Restarting system
```

4.3 常见问题汇总

3.3.1 “cannot find/open a drm device”

常见在非 RK3308 平台，进入 recovery 模式后串口打印如下 log：

```
we are in recovery, skip init oem/userdata
```

```
start debug recovery...
```

```
Starting recovery on Fri Jan 18 09:19:51 2013
```

```
failed to read font: res=-1, fall back to the compiled-in font
```

```
Starting network: cannot find/open a drm device: No such file or directory
```

遇到此情况时，解决方法是：接上设备支持的显示屏，或者 HDMI 设备。

原因分析：从提示的 log 看是找不到或者打开一个 drm 设备失败。因为，默认非 RK3308 平台 recovery 程序的编译是打开支持 UI 显示的，如果进入 recovery 模式之后，打开显示设备失败，则会导致 recovery 执行失败。

如果用户想在非 RK3308 平台上（如 RK3399、RK3288 等）支持不带屏显示的 recovery 升级功能，可以按如下方式操作：

1、编辑修改 recovery 的 mk 文件

```
vim buildroot/package/rockchip/recovery/? recovery.mk
```

```
ifeq ($(BR2_PACKAGE_RK3308),y)
    TARGET_MAKE_ENV += RecoveryNoUi=true
else
    RECOVERY_BUILD_OPTS += -lz -lpng -ldrm
    RECOVERY_DEPENDENCIES += libzlib libpng libdrm
Endif
```

参照此处写法，根据自身芯片平台，定义 RecoveryNoUi=true。



```
PROJECT_DIR := $(shell pwd)
CC = gcc
PROM = recovery
OBJ = recovery.o \
    default_recovery_ui.o \
    rktools.o \
    roots.o \
    bootloader.o \
    safe_iop.o \
    strlcpy.o \
    strlcat.o \
    rkupdate.o \
    mtdutils/mounts.o \
    mtdutils/mtdutils.o \
    mtdutils/rk29.o \
    minzip/DirUtil.o

ifdef RecoveryNoUi
OBJ += noui.o
else
OBJ += ui.o \
    minzip/Hash.o \
    minzip/Inlines.o \
    minzip/SysUtil.o \
    minzip/Zip.o \
    minui/events.o \
    minui/graphics.o \
    minui/resources.o \
    minui/graphics_drm.o
endif
```

图 4 - 3 Recovery Makefile 相关

这样的话，external/recovery/Makefile 中就不会编译与显示相关的代码，如上图 4-3 所示。

2、重新编译 recovery

```
make recovery-rebuild
```

3、重新生成 recovery 固件

```
./build.sh recovery
```

4、生成固件。

```
./mkfirmware.sh
```

3.3.2 “E:Can't open /dev/block/by-name/misc”

在使用 recovery 功能升级固件过程中，如果遇到使用 emmc flash 时，串口有如下的 log:

```
start debug recovery...
Starting recovery on Thu Jan  1 00:00:01 1970

recovery filesystem table
=====
 0 (null) /tmp ramdisk (null) (null) (null)
 1 /dev/root / ext2 rw,noauto 0 1
 2 proc /proc proc defaults 0 0
 3 devpts /dev/pts devpts defaults,gid=5,mode=620 0 0
 4 tmpfs /dev/shm tmpfs mode=0777 0 0
 5 tmpfs /tmp tmpfs mode=1777 0 0
 6 tmpfs /run tmpfs mode=0755,nosuid,nodev 0 0
 7 sysfs /sys sysfs defaults 0 0
Starting network:  8 debug /sys/kernel/debug debugfs defaults 0 0
 9 pstore /sys/fs/pstore pstore defaults 0 0
10 /dev/block/by-name/misc /misc emmc defaults 0 0
11 /dev/block/by-name/oem /oem ext2 defaults 0 2
12 /dev/block/by-name/userdata /userdata ext2 defaults 0 2

emmc_point is
sd_point is (null)
sd_point_2 is (null)
buf = /dev/block/by-name/misc
### get mount_ponit = /dev/block/by-name/misc ###
===path = /misc, v-mount_point = /misc ===
E:Can't open /dev/block/by-name/misc
(No such file or directory)
No link to path!!!
===path = /userdata/recovery/command, v-mount_point = /userdata ===
E:failed to mount /userdata (No such file or directory)
E:Can't mount /userdata/recovery/command
buf = /dev/block/by-name/misc
```

```
### get mount_ponit = /dev/block/by-name/misc ###
===path = /misc, v-mount_point = /misc ===
E:Can't open /dev/block/by-name/misc
(No such file or directory)
Command: "/usr/bin/recovery"
```

遇到此情况时，解决方法是：打上如下的补丁。

原因分析：从提示的 log 看是进入 recovery 模式之后打开 misc 分区失败。由于 misc 分区是 udev 异步初始化完成之后才会软链到 by-name 相应分区上的，mmc 识别是异步的，如果在 recovery 启动时候 mmc 还没有初始化完成，则 recovery 打开 msic 的 by-name 就会失败。

Recovery/bootloader.c 中给函数 get_bootloader_message_block 加上 wait_for_device 等待的操作即可。

```
diff --git a/bootloader.c b/bootloader.c
index fbf01ab2..7681507c 100644
--- a/bootloader.c
+++ b/bootloader.c
@@ -30,7 +30,7 @@ static int set_bootloader_message_block(const struct
bootloader_message *in, con

    int get_bootloader_message(struct bootloader_message *out) {
        Volume* v = volume_for_path("/misc");
-
+
        if(!v) return -1;
        if (strcmp(v->fs_type, "mtd") == 0) {
            return get_bootloader_message_mtd(out, v);
@@ -133,9 +133,26 @@ static int set_bootloader_message_mtd(const struct
bootloader_message *in,
    // -----
    // for misc partitions on block devices
    // -----
+static void wait_for_device(const char* fn) {
+    int tries = 0;
+    int ret;
+    struct stat buf;
+    do {
+        ++tries;
+        ret = stat(fn, &buf);
+        if (ret) {
+            printf("stat %s try %d: %s\n", fn, tries, strerror(errno));
+            sleep(1);
+        }
+    } while (ret && tries < 10);
+    if (ret) {
+        printf("failed to stat %s\n", fn);
```

```
+    }
+}

static int get_bootloader_message_block(struct bootloader_message *out,
                                       const Volume* v) {
+    wait_for_device(v->device);
    FILE* f = fopen(v->device, "rb");
    if (f == NULL) {
        LOGE("Can't open %s\n(%s)\n", v->device, strerror(errno));
    }
    --
```