android recovery模式及ROM制作

1 总述

为了方便客户日后的固件升级，本周研究了一下android的recovery模式。网上有不少这类的资料，但都比较繁杂，没有一个系统的介绍与认识，在这里将网上所找到的和自己通过查阅代码所掌握的东西整理出来，给大家一个参考！

2 Android启动过程

在这里有必要理一下android的启动过程：

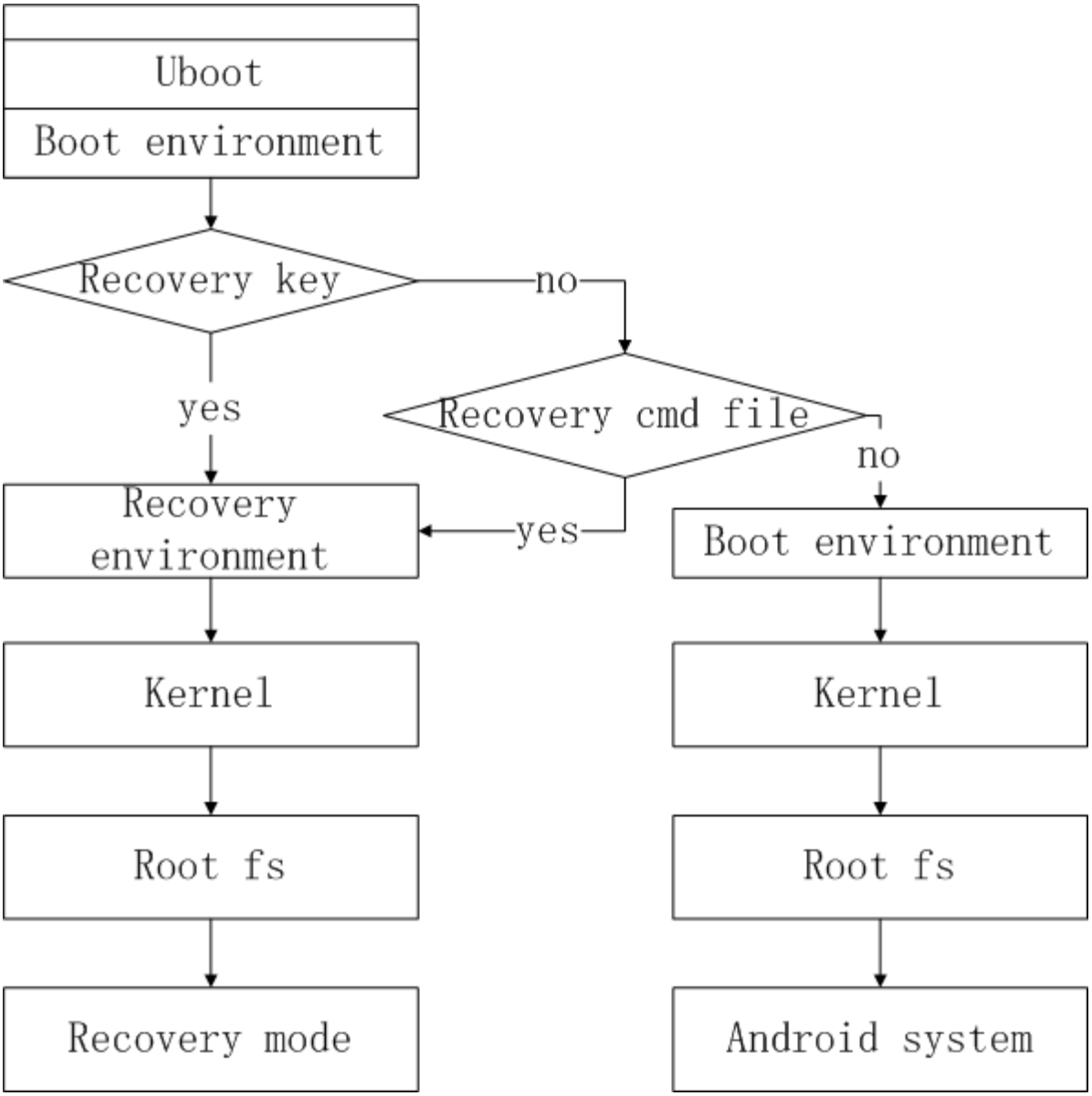

图1 android启动过程

系统上电之后，首先是完成一系列的初始化过程，如cpu、串口、中断、timer、DDR等等硬件设备，然后接着加载boot default environmet，为后面内核的加载作好准备。在一些系统启动必要的初始完成之后，将判断是

否要进入recovery模式，从图1中可以看出，进入recovery模式有两种情况。一种是检测到有组合按键按下时；另一种是检测到cache/recovery目录下有command这个文件，这个文件有内容有它特定的格式，将在后面讲到。

3 Uboot启动

下面来看看uboot中lib_arm/board.c这个文件中的start_armboot这个函数，这个函数在start.s这个汇编文件中完成堆栈等一些基础动作之后被调用，进入到c的代码中，start_armboot部分代码如下：

```
void start_armboot (void)
{
	.
	.
	.
	for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr) {
		if ((*init_fnc_ptr)() != 0) {
			hang ();
		}
#ifdef CONFIG_ANDROID_RECOVERY
	check_recovery_mode();
#endif
	/* main_loop() can return to retry autoboot, if so just run it again. */
	for (;;) {
		main_loop ();
	}
}
```

init_sequence是一个函数结构体指针，里面存放的是一些必备的初始化函数，其代码如下：

```
init_fnc_t *init_sequence[] = {
#if defined(CONFIG_ARCH_CPU_INIT)
	arch_cpu_init,		/* basic arch cpu dependent setup */
#endif
	board_init,		/* basic board dependent setup */
#if defined(CONFIG_USE_IRQ)
	interrupt_init,		/* set up exceptions */
#endif
	timer_init,		/* initialize timer */
	env_init,		/* initialize environment */
	init_baudrate,		/* initialze baudrate settings */
	serial_init,		/* serial communications setup */
	console_init_f,		/* stage 1 init of console */
	display_banner,		/* say that we are here */
#if defined(CONFIG_DISPLAY_CPUINFO)
	print_cpuinfo,		/* display cpu info (and speed) */
#endif
#if defined(CONFIG_DISPLAY_BOARDINFO)
	checkboard,		/* display board info */
#endif
#if defined(CONFIG_HARD_I2C) || defined(CONFIG_SOFT_I2C)
	init_func_i2c,
#endif
	dram_init,		/* configure available RAM banks */
#if defined(CONFIG_CMD_PCI) || defined (CONFIG_PCI)
	arm_pci_init,
#endif
	display_dram_config,
	NULL,
};
```

我们来看看env_init这个函数，其代码如下：

```
int env_init(void)
{
	/* use default */
	gd->env_addr = (ulong)&default_environment[0];
	gd->env_valid = 1;
#ifdef CONFIG_DYNAMIC_MMC_DEVNO
	extern int get_mmc_env_devno(void);
	mmc_env_devno = get_mmc_env_devno();
#else
	mmc_env_devno = CONFIG_SYS_MMC_ENV_DEV;
#endif
	return 0;
}
```

可以看出在这里将default_environment加载进入系统，default_environment对应的部分代码如下：

```
uchar default_environment[] = {
	.
	.
	.
#ifdef  CONFIG_EXTRA_ENV_SETTINGS
	CONFIG_EXTRA_ENV_SETTINGS
#endif
	"\0"
};
```

而CONFIG_EXTRA_ENV_SETTINGS则是在我们对应的BSP的头文件中定义了，如下：

```
#define  CONFIG_EXTRA_ENV_SETTINGS                                          \
	"netdev=eth0\0"                                                 \
	"ethprime=FEC0\0"                                               \
	"bootfile=uImage\0"  \                                          \
	"loadaddr=0x70800000\0"                                           \
	"rd_loadaddr=0x70D00000\0"        \
	"bootargs=console=ttymxc0 init=/init " \
		"androidboot.console=ttymxc0 video=mxcdi1fb:RGB666,XGA " \
		"ldb=di1 di1_primary pmem=32M,64M fbmem=5M gpu_memory=64M\0" \
	"bootcmd_SD=mmc read 0 ${loadaddr} 0x800 0x2000;" \
		"mmc read 0 ${rd_loadaddr} 0x3000 0x300\0" \
	"bootcmd=run bootcmd_SD; bootm ${loadaddr} ${rd_loadaddr}\0" \
```

再来看看check_recovery_mode这个函数中的代码，具体代码如下：

```
/* export to lib_arm/board.c */
void check_recovery_mode(void)
{
	if (check_key_pressing())
		setup_recovery_env();
	else if (check_recovery_cmd_file()) {
		puts("Recovery command file founded!\n");
		setup_recovery_env();
	}
}
```

可以看到在这里通过check_key_pressing这个函数来检测组合按键，当有对应的组合按键按下时，将会进入到recovery模式，这也正是各大android论坛里讲到刷机时都会提到的power+音量加键进入recovery模式的原因。那么check_recovery_cmd_file又是在什么情况下执行的呢？这个也正是这篇文章所要讲的内容之处。

先来看看check_recovery_cmd_file这个函数中的如下这段代码：

```
int check_recovery_cmd_file(void)
{
        .
        .
        .
        switch (get_boot_device()) {
        case MMC_BOOT:
        case SD_BOOT:
                {
                        for (i = 0; i < 2; i++) {
                                block_dev_desc_t *dev_desc = NULL;
                                struct mmc *mmc = find_mmc_device(i);
                                dev_desc = get_dev("mmc", i);
                                if (NULL == dev_desc) {
                                        printf("** Block device MMC %d not supported\n", i);
                                        continue;
                                }
                                mmc_init(mmc);
                                if (get_partition_info(dev_desc, CONFIG_ANDROID_CACHE_PARTITION_MMC,
                                                        &info)) {
                                        printf("** Bad partition %d
**\n",CONFIG_ANDROID_CACHE_PARTITION_MMC);
                                        continue;
                                }
                                part_length = ext2fs_set_blk_dev(dev_desc,
CONFIG_ANDROID_CACHE_PARTITION_MMC);
                                if (part_length == 0) {
                                        printf("** Bad partition - mmc %d:%d **\n", i,
CONFIG_ANDROID_CACHE_PARTITION_MMC);
                                        ext2fs_close();
                                        continue;
                                }
                                if (!ext2fs_mount(part_length)) {
                                        printf("** Bad ext2 partition or "
                                                "disk - mmc %d:%d **\n",
                                                i, CONFIG_ANDROID_CACHE_PARTITION_MMC);
                                        ext2fs_close();
                                        continue;
                                }
                                filelen = ext2fs_open(CONFIG_ANDROID_RECOVERY_CMD_FILE);
                                ext2fs_close();
                                break;
                        }
                }
                break;
        .
        .
        .
}
```

主要来看看下面这个ext2fs_open所打开的内容，CONFIG_ANDROID_RECOVERY_CMD_FILE，这个正是上面所提到的rocovery cmd file的宏定义，内容如下：

```
#define CONFIG_ANDROID_RECOVERY_CMD_FILE  "/recovery/command"
```

当检测到有这个文件存在时，将会进入到setup_recovery_env这个函数中，其相应的代码如下：

```
void setup_recovery_env(void)
{
        char *env, *boot_args, *boot_cmd;
        int bootdev = get_boot_device();
```

```
        boot_cmd = supported_reco_envs[bootdev].cmd;
        boot_args = supported_reco_envs[bootdev].args;
        if (boot_cmd == NULL) {
                printf("Unsupported bootup device for recovery\n");
                return;
        }
        printf("setup env for recovery..\n");
        env = getenv("bootargs_android_recovery");
        /* Set env to recovery mode */
        /* Only set recovery env when these env not exist, give user a
         * chance to change their recovery env */
        if (!env)
                setenv("bootargs_android_recovery", boot_args);
        env = getenv("bootcmd_android_recovery");
        if (!env)
                setenv("bootcmd_android_recovery", boot_cmd);
        setenv("bootcmd", "run bootcmd_android_recovery");
}
```

在这里主要是将bootcmd_android_recovery这个环境变量加到uboot启动的environment中，这样当系统启动加载完root fs之后将不会进入到android的system中，而是进入到了recovery这个轻量级的小UI系统中。

下面我们来看看为什么在uboot的启动环境变量中加入bootcmd_android_recovery这些启动参数的时候，系统就会进入到recovery模式下而不是android system，先看看bootcmd_android_recovery相应的参数：

```
#define CONFIG_ANDROID_RECOVERY_BOOTARGS_MMC \
        "setenv bootargs ${bootargs} init=/init root=/dev/mmcblk1p4"     \
        "rootfs=ext4 video=mxcdi1fb:RGB666,XGA ldb=di1 di1_primary"
#define CONFIG_ANDROID_RECOVERY_BOOTCMD_MMC  \
        "run bootargs_android_recovery;"  \
        "mmc read 0 ${loadaddr} 0x800 0x2000;bootm"
```

可以看到在进入recovery模式的时候这里把root的分区设置成了/dev/mmcblk1p4，再来看看在系统烧录的时候对整个SD卡的分区如下：

```
sudo mkfs.vfat -F 32 ${NODE}${PART}1 -n sdcards
sudo mkfs.ext4 ${NODE}${PART}2 -O ^extent -L system
sudo mkfs.ext4 ${NODE}${PART}4 -O ^extent -L recovery
sudo mkfs.ext4 ${NODE}${PART}5 -O ^extent -L data
sudo mkfs.ext4 ${NODE}${PART}6 -O ^extent -L cache
```

这里NODE = /dev/mmcblk1为挂载点，PART = p或者为空，作为分区的检测。可以看出上面在给recovery分区的时候，用的是/dev/mmcblk1p4这个分区，所以当设置了recovery启动模式的时候，root根目录就被挂载到/dev/mmcblk1p4这个recovery分区中来，从而进入recovery模式。

4 recovery

关于android的recovery网上有各种版本的定义，这里我总结一下：所谓recovery是android下加入的一种特殊工作模式，有点类似于windows下的gost，系统进入到这种模式下时，可以在这里通过按键选择相应的操作菜单实现相应的功能，比如android系统和数据区的快速格式化(wipe)；系统和用户数据的备份和恢复；通过sd卡刷新的rom等等。典型的recovery界面如下：

图2 recovery界面

Recovery的源代码在bootable/recovery这个目录下面，主要来看看recovery.c这个文件中的main函数：

```
Int main(int argc, char **argv) {
        .
        .
        .
    ui_init();
    ui_set_background(BACKGROUND_ICON_INSTALLING);
    load_volume_table();
        .
        .
        .
    while ((arg = getopt_long(argc, argv, "", OPTIONS, NULL)) != -1) {
        switch (arg) {
        case 'p': previous_runs = atoi(optarg); break;
        case 's': send_intent = optarg; break;
        case 'u': update_package = optarg; break;
        case 'w': wipe_data = wipe_cache = 1; break;
        case 'c': wipe_cache = 1; break;
        case 'e': encrypted_fs_mode = optarg; toggle_secure_fs = 1; break;
        case 't': ui_show_text(1); break;
        case '?':
            LOGE("Invalid command argument\n");
            continue;
        }
    }
    device_recovery_start();
        .
        .
        .
    if (update_package)
        {
        // For backwards compatibility on the cache partition only, if
        // we're given an old 'root' path "CACHE:foo", change it to
        // "/cache/foo".
        if (strncmp(update_package, "CACHE:", 6) == 0)
            {
            int len = strlen(update_package) + 10;
```

```
            char* modified_path = malloc(len);
            strlcpy(modified_path, "/cache/", len);
            strlcat(modified_path, update_package+6, len);
            printf("(replacing path \"%s\" with \"%s\")\n",
                   update_package, modified_path);
            update_package = modified_path;
        }
              //for update from "/mnt/sdcard/update.zip",but at recovery system is "/sdcard" so
change it to "/sdcard"
              //ui_print("before:[%s]\n",update_package);
        if (strncmp(update_package, "/mnt", 4) == 0)
        {
        //jump the "/mnt"
                    update_package +=4;
              }
              ui_print("install package from[%s]\n",update_package);
     }
    printf("\n");
    property_list(print_property, NULL);
    printf("\n");
    int status = INSTALL_SUCCESS;
       .
       .
       .
// Recovery strategy: if the data partition is damaged, disable encrypted file systems.
        // This preventsthe device recycling endlessly in recovery mode.
       .
       .
       .
    if (update_package != NULL)
    {
        status = install_package(update_package);
        if (status != INSTALL_SUCCESS)
                    ui_print("Installation aborted.\n");
              else
              {
                    erase_volume("/data");
                    erase_volume("/cache");
              }
    } else if (wipe_data) {
        if (device_wipe_data()) status = INSTALL_ERROR;
        if (erase_volume("/data")) status = INSTALL_ERROR;
        if (wipe_cache && erase_volume("/cache")) status = INSTALL_ERROR;
        if (status != INSTALL_SUCCESS) ui_print("Data wipe failed.\n");
    } else if (wipe_cache) {
        if (wipe_cache && erase_volume("/cache")) status = INSTALL_ERROR;
        if (status != INSTALL_SUCCESS) ui_print("Cache wipe failed.\n");
    } else {
        status = INSTALL_ERROR;  // No command specified
    }
    if (status != INSTALL_SUCCESS) ui_set_background(BACKGROUND_ICON_ERROR);
       //Xandy modify for view the install infomation
    //if (status != INSTALL_SUCCESS || ui_text_visible())
    if(status != INSTALL_SUCCESS)
       {
        prompt_and_wait();
    }
    // Otherwise, get ready to boot the main system...
    finish_recovery(send_intent);
    ui_print("Rebooting...\n");
    sync();
    reboot(RB_AUTOBOOT);
    return EXIT_SUCCESS;
}
```

在这里首先完成recovery模式轻量级的UI系统初始化，设置背景图片，然后对输入的参数格式化，最后根据输入的参数进行相应的操作，如：安装新的ROM、格式化(wipe)data及cache分区等等；值得注意的是刷新ROM的时候，要制作相应的update.zip的安装包，这个在最后一章节讲述，这里遇到的一个问题是在recovery模式下sd卡的挂载点为/sdcard而不是android系统下的/mnt/sdcard，所以我在这里通过：

```
//for update from "/mnt/sdcard/update.zip",but at recovery system is "/sdcard" so change it to
"/sdcard"
```

```
            //ui_print("before:[%s]\n",update_package);
        if (strncmp(update_package, "/mnt", 4) == 0)
        {
        //jump the "/mnt"
                    update_package +=4;
              }
```

这样的操作跳过了上层传过来的/mnt这四个字符。另外一个值得一提的是，传入这里的这些参数都是从/cache/recovery/command这个文件中提取。具体对command文件的解析过程这里不再讲述，可能通过查看recovery.c这个文件中的get_args函数。

那么command这个文件是在什么情况下创建的呢？下面我们就来看看吧！

5 恢复出厂设置和固件升级

在android的系统设备中进入“隐私权->恢复出厂设置->重置手机”将为进入到恢复出厂设置的状态，这时将会清除data、cache分区中的所有用户数据，使得系统重启后和刚刷机时一样了。另外为了方便操作我们还可在“隐私权->固件升级->刷新ROM”这里加入了固件升级这一项。

在讲述这些内容之前，我们有必要来看看/cache/recovery/command这个文件相应的一些recovery命令，这些命令都由android系统写入。所有的命令如下：

```
*       --send_intent=anystring -- write the text out to recovery.intent
*       --update_package=root:path -- verify install an OTA package file
*       --wipe_data -- erase user data (and cache), then reboot
*       --wipe_cache -- wipe cache (but not user data), then reboot
```

5.1 恢复出厂设置

在frameworks/base/services/java/com/android/server/masterClearReceiver.java

这个文件中有如下代码：

```
public class MasterClearReceiver extends BroadcastReceiver {
    private static final String TAG = "MasterClear";
    @Override
    public void onReceive(final Context context, final Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_REMOTE_INTENT)) {
            if (!"google.com".equals(intent.getStringExtra("from"))) {
                Slog.w(TAG, "Ignoring master clear request -- not from trusted server.");
                return;
            }
        }
        Slog.w(TAG, "!!! FACTORY RESET !!!");
        // The reboot call is blocking, so we need to do it on another thread.
        Thread thr = new Thread("Reboot") {
            @Override
            public void run() {
                try {
                    if (intent.hasExtra("enableEFS")) {
                        RecoverySystem.rebootToggleEFS(context,
intent.getBooleanExtra("enableEFS", false));
                    } else {
                        RecoverySystem.rebootWipeUserData(context);
                    }
                    Log.wtf(TAG, "Still running after master clear?!");
                } catch (IOException e) {
```

```
                    Slog.e(TAG, "Can't perform master clear/factory reset", e);
                }
            }
        };
        thr.start();
    }
}
```

当app中操作了“恢复出厂设置”这一项时，将发出广播，这个广播将在这里被监听，然后进入到恢复出厂设置状态，我们来看看rebootWipeUserData这个方法的代码:

```
public static void rebootWipeUserData(Context context) throws IOException {
        final ConditionVariable condition = new ConditionVariable();
        Intent intent = new Intent("android.intent.action.MASTER_CLEAR_NOTIFICATION");
        context.sendOrderedBroadcast(intent, android.Manifest.permission.MASTER_CLEAR,
                new BroadcastReceiver() {
                    @Override
                    public void onReceive(Context context, Intent intent) {
                        condition.open();
                    }
                }, null, 0, null, null);
        // Block until the ordered broadcast has completed.
        condition.block();
        bootCommand(context, "--wipe_data");
    }
```

我们可以看到在这里参入了“--wipe_data”这个参数，并把这条命令写入到command这个文件中去了，在进入recovery模式的时候解析到这条命令时就会清除data和cache中的数据了。

再来看看bootCommand这个方法里的代码:

```
private static void bootCommand(Context context, String arg) throws IOException {
        RECOVERY_DIR.mkdirs();  // In case we need it
        COMMAND_FILE.delete();  // In case it's not writable
        LOG_FILE.delete();
        FileWriter command = new FileWriter(COMMAND_FILE);
        try {
            command.write(arg);
            command.write("\n");
        } finally {
            command.close();
        }
        // Having written the command file, go ahead and reboot
        PowerManager pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE);
        pm.reboot("recovery");
        throw new IOException("Reboot failed (no permissions?)");
    }
```

其中COMMAND_FILE这个成员的定义如下:

```
/** Used to communicate with recovery.  See bootable/recovery/recovery.c. */
private static File RECOVERY_DIR = new File("/cache/recovery");
private static File COMMAND_FILE = new File(RECOVERY_DIR, "command");
```

至此恢复出厂设置的命令就写入了recovery cmd file中去了，通过pm.reboot(“recovery”);重启系统，系统就自动进入到recovery模式自动清除用户数据后再重启系统。

5.2 固件升级

固件升级的流程和恢复出厂设置差不多，不同之处是入command这个文件中写入的命令不一样，下面是恢复出厂设置时的写命令的代码：

```
public static void installPackage(Context context, File packageFile)
      throws IOException {
      String filename = packageFile.getCanonicalPath();
      Log.w(TAG, "!!! REBOOTING TO INSTALL " + filename + " !!!");
      String arg = "--update_package=" + filename;
      bootCommand(context, arg);
  }
```

这里的packageFile是由上层app传入的，内容如下：

```
File packageFile = new File("/sdcard/update.zip");
RecoverySystem.installPackage(context, packageFile);
```

这样当系统重启进入到recovery模式时将会自动查找sdcard的根目录下是否有update.zip这个文件，如果有将会进入到update状态，否则会提示无法找到update.zip！

至此我们已经明白了android的整个recovery流程，下面将讲讲update.zip也就是各大论坛里讲到的ROM的制作过程。

6 ROM的制作

我们解压update.zip这个文件，可发现它一般打包了如下这几个文件：

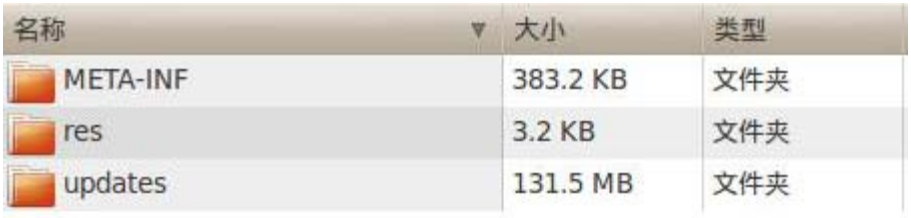

图3 ROM包中的内容

或者没有updates而是system这个目录，不同的原因是我这里在updates里放置的是system.img等镜像文件，这些文件都由源码编译而来。而如果是system目录，这里一般放的是android系统的system目录下的内容，可以是整个android系统的system目录，也可以是其中的一部分内容，如一些so库等等，这样为补丁的发布提供了一个很好的解决办法，不需要更新整个系统，只需要更新一部分内容就可以了！

来看看META-INF/com/google/android这个目录下的内容，在这里就两个文件，一个是可执行的exe文件update-binary，这个文件在进入update状态的用于控制ROM的烧入，具体的代码在recovery下的install.c文件中的try_update_binary这个函数中；另一个是updater-script，这个文件里是一些脚本程序，具体的代码如下：

```
# Mount system for check figurepoint etc.
# mount("ext4", "EMMC","/dev/block/mmcblk0p2", "/system");
# Make sure Check system image figurepoint first.
# uncomment below lines to check
# assert(file_getprop("/system/build.prop", "ro.build.fingerprint") ==
"freescale/imx53_evk/imx53_evk/imx53_evk:2.2/FRF85B/eng.b33651.20100914.145340:eng/test-keys");
# assert(getprop("ro.build.platform) == "imx5x");
```

```
# unmount("/system");

show_progress(0.1, 5);
package_extract_dir("updates", "/tmp");
#Format system/data/cache partition
ui_print("Format disk...");
format("ext4","EMMC","/system");
format("ext4","EMMC","/data");
format("ext4","EMMC","/cache");
show_progress(0.2, 10);
# Write u-boot to 1K position.
# u-boot binary should be a no padding uboot!
# For eMMC(iNand) device, needs to unlock boot partition.
ui_print("writting u-boot...");
sysfs_file_write(" /sys/class/mmc_host/mmc0/mmc0:0001/boot_config", "1");
package_extract_file("files/u-boot.bin", "/tmp/u-boot.bin");
#ui_print("Clean U-Boot environment...");
show_progress(0.2, 5);
#simple_dd("/dev/zero","/dev/block/mmcblk0",2048);
simple_dd("/tmp/u-boot.bin", "/dev/block/mmcblk0", 2048);
#access user partition,and enable boot partion1 to boot
sysfs_file_write("/sys/class/mmc_host/mmc0/mmc0:0001/boot_config", "8");

#Set boot width is 8bits
sysfs_file_write("/sys/class/mmc_host/mmc0/mmc0:0001/boot_bus_config", "2");
show_progress(0.2, 5);

ui_print("extract kernel image...");
package_extract_file("files/uImage", "/tmp/uImage");
# Write uImage to 1M position.
ui_print("writting kernel image");
simple_dd("/tmp/uImage", "/dev/block/mmcblk0", 1048576);

ui_print("extract uramdisk image...");
package_extract_file("files/uramdisk.img", "/tmp/uramdisk.img");
# Write uImage to 1M position.
ui_print("writting uramdisk image");
simple_dd("/tmp/uramdisk", "/dev/block/mmcblk0", 6291456);
show_progress(0.2, 50);

# You can use two way to update your system which using ext4 system.
# dd hole system.img to your mmcblk0p2 partition.
package_extract_file("files/system.img", "/tmp/system.img");
ui_print("upgrading system partition...");
simple_dd("/tmp/system.img", "/dev/block/mmcblk0p2", 0);
show_progress(0.1, 5);
```

相应的脚本指令可在说明可对应源码可在recovery包中的install.c这个文件中找到。

在bootable/recovery/etc下有原始版的脚本代码update-script，但在recovery下的updater.c这个文件中有如下定义：

```
// Where in the package we expect to find the edify script to execute.
// (Note it's "updateR-script", not the older "update-script".)
#define SCRIPT_NAME "META-INF/com/google/android/updater-script"
```

所在使用这个原版的脚本的时候要将update-script更成updater-script，需要注意！

我们可以发现在bootable/recovery/etcMETA-INFO/com/google/android目录下少了一个update-binary的执行文件，在out/target/product/YOU_PRODUCT/system/bin下面我们可以找到updater，只要将其重名字为update-binary就可以了！

有了这些准备工作，我们就可以开始制作一个我们自己的ROM了，具体步骤如下：

```
*       Xandy@ubuntu:~$ mkdir recovery
*       Xandy@ubuntu:~$ cd recovery 然后将上面提到的bootable/recovery/etc下的所有内容拷贝到当前目录下并删掉init.rc这个文件
*       编译./META-INF/com/google/android/updater-script这个文件使达到我们想要的烧写控制，如果是烧
  system.img                                    updater-script
```

写　　这样的镜像文件，可以直接用我上面提到的　　这个脚本代码。
*　　拷贝相应的需要制作成ROM的android文件到updates目录或者system目录下，这个得根据系统的需要决定。
*　　Xandy@ubuntu:~/recovery$ mkdir res
*　　Xandy@ubuntu:~/recovery$ ~/myandroid/out/host/linux-x86/framework/dumpkey.jar ~/myandroid/build/target/product/security/testkey.x509.pem > res/keys 这里创建一个目录用于存储系统的key值
*　　zip /tmp/recovery.zip -r ./META-INF ./updates ./res 将所有文件打包
*　　java -jar ./tools/signapk.jar -w ./tools/testkey.x509.pem ./tools/testkey.pk8 /tmp/recovery.zip update.zip 我在recovery目录下创建了一个tools目录，里面放置了sygnapk.jar、testkey.pk8、testkey.x509.pem这几个文件用于java签名时用

经过上面这几步之后就会在recovery目录生成一个update.zip的文件，这个就是我们自己制作的ROM文件，将它拷到sdcard的根目录下，在系统设置里操作进入到“固件升级状态”，等到系统重启时，就会看到已经开始自行格式化data和cache分区，稍后就开始出现进度条向相应分区里烧写uboot、kernel、android system的文件了！

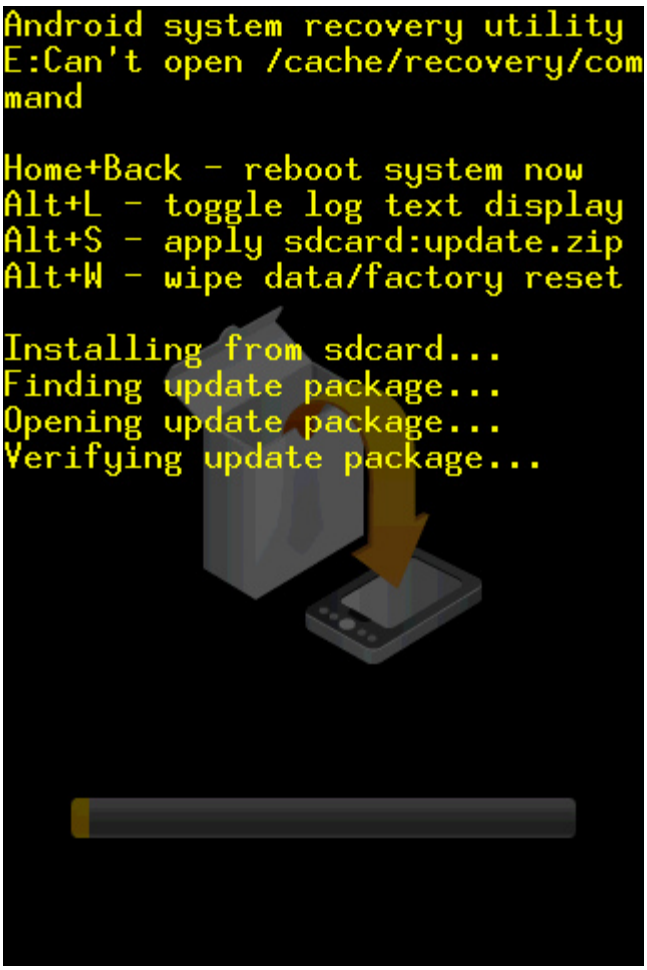

图4 烧入新ROM