TinaLinux

OTA 开发指南

1.0

2019.07.03

文档履历

版本号	日期	制/修订人	内容描述
1.0	2019.07.03	AWA1531	初始版本

目录

1. misc-upgrade 介绍

1.1 概述概述

1.1.1 编写目的

本文主要服务于使用 Tina 软件平台的广大客户，以冀帮助客户使用 Tina 平台的 OTA 升级系统并做二次开发。

1.1.2 适用范围

Allwinner 软件平台 Tina
Allwinner 硬件平台 R6 R11 R16 R18 R30 R40 R328

1.1.3 相关人员

适用 Tina 平台的广大客户和关心 OTA 的相关人员。

1.2 OTA 方案选择

OTA 是 Over The Air 的简称，顾名思义就是通过无线网络从服务器上下载更新文件对本地系统或文件进行升级，便于客户为其用户及时更新系统和应用以提供更好的产品服务，这对于客户和消费者都极其重要。
由于在实际应用中，存储操作系统和持久文件的存储介质（如 nand、emmc、spinor）大小各异，在 OTA 中需要单独在存储介质上开辟 recovery 分区 [1]，以防备在更新中意外断电，造成系统更新失败无法重启的问题。
所以在选择 OTA 方案时一定要考虑到 recovery 分区大小对分区规划的影响，避免在小容量时 recovery 分区太大导致分区规划难题。
综上因素，我们在 Tina 上针对大容量和小容量设计了不同的方案。

1.2.1 小容量方案

小容量介质一般指存储介质容量小于 32M（一般为 spi nor）。
在命令行中进入 Tina 根目录，执行命令进入配置主界面：

```
source build/envsetup.sh (见详注1)
lunch (见详注2)
make menuconfig (见详注3)
详注：
1 加载环境变量及tina提供的命令
2 输入编号，选择方案
3 进入内核配置主界面(对一个shell而言，前两个命令只需要执行一次)
```

配置路径：

```
Target Image
    └─> *** Image Options ***
        [*] For storage less than 32M, enable this when using ota
```

选中该配置项后，rootfs 的/usr 会被分拆出一部分生成 usr.squashfs(usr.img)，并建立软链接 usr.fex。通过配置分区表将 usr.fex 放在 extend 分区，开机后自动挂载到 usr 目录。这种设置的目的是可与 recovery 镜像（boot_initramfs）复用该分区，以此起到节省存储空间的作用。因此小容量方案中，并无单独的 recovery 分区，而是在 OTA 升级时与 extend 分区复用。

1.2.2 大容量方案

大容量介质一般指存储介质容量大于 32M（一般是 nand、emmc）。
对于大容量方案，不建议选中小容量方案中所述配置项，即不需要 usr.img 和 extend 分区，而只需要添加 recovery 分区，这样在 OTA 升级时会省去很多麻烦。

1.2.3 misc-upgrade

不管是小容量还是大容量，都要在 make 之前选中应用包 misc-upgrade：

```
make menuconfig
    └─> Allwinner
    └─> <*> misc-upgrade..... read and write the misc partition
```

misc-upgrade 包主要功能是从指定服务器下载更新镜像到本地，然后升级相应分区，期间会向 misc 分区 [1] 写入升级阶段的标志，出现意外无法重启时 uboot 或内核（如果能够启动）可以根据 misc 分区的状态标志进行下一步的决策。

1.2.4 OTA 的升级流程

1.2.4.1 基本步骤

Tina3.0 及之前版本处理流程

1. 备份busybox等资源到ram中, 使得OTA过程不依赖rootfs
2. 设置开始OTA的标志，upgrade_pre
3. 将recovery系统写入recovery分区
4. 设置标志boot-recovery
5. 更新boot和rootfs分区
6. 设置标志 upgrade_post
7. 对于小容量方案，更新usr分区
8. 设置标志upgrade_end
9. 重启，重启后为新系统

Tina3.0.1 版本处理流程

1. 设置开始OTA的标志，upgrade_pre
2. 将recovery系统写入recovery分区
3. 设置标志boot-recovery
4. 主动重启，重启后进入recovery系统
5. 更新boot和rootfs分区
6. 设置标志upgrade_post
7. 对于小容量方案，更新usr分区
8. 设置标志upgrade_bootloader
9. 更新boot0/uboot
10. 设置标志upgrade_end
11. 重启，重启后为新系统

1.2.4.2 特殊步骤

对于使用 busybox init 启动的平台，如 R6 等，其 rootfs_data 分区是启动后拷贝 rootfs 中的 etc 目录生成的，OTA 之后，rootfs 发生了变化，需要重新生成 rootfs_data 分区的内容。
因此，在 upgrade_end 前新增一个状态，upgrade_etc. 重启后 busybox-init 的启动脚本判断此标志，如果启动是标志为 upgrade_etc，则进行 etc 分区文件的更新，更新后设置系统状态为 upgrade_end。

1.2.4.3 中途掉电

OTA 中途掉电后，下次启动会根据标志，继续完成 OTA
当设置 upgrade_pre 标志之后掉电，重启后仍是旧系统，从该标志之后的步骤开始执行
当设置 boot-recovery 标志之后掉电，重启时，uboot 判断到这个标志，并直接启动 recovery 系统，启动后，从该标志之后的步骤开始执行
当设置 upgrade_post 标志之后掉电，重启时后为新系统，从该标志之后的步骤开始执行

1.3 分区处理

1.3.1 分区定义

升级分区	
boot 分区	基础系统镜像分区（/lib, /bin, /etc, /sbin 等非/usr，非挂载其他分区的路径，wifi 支持环境，alsa 支持环境、OTA 环境）
extend 分区	扩展系统镜像分区（/usr 应用分区）[仅小容量方案有]
recovery 分区	存放恢复系统镜像 [仅大容量方案有]
不升级分区	
private 分区	存储 SN 号分区
misc 分区	系统状态、刷机状态分区
UDISK 分区	用户数据分区（/mnt/UDISK）
overlayfs 分区	存储 overlayfs 覆盖数据

1.3.2 分区大小配置

1.3.2.1 配置 boot 分区大小

boot 分区镜像的大小依赖内核配置, 必须小于等于 sys_partition.fex/sys_partition_nor.fex [3] 中定义的 boot 分区大小。

boot 分区镜像大小设定:

```
make menuconfig
    └─> Target Images
        └─> *** Image Options ***
            (4) Boot (SD Card) filesystem partition size (in MB)
```

boot 分区大小设定:

```
[partition]
    name =boot
    size =8192[4]
    downloadfile ="boot.fex"
    user_type =0x8000
```

对于大容量方案, 需要在 sys_partition.fex 中添加 recovery 分区:

```
recovery 分区说明
如果启用了 OTA 升级,需要去掉下面 recovery 分区的注释以提供恢复分区存储恢复系统镜像,
默认以 boot_initramfs.img 作为 recovery.fex

[partition]
name =recovery
size =32768
downloadfile ="recovery.fex"
user_type =0x8000
```

其中 recovery.fex 是生成的 boot_initramfs.img 软链接而成。

1.3.2.2 rootfs 分区的大小

rootfs 分区不需要通过 make menuconfig 去设定, 直接根据镜像大小修改分区文件即可。

1.3.2.2.1 小容量　对于一些小容量 flash 的方案 (如 16M), 需在/bin 下存放联网逻辑程序、版本控制程序、下载镜像程序、播报语音程序以及语音文件 (这些文件在编译时应该 install 到 /bin 或者 /lib 下), 可以在固件编译完后, 查看 rootfs.img 的大小再决定 sys_partition.fex 中 rootfs 分区的大小。

1.3.2.2.2 大容量　对于大容量 flash 的方案 (如 128M 以上, 或者有足够的 flash 空间存相关镜像), 不需要小容量中那些 OTA 额外的程序, 直接查看 rootfs.img 的大小设定分区文件即可。

1.3.2.3 extend 分区的大小

extend 分区用于小容量方案下 boot_initramfs.img 和 usr.img 的复用, 其大小需要考虑多个方面:

```
1 . 编译后 usr.img 的大小
2 . make_ota_image 后 initramfs 镜像的大小 [5]
```

因此, extend 分区略大于 boot_initramfs.img 和 usr.img 两个的最大值, 并把 extend 分区的大小值设置为 initramfs 镜像的大小:

```
make menuconfig
    └─> Target Images
        └─> *** Image Options ***
            (8) Boot-Recovery initramfs filesystem partition size (in MB)
```

1.3.2.4 其他分区

如 private 、misc 等使用默认的大小即可。

1.3.2.5 UDISK 分区

sys_partition.fex 中不指定 UDISK 分区大小, 则剩下的空间全部自动分配进入 UDISK 分区。需要注意的是, 因为 OTA 过程会在里面写一些中间文件, 所以一定要留取一定空间给 UDISK 分区, 至少可以格式化, 挂载, 而小容量 flash 的方案, 也要保证有 256K~512K 的空间。

1.3.2.6 其他说明

在 OTA 升级过程中并不能修改上述分区的修改, 因此应在满足分区大小条件限制 (如 3.2.1-3.2.3) 的情况下尽可能留有足够的空余空间, 以满足 OTA 升级添加内容的需求。
修改分区大小时, 尽量对齐到所用存储介质的块大小。
对于大容量, 使用 recovery 分区, 对应的, 其 env 定义中, boot_recovery 命令需定义为从 recovery 分区启动系统。对于小容量, 使用 extend 分区, 对应的, 其 env 定义中, boot_recovery 命令需定义为从 extend 分区启动系统。

1.4 misc-upgrade 升级

1.4.1 misc-upgrade 构成

misc-upgrade 是 Tina 下的一个应用, 其路径为:

```
tina/package/allwinner//misc-upgrade
```

Makefile 符合 tina 安装包的书写规范。
misc-upgrade 目录结构如下:

```
├── aw_fstab.init
├── aw_reboot.sh
├── aw_upgrade_autorun.init
├── aw_upgrade_image.sh
├── aw_upgrade_log.sh
├── aw_upgrade_normal.sh
```

```
├── aw_upgrade_process.sh
├── aw_upgrade_utils.sh
├── aw_upgrade_vendor_default.sh
├── Makefile
├── readme.txt
└── tools
    ├── Makefile
    ├── misc_message.c
    ├── misc_message.h
    ├── read_misc.c
    └── write_misc.c
```

其中 aw_upgrade_normal.sh 是专用于大容量的方案, 而 aw_upgrade_process.sh 是用于小容量的方案。

1.4.2 OTA 镜像包编译

在命令行中进入 Tina 根目录, 执行命令进入配置主界面 (环境配置):

```
source build/envsetup.sh ( 详见1 )
lunch ( 详见2 )
make ota_menuconfig (可选) ( 详见3 )
详注:
1 加载环境变量及 tina 提供的命令
2 输入编号,选择方案
3 进入 OTA config 配置界面。直接保存退出即可。
此步骤可选,目的是解决开发过程中 defconfig_ota 未能及时更新而可能引发的编译问题
```

编译命令:

```
make_ota_image ( 详见1 )
make_ota_image --force ( 详见2 )
详注:
1 在新版本代码已经成功编译出烧录固件的环境的基础上,打包 OTA 镜像
2 重新编译新版本代码,然后再打包 OTA 镜像
```

执行 make_ota_image 之前, 可通过 make ota_menuconfig 对 ota 的恢复系统镜像 boot_initramfs.img 进行配置, 可根据实际情况, 配置 ota 恢复系统包含的功能。
如以下配置支持 ramdisk 并选用 xz 压缩 cpio :

```
make ota_menuconfig
└─> target Images
    └─> [*] ramdisk
        └─> --- ramdisk
            └─> Compression (xz) --->
```

OTA 镜像包路径为: tina/out/xxx/ota/
目录结构如下:

```
├── ramdisk_sys
│   └── boot_initramfs.img.md5
├── ramdisk_sys.tar.gz
├── target_sys
│   ├── boot.img
│   ├── boot.img.md5
│   ├── rootfs.img
│   └── rootfs.img.md5
└── target_sys.tar.gz
```

其中," .img.md5 ``是" .img ``的校验值文件

1.4.3 小机端 OTA 升级命令

必选参数: -f -p 二选一
aw_upgrade_process.sh -f 升级完整系统 (内核分区、rootfs 分区、extend 分区)
aw_upgrade_process.sh -p 升级应用分区 (extend 分区)
可选参数: -l, -d -u, -n
注: 对于大容量, 用 aw_upgrade_normal.sh 替代 aw_upgrade_process.sh , 且一般用 -f 参数而不用 -p

1.4.3.1 大容量 flash 方案

可以使用本地镜像测试, 如主程序下载校验好 2 个镜像后 (ramdisk_sys.tar.gz 、target_sys.tar.gz), 存在 /mnt/UDISK/misc-upgrade 中, 调用如下命令。
OTA 升级期间掉电, 重启后升级程序也能自动完成烧写, 不需要依赖联网重新下载镜像。

1.4.3.1.1 -l 选项 -l < 路径 >

如:

aw_upgrade_normal.sh -f -l /mnt/UDISK/misc-upgrade

(注: mnt 前的根目录 ``/'' 最好带上, misc-upgrade 后不要带 ``/'') (-l 参数, 默认使用压缩镜像包)

不使用 -n 参数的方案需要部署上服务器上的镜像是: ramdisk_sys.tar.gz 、target_sys.tar.gz

1.4.3.1.2 -n 选项 如:

aw_upgrade_normal.sh -f -n -l /mnt/UDISK/misc-upgrade

一般情况下不使用 -n 选项, 而是下载 ramdisk_sys.tar.gz 、target_sys.tar.gz 。但对于某些 ram 较小的平台, 如 R6 spinand 的情况, flash 的容量足够放下大容量的 OTA 包, 但升级过程可能因为 ram 不足而失败。对于这种情况, 可以选择下载未压缩的数据, 即 ramdisk_sys.tar.gz 、target_sys.tar.gz 解压出来的所有内容。

将多个分区数据文件下载到 /mnt/UDISK/misc-upgrade 中, 调用上述命令。

使用 -n 参数的方案需要部署上服务器上的镜像是: boot_initramfs.img 、boot.img 、rootfs.img 及其对应的 md5 后缀的校验文件

1.4.3.2 小容量 flash 方案

不能使用 -l 参数, 升级区间出错重启后, 需要联网下载程序获取镜像。

```
-d arg1 -u arg2
```

同时使用, -d 参数为可以 ping 通的 OTA 服务器的地址、-u 参数为镜像的下载地址

```
-n
```

-n 一些小 ddr 的方案 (如剩余可使用内存在 20m 以下的方案), 可以使用这个参数, shell 会直接请下载不压缩的 4 个 img 文件, 这样子设备下载后不需要 tar 解压, 减少内存使用。

如:

```
aw_upgrade_process -f -d 192.168.1.140 -u http://192.168.1.140/
```

升级 shell 会先 ping -d 参数 (ping 192.168.1.140), ping 通过后, 会根据升级命令和系统当前场景请求
下载:
无 -n 参数:

```
http://192.168.1.140/ramdisk_sys.tar.gz
http://192.168.1.140/target_sys.tar.gz
http://192.168.1.140/usr_sys.tar.gz
```

有 -n 参数:

```
http://192.168.1.140/boot_initramfs.img
http://192.168.1.140/boot.img
http://192.168.1.140/rootfs.img
http://192.168.1.140/usr.img
```

使用 -n 参数的方案需要部署上服务器上的镜像是: boot_initramfs.img 、boot.img 、rootfs.img 、usr.img 及其对应的 md5 后缀的校验文件
不使用 -n 参数的方案需要部署上服务器上的镜像是: ramdisk_sys.tar.gz 、target_sys.tar.gz 、usr_sys.tar.gz

1.5 脚本接口说明

对于小容量 flash 的方案, 没有空间存储镜像, 相关镜像只会存在 ram 中, 断电就会丢失。
假如升级过程断电, 需要在重启后重新下载镜像。
aw_upgrade_vendor.sh 设计为各个厂家实现的钩子, SDK 上只是个 demo 可以随意修改。

1.5.1 实现联网逻辑

```
check_network_vendor(){
    return 0 联网成功(如:可以 ping 通 OTA 镜像服务器)
    return 1 联网失败
}
```

1.5.2 请求下载目标镜像

$1 : ramdisk_sys.tar.gz $2 : /tmp

```
download_image_vendor(){
    # $1 image name $2 DIR $@ others
    rm -rf $2/$1
    echo "wget $ADDR/$1"
    wget $ADDR/$1 -P $2
}
```

1.5.3 开始烧写分区状态

```
aw_upgrade_process.sh -p 主动升级应用分区的模式下,返回 0 开始写分区 1 不写分区
aw_upgrade_process.sh -f 不理会这个返回值
upgrade_start_vendor(){
    # $1 mode: upgrade_pre,boot-recovery,upgrade_post
    #return 0 -> start upgrade; 1 -> no upgrade
    #reutrn value only work in nornal mode
    #nornal mode: $NORMAL_MODE
    echo upgrade_start_vendor $1
    return 0
}
```

1.5.4 写分区完成

```
upgrade_finish_vendor(){
    #set version or others
    reboot -f
}
```

1.5.5 -f (-n) 调用顺序

```
1 . check_network_vendor ->
2 . upgrade_start_vendor ->
3 . download_image_vendor (ramdisk_sys.tar.gz, -n 为 boot_initramfs.img)->
4 .内部烧写、清除镜像逻辑(不让已经使用镜像占用内存) ->
5 . download_image_vendor(target_sys.tar.gz, -n 为 boot.img rootfs.img) ->
6 .内部烧写、清除镜像逻辑(不让已经使用镜像占用内存) ->
7 . download_image_vendor(usr_sys.tar.gz, -n 为 usr.img) ->
8 .内部烧写、清除镜像逻辑(不让已经使用镜像占用内存) ->
9 . upgrade_finish_vendor
```

1.5.6 -p 调用顺序

```
1 . check_network_vendor ->
2 . download_image_vendor (usr_sys.tar.gz) ->
3 . upgrade_start_vendor ->
4 .检测返回值,烧写 ->
5 . upgrade_finish_vendor
```

1.6 相关系统状态读写

相关的信息存储在 misc 分区, OTA 升级不会清除这个分区 (重新烧写镜像会擦除)

读:

read_misc [command] [status] [version]

其中:

```
command 表示升级的系统状态( shell 脚本处理使用)
status 自由使用,表示用户自定义状态
version 自由使用,表示用户自定义状态
```

写:

write_misc [-c command] [-s status] [-v version]

其中:

```
-c 不能随意修改,只能有 aw-upgrade shell 修改
-s -v 自定义使用
```

1.7 OTA 配置

一般来说, 默认方案目录下会有一份 defconfig_ota 配置文件, 该文件用于编译生成一个带 ramdisk 的 kernel , 即 boot_initramfs.img , 作为 OTA 期间烧写到 recovery 分区或 extend 分区的备份系统。

用户可以基于原有的 defconfig_ota 进行配置, 也可以自行拷贝 defconfig 为新的 defconfig_ota , 然后进行适当修改和配置。

执行 make ota_menuconfig 可进行 OTA 配置。

选上 _recovery 后缀，避免编译 recovery 系统时，影响到主系统。

```
make ota_menuconfig
    ---> Target Images
        ---> [*] customize image name
            ---> Boot Image(kernel) name suffix (boot_recovery.img/boot_initramfs_recovery.img)
                --->    Rootfs Image name suffix (rootfs_recovery.img)
```

如果方案进行了较多的修改, 建议删除原本的 defconfig_ota , 然后拷贝 defconfig 为新的 defconfig_ota , 再进行配置。

如上文所述, 必须选上 misc-upgrade 包, 以及 ramdisk 选项, 保留 wifi 功能。其他选项可以关掉, 以对生成的 boot_initramfs.img 进行裁剪。

1.8 对 overlayfs 的处理

Tina 默认使用 overlayfs , 则用户对原 rootfs 的修改会记录在 rootfs_dafa 分区中。而 OTA 更新的是 rootfs 分区, 默认不会修改 rootfs 分区。则用户对 rootfs 文件的增加, 删除, 修改操作都会保留, 假如原本的 rootfs 中有文件 A , 用户将其修改为 A1 , 而 OTA 更新将该文件修改为 B , 则最终看到的仍然是 A1 。这是由 overlayfs 的特性决定的, 上层目录的文件会屏蔽下层目录。
如果希望 OTA 之后, 以 OTA 更新的文件为准, 移除所有用户的修改。则可以在 OTA 之后, 重新格式化 rootfs_data 分区。

1.9 对 busybox-init 的处理

有些平台使用了 busybox-init , 以 R6 为例子:
R6 方案将启动方式从 procd 修改为 busybox-init , 不再使用 procd 和 overlayfs , 由此带来一系列变化。
OTA 脚本通过 1 号进程的名字来判断启动方式。并根据结果。在后续脚本中做区别处理。
对于 busybox-init , 在第一次启动之后, 会将 etc 的文件拷贝到 rootfs_data 分区中, 并在后续挂载该分区作为 etc 目录。OTA 的过程不会更新 rootfs_data 分区。为了支持更新 etc 目录, 增加了一个系统状态, upgrade_etc , 并在原本设置 upgrade_end 的地方, 改为设置成 upgrade_etc 。而 busybox-init 的启动脚本会判断此标志, 如果启动是标志为 upgrade_etc , 则进行 etc 分区文件的更新, 更新后设置系统状态为 upgrade_end 。
主系统指定了启动脚本 init=/pseudo_init. 对于 OTA 使用的 recovery 系统也需要指定启动脚本, 若所使用的方案未配置, 可自行修改 env , 在 cmdline 中传递 rdinit=/pseudo_init 进行指定。若所使用的文件系统中, 已经有 rdinit 作为 pesudo_init 的软链接, 则可使用 rdinit=/rdinit

1.10 常见问题

1.10.1 OTA 时出现 SQUASHFS ERROR

此问题是由于 Tina3.0 及更早版本的 OTA , 在更新 rootfs 分区时, 只是将 busybox 等备份到 ram 中, rootfs，分区尚处于挂载状态，此时如果有进程访问 rootfs , 则会出现错误。
解决方式
1. OTA 之前, 关闭其他进程, 避免有进程访问 rootfs 。如果确有进程需要保留, 将其依赖的资源提前备

份到 ram 中 (相关代码可参考 OTA 脚本中的 prepare_env 函数)
2. 参照 3.1 的做法, 在更新完 recovery 分区后, 主动 reboot , 进入 recovery 系统, 在 recovery 系统中更新 boot 和 rootfs 分区
3. 参考原生 openwrt 的做法, 在内存中构建 ram 文件系统后, 执行切换根文件系统的操作, 再进行更新

1.10.2 编译 OTA 包之后, 正常编译出错

出现编译问题, 可能是由于 defconfig 被替换了
请检查下 target 仓库中方案对应目录的 defconfig 和 defconfig_ota 的状况。当前的 OTA 包编译过程, 实质上是备份 defconfig , 使用 defconfig_ota 替换 defconfig , 执行 make , 最终还原 defconfig。
如果此过程中断, 则 defconfig 未被还原, 会导致下次正常编译出问题。
解决方式
还原方案目录下的 defconfig 文件。
建议 make menuconfig 和 make ota_menuconfig 之后, 及时在 target 仓库下将修改提交入 git 仓库中, 避免修改丢失, 方便在必要时进行还原。
(此问题为 Tina3.0 之前的问题，最新版本没有此问题。)

1.10.3 是否可更新 boot0/uboot

请参考 ota-burnboot 部分，尚未集成到 misc-upgrade 中。

2. ota-burnboot 介绍

2.1 文档说明

此文档主要介绍如何在 OTA 时升级 boot0/uboot。
升级工具包含两个方面内容:
OTA 命令升级 boot0 和 uboot
OTA 升级 boot0 和 uboot 的 C/C++ APIs

2.2 概念说明

概念	说明
boot0	较为简单, 主要作用是初始化 dram 并加载运行 uboot。一般不需修改。
uboot	功能较丰富, 支持烧写, 启动内核, 烧 key 及其他一些定制化的功能。
sys_config	全志特有的配置文件, 对于使用 linxu3.4/uboot2011 的平台, 在打包之后 sys_config 会跟 uboot 拼接到一起。对于使用 linux3.10/uboot2014 及更高版本的平台,sys_config 会在打包阶段, 跟设备树的配置合并, 生成最终的 dtb。
dtb	设备树, 由 dts 配置和 sys_config 配置综合得到。
u-boot.fex	使用 linxu3.4/uboot2011 的平台最终用到的 uboot, 其实是 uboot+sys_config。
boot_package.fex	使用 linux3.10/uboot2014 及更高版本的平台最终用到的 uboot, 其实包含的文件由配置文件 boot_package.cfg 决定, 一般至少包含了 uboot 和 dtb, 安全方案会包含一些安全所需文件文件, 可能还有 bootlogo 等文件。
toc0.fex	安全方案使用的 boot0。
toc1.fex	安全方案使用的 uboot, 类似 boot_package.fex 说明, 其中实际也包含了 dts 等多个文件。

即, 本文介绍的升级 uboot, 其实是升级 uboot+sys_config+dtb 这样的一个整体文件。后文不再区分更新 uboot, 更新 sys_config, 更新 dtb。这几个打包完毕是合成一个文件的, 暂不支持单独更新其中一个, 需整体更新。

2.3 用于更新的 bin 文件

获取用于 OTA 的 boot0 与 uboot 的 bin 文件, 用于加入 OTA 包中。

2.3.1 编译 uboot

如果原本的固件生成流程已经包含编译 uboot, 则正常编译固件即可。
否则可按照如下步骤编译生成 uboot

```
$ source build/envsetup.sh
    => 设置环境变量
$ lunch
    => 选择方案
$ muboot
    => 编译 uboot
```

编译后会自行拷贝 bin 文件到该平台的目录下, 即

```
target/allwinner/xxx-common/bin
```

编译出的 uboot 还不能直接用于 OTA, 请继续编译和打包固件, 如执行

```
$ make -j <N>
=> 编译命令,若只修改 uboot/sys_sys_config 无需重新编译,可跳过
=> 若修改了 dts 则需要执行,重新编译
$ pack [-d]
=> 非安全方案的打包命令
$ pack -s [-d]
=> 安全方案的打包命令
```

2.3.2 关于更新 boot0

客户的代码环境中并不包含 boot0 相关代码, 因此无法编译 boot0。
一般情况下, 客户并不需要修改 boot0, 而是直接使用提供的 boot0 的 bin 文件即可。

2.3.3 Bin 文件路径

2.3.3.1 使用 **uboot2011** 的非安全方案

以 R16 的 astart-parrot 方案为例
根据对应存储介质选择 bin
boot0:

```
out/astar-parrot/image/boot0_nand.fex :nand 方案使用的 boot0
out/astar-parrot/image/boot0_sdcard.fex :mmc 方案使用的 boot0
out/astar-parrot/image/boot0_spinor.fex :nor 方案使用的 boot0
```

uboot:

```
out/astar-parrot/image/u-boot.fex :nand/mmc 方案使用的 uboot
out/astar-parrot/image/u-boot-spinor.fex :nor 方案使用的 uboot
```

2.3.3.2 使用 **uboot2014** 及更高版本的非安全方案

以 R6 的 sitar-evb 方案为例
根据对应存储介质选择 bin
boot0:

```
out/sitar-evb/image/boot0_nand.fex :nand 方案使用的 boot0
out/sitar-evb/image/boot0_sdcard.fex :mmc 方案使用的 boot0
out/sitar-evb/image/boot0_spinor.fex :nor 方案使用的 boot0
```

uboot:

```
out/sitar-evb/image/boot_package.fex :nand/mmc 方案使用的 uboot
out/sitar-evb/image/boot_package_nor.fex :nor 方案使用的 uboot
```

2.3.3.3 安全方案

以 R18 的 tulip-noma 方案为例

boot0:

```
out/tulip-noma/image/toc0.fex :安全方案使用的 boot0
```

uboot:

```
out/tulip-noma/image/toc1.fex :安全方案使用的 uboot
```

2.4 OTA 升级命令

2.4.1 支持 OTA 升级命令

升级 boot0 与 uboot 分别使用 ota-burnboot0 与 ota-burnuboot 命令。
两个命令都是 OTA 升级 boot0 和 uboot 的 C/C++ APIs 的封装。
要支持本功能, 需要选中 ota-burnboot 的包, 即

```
Make menuconfig --> Allwinner ---> <*>ota-burnboot
```

2.4.2 ota-burnboot0

2.4.2.1 命令说明

```
$ Usage: ota-burnboot0 <boot0-image>
```

升级 boot0, 其中是镜像的路径
请注意, 安全和非安全方案所使用的是不同的, 具体见第二章。

2.4.2.2 使用示例

```
root@TinaLinux:/# ota-burnboot0 /tmp/boot0_nand.fex
Burn Boot0 Success
```

2.4.3 ota-burnuboot

2.4.3.1 命令说明

```
$ Usage: ota-burnuboot <uboot-image>
```

升级 uboot, 其中是镜像的路径。请注意, 安全和非安全方案, 不同的 uboot 版本, 所使用的是不同的, 具体见第二章。

2.4.3.2 使用示例

```
root@TinaLinux:/# ota-burnuboot /tmp/u-boot.fex
Burn Uboot Success
```

2.5 OTA 升级 C/C++ APIs

包含头文件 <OTA_BurnBoot.h>, 使用库 libota-burnboot.so

2.5.1 int OTA_burnboot0(const char *img_path)

函数原型	int OTA_burnboot0(const char *img_path);
参数说明	img_path:boot0 镜像路径
返回说明	0: 成功非零: 失败
功能描述	烧写 boot0

2.5.2 int OTA_burnuboot(const char *img_path)

函数原型	int OTA_burnuboot(const char *img_path);
参数说明	img_path:uboot 镜像路径
返回说明	0: 成功非零: 失败
功能描述	烧写 uboot

2.6 底层实现

2.6.1 如何保证安全更新 boot0/uboot

前提条件是,flash 中存有不止一份 boot0/uboot。在这个基础上, 启动流程需支持校验并选择完整的 boot0/uboot 进行启动, 更新流程需保证任意时刻掉电,flash 上总存在至少一份可用的 boot0/uboot。

2.6.2 Nand Flash 实现

在 nand 方案中,boot0 和 uboot 是由 nand 驱动管理, 保存在物理地址中, 逻辑分区不可见。
Nand 驱动会保存多份 boot0 和 uboot, 启动时, 从第一份开始依次尝试, 直到找到一份完整的 boot0/uboot 进行使用。
更新 boot0/uboot 时, 上层调用 nand 驱动提供的接口, 驱动中会从第一份开始依次更新, 多份全部更新完毕后返回。因此可保证在 OTA 过程中任意时刻掉电,flash 中均有至少一份完整的 boot0/uboot 可用。再次启动后, 只需重新调用更新接口进行更新, 直到调用成功返回即可。
目前 nand 中的多份 boot0/uboot 是由 nand 驱动管理的, 只能整体更新, 暂不支持单独更新其中的一份。

2.6.3 MMC Flash 实现

在 mmc 方案中,boot0 和 uboot 各有两份, 存在 mmc 上的指定偏移处, 逻辑分区不可见。
启动时, 会从第一份开始依次尝试, 找到一份完整的 boot0/uboot 进行使用。
更新时, 默认只更新第一份, 而第二份是保持在出厂状态的。只要第一份正常更新了, 则启动时会优先使用第一份。如果在更新第一份的过程中掉电导致数据损坏, 则自动使用第二份进行启动。

2.6.4 NOR Flash 实现

nor 方案中, 只保存一份 boot0 和 uboot, 更新过程中掉电可能导致无法启动, 只能进行刷机。故目前未实现 ota 更新, 需后续扩展。

3. Tina SWUpdate OTA 介绍

3.1 swupdate 介绍

3.1.1 开源部分

源码

https://github.com/sbabic/swupdate

文档

http://sbabic.github.io/swupdate/

3.1.2 移植到 tina

移植到 package/allwinner/swupdate

仿照 busybox，添加了配置项

添加 patch，支持了更新 boot0,uboot

添加了自启动脚本

默认启动 progress 在后台，输出到串口。这样升级时会打印进度条。实际方案不需要的话，可去除。客户应用可参考 progress 源码，自行获取进度信息。

默认启动一个脚本 swupdate_cmd.sh，负责完善参数，最终调用 swupdate。实际方案可不用此脚本，自行确定参数保存的方式，并自行在脚本或应用中去调用 swupdate

3.2 主系统和 recovery 系统

3.2.1 recovery 系统介绍

默认 Tina 支持主系统 +recovery 系统的方式。

recovery 系统是一个带 initramfs 的 kernel。对应的配置文件是 target/allwinner/xxx/defconfig_ota

如果没有此文件，可以拷贝 defconfig 为 defconfig_ota，再做配置裁剪。

3.2.2 配置

对于主系统，使用 make menuconfig

对于 recovery 系统，使用 make ota_menuconfig

3.2.3 主系统和 recovery 都需要的 swupdate 包

选上 swupdate 包

```
Allwinner --> [*]swupdate
```

swupdate 中还有很多细分选项，一般用默认配置即可。需要的话可以做一些调整，比如裁剪掉网络部分。

3.2.4 主系统和 recovery 都需要的 wifimanager daemon

如果想从网络升级，则需要启动系统自动联网。

一种实现方式是，使用 wifimanager daemon 。当然，如果用户自己在脚本或应用中去做联网，则不需要此选项。

```
Allwinner
---> <*> wifimanager
        ---> [*] Enable wifimanager daemon support
                ---> <*> wifimanager-daemon-demo..... Tina wifimanager daemon demo
```

3.3 配置主系统

选上 swupdate 包, 目前默认配置支持本地升级和网络升级, 正常编译打包系统即可。

3.4 配置 recovery 系统

选上 ramdisk，建议使用 xz 压缩方式以节省 flash 空间。

```
make ota_menuconfig
    ---> Target Images
        ---> [*] ramdisk
            ---> Compression (xz)
```

选上 _recovery 后缀，避免编译 recovery 系统时，影响到主系统。

```
make ota_menuconfig
    ---> Target Images
        ---> [*] customize image name
            ---> Boot Image(kernel) name suffix (boot_recovery.img/boot_initramfs_recovery.img)
```

swupdate 使用的 recovery 系统配置文件为 defconfig_ota

```
target/allwinner/xxx/defconfig_ota
```

使用

```
swupdate_make_recovery_img
```

或手工调用

```
make -j16 TARGET_CONFIG=./target/allwinner/xxx/defconfig_ota
```

编译得到

```
out/xxx/boot_initramfs_recovery.img
```

3.5 配置 env

修改 boot_normal，增加 boot_partition 变量。

并使得 boot_normal 可根据 boot_partition 变量，启动主系统或 recovery 系统

即从

```
boot_normal=fatload sunxi_flash boot 40007fc0 uImage;bootm 40007fc0
```

改成

```
boot_partition=boot
boot_normal=fatload sunxi_flash ${boot_partition} 40007fc0 uImage;bootm 40007fc0
```

这一步的作用在于，让内核可以通过控制 boot_partition 来直接选择下次要启动的系统，无需 uboot 介入.

3.6 配置启动脚本

procd-init 是默认配置好的。

busybox-init 需要手工配置下。

参考《Tina System init 使用说明文档》, 拷贝

```
<tina>/package/busybox-init-base-files/files/etc/init.d/load_script.conf
```

到

```
<tina>/target/allwinner/<platform>/busybox-init-base-files/etc/init.d/
```

并在其中添加一行

```
swupdate_autorun
```

3.7 配置 OTA 包

3.7.1 方法 V0.2

3.7.1.1 swupdate_pack_swu

同步最新代码，在 build/envsetup.sh 中提供了一个 swupdate_pack_swu 函数。

可以参考该函数，自行实现一套打包 swupdate 升级包的脚本。也可以直接使用，使用方式如下 1. 准备好 sw-descrition 文件，具体作用和语法请参考 swupdate 说明文档
2. 准备好 sw-subimgs.cfg 文件，里面需要每一行列出一个打包需要的子镜像文件，即内核，rootfs 等。使用相对于 tina 根目录的相对路径进行描述。
3. 编译好所需的子镜像，例如主系统的内核和 rootfs，recovery 系统等。

4. 执行 swupdate_pack_swu 生成 swupdate 升级包

3.7.1.2 sw-description

文件

target/allwinner/cowbell-perf1/configs/sw-description

内容

```
software =
{
    version = "0.1.0";
    description = "Firmware update for Tina Project";

    stable = {
        boot = {
            images: (
                {
                    filename = "recovery"
                    device = "/dev/by-name/recovery";
                    installed-directly = true;
                },
                {
                    filename = "uboot"
                    type = "awuboot";
                },
                {
                    filename = "boot0"
                    type = "awboot0";
                }

            );

            bootenv: (
                {
                    name = "boot_partition";
                    value = "recovery";
                }
            );
        };

        recovery = {
            images: (
```

```
                {
                    filename = "kernel";
                    device = "/dev/by-name/boot";
                    installed-directly = true;
                },
                {
                    filename = "rootfs";
                    device = "/dev/by-name/rootfs";
                    installed-directly = true;
                }
            );
            bootenv: (
                {
                    name = "boot_partition";
                    value = "boot";
                },
                {
                    name = "swupdate_cmd";
                    value = "";
                }
            );
        };
    };
}
```

说明

升级过程会进行两次重启。具体的

（1）升级 recovery 分区 (recovery)，uboot(uboot)，boot0(boot0) 。设置 boot_partition 为 recovery

（2）重启，进入 recovery 系统

（3）升级内核 (kernel) 和 rootfs(rootfs) 。设置 boot_partition 为 boot

（4）重启，进入主系统，升级完成

3.7.1.3 sw-subimgs.cfg

文件

```
target/allwinner/cowbell-perf1/configs/sw-subimgs.cfg
```

内容

```
swota_file_list=(
target/allwinner/${TARGET_BOARD}/configs/sw-description
out/${TARGET_BOARD}/boot.img:kernel
out/${TARGET_BOARD}/rootfs.img:rootfs
out/${TARGET_BOARD}/boot_initramfs_recovery.img:recovery
out/${TARGET_BOARD}/image/boot_package.fex:uboot
out/${TARGET_BOARD}/image/boot0_nand.fex:boot0
)
```

说明

指明打包 swupdate 升级包所需的各个文件的位置。这些文件会被拷贝到 out 目录下，再生成 swupdate OTA 包

3.8 OTA 版本号

3.8.1 使用方式

在 sw-descriptionwen 文件中，会配置一个版本号字符串，如

```
software =
{
    version = "0.1.0";
    ...
}
```

如果需要在升级时，检查版本号，则可使用 -N 参数，传入的参数代表小机端当前的版本号。swupdate 会进行比较，如果 OTA 包中 sw-descriptionwen 文件配置的版本号小于当前版本号，则不允许升级。如何在小机端保存，获取，更新版本号，可以自定义。

3.8.2 例子

一种可选的方式是，保存到 env 中。

具体地，

1. 在烧录固件之前，可以在 env-x.x.cfg 中添加一行

```
swu_version=0.1.0
```

2. 之后想添加版本号，也可以在 sw-description 中，添加一个 swu_version 的 env 操作

```
software =
{
        version = "0.1.0";
        ...
                bootenv: (
                        ...
+ {
+ name = "swu_version";
+ value = "0.2.0";
                }
                ...
                );
        ...
}
```

注意，这么做的话，下次更新 sw-description 要修改两个位置，一处是最上面的 version = ``0.1.0"; 的版本号，一处是刚刚添加的这个写 env 的操作中的版本号。

2. 在小机端读取并将版本号传给 swupdate

假如小机端是用脚本调用，则可用如下方式读取并传给 swupdate

```
swu_version=$(fw_printenv -n swu_version)
swupdate ... -N $swu_version
```

3.9 OTA 签名校验

3.9.1 配置

swupdate 支持使用签名校验功能，需要在编译时选中对应功能。
出于安全考虑，一旦使能了校验，则 swupdate 不再支持不使用签名的更新调用。

```
make menuconfig --->
    Allwinner --->
        <*> swupdate --->
            [*] Enable verification of signed images
                Signature verification algorithm (RSA PKCS#1.5) --->(选择校验算法，此处以RSA为例)
```

注意，recovery 系统也需要对应进行配置，即

```
make ota_menuconfig --->
```

重复如上配置

3.9.2 使用方法

在 PC 端使用私钥签名 OTA 包
在小机端调用 swupdate 时，使用 -k 参数传入公钥

3.9.3 初始化 key

封装了一条命令，生成默认的密钥对。执行

```
swupdate_init_key
```

会在方案的 configs 目录下，使用默认密码和密钥对。
此步骤仅为方面调试使用，可跳过，自行生成密钥
正式产品中，需要自己重新生成密钥，并保护好私钥

3.9.4 修改 sw-description

在启动了签名校验的固件中，需要在为每个更新文件，添加 sha256 属性。
目前脚本支持自动在生成 OTA 包时，更新 sha256 的值。但需要在 sw-description 中，手工添加

```
sha256 = @文件名
```

如

```
$ git diff sw-description
diff --git a/allwinner/cowbell-perf1/configs/sw-description b/allwinner/cowbell-perf1/configs/sw-description
index ed04b64..467ac3b 100644
--- a/allwinner/cowbell-perf1/configs/sw-description
+++ b/allwinner/cowbell-perf1/configs/sw-description
@@ -9,14 +9,17 @@ software =
                        {
                                filename = "boot_initramfs_recovery.img"
                                device = "/dev/by-name/recovery";
+ sha256 = "@boot_initramfs_recovery.img"
                        },
                        {
                                filename = "boot_package.fex"
                                type = "awuboot";
+ sha256 = "@boot_package.fex"
                        },
                        {
                                filename = "boot0_nand.fex"
                                type = "awboot0";
+ sha256 = "@boot0_nand.fex"
                        }

                );
@@ -34,10 +37,12 @@ software =
                        {
                                filename = "boot.img";
                                device = "/dev/by-name/boot";
+ sha256 = "@boot.img"
                        },
                        {
                                filename = "rootfs.img";
                                device = "/dev/by-name/rootfs";
+ sha256 = "@rootfs.img"
                        }
                );
                bootenv: (
```

3.9.5 添加 sw-description.sig

签名的 OTA 包，需要生成签名文件 sw-description.sig，并使其在 OTA 中，紧随在 sw-description 后面。目前脚本中自动处理

3.9.6 生成 OTA 包

方法不变，脚本中会检测 defconfig 的配置，并自动完成签名等动作

3.9.7 将公钥放置到小机端

目前脚本中生成 key 的时候，自动拷贝了。如需手工处理，可参考如下方式

对于 procd-init

```
cconfigs
mkdir -p ../base-files
cp swupdate_public.pem ../busybox-init-base-files/etc/
```

对于 busybox-init

```
cconfigs
mkdir -p ../busybox-init-base-files/
cp swupdate_public.pem ../busybox-init-base-files/etc/
```

3.9.8 在小机端调用

在原本的命令基础上，加上 -k /etc/swupdate_public.pem 即可, 如

```
swupdate_cmd.sh -v -i /mnt/UDISK/tina-cowbell-perf1.swu -k /etc/swupdate_public.pem
```

3.10 调用 OTA

swupdate_cmd.sh , 用于给 swupdate 传入相关参数，切换更新状态，以及不断重试。

如果调用 swupdate 的时候加上 -p reboot , 则 swupdate 更新完毕后，会执行 reboot。

开机启动 progress，该程序会在后台运行，负责打印进度条并在 swupdate 更新完成后。

如果调用 progress 的时候加上 -r 参数，则 progress 会在检测到更新完成后，执行 reboot。

3.10.1 本地升级

将生成的 OTA 包推送到小机端，如放在/mnt/UDISK 目录下

PC 端执行

```
adb push out/cowbell-perf1/swupdate/tina-cowbell-perf1.swu /mnt/UDISK
```

小机端执行 (不带签名校验版本)

```
swupdate_cmd.sh -i /mnt/UDISK/tina-cowbell-perf1.swu -e stable,upgrade_recovery
```

小机端执行 (带签名校验版本)

```
swupdate_cmd.sh -i /mnt/UDISK/tina-cowbell-perf1.swu -k /etc/swupdate_public.pem -e stable,upgrade_recovery
```

3.10.2 网络升级

启动服务器

```
cd out/cowbell-perf1/swupdate/
sudo python -m SimpleHTTPServer 80 #启动一个服务器
```

小机端命令，使用 -d -uxxx，xxx 为 url，

例如 (不带签名校验版本)

```
swupdate_cmd.sh -d -uhttp://192.168.35.112/tina-cowbell-perf1.swu -e stable,upgrade_recovery
```

例如 (带签名校验版本)

```
swupdate_cmd.sh -d -uhttp://192.168.35.112/tina-cowbell-perf1.swu -k /etc/swupdate_public.pem -e stable,upgrade_recovery
```

注：需依赖外部程序，提供自动联网支持。OTA 本身不处理联网。

3.11 裁剪

swupdate 本身是可配置的, 不需要某些功能时，可将其裁剪掉。

```
make menuconfig --->
    Allwinner --->
        <*> swupdate --->
```

例如，不需要使用 swupdate 来从网络下载 OTA 包的话，则可将

```
[*] Enable image downloading
```

取消掉。

不需要更新 boot0/uboot 的话，则将

```
Image Handlers --->
    [*] allwinner boot0/uboot
```

取消掉

3.12 调试

3.12.1 直接调用 swupdate

目前 swupdate_cmd.sh 主要有两个作用

1. 自启动，无限重试
2. 在主系统和 recovery 系统中，传入不同的 -e 参数给 swupdate

可以不使用 swupdate_cmd.sh，手工直接调用 swupdate，在后面加上合适的 -e 参数，观察输出 log

如

```
swupdate -v -i xxx.swu -e stable,boot
swupdate -v -i xxx.swu -e stable,recovery
```

3.12.2 手工切换系统

按上述 env 的配置，启动的系统，是由 boot_partition 变量控制的

注意，需要/var/lock 目录存在且可写

切换到主系统

```
fw_setenv boot_partition boot
reboot
```

切换到 recovery 系统

```
fw_setenv boot_partition recovery
reboot
```

观察当前变量

```
fw_printenv
```

3.12.3 更新 boot0/uboot

目前更新 boot0，uboot 实际功能是由另一个软件包 ota-burnboot 完成的，swupdate 只是准备数据，并调用 ota-burnboot 提供的动态库。
如果更新失败，先尝试手工使用 ota-burnboot0 xxx 和 ota-burnuboot xxx 能否正常更新。以确定是 ota-burnboot 的问题，还是 swupdate 的问题

3.13 测试固件

3.13.1 生成方式

一般而言，测试需要两个有差异的 OTA 包，如，uboot 和 kernel 的 log 有差异，rootfs 的文件有差异。
这样方法测试人员根据 log 判断是否升级成功。

3.13.1.1 准备工作

如果需要网络更新，OTA 不负责联网，所以需要选上 wifimanager-daemon

```
Allwinner
---> <*> wifimanager
        ---> [*] Enable wifimanager daemon support
```

```
---> <*> wifimanager-daemon-demo..... Tina wifimanager daemon demo
```

3.13.1.2 生成固件 1 和 OTA 包 1

```
重新编译boot，使得编译时间更新
```

mboot

```
创建文件以标记rootfs
```

cd target/allwinner/cowbell-ailabs_c1c/base-files rm -f OTA1 OTA2 echo OTA1 > OTA1

```
重新编译recovery系统
```

swupdate_make_recovery_img

```
重新编译打包，使得编译时间更新
```

mp -j32

```
生成OTA包1
```

swupdate_pack_swu

```
得到产物
```

cp out/cowbell-perf1/tina_cowbell-perf1_uart0.img tina_cowbell-perf1_uart0_OTA1.img cp out/cowbell-perf1/swupdate/tina-cowbell-perf1.swu tina-cowbell-perf1_OTA1.swu

```
####3.13.1.3 生成固件2和OTA包2
```

重新编译 uboot，使得编译时间更新

```
muboot
```

创建文件以标记 rootfs

```
cd target/allwinner/cowbell-ailabs_c1c/base-files
rm -f OTA1 OTA2
echo OTA2 > OTA2
```

重新编译 recovery 系统

```
swupdate_make_recovery_img
```

重新编译打包，使得编译时间更新

```
mp -j32
```

生成 OTA 包 2

```
swupdate_pack_swu
```

得到产物

```
cp out/cowbell-perf1/tina_cowbell-perf1_uart0.img tina_cowbell-perf1_uart0_OTA2.img
cp out/cowbell-perf1/swupdate/tina-cowbell-perf1.swu tina-cowbell-perf1_OTA2.swu
```

3.13.2 使用方式

任意选择一个 OTA 固件烧录后，可在此基础上进行本地升级或网络升级。

3.13.2.1 本地升级方式

PC 端执行

```
adb push out/astar-parrot/swupdate/tina-cowbell-perf1_OTA1.swu /mnt/UDISK
```

小机端执行

```
swupdate_cmd.sh -i /mnt/UDISK/tina-cowbell-perf1_OTA1.swu
```

3.13.2.2 网络升级方式

PC 端搭建服务器

```
sudo python -m SimpleHTTPServer 80
```

小机端联网

```
wifi_connect_ap_test SSID 密码
```

小机端执行

```
swupdate_cmd.sh -d -uhttp://192.168.xxx.xxx/tina-cowbell-perf1_OTA1.swu
```

注明：启动后会自动联网，连网后等待 OTA 后台脚本尝试更新。中途掉电重启后，正常会在启动后几十秒内，成功联网并开始继续更新。

3.13.2.3 升级过程

升级过程会进行两次重启。具体的

（1）升级 recovery 分区 (boot_initramfs_recovery.img)，uboot(boot_package.fex)，boot0(boot0_nand.fex)

（2）重启，进入 recovery 系统

（3）升级内核 (boot,img) 和 rootfs(rootfs.img)

（4）重启，进入主系统，升级完成

3.13.2.4 判断升级

从 log 中的时间和 rootfs 文件可以判断当前运行的版本。

例如

```
OTA_1:
boot0：[66]HELLO! BOOT0 is starting Dec 29 2018 16:15:59!
uboot：U-Boot 2018.05-00015-g5068c23-dirty (Dec 29 2018 - 16:15:41 +0800) Allwinner Technology
kernel：[ 0.000000] Linux version 4.9.118 (zhuangqiubin@Exdroid5) (gcc version 6.4.1 (OpenWrt/Linaro GCC 6.4-2017.11 2017-11) ) #189 SMP Sat
    Dec 29 08:13:38 UTC 2018 （注，进入控制台cat /proc/version 也可看到）
rootfs：ls查看下，在根目录下有一个文件 OTA1

OTA_2:
boot0: [66]HELLO! BOOT0 is starting Dec 29 2018 16:17:35!
uboot: U-Boot 2018.05-00015-g5068c23-dirty (Dec 29 2018 - 16:17:17 +0800) Allwinner Technology
kernel:[ 0.000000] Linux version 4.9.118 (zhuangqiubin@Exdroid5) (gcc version 6.4.1 (OpenWrt/Linaro GCC 6.4-2017.11 2017-11) ) #190 SMP Sat
    Dec 29 08:18:33 UTC 2018（注，进入控制台cat /proc/version 也可看到）
rootfs：ls查看下，在根目录下有一个文件 OTA2
```

4. Tina upgrade app 介绍

4.1 功能简介

有些客户，需要单独更新应用程序。一种解决方式是，将应用程序单独放到一个分区中，并在启动时挂载该分区。为了保证更新过程掉电重启，仍有可用的应用程序，可设置两个应用分区，并配合环境变量等选择挂载。

4.2 应用源码

需修改应用源码的 Makefile，将应用涉及文件，全部安装到/mnt/app 路径
如果应用是二进制形式，则请放到

```
target/allwinner/xxx/busybox-init-base-files/mnt/app 目录
```

4.3 menuconfig 设置

选中

```
make menuconfig ---> Target Images ---> [*] Separate /mnt/app from rootfs
```

使得/mnt/app 目录，从 rootfs 中分离出来。打包的时候，会被制作成一个单独的文件 app.fex 选中

```
make menuconfig ---> Allwinner ---> <*> tina-app-upgrade
```

选中后，可使用

```
/sbin/aw_upgrade_dual_app.sh
```

进行 OTA

4.4 分区设置

增加两个分区，名字固定为 app 和 app_sub，downloadfile 固定为 app.fex，size 根据实际情况调整

```
[partition]
    name = app
    size = 51200
    downloadfile = "app.fex"
    user_type = 0x8000

[partition]
    name = app_sub
    size = 51200
    downloadfile = "app.fex"
    user_type = 0x8000
```

4.5 env 设置

在 env 中

```
target/allwinner/xxx/configs/env-xxx.cfg
```

增加配置

```
appAB=A
#set applimit=0 to disable appcount check
applimit=0
appcount=0
```

其中 appAB 指定要启动时要挂载哪个分区

```
appAB=A 挂载/dev/by-name/app到/mnt/app
appAB=B 挂载/dev/by-name/app到/mnt/app
```

applimit 和 appcount 用于支持应用的自动回退功能

4.6 自动回退

applimit=0 时，没有任何作用
applimit 非 0 时，会在每次启动，递增 appcount 值，并检测 appcount 是否大于 applimit，若大于，则切换挂载另一个 app 分区。
例如，当前挂载/dev/by-name/app，设置 applimit=2，appcount=0
则每次启动，appcount 加一，两次启动后，appcount ＝ 2 ，再次重启，appcount+1=3>applimit，超过限制，自动改成挂载/dev/by-name/app_sub
这个功能主要是要解决，更新了一个有问题的应用，导致无法正常启动应用的问题。例如
当前处于 app，OTA 更新了一个有问题的新应用到 app_sub 分区，重启，重启后无法正常启动 app_sub 中的新应用。则 applimit 次重启后，会自动改回使用 app 分区中，旧的可用的应用。
使用此功能，需要应用在正常启动后，主动使用

```
fw_setenv appcount 0
```

清空计数值，避免累积到 applimit 切换分区

4.7 OTA 步骤

4.7.1 生成 OTA 包

tina 目录下，执行

```
make_ota_package_for_dual_app
```

生成 OTA 包

```
out/xxx/ota_dual_app/app_ota.tar
```

4.7.2 下载 OTA 包

通过网络或 ADB 等方式，将 app_ota.tar 放到小机端/mnt/UDISK/app_ota.tar

4.7.3 执行 OTA

执行脚本

```
aw_upgrade_dual_app.sh
```

4.8 调试

生成 OTA 包的函数，位于

```
build/envsetup.sh
function make_ota_package_for_dual_app()
```

可从生成的 tar 文件中，将 app.fex 解出来

```
tar -xf out/xxx/ota_dual_app/app_ota.tar
```

解压得到的 app.fex，是一个 ext4 文件系统，可在 linux 主机，或推送到小机端，挂载，查看文件系统中的内容

```
mkdir app
mount app.fex app
ls aaa
```

5. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.