Tina Linux OTA
开发指南

版本号: 2.6
发布日期: 2021.04.12

版本历史

版本号	日期	制/修订人	内容描述
1.0	2019.07.03	AWA1531	初始版本。
1.1	2020.01.01	AWA1046	补充例子。
1.2	2020.03.10	AWA1046	修改路径，补充 readback, ubi, rdiff 等说明。
1.3	2020.04.27	AWA1046	完善升级 bootloader 的说明。
1.4	2020.05.14	AWA1046	介绍 rdiff 和 ubi 的配合。
1.5	2020.06.19	AWA1046	补充 awboot 说明，修复格式，完善描述。
1.6	2020.06.28	AWA1046	补充 misc-upgrade 升级 boot0/uboot 说明，修改 busybox-init 说明。
1.7	2020.07.08	AWA1046	补充一些升级 boot0/uboot 的说明。
1.8	2020.07.14	AWA1046	补充 swupdate 定制分区说明和其他细节说明。
1.9	2020.09.09	AWA1046	完善 rdiff 和 readback 介绍，补充错误判断的介绍
2.0	2020.09.18	AWA1046	介绍系统切换
2.1	2020.10.12	AWA1046	介绍压缩特性
2.2	2020.12.15	AWA1046	补充 rsync 库的介绍
2.3	2021.01.12	AWA1046	补充 busybox-init 的说明
2.4	2021.01.14	AWA1046	补充 env-redund 分区说明
2.5	2021.03.08	AWA1046	补充 emmc boot0/uboot 的具体偏移
2.6	2021.04.12	AWA1615	添加升级失败问题排查

版权所有 © 珠海全志科技股份有限公司。保留一切权利

目　录

ALLWINNER®

版权所有 © 珠海全志科技股份有限公司。保留一切权利

版权所有 © 珠海全志科技股份有限公司。保留一切权利

版权所有 © 珠海全志科技股份有限公司。保留一切权利

版权所有 © 珠海全志科技股份有限公司。保留一切权利

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1 概述

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。

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。

1.2 适用范围

Allwinner 软件平台 Tina。

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。

1.4 OTA 方案

1.4.1 recovery 系统方案

recovery 系统方案，是在主系统之外，增加一个 recovery 系统。升级时，主系统负责升级 recovery 系统，recovery 系统负责升级主系统。

这样如果升级中途发生掉电，也不会影响当前正在使用的这个系统。重启后仍可正常进入系统，继续完成升级。

一般 recovery 系统会使用 intiramfs 功能，并大量裁剪不必要的应用，只保留 OTA 必需的功能，把 size 尽量减小。

recovery 系统方案优点：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1. recovery 系统可以做得比较小，省 flash 空间。

recovery 系统方案缺点：

1. recovery 系统一般不包含主应用，所以 OTA 期间，处于 recovery 系统中时，无法为用户正常提供服务。
2. 需要重启两次。
3. 需要维护两份系统配置，即主系统和 recovery 系统。

1.4.2 AB 系统方案

AB 系统方案，是将原有的系统，增加一份。即 flash 上总共有 AB 两套系统。两套系统互相升级。OTA 时，若当前运行的是 A 系统，则升级 B 系统，升级完成后，设置标志，重启切换到 B 系统。OTA 时，若当前运行的是 B 系统，则升级 A 系统，升级完成后，设置标志，重启切换到 A 系统。

AB 系统方案优点：

1. 更新过程是在完整系统中进行的，更新期间可正常提供服务，用户无感知。最终做一次重启即可。
2. 逻辑简单，只重启一次。
3. 只维护一套系统配置。

AB 系统方案缺点：

1. flash 占用较大。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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

表 2-1: ota-burnboot 相关概念说明表

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。linux5.4 开始不再合并到 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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

即，本文介绍的升级 uboot，其实是升级 uboot+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 boot0 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。
$ mboot0
    => 编译 boot0 (注意，此命令在大多数平台无效，因为boot0不开源，SDK中提供了编译好的bin文件)。
```

编译后会自行拷贝 bin 文件到该平台的目录下，即：

对于 tina3.5.0 及之前版本，路径为：

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

对于 tina3.5.1 及之后版本，路径为：

```
device/config/chips/${CHIP}/bin
```

编译出的 boot0/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 boot0/uboot/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。
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.3.2 关于更新 boot0

大多数平台，代码环境中并不包含 boot0 相关代码, 因此无法编译 boot0。

一般情况下并不需要修改 boot0, 而是直接使用提供的 boot0 的 bin 文件即可。

少部分平台提供了可编译的 boot0 代码，可使用 mboot0 编译。

2.3.3 Bin 文件路径

2.3.3.1 使用 uboot2011 的非安全方案

以 R16 的 astar-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。
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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 boot0-image 是镜像的路径。

请注意，安全和非安全方案所使用的 boot0-image 是不同的，具体见“用于更新的 bin 文件”章节。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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 uboot-image 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 uboot-image 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)

表 2-2: OTA_burnboot0 函数说明表

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)

版权所有 © 珠海全志科技股份有限公司。保留一切权利

表 2-3: OTA_burnuboot 函数说明表

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 NFTL 方案实现

在 nand nftl 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 Nand Flash UBI 方案实现

在 nand ubi 方案中，boot0 一般存放于 mtd0 中，uboot 存放于 mtd1 中。

与 nftl 方案一样，底层实际是保存多份 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。对上提供多份统一的更新接口，软件包会通过对 mtd 的 iotcl 接口发起更新。

注：用户空间直接读写/dev/mtdx 节点，需要内核使能 CONFIG_MTD_CHAR=y。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.6.4 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。需要读写可直接操作/dev/mmcblk0 节点的指定偏移。

具体位置:

```
1 sector = 512 bytes = 0.5k。

boot0/toc0 保存了两份，offset1: 16 sector,  offset2: 256 sector。

uboot/toc1 保存了两份，offset1: 32800 sector, offset2: 24576 sector。
```

启动时会先读取 offset1，如果完整性校验失败，则读取 offset2。

更新时, 默认只更新 offset1, 而 offset2 是保持在出厂状态的。只要 offset1 正常更新了, 则启动时会优先使用。如果在更新 offset1 的过程中掉电导致数据损坏, 则自动使用 offset2 进行启动。

如需定制策略，例如改成每次 offset1 和 offset2 均更新，可自行修改 ota-burnboot 代码。

2.6.5 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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 Tina SWUpdate OTA 介绍

3.1 swupdate 介绍

3.1.1 简介

SWUpdate 是一个开源的 OTA 框架，提供了一种灵活可靠的方式来更新嵌入式系统上的软件。

官方源码：

```
https://github.com/sbabic/swupdate
```

官方文档：

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

非官方翻译的中文文档：

```
https://zqb-all.github.io/swupdate/
```

源码自带文档：

```
解压 tina/dl/swupdate-xxx.tar.xz ，解压后的 doc 目录下即为此版本源码附带的文档。
```

社区论坛：

```
https://groups.google.com/forum/#!forum/swupdate
```

3.1.2 移植到 tina 的改动

移植到 tina 主要做了以下修改：

- 位置在 package/allwinner/swupdate。
- 仿照 busybox，添加了配置项，可通过 make menuconfig 直接配置。
- 添加 patch，支持了更新 boot0,uboot。
- 添加了自启动脚本。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- 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。脚本介绍详见后续章节。

3.2 配置

3.2.1 recovery 系统介绍

若选用主系统 +recovery 系统的方式，则需要一个 recovery 系统。

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 系统配置命令

对于主系统，使用：

```
make menuconfig
```

配置结果保存在：

```
target/allwinner/xxx/defconfig
```

对于 recovery 系统，使用：

```
make ota_menuconfig
```

配置结果保存在：

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

3.2.3 主系统和 recovery 都需要的 swupdate 包

选上 swupdate 包。

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

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

swupdate 会依赖选中 uboot-envtools 包，以提供用户空间读写 env 分区的功能。

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
```

3.2.5 配置主系统

就以上提到的几个包，暂时没有只针对主系统的需要选的包。

3.2.6 编译主系统

正常 make 即生成主系统。

```
make
```

3.2.7 配置 recovery 系统

对于 recovey 系统，需要选上 ramdisk，同时建议使用 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.
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    img)
            ---> Rootfs Image name suffix (rootfs_recovery.img)
```

3.2.8 编译 recovery 系统

要编译生成 recovery 系统，可使用：

```
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
```

3.2.9 配置 env

本方案推荐使用 env 来保存信息，不使用 misc 分区。

uboot 会从 env 分区读取启动命令，并根据启动命令来启动系统。只要我们能在用户空间改动到 env，即可控制下次启动的系统。

3.2.9.1 boot_partition 变量

增加一个 boot_partition 变量，用于指定要启动的内核所在分区。

配置 env 主要是修改 boot_normal 命令，将要启动的分区独立成 boot_partition 变量。

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
```

这样可以通过控制 boot_partition 来直接选择下次要启动的系统，无需 uboot 介入。uboot 只需按照 boot_normal 启动即可。

对于 recovery 方案，可设置 boot_partition 为 boot 或 recovery。OTA 切换系统时，只需要改变此变量即可达到切换主系统和 recovery 系统的目的。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

对于 AB 系统方案，可设置为 boot_partition 为 bootA 或 bootB。OTA 切换系统时，只需要改变此变量即可达到切换 kernel 的目的。

3.2.9.2 root_partition 变量

增加一个 root_partition 变量，用于指定要启动的 rootfs 所在分区。

uboot 会解析分区表，找出此变量指定的分区并在 cmdline 中指定 root 参数。

例如，在 env 中设置：

```
root_partition=rootfs
```

则启动时 uboot 会遍历分区表，找到名字为 rootfs 的分区，假设找到的分区为/dev/nand0p4，则在 cmdline 中增加 root=/dev/nand0p4。

kernel 需要挂载 rootfs 时，取出 root 参数，则得知需要挂载/dev/nand0p4 分区。

对于 recovery 方案，就一直设置 root_partition 为 rootfs 即可。主系统需要从 rootfs 分区读取数据，而 recovery 系统使用 initramfs，无需从 rootfs 分区读取数据即可正常运行 OTA 应用等。当然，recovery 系统中要更新 rootfs 的话，还是会访问 (写入)rootfs 分区的，但这个动作就跟 env 的 root_partition 无关了。

对于 AB 系统方案，可设置 root_partition 为 rootfsA 或 rootfsB，以匹配不同的系统。OTA 切换系统时，只需要改变此变量即可达到切换 rootfs 的目的。

3.2.10 配置备份 env

由于写入 env 时断电，可能导致 env 的数据被破坏，因此需要支持备份 env。

3.2.10.1 方式一：env 分区扩展为存放两份 env

此方式是在 uboot 中进行定制实现，非社区原生方案。

可在uboot源码中搜索CONFIG_SUNXI_ENV_NOT_BACKUP, 若存在则说明支持此功能。

支持此功能后，只要uboot不配置CONFIG_SUNXI_ENV_NOT_BACKUP，则此功能默认开启。uboot会将env数据在同一分区中进行备份。

启用方法：

1. 修改分区表，将 env 分区扩大到 128k*2=256k。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

工作方式：

uboot 检测到 env 分区足够大，则激活 env 备份功能，认为 0-128k 存放第一份 env 数据，128k-256k 存放第二份 env 数据。由于 env 数据本身带有 CRC 校验，所以可判断一份 env 是否完整。启动时，uboot 会对两份 env 进行同步，若某一份损坏则取另一份进行覆盖，若两份均完整，则以第一份为准。

对于用户空间的 fw_printenv，默认只会更新第一份 env，即执行 fw_setenv 之后两份 env 就有差异了，要到下次启动才由 uboot 进行同步。若更新 env 的过程中发生掉电，则第一份 env 不完整，重新启动时，uboot 会识别到并用第二份 env 覆盖第一份。

确认是否生效：

1. 直接观察。

用户空间控制台执行：

```
hexdump -C /dev/by-name/env
```

确认是否存在两份。

2. 尝试破坏 env。

```
dd if=/dev/zero of=/dev/by-name/env bs=1k count=1
```

损坏第一份 env，重启看能否正常启动，并尝试

```
fw_printenv
hexdump -C /dev/by-name/env
```

确认 env 分区数据是否正常。

3.2.10.2 方式二：增加 env-redund 分区

此方式是 uboot 原生功能。虽然也需要修改 sunxi 的 env 读取代码进行适配，但总体读写逻辑是社区原生的。

启用方法：

1. 增加 env-redund 分区。

将 env 分区复制一份，分区名改为 env-redund。

注意只是分区名修改为 env-redund，其 downloadfile 仍然指定为 env.fex

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2. 支持在打包时制作冗余 env。

```
make menuconfig --> Global build settings --> [*] sunxi make redundant env data
```

注意事项：

启用上述选项之后，打包时会调用 mkenvimage 工具来制作 env，对 env 的格式有一定要求。如注释和有效配置不能合并在一行。

若 env-x.x.cfg 中存在类似如下配置

```
bootcmd=run setargs_nand boot_normal#default nand boot
```

则需要改成：

```
#default nand boot
bootcmd=run setargs_nand boot_normal
```

3. 配置 uboot 并重新编译 uboot bin。

以 r328 spinand 方案为例。

在

```
lichee/brandy-2.0/u-boot-2018/configs/sun8iw18p1_defconfig
```

中增加配置：

```
CONFIG_SUNXI_REDUNDAND_ENVIRONMENT=y
```

重新编译 uboot。

4. 修改 fw_env.config

拷贝

```
package/utils/uboot-envtools/files/fw_env.config
```

到

```
target/allwinner/<board>/base-files/etc/  (若使用procd-init)
target/allwinner/<board>/busybox-init-base-files/etc/  (若使用busybox-init)
```

修改拷贝后的 fw_env.config，增加备份 env 的配置。

例如原本最后一行为：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
/dev/by-name/env        0x0000          0x20000
```

则增加一行：

```
/dev/by-name/env-redund        0x0000          0x20000
```

这样用户空间的 fw_printenv 和 fw_setenv 即可正确处理两份 env。

3.2.11 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
```

3.3 OTA 包

OTA 包中，需要包含 sw-description 文件，以及本次升级会用到的各个文件，例如 kernel，rootfs。

整个 OTA 包是 cpio 格式，且要求 sw-description 文件在第一个。

3.3.1 OTA 策略描述文件：sw-description

sw-description 文件是 swupdate 官方规定的，OTA 策略的描述文件，具体语法可参考 swupdate 官方文档。

tina 提供了几个示例：

```
target/allwinner/generic/swupdate/sw-description-ab
target/allwinner/generic/swupdate/sw-description
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

也可以自行为具体的方案编写描述文件：

```
target/allwinner/<board>/swupdate/sw-description
```

本文件在 SDK 中的存放路径和名字没有限定，只要最终打包进 OTA 包中，重命名为 sw-description 并放在第一个文件即可。

3.3.2 OTA 包配置文件：sw-subimgs.cfg

sw-subimgs.cfg 是 tina 提供的，用于指示如何生成 OTA 包。

基本格式为

```
swota_file_list=(
#表示把文件xxx拷贝到swupdate目录下，重命名为yyy，并把yyy打包到最终的OTA包中
xxx:yyy
)

swota_copy_file_list=(
#表示把文件xxx拷贝到swupdate目录下，重命名为yyy，但不把yyy打包到最终的OTA包中
xxx:yyy
)
```

swota_copy_file_list 存在的原因是，有一些文件我们只需要其 sha256 值，而不需要文件本身。例如使用差分包配合 readback handler 时，readback handler 需要原始镜像的 sha256 值用于校验。

例子:

```
swota_file_list=(
#将target/allwinner/generic/swupdate/sw-description-ab-sign拷贝成sw-description，后续同理。
target/allwinner/generic/swupdate/sw-description-ab-sign:sw-description
out/${TARGET_BOARD}/uboot.img:uboot
out/${TARGET_BOARD}/boot0.img:boot0
out/${TARGET_BOARD}/image/boot.fex.gz:kernel.gz
out/${TARGET_BOARD}/image/rootfs.fex.gz:rootfs.gz
out/${TARGET_BOARD}/image/rootfs.fex.zst:rootfs.zst
)
```

tina 提供了几个示例：

```
target/allwinner/generic/swupdate/sw-subimgs.cfg                # 普通系统，recovery系统，整包升
    级。这是其余demo的基础版本。
target/allwinner/generic/swupdate/sw-subimgs-ab.cfg             # 改为AB系统。
target/allwinner/generic/swupdate/sw-subimgs-secure.cfg         # 改为安全系统。
target/allwinner/generic/swupdate/sw-subimgs-ab-rdiff.cfg       # 改为AB方案，差分方案。
target/allwinner/generic/swupdate/sw-subimgs-readback.cfg       # 增加回读校验。
target/allwinner/generic/swupdate/sw-subimgs-sign.cfg           # 增加签名校验。
target/allwinner/generic/swupdate/sw-subimgs-ubi.cfg            # 改为ubi方案。
```

也可以自行为具体的方案编写描述文件。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
target/allwinner/<board>/swupdate/sw-subimgs.cfg
```

本文件在 SDK 中的路径需位于 target/allwinner/<board>/swupdate 目录下，或 target/allwinner/generic/swupdate 目录下。

名字需要命名为 sw-subimg.cfg 或 sw-subimgsxxx.cfg，其中 xxx 可自定义。

这个限定主要是为了方便打包函数处理。在打包时，命令行传入参数 xxx，则会使用 sw-subimgsxxx.cfg 进行打包。

3.3.3 OTA 包生成：swupdate_pack_swu

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。可以使用冒号分隔，前面为 SDK 中的文件，后面为打包进 OTA 包的文件名。若没有冒号则使用原文件名字。使用相对于 tina 根目录的相对路径进行描述。其中第一个必须为 sw-description。

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。不带参数执行，则会在特定路径下寻找 sw-subimgs.cfg，解析配置生成 OTA 包。带参数-xx 执行，则会在特定路径下寻找 sw-subimgs-xx.cfg，解析配置生成 OTA 包。例如执行 swupdate_pack_swu -sign，则会寻找 sw-subimgs-sign.cfg，如此方便配置多个不同用途的 sw-subimgs-xx.cfg。

注：不同介质使用的 boot0/uboot 镜像不同，swupate_pack_swu 需要 sys_config.fex 中的 storage_type 配置明确指出介质类型，才能取得正确的 boot0.img 和 uboot.img 具体可直接查看 build/envsetup.sh 中 swupdate_pack_swu 的实现。

3.4 recovery 系统方案举例

3.4.1 配置分区和 env

在分区表中，增加一个 recovery 分区，用于保存 recovery 系统。

size 根据实际 recovery 系统的大小，再加点裕量。

download_file 可以留空，因为 OTA 第一步就是写入一个 recovery 系统。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

当然也可以配置上 download_file，并在打包固件之前先编译好 recovery 系统，一并打包到固件中，这样出厂就带 recovery 系统，后续的 OTA 执行过程，可以考虑不写入 recovert 系统，用现成的，直接重启并升级主系统。

在 env 中指定：

```
boot_partition=boot
root_partition=rootfs
```

并配置 boot_normal 命令，从 $boot_partition 变量指定的分区加载系统。

3.4.2 配置主系统

lunch 选择方案后, make menuconfig, 选上 swupdate。

3.4.3 配置 recovery 系统

假设没有现成的 recovery 系统配置，则我们从主系统配置修改得到。lunch 选择方案后, 拷贝配置文件。

```
cdevice
cp defconfig defconfig_ota
```

根据上文介绍，make ota_menuconfig 选上 swupdate, ramdisk, recovery 后缀等必要的配置。

recovery 系统整个运行在 ram 中，如果系统过大会无法启动，所以需要进行裁剪。make ota_menuconfig, 将不必要的包尽量从 recovery 系统中去掉。

3.4.4 准备 sw-description

这里我们直接使用：

```
target/allwinner/generic/swupdate/sw-description
```

内容如下，中文部分是注释，原文件中没有。

```
/* 固定格式，最外层为software = { } */
software =
{
    /* 版本号和描述 */
    version = "0.1.0";
    description = "Firmware update for Tina Project";

    /*
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
     * 外层tag，stable，
     * 没有特殊含义，就理解为一个字符串标志即可。
     * 可以修改，调用的时候传入匹配的字符串即可
     */
    stable = {

        /*
         * 内层tag，upgrade_recovery，
         * 当调用swupdate xxx -e stable,upgrade_recovery时，就会匹配到这部分，执行 {} 内的动作，
        * 可以修改，调用的时候传入匹配的字符串即可
         */
         /* upgrade recovery,uboot,boot0 ==> change swu_mode,boot_partition ==> reboot */
         upgrade_recovery = {
             /* 这部分是为了在主系统中，升级recovery系统，升级uboot和boot0 */
             /* upgrade recovery */
             images: (  /* 处理各个image */
                 {
                     filename = "recovery";   /* 源文件是OTA包中的recovery文件 */
                     device = "/dev/by-name/recovery"; /* 要写到/dev/by-name/recovery节点中，这
个节点在tina上就对应recovery分区 */
                     installed-directly = true; /* 流式升级，即从网络升级时边下载边写入，而不是先完
整下载到本地再写入，避免占用额外的RAM或ROM */
                 },
                 {
                     filename = "uboot"; /* 源文件是OTA包中的uboot文件 */
                     type = "awuboot"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
                 },
                 {
                     filename = "boot0"; /* 源文件是OTA包中的boot0文件 */
                     type = "awboot0"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
                 }
             );
             /* image处理完之后，需要设置一些标志，切换状态 */
             /* change swu_mode to upgrade_kernel,boot_partition to recovery & reboot*/
             bootenv: ( /* 处理bootenv，会修改uboot的env分区 */
                 {
                     /* 设置env:swu_mode=upgrade_kernel，这是为了记录OTA进度 */
                     name = "swu_mode";
                     value = "upgrade_kernel";
                 },
                 {
                     /* 设置env:boot_partition=recovery，这是为了切换系统，下次uboot就会启动
recovery系统(kernel位于recovery分区) */
                     name = "boot_partition";
                     value = "recovery";
                 },
                 {
                     /* 设置env:swu_next=reboot，这是为了跟外部脚本配合，指示外部脚本做reboot动作 */
                     name = "swu_next";
                     value = "reboot";
                 }
                 /* 实际有什么其他需求，都可以灵活增删标志来解决，外部脚本和应用可通过fw_setenv/
fw_printenv操作env */
                 /* 注意，以上几个env，是一起在ram中修改好再写入的，不会出现部分写入部分未写入的情况 */
             );
         };

        /*
         * 内层tag，upgrade_kernel，
         * 当调用swupdate xxx -e stable,upgrade_kernel时，就会匹配到这部分，执行 {} 内的动作，
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
        * 可以修改，调用的时候传入匹配的字符串即可。
        */
        /* upgrade kernel,rootfs ==> change sw_mode */
        upgrade_kernel = {
            /* upgrade kernel,rootfs */
            /* image部分，不赘述 */
            images: (
                {
                    filename = "kernel";
                    device = "/dev/by-name/boot";
                    installed-directly = true;
                },
                {
                    filename = "rootfs";
                    device = "/dev/by-name/rootfs";
                    installed-directly = true;
                }
            );
            /* change sw_mode to upgrade_usr,change boot_partition to boot */
            bootenv: (
                {
                    /* 设置env:swu_mode=upgrade_usr, 这是为了记录OTA进度 */
                    name = "swu_mode";
                    value = "upgrade_usr";
                },
                {
                    /* 设置env:boot_partition=boot, 这是为了切换系统，下次uboot就会启动主系统(
    kernel位于boot分区) */
                    name = "boot_partition";
                    value = "boot";
                }
            );
        };

       /* 内层tag，upgrade_usr,
          当调用swupdate xxx -e stable,upgrade_usr时，就会匹配到这部分，执行 {} 内的动作，
          可以修改，调用的时候传入匹配的字符串即可 */
        /* upgrade usr ==> clean ==> reboot */
        upgrade_usr = {
            /*
             * misc-upgrade的小容量方案，将usr拆成独立分区了。
            * 这里我们不需要，如果保留的话，不做任何image操作即可。
             * 也可以彻底删除这一部分，并将上面的upgrade_usr改掉。
            */
            /* upgrade usr */

            /* OTA结束，清空各种标志 */
            /* clean swu_param,swu_software,swu_mode & reboot */
            bootenv: (
                {
                    name = "swu_param";
                    value = "";
                },
                {
                    name = "swu_software";
                    value = "";
                },
                {
                    name = "swu_mode";
                    value = "";
```

ALLWINNER®

```
                },
                {
                    name = "swu_next";
                    value = "reboot";
                }
            );
        };

    };

    /* 当没有匹配上面的tag，进入对应的处理流程时，则运行到此处。我们默认清除掉一些状态 */
    /* when not call with -e xxx,xxx    just clean */
    bootenv: (
        {
            name = "swu_param";
            value = "";
        },
        {
            name = "swu_software";
            value = "";
        },
        {
            name = "swu_mode";
            value = "";
        },
        {
            name = "swu_version";
            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。

3.4.5 准备 sw-subimgs.cfg

我们直接看下 tina 默认的：

```
target/allwinner/generic/swupdate/sw-subimgs.cfg
```

内容如下，中文部分是注释，原文件中没有。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
swota_file_list=(
#取得sw-description，放到OTA包中。
#注意第一行必须为sw-description。如果源文件不叫sw-description，可在此处加:sw-description做一次重命名
target/allwinner/generic/swupdate/sw-description
#取得boot_initramfs_recovery.img，重命名为recovery，放到OTA包中。以下雷同
out/${TARGET_BOARD}/boot_initramfs_recovery.img:recovery
#uboot.img和boot0.img是执行swupdate_pack_swu时自动拷贝得到的，需配置sys_config.fex中的
    storage_type
out/${TARGET_BOARD}/uboot.img:uboot
#注：boot0没有修改的话，以下这行可去除，其他雷同，可按需升级
out/${TARGET_BOARD}/boot0.img:boot0
out/${TARGET_BOARD}/boot.img:kernel
out/${TARGET_BOARD}/rootfs.img:rootfs
#下面这行是给小容量方案预留的，目前注释掉
#out/${TARGET_BOARD}/usr.img:usr
)
```

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。

3.4.6 编译 OTA 包所需的子镜像

编译 kernel 和 rootfs。

```
make
```

编译 recovery 系统。

```
swupdate_make_recovery_img
```

编译 uboot。

```
muboot
```

打包，若需要升级 boot0/uboot，则是必要步骤，打包会将 boot0 和 uboot 拷贝到 out 目录下，并对头部参数等进行修改。生成的固件也可用于测试。注：如果希望生成的固件的 recovery 分区是有系统的，则需要先编译 recovery 系统，再打包。

```
pack / pack -s
```

生成 OTA 包。因为我们使用的就是 sw-subimgs.cfg，所以不同带参数。

注意，如果方案目录下存在 sw-subimgs.cfg，则优先用方案目录下的。没有方案特定配置才用 generic 下的。如果需要升级 boot0/uboot，需要配置好 sys_config.fex 中的 storage_type 参数，swupdate_pack_swu 才能正确拷贝对应的 boot0/uboot。

```
swupdate_pack_swu
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.4.7 执行 OTA

3.4.7.1 准备 OTA 包

对于测试来说，直接推入。

```
adb push  out/<board>/swupdate/<board>.swu  /mnt/UDISK
```

实际应用时，可从先从网络下载到本地，再调用 swupdate，也可以直接传入 url 给 swupdate。

3.4.7.2 调用 swupdate

若使用原生的 swupdate，则调用：

```
swupdate -i /mnt/UDISK/<board>.swu -e stable,upgrade_recovery
```

但这样不会在自启动的时候帮我们准备好 swupdate 所需的-e 参数。

我们可以使用辅助脚本：

```
swupdate_cmd.sh -i /mnt/UDISK/<board>.swu -e stable,upgrade_recovery
```

3.5 AB 系统方案举例

3.5.1 配置分区和 env

在分区表中，将原有的 boot 分区和 rootfs 分区，分区名改为 bootA 和 rootfsA。

将这两个分区配置拷贝一份，即新增两个分区，并把名字改为 bootB 和 rootfsB。

这样 flash 中就存在 A 系统 (bootA+rootfsA) 和 B 系统 (bootB+rootfsB)。

一般是一个系统烧录两份。即分区表中的 bootA 和 bootB 都指定的 boot.fex，rootfsA 和 rootfsB 都指定的 rootfs.fex。

在 env 中，指定：

```
boot_partition=bootA
root_partition=rootfsA
```

并配置 boot_normal 命令，从 $boot_partition 变量指定的分区加载系统。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.5.2 配置主系统

lunch 选择方案后, make menuconfig, 选上 swupdate。

3.5.3 配置 recovery 系统

AB 系统方案没有使用 recovery 系统，无需配置和生成。

3.5.4 准备 sw-description

这里我们直接使用：

```
target/allwinner/generic/swupdate/sw-description-ab
```

内容如下，中文部分是注释，原文件中没有。

```
/* 固定格式，最外层为software = { } */
software =
{
    /* 版本号和描述 */
    version = "0.1.0";
    description = "Firmware update for Tina Project";

    /*
     * 外层tag，stable，
     * 没有特殊含义，就理解为一个字符串标志即可。
     * 可以修改，调用的时候传入匹配的字符串即可。
     */
    stable = {

        /*
         * 内层tag，now_A_next_B，
         * 当调用swupdate xxx -e stable,now_A_next_B时，就会匹配到这部分，执行 {} 内的动作，
         * 可以修改，调用的时候传入匹配的字符串即可。
         */
        /* now in systemA, we need to upgrade systemB(bootB, rootfsB) */
        now_A_next_B = {
            /* 这部分是描述，当前处于A系统，需要更新B系统，该执行的动作。执行完后下次启动为B系统 */
            images: ( /* 处理各个image */
                {
                    filename = "kernel"; /* 源文件是OTA包中的kernel文件 */
                    device = "/dev/by-name/bootB"; /* 要写到/dev/by-name/bootB节点中，这个节点
    在tina上就对应bootB分区 */
                    installed-directly = true; /* 流式升级，即从网络升级时边下载边写入，而不是先完
    整下载到本地再写入，避免占用额外的RAM或ROM */
                },
                {
                    filename = "rootfs"; /* 同上，但处理rootfs，不赘述 */
                    device = "/dev/by-name/rootfsB";
                    installed-directly = true;
                },
```

ALLWINNER®

```
            {
                filename = "uboot"; /* 源文件是OTA包中的uboot文件 */
                type = "awuboot"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
            },
            {
                filename = "boot0"; /* 源文件是OTA包中的boot0文件 */
                type = "awboot0"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
            }
        );
        /* image处理完之后，需要设置一些标志，切换状态 */
        bootenv: ( /* 处理bootenv，会修改uboot的env分区 */
            {
                /* 设置env:swu_mode=upgrade_kernel，这是为了记录OTA进度，对于AB系统来说，此时
已经升级完成，置空 */
                name = "swu_mode";
                value = "";
            },
            {
                /* 设置env:boot_partition=bootB，这是为了切换系统，下次uboot就会启动B系统(
kernel位于bootB分区) */
                name = "boot_partition";
                value = "bootB";
            },
            {
                /* 设置env:root_partition=rootfsB，这是为了切换系统，下次uboot就会通过cmdline
指示挂载B系统的rootfs */
                name = "root_partition";
                value = "rootfsB";
            },
            {
                /* 兼容另外的切换方式，可以先不管 */
                name = "systemAB_next";
                value = "B";
            },
            {
                /* 设置env:swu_next=reboot，这是为了跟外部脚本配合，指示外部脚本做reboot动作 */
                name = "swu_next";
                value = "reboot";
            }
        );
    };

   /*
    * 内层tag，now_B_next_A,
    * 当调用swupdate xxx -e stable,now_B_next_A时，就会匹配到这部分，执行 {} 内的动作，
   * 可以修改，调用的时候传入匹配的字符串即可
    */
    /* now in systemB, we need to upgrade systemA(bootA, rootfsA) */
    now_B_next_A = {
        /* 这里面就不赘述了，跟上面基本一致，只是AB互换了 */
        images: (
            {
                filename = "kernel";
                device = "/dev/by-name/bootA";
                installed-directly = true;
            },
            {
                filename = "rootfs";
                device = "/dev/by-name/rootfsA";
                installed-directly = true;
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
                },
                {
                    filename = "uboot";
                    type = "awuboot";
                },
                {
                    filename = "boot0";
                    type = "awboot0";
                }
            );
            bootenv: (
                {
                    name = "swu_mode";
                    value = "";
                },
                {
                    name = "boot_partition";
                    value = "bootA";
                },
                {
                    name = "root_partition";
                    value = "rootfsA";
                },
                {
                    name = "systemAB_next";
                    value = "A";
                },
                {
                    name = "swu_next";
                    value = "reboot";
                }
            );
        };
    };

    /* 当没有匹配上面的tag，进入对应的处理流程时，则运行到此处。我们默认清除掉一些状态 */
    /* when not call with -e xxx,xxx    just clean */
    bootenv: (
        {
            name = "swu_param";
            value = "";
        },
        {
            name = "swu_software";
            value = "";
        },
        {
            name = "swu_mode";
            value = "";
        },
        {
            name = "swu_version";
            value = "";
        }
    );

}
```

说明：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

升级过程会进行一次重启。具体的：

（1）升级 kernel 和 rootfs 到另一个系统所在分区，升级 uboot(uboot)，boot0(boot0) 。设置 boot_partition 为切换系统。

（2）重启，进入新系统。

3.5.5 准备 sw-subimgs.cfg

我们直接看下 tina 默认的：

```
target/allwinner/generic/swupdate/sw-subimgs-ab.cfg
```

内容如下，中文部分是注释，原文件中没有。

```
swota_file_list=(
#取得sw-description-ab，重命名成sw-description，放到OTA包中。
#注意第一行必须为sw-description
target/allwinner/generic/swupdate/sw-description-ab:sw-description
#取得uboot.img，重命名为uboot，放到OTA包中。以下雷同
#uboot.img和boot0.img是执行swupdate_pack_swu时自动拷贝得到的，需配置sys_config.fex中的
    storage_type
out/${TARGET_BOARD}/uboot.img:uboot
#注：boot0没有修改的话，以下这行可去除，其他雷同，可按需升级
out/${TARGET_BOARD}/boot0.img:boot0
out/${TARGET_BOARD}/boot.img:kernel
out/${TARGET_BOARD}/rootfs.img:rootfs
)
```

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。

3.5.6 编译 OTA 包所需的子镜像

编译 kernel 和 rootfs。

```
make
```

编译 uboot。

```
muboot
```

打包，若需要升级 boot0/uboot，则是必要步骤，打包会将 boot0 和 uboot 拷贝到 out 目录下，并对头部参数等进行修改。生成的固件也可用于测试。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
pack / pack -s
```

生成 OTA 包。因为我们使用的是 sw-subimgs-ab.cfg，所以调用时带参数-ab。

注意，如果方案目录下存在 sw-subimgs-ab.cfg，则优先用方案目录下的。没有方案特定配置才用 generic 下的。

```
swupdate_pack_swu -ab
```

3.5.7 执行 OTA

3.5.7.1 准备 OTA 包

对于测试来说，直接推入。

```
adb push  out/<board>/swupdate/<board>.swu  /mnt/UDISK
```

实际应用时，可从先从网络下载到本地，再调用 swupdate，也可以直接传入 url 给 swupdate。

3.5.7.2 判断 AB 系统

对于 AB 系统方案来说，必须判断当前所处系统，才能知道需要升级哪个分区的数据。

判断当前是处于 A 系统还是 B 系统。

方式一：直接使用 fw_printenv 读取判断当前的 boot_partition 和 root_partition 的值。

3.5.7.3 调用 swupdate

若使用原生的 swupdate，则调用：

```
当前处于A系统：
swupdate -i /mnt/UDISK/<board>.swu -e stable,now_A_next_B
当前处于B系统：
swupdate -i /mnt/UDISK/<board>.swu -e stable,now_B_next_A
```

但这样不会在自启动的时候帮我们准备好 swupdate 所需的-e 参数。

我们可以使用辅助脚本：

```
当前处于A系统：
swupdate_cmd.sh -i /mnt/UDISK/<board>.swu -e stable,now_A_next_B
当前处于B系统：
swupdate_cmd.sh -i /mnt/UDISK/<board>.swu -e stable,now_B_next_A
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.6 辅助脚本 swupdate_cmd.sh

为什么需要辅助脚本?

因为我们需要启动时能自动调用 swupdate，自动传递合适的-e 参数给 swupdate, 需要在合适的时候调用重启。

具体可直接看下脚本内容。

其基本思路是，当带参数调用时，脚本从传入的参数中，取出”-e xxx,yyy” 部分，将其余参数原样保存为 env 的 swu_param 变量。

取出的”-e xxx,yyy” 中的 xxx 保存到 env 的 swu_software 变量, yyy 保存为 env 的 swu_mode 变量。

然后就取出变量，循环调用。

```
swupdate $swu_param -e "$swu_software,$swu_mode"
```

sw-description 中可以通过改变 env 的 swu_software 和 swu_mode 变量，来影响下次的调用参数。

实际应用时，可不使用此脚本，直接在主应用中，调用 swupdate 即可。但要自行做好-e 参数的处理。

3.7 版本号

3.7.1 使用方式

在 sw-descriptionwen 文件中，会配置一个版本号字符串，如:

```
software =
{
    version = "1.0.0";
    ...
}
```

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。如果不需要，则不传递 -N 参数，忽略版本号即可。

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，需要自定义, swupdate 没有规定具体的方式。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.7.2 实现例子

应用可以按自己的逻辑维护版本号，不依赖系统 env 等，只需按照 swupate 要求传递参数即可。

此处提供一种依赖系统 env 的实现方式供参考。

1. 初始化设备端版本号。

首先需要定义设备端的版本号存放在哪，如何获取。

本方法定义设备端的版本号保存于 env 之中，用 swu_version 记录。

则在 SDK 中，需在 env-x.x.cfg 中添加一行：

```
swu_version=1.0.0
```

表示此时版本为 1.0.0，烧录固件后可执行 fw_printenv 查看。

此步骤如果不做，则第一次烧录固件后 env 中不存在 swu_version，调用 swupdate 时也无法传入获得并版本号，则第一次升级时不会检查版本。

注：这是 tina 自定义的，可修改。只要读写这个版本号的地方均配套修改即可。实际应用时版本号可以存在任意分区中，或者存放在文件系统的文件中，或者硬编码在系统和应用的二进制中，swupdate 未做限制。

2. 在 sw-description 中，设置 OTA 包版本号。

升级时如果检查到 OTA 包的 sw-description 中的 version，小于通过 -N 参数传入的版本号，则不允许升级。

```
software =
{
    version = "2.0.0";
    ...
```

例如当设备端的 env 中设置了`swu_version=2.0.0`, 则调用 swupdate_cmd.sh 时，会自动获取此参数并在调用 swupdate 时传入`-N 2.0.0`。

此时若 OTA 包中定义了 version = “1.0.0”，则此时升级会降低版本号，拒绝升级。

此时若 OTA 包中定义了 version = “2.0.0”，则此次升级不会降低版本号，可以升级。

此时若 OTA 包中定义了 version = “3.0.0”，则此次升级不会降低版本号，可以升级。

注：这是 swupdate 原生的 OTA 包版本号规则，不是 tina 自定义的。

3. 更新设备端版本号。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

本方式版本号定义在 env 中，则升级 kernel 和 rootfs 分区不会自动更新版本号，需要主动修改 env。

若版本号是记录于 rootfs 的某个文件，则不必在 sw_description 中添加这种操作，因为更新 rootfs 时版本号就自然更新了。但缺点是版本号跟 rootfs 绑定了，每次 OTA 必须升级 rootfs 才能更新版本号。

添加一个设置 version 代表 swu_version 的 env 操作, 在 OTA 时自动更新版本号。

```
software =
{
    #表示这个OTA包的版本号，给swupdate读取检查的。原生规定的。
    version = "2.0.0";
    ...
            bootenv: (
                ...
                {
                    #表示这个OTA包的版本号，OTA时会写入env分区，用于在下次OTA时读出作为-N参数的值。
    Tina自定义的。
                    name = "swu_version";
                    value = "2.0.0";
                }
                ...
            );
    ...
}
```

注意，这么做的话，更新版本时需要修改 env 中的版本号，以使得新的固件包拥有新的版本号，以及更新 sw-description 的两个位置，一处是最上面的 version = xxx 的版本号，一处是 bootenv 操作中的版本号，以使得 OTA 包拥有新的版本号，以及能在 OTA 时写入新版本号。

4. 读取设备端版本号传给 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
```

更好的方式是判断非空才传入，如此可支持不在 env 中提前配置好 swu_version。

```
check_version_para=""
[ x"$swu_version" != x"" ] && {
    echo "now version is $swu_version"
    check_version_para="-N $swu_version"
}
swupdate  ... $check_version_para
```

注：

如果不使用版本号，则不在 env 中设置 swu_version，也不在 bootenv 中写 swu_version 即可。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

如果 sw_description 中的版本号一直保持 v1.0.0，也总是能升级。

3.8 签名校验

3.8.1 检验原理

OTA 包中包含了 sw-decsription 文件和各个具体的镜像，如 kernel，rootfs。

如果对整个 OTA 包进行完整校验，则会对流式升级造成影响，要求必须把整个 OTA 包下载下来，才能判断出校验是否通过。

为了避免上述问题，swupdate 的校验是分镜像的，首先从 OTA 包最前面取出两个文件，即 sw-description 和 sw-description.sig，使用传入的公钥校验 sw-description，校验通过则认为 sw-description 可信，则说明其中描述的 image 和 sha256 也是可信的。

后续无需再使用公钥，直接校验每个镜像的 sha256 即可。因此可以逐个镜像处理，无需全部下载完毕再处理。

3.8.2 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 ---> ...(重复以上配置)
```

3.8.3 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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.8.4 初始化 key

Tina 封装了一条命令，生成默认的密钥对。执行：

```
swupdate_init_key
```

执行后会使用默认密码生成密钥对并拷贝到指定目录：

```
密码/私钥/公钥:
password:tina/target/allwinner/方案名/swupdate/swupdate_priv.password
private key:tina/target/allwinner/方案名/swupdate/swupdate_priv.pem
public key:tina/target/allwinner/方案名/swupdate/swupdate_public.pem
公钥拷贝到base-files中，供使用procd-init的方案使用
public key:tina/target/allwinner/方案名/base-files/swupdate_public.pem
公钥拷贝到busybox-init-base-files中，供使用busybox-init的方案使用
public key:tina/target/allwinner/方案名/busybox-init-base-files/swupdate_public.pem
```

此步骤仅为方便调试使用，只需要做一次。

用户也可使用自己的密码自行生成密钥，生成密钥的具体命令可参考 build/envsetup.sh 中 swupdate_init_key 的实现：

```
    local password="swupdate";
    echo "$password" > swupdate_priv.password;
    echo "-------------------- init priv key --------------------";
    openssl genrsa -aes256 -passout file:swupdate_priv.password -out swupdate_priv.pem;
    echo "-------------------- init public key --------------------";
    openssl rsa -in swupdate_priv.pem -passin file:swupdate_priv.password -out
    swupdate_public.pem -outform PEM -pubout;
```

生成的密钥如 swupdate_init_key 一般放到 tina/target/allwinner/方案名/swupdate/ 中，即可在打包 OTA 包时自动使用。

主要就是调用 openssl 生成，私钥拷贝到 SDK 指定目录，供生成 OTA 包时使用。公钥放到设备端，供设备端执行 OTA 时使用。

密钥的作用是校验 OTA 包，意味着拿到密钥的人即可生成可通过校验的 OTA 包，因此正式产品中一般密钥只掌握在少数人手中，并采取适当措施避免泄漏或丢失。

一种可参考的实践方式是，正式密钥做好备份，并仅部署在有权限管控的服务器上，只能代码入库后通过自动构建生成 OTA 包，普通工程师无法拿到密钥自行本地生成用于正式产品的 OTA 包。

3.8.5 修改 sw-description

如上文所述，每个 image 在使用时会校验 sha256，因此需要在为每个更新文件在 sw-desctiption 中添加 sha256 属性，指定 sha256 的值供更新过程校验。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

有独立镜像的文件才需要 sha256 属性，例如 images 中配置的文件。而 bootenv 等直接写在 sw-description 中的，则无需 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: (
```

在打包 OTA 包时，脚本自动算出 sha256 的值，并替换到上述位置，再完成 OTA 包的生成。

可参考：

```
target/allwinner/generic/swupdate/sw-description-sign
target/allwinner/generic/swupdate/sw-subimgs-sign.cfg

注:
调用swupdate_pack_swu 则会使用sw-subimgs.cfg，其中默认指定了使用sw-description做为最终的sw-
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    description。
调用swupdate_pack_swu -sign则会使用sw-subimgs-sign.cfg，其中默认指定了使用sw-description-sign做为
    最终的sw-description。
即关键还是看使用哪份sw-subimgs.cfg，以及sw-subimgs.cfg中如何指定。
```

3.8.6 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 包中，紧随在 sw-description 后面。

目前脚本中自动处理。

3.8.7 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。

3.8.8 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：

```
cdevice
mkdir -p ./base-files
cp swupdate_public.pem ./base-files/etc/
```

对于 busybox-init

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

3.8.9 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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.9 压缩

swupdate 支持对镜像先解压，再写入目标位置，当前支持 gzip 和 zstd 两种压缩算法。

3.9.1 配置

使用 gzip 压缩无需配置，使用 zstd 则需选上

```
make menuconfig --> Allwinner  ---> <*> swupdate --> [*]   Zstd compression support
```

3.9.2 生成压缩镜像

如果希望每次打包固件自动生成，则可修改 scripts/pack_img.sh, 在 function do_pack_tina() 函数的最后加上压缩的动作。

```
生成gz镜像:
gzip -k -f boot.fex
gzip -k -f rootfs.fex
gzip -k -f recovery.fex #如果使用AB方案，则无需recovery

生成lzma镜像:
zstd -k -f boot.fex -T0
zstd -k -f rootfs.fex -T0
zstd -k -f recovery.fex -T0
```

对于 lzma，若需要调整压缩率，可指定 0-19 的数字 (数字越大，压缩率越高，耗时越长)，如

```
zstd -19 -k -f boot.fex -T0
zstd -19 -k -f rootfs.fex -T0
zstd -19 -k -f recovery.fex -T0
```

如果不希望每次打包固件多耗时间，则需自行在生成 OTA 包之前，使用上述命令制作好压缩镜像。原始的 boot.fex，rootfs.fex, recovery.fex 在 out/方案/image/目录下。

3.9.3 sw-subimgs.cfg 配置压缩镜像

以 rootfs 为例，将原本未压缩的版本

```
out/${TARGET_BOARD}/rootfs.img:rootfs
```

改成压缩的

```
out/${TARGET_BOARD}/image/rootfs.fex.gz:rootfs.gz
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

或

```
out/${TARGET_BOARD}/image/rootfs.fex.zst:rootfs.zst
```

注：为了方便差分包的处理，此处约定压缩镜像需以.gz 或.zst 结尾，生成差分项的脚本会检查后缀名，并自动解压。

3.9.4 sw-description 配置压缩镜像

以 rootfs 为例，将原本未压缩的版本

```
                {
                    filename = "rootfs";
                    device = "/dev/by-name/rootfs";
                    installed-directly = true;
                    sha256 = "@rootfs";
                },
```

换成

```
                {
                    filename = "rootfs.gz";
                    device = "/dev/by-name/rootfs";
                    installed-directly = true;
                    sha256 = "@rootfs.gz";
                    compressed = "zlib";
                },
```

或

```
                {
                    filename = "rootfs.zst";
                    device = "/dev/by-name/rootfsB";
                    installed-directly = true;
                    sha256 = "@rootfs.zst";
                    compressed = "zstd";
                },
```

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。

3.10.1 进度条

swupdate 提供了 progress 程序，该程序会在后台运行，从 socket 获取进度信息，打印进度条到串口。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

具体方案可参考其实现 (在 swupdate 源码中搜索 progress)，自行在应用中获取进度，通过屏幕等其他方式进行指示。

3.10.2 重启

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

2.swupdate_cmd.sh 支持检测 env 中的 swu_next 变量，如果为 reboot，则脚本中执行 reboot。可在 sw-description 中设置此变量。

3. 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。

3.10.3 本地升级示例

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
```

3.10.4 网络升级示例

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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

3.10.5 错误处理

如何判断 swupdate 升级出错?

1. 调用 swupdate 时获得并判断返回值是否为 0。
2. 读取 env 变量 recovery_status。根据 swupdate 官方文档，swupdate 开始执行时，会设置 recovery_status=“progress”，升级完成会清除这个变量，升级失败则设置 recovery_status=“failed”。

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

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

出问题时，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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

3.12.4 解压 OTA 包

swupdate 的 OTA 包，本质上是一个 cpio 格式的包，直接使用通用的 cpio 解包命令即可。

```
cpio -idv < xxx.swu
```

3.12.5 校验 OTA 包

当使能了签名校验，会对 sw-description 签名生成 sw-description.sig，如果校验失败，可以在 PC 端手工验证下：

使用 RSA 时, build/envsetup.sh 中调用的命令是：

```
openssl dgst -sha256 -sign "$priv_key_file" $password_para "$SWU_DIR/sw-description" > "
    $SWU_DIR/sw-description.sig"
```

则对应的密钥验证签名命令为：

```
openssl dgst -prverify swupdate_priv.pem -sha256 -signature sw-description.sig sw-
    description
```

公钥验证签名的命令为：

```
openssl dgst -verify swupdate_public.pem -sha256 -signature sw-description.sig sw-
    description
```

3.13 测试固件示例

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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
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
```

3.14 升级定制分区

如果定制了一个分区，并需要对此分区进行 OTA，则需要：

1. 确认需不需要备份。
2. 将分区文件加入 OTA 包。
3. 确定升级策略，在 sw-description 中增加对此分区的处理。

3.14.1 备份

在 OTA 过程随时可能掉电，如果掉电时正在升级分区 mypart ，则重启后 mypart 的数据是不完整的。

是否需要备份主要取决于 mypart 所保存的文件是否影响继续进行 OTA。

例如 mypart 中保存的是开机音乐，被损坏的结果只是开机无声音，但开机后能正常继续 OTA，则无需备份。

例如 mypart 中保存的是应用，且 OTA 中途掉电后重启，需要应用负责联网从网络重新下载 OTA 包，则需要备份，否则无法继续 OTA，设备就无法恢复了。

例如 mypart 中保存的是 dsp，启动过程没有有效的 dsp 镜像会无法启动，则需要备份，否则无法启动也就无法继续 OTA，设备就无法恢复了。

3.14.2 无需备份

假设分区表中定义了：

```
[partition]
    name         = mypart
    size         = 512
    downloadfile = "mypart.fex"
    user_type    = 0x8000
```

文件放在：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
out/${TARGET_BOARD}/image/mypart.fex
```

则在 sw-subimg.cfg 中增加一行（注意要放到 sw-description 之后，因为 sw-description 必须是第一个文件），将 mypart.fex 打包到 OTA 包中，重命名为 mypart：

```
out/${TARGET_BOARD}/image/mypart.fex:mypart
```

在 sw-description 中增加升级动作的定义。例如可以在升级 rootfs 之后，升级该分区：

```
software =
{
    ...
    stable = {
        ...
        upgrade_kernel = {
            images: (
                ...
                {
                    filename = "rootfs";
                    device = "/dev/by-name/rootfs";
                    installed-directly = true;
                },
                {
                    filename = "mypart";
                    device = "/dev/by-name/mypart";
                    installed-directly = true;
                }
        ...
            );
        ...
```

3.14.3 需要备份

在分区表中定义好两个分区，这样升级过程掉电，总有一份是完整的。

```
[partition]
    name         = mypart
    size         = 512
    downloadfile = "mypart.fex"
    user_type    = 0x8000

[partition]
    name         = mypart-r
    size         = 512
    downloadfile = "mypart.fex"
    user_type    = 0x8000
```

文件放在：

```
out/${TARGET_BOARD}/image/mypart.fex
```

则在 sw-subimg.cfg 中增加一行（注意要放到 sw-description 之后，因为 sw-description 必

版权所有 © 珠海全志科技股份有限公司。保留一切权利

须是第一个文件），将 mypart.fex 打包到 OTA 包中，重命名为 mypart：

```
out/${TARGET_BOARD}/image/mypart.fex:mypart
```

有两个分区，则需要条件决定使用哪一个，可考虑在 env 中定义一个变量：

```
mypart_partition=mypart
```

使用 mypart 时，要先读取 env 的 mypart_partition 的值来决定要使用哪个分区。

在 sw-description 中，定义好要升级的分区和 bootenv，保证每次升级那个未在使用的分区。这样即使掉电也无妨。

```
software =
{
    ...
    stable = {
        upgrade_recovery = {
            images:
                {
                    filename = "recovery";
                    device = "/dev/by-name/recovery";
                    installed-directly = true;
                },
        /* 更新mypart-r， 此时掉电，mypart分区是完整的 */
                {
                    filename = "mypart-r";
                    device = "/dev/by-name/mypart-r";
                    installed-directly = true;
                }

            );
            bootenv: (
                {
                    name = "swu_mode";
                    value = "upgrade_kernel";
                },
                {
                    name = "boot_partition";
                    value = "recovery";
                },
        /* 设置这个env，指示下次启动，即启动到recovery分区时，配套使用mypart-r */
                {
                    name = "mypart_partition";
                    value = "mypart-r";
                },
                {
                    name = "swu_next";
                    value = "reboot";
                }
            );
        };

        upgrade_kernel = {
            images: (
                {
                    filename = "kernel";
```

ALLWINNER®

```
                    device = "/dev/by-name/boot";
                    installed-directly = true;
                },
                {
                    filename = "rootfs";
                    device = "/dev/by-name/rootfs";
                    installed-directly = true;
                },
        /* 更新mypart， 此时掉电，mypart分区还是完整的 */
                {
                    filename = "mypart-r";
                    device = "/dev/by-name/mypart-r";
                    installed-directly = true;
                }
            );
            bootenv: (
                {
                    name = "swu_mode";
                    value = "upgrade_usr";
                },
        /* 设置这个env，指示下次启动，即启动到正常系统时，使用mypart */
                {
                    name = "mypart_partition";
                    value = "mypart";
                },
                {
                    name = "boot_partition";
                    value = "boot";
                }
            );
        };
    ...
```

3.15 handler 说明

此处只对一些 handler 做简介。

具体的 handler 的用法，请参考 swupdate 官方文档说明。

3.15.1 awboot

3.15.1.1 nand/emmc

全志拓展的 handler，用于支持升级全志 boot0 和 uboot，实质上会调用外部的 ota-burnboot 包完成升级。

目前只支持 nand/emmc，详见 ota-burnboot 章节。

使用方式：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

选上 handler 支持：

```
make menuconfig --->
    Allwinner --->
        <*> swupdate --->
            Image Handlers -->
                [*] allwinner boot0/uboot
```

注意，recovery 系统也需要对应进行配置，即：

```
make ota_menuconfig ---> ...(重复以上配置)
```

在 sw-description 中指定 type 即可：

```
{
    filename = "uboot"; /* 源文件是OTA包中的uboot文件 */
    type = "awuboot"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
},
{
    filename = "boot0"; /* 源文件是OTA包中的boot0文件 */
    type = "awboot0"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
}
```

3.15.1.2 nor

对于 nor 方案，升级 boot0/uboot 有掉电风险。如确需升级，可直接配置合适偏移即可，不使用 awboot handler。

例如已知 boot0 存放在偏移为 0 处，uboot 存放在偏移为 24k 处，则配置：

```
{
    filename = "boot0";
    device = "/dev/mtdblock0";
    installed-directly = true;
},
{
    filename = "uboot";
    device = "/dev/mtdblock0";
    offset = "24k"
    installed-directly = true;
}
```

3.15.2 readback

用于支持在 sw-description 中配置 sha256，在升级后读出数据进行校验。

一种应用场景是，在 AB 系统差分升级时，应用差分包后读出校验，以确认差分得到的结果是对的，再切换系统。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

需选上对应 handler。

```
make menuconfig/make ota_menuconfig
--> Allwinner
  -->  swupdate
    --> <*>   Allow to add sha256 hash to each image

make menuconfig/make ota_menuconfig
--> Allwinner
  -->  swupdate
    --> Image Handlers
      --> [*] readback
```

3.15.2.1 示例

```
target/allwinner/generic/swupdate/sw-description-readback
target/allwinner/generic/swupdate/sw-subimgs-readback.cfg
```

3.15.3 ubi

用于 ubi 方案。

需先选上 MTD 支持：

```
make menuconfig/make ota_menuconfig
--> swupdate
  --> Swupdate Settings
    ---> General Configuration
     ---> [*] MTD support
```

再选上对应 handler：

```
make menuconfig/make ota_menuconfig
-->  swupdate
  --> Image Handlers
    --> [*] ubivol
```

3.15.3.1 示例

请参考：

```
target/allwinner/generic/swupdate/sw-subimgs-ubi.cfg
target/allwinner/generic/swupdate/sw-description-ubi
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.15.4 rdiff

rdiff handle 用于差分包升级。

需选上对应 handler：

```
make menuconfig/make ota_menuconfig
-->  swupdate
  --> Image Handlers
    --> [*] rdiff

#若使用AB系统方案，则无recovery系统，则make ota_menuconfig的配置可不做。
```

3.15.4.1 特性

在 https://librsync.github.io/page_rdiff.html 中指出

```
rdiff cannot update files in place: the output file must not be the same as the input file.
```

即不支持原地更新，即应用差分包将 A0 更新成 A1，需要有足够空间存储 A0 和 A1，不能直接对 A0 进行改动。

```
rdiff does not currently check that the delta is being applied to the correct file. If a
    delta is applied to the wrong basis file, the results will be garbage.
```

不校验原文件，如果将差分包应用于错误的文件，则会得到无效的输出文件，但不会报错。

```
The basis file must allow random access. This means it must be a regular file rather than a
     pipe or socket.
```

原文件必须支持随机访问，因此不能从管道或 socket 中获取原文件。

更多介绍请参考：https://librsync.github.io/

3.15.4.2 示例

首先需要将方案修改为 AB 系统的方案，请参考上文的 “AB 系统方案举例”。

swupdate 的配置文件请参考：

```
target/allwinner/generic/swupdate/sw-description-ab-rdiff
target/allwinner/generic/swupdate/sw-subimgs-ab-rdiff.cfg
```

差分文件的生成使用 rdiff：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
$ rdiff -h
Usage: rdiff [OPTIONS] signature [BASIS [SIGNATURE]]
             [OPTIONS] delta SIGNATURE [NEWFILE [DELTA]]
             [OPTIONS] patch BASIS [DELTA [NEWFILE]]
```

tina 封装了一条命令，用于解出两个 swu 包并生成各个子镜像的差分文件。

一种参考的差分包生成方式是，先按普通的升级方式生成整包。

假设旧固件对应 V1.swu，新固件对应 V2.swu，则可使用：

```
swupdate_make_delta V1.swu V2.swu
```

生成差分文件。

再将生成的差分文件，用于生成差分的 OTA 包。

例如：

```
make && pack   #生成V1固件
swupdate_pack_swu -ab  #得到out/r328s2-perf1/swupdate/tina-r328s2-perf1-ab.swu
cp out/r328s2-perf1/swupdate/tina-r328s2-perf1-ab.swu V1.swu #保存V1的OTA包

#进行一些修改
make && pack   #生成V2固件
swupdate_pack_swu -ab  #得到out/r328s2-perf1/swupdate/tina-r328s2-perf1-ab.swu
cp out/r328s2-perf1/swupdate/tina-r328s2-perf1-ab.swu V2.swu #保存V2的OTA包

swupdate_make_delta V1.swu V2.swu #生成差分镜像
swupdate_pack_swu -ab-rdiff  #生成差分OTA包，这一步用到了刚刚生成的差分镜像
```

3.15.4.3 开销问题

差分升级的主要好处在于节省传输的文件大小。而不是节省 ram 和 rom 的占用。

设备端在进行差分升级时，需要使用版本匹配的旧版本镜像，加上差分包，生成新版本镜像。

rsync 不支持原地更新，必须有额外的空间保存新生成的镜像。

从掉电安全的角度考虑，在新版本镜像完整保存到 flash 之前，旧版本镜像不能破坏，否则一旦中途掉电，将无法再次使用旧镜像 + 差分包生成新镜像，只能联网下载完整的 OTA 包。

以上限制，导致 flash 必须在旧镜像之外，有足够 flash 空间用于存放新的镜像。

对于 recovery 方案，原本的：

```
从OTA包获取新recovery写入recovery分区 -->
reboot -->
从OTA包获取新kernel写入boot分区 -->
从OTA包获取新rootfs升级rootfs分区 -->
reboot
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

就需要变成：

```
从OTA包获取recovery差分包，读recovery分区，生成新recovery暂存到文件系统中 -->
从文件系统获取新recovery写入recovery分区 -->
reboot -->
...
```

总体较为麻烦，且需要文件系统足够大。

对于 AB 方案，原本的：

```
从OTA包获取新kernel写入bootB分区 -->
从OTA包获取新rootfs写入rootfsB分区 -->
reboot
```

就需要变成：

```
从OTA包获取kernel差分包，读出bootA分区，合并生成新kernel写入bootB分区 -->
从OTA包获取rootfs差分包，读出rootfsA分区，合并生成新rootfs写入bootB分区 -->
reboot
```

不需要依赖额外的文件系统空间。

因此，若希望节省 ram/rom 占用，差分包并非解决的办法。若希望使用差分包，建议配合 AB 系统使用。

3.15.4.4 管理问题

差分包的一个麻烦问题在于，差分包必须跟设备端的版本匹配。而出厂之后的设备，可能存在多种版本。

例如出厂为 V1，当前最新为 V4，则设备可能处于 V1，V2，V3。此时若使用整包升级，则无需区分。

若使用差分升级，一种策略是为每个旧版本生成一个差分包，则需要制作三个差分包 V1_4，V2_4，V3_4，并在 OTA 时先判断设备端和云端版本，再使用对应的差分包。

另一种策略是，只为上一个版本生成差分包 V3_4，并额外准备一个整包。在 OTA 时先判断设备端版本和云端版本，若可相差一个版本则使用差分包，若跨版本则使用整包。

不管哪一种，都需要应用做出额外的判断。这一点需要主应用和云端服务器做好处理。

3.15.4.5 校验问题

目前社区支持的 rdiff 本身并不包含较好的校验机制，即应用一个版本不对的差分包，也能跑完升级流程。这样一旦出错，就会导致机器变砖。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

一种可考虑的方式是搭配 readback 使用，即在应用差分包，写入目标分区之后，将更新后的目标分区数据读出，校验其 sha256 是否符合预期，校验成功才切换系统，校验失败则报错。

可参考

```
target/allwinner/generic/swupdate/sw-description-ab-rdiff-sign
target/allwinner/generic/swupdate/sw-subimgs-ab-rdiff-sign.cfg
```

其中增加了 readback 的处理。

具体的，sw-subimgs-xxx.cfg 中可以配置 swota_copy_file_list，指定一些文件只拷贝到 swupdate 目录，不打包到最终的 OTA 包（swu 文件）中。因为在这个场景下，我们需要原始的 kernel, rootfs 等文件来计算 sha256，但并不需要将其加入最终的 OTA 包中。

3.15.4.6 跟 ubi 的配合问题

swupdate 官方目前没有支持 rdiff 用于 ubi 卷。

目前采用增加预处理脚本的方式来兼容，即在执行 rdiff handler 之前，先调用脚本使用 ubiupdatevol 创建一个可用于升级目标 ubi 卷的 fifo。随后 rdiff handler 即可将此 fifo 当作目标裸设备，无需特殊处理 ubi 卷。在后处理脚本中，再通过填 0 的方式，结束所有的 ubiupdatevol 。

具体可参考如下文件夹中的配置和脚本：

```
target/allwinner/r329-evb5/swupdate
```

前提仍然是配置好 AB 系统，使用方式：

```
# 编译系统，得到要烧录的固件，记为V1
make && pack

# 生成OTA包，此OTA包的各个分区镜像跟V1固件的镜像一致
swupdate_pack_swu -ab
cp out/r329-evb5/swupdate/tina-r329-evb5-ab.swu tina-r329-evb5-ab_V1.swu

# 进行修改，编译出V2的系统
make && pack

# 生成OTA包，此OTA包的各个分区镜像跟V2固件的镜像一致
swupdate_pack_swu -ab
cp out/r329-evb5/swupdate/tina-r329-evb5-ab.swu tina-r329-evb5-ab_V2.swu

# 此时 tina-r329-evb5-ab_V2.swu 可用于正常的OTA升级
# 生成各个镜像的差分文件
swupdate_make_delta ./tina-r329-evb5-ab_V1.swu ./tina-r329-evb5-ab_V2.swu

# 基于上一步得到的差分文件，生成差分OTA包
swupdate_pack_swu -ab-rdiff
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4 Tina misc-upgrade 介绍 (建议改用 swupdate)

4.1 方案选择

misc-upgrade 只支持 recovery 系统方案。

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，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。

4.1.1 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。

为了达到启动后自动挂载 extend 分区的效果。需要进行配置。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

对于 busybox-init，默认在 pseudo_init 中会尝试挂载 usr。如果发现没有正常挂载，请检查/pseudo_init 中的 mount_usr。

对于 procd-init，可在

```
target/allwinner/<board>/base-files/etc/config/fstab
```

配置文件中，增加：

```
config 'mount'
    option  target          '/usr'
    option  device          '/dev/by-name/extend'
    option  options         'ro,sync'
    option  enabled         '1'
```

4.1.2 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。

4.1.3 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。

4.1.4 OTA 的升级流程

4.1.4.1 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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
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
```

最新版本处理流程：

```
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_boot0
7. 如果存在boot0镜像，则更新boot0
8. 设置标志upgrade_uboot
9. 如果存在uboot镜像，则更新uboot
10. 设置标志upgrade_post
11. 对于小容量方案，更新usr分区
12. 设置标志upgrade_end
13. 重启，重启后为新系统
```

4.1.4.2 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。

4.2 分区处理

4.2.1 分区定义

表 4-1：misc-upgrade 升级分区说明表

升级分区	
boot 分区	基础系统镜像分区，即/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，即 /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	

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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，一般挂载在 /mnt/UDISK。
overlayfs 分区	存储 overlayfs 覆盖数据。

4.2.2 分区大小配置

4.2.2.1 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 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
    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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.2.2.2 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。

4.2.2.2.1 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。

4.2.2.2.2 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。

4.2.2.3 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
```

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)
```

4.2.2.4 其他分区

如 private 、misc 等使用默认的大小即可。

4.2.2.5 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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.2.2.6 其他说明

在 OTA 升级过程中并不能修改上述分区的大小，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。

4.3 misc-upgrade 升级

4.3.1 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       #小容量方案挂载用。
├── aw_reboot.sh
├── aw_upgrade_autorun.init  #自启动脚本。
├── aw_upgrade_image.sh
├── aw_upgrade_lite.sh
├── aw_upgrade_log.sh
├── aw_upgrade_normal.sh     #大容量方案。
├── aw_upgrade_plus.sh
├── aw_upgrade_process.sh    #小容量方案。
├── aw_upgrade_utils.sh
├── aw_upgrade_vendor_default.sh
├── Makefile
├── readme.txt
└── tools               #编译出write_misc和read_misc应用。
    ├── Makefile
    ├── misc_message.c
    ├── misc_message.h
    ├── read_misc.c
    └── write_misc.c
```

4.3.2 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)：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
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。

注：不同介质使用的boot0/uboot镜像不同，
make_ota_image需要sys_config.fex中的storage_type配置明确指出介质类型，
才能取得正确的boot0.img和uboot.img。
具体可直接查看build/envsetup.sh中make_ota_image的实现。
```

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：

```
├── boot0_sys               #boot0升级文件。
│   ├── boot0.img
│   └── boot0.img.md5
├── boot0_sys.tar.gz        #boot0升级文件的压缩包。
├── package_sys             #某定制版OTA脚本使用，不在本文档描述范围。
│   ├── boot0.img
│   ├── boot0.img.md5
│   ├── boot.img
│   ├── boot.img.md5
│   ├── ota.tar
│   ├── recovery.img
│   ├── recovery.img.md5
│   ├── rootfs.img
│   ├── rootfs.img.md5
│   ├── uboot.img
│   └── uboot.img.md5
├── ramdisk_sys                 #recovery系统升级文件。
│   ├── boot_initramfs.img
│   └── boot_initramfs.img.md5
```

ALLWINNER®

```
├── ramdisk_sys.tar.gz          #recovery系统升级压缩包。
├── target_sys                  #kernel和rootfs升级文件。
│   ├── boot.img
│   ├── boot.img.md5
│   ├── rootfs.img
│   └── rootfs.img.md5
├── target_sys.tar.gz           #kernel和rootfs升级压缩包。
├── uboot_sys                   #uboot升级文件。
│   ├── uboot.img
│   └── uboot.img.md5
└── uboot_sys.tar.gz            #uboot升级压缩包。
```

其中”.img.md5“是”.img“的校验值文件。

升级脚本不带-n 参数，则使用压缩包，带-n 则直接使用不压缩的升级文件。

4.3.3 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。

4.3.3.1 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镜像后，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。

4.3.3.1.1 -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：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- recovery 系统和主系统（必选）：ramdisk_sys.tar.gz 、target_sys.tar.gz。
- uboot 和 boot0（可选，调用脚本时镜像不存在则自动跳过）：uboot_sys.tar.gz，boot0_sys.tar.gz。

4.3.3.1.2 -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 及可选的 uboot_sys.tar.gz、boot0_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，即上述 xxx.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：

- recovery 系统和主系统（必选）：boot_initramfs.img 、boot.img 、rootfs.img 及其对应的 md5 后缀的校验文件。
- uboot 和 boot0（可选，调用脚本时镜像不存在则自动跳过）：uboot.img、boot0.img 及其对应的 md5 后缀的校验文件。

4.3.3.2 小容量 flash 方案

4.3.3.2.1 -l 选项 原始的设计是用于网络升级，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。

后续考虑存在小容量设备插 SD 卡升级的情况，支持了-l 参数，命令类似上述的大容量方案，不赘述。

4.3.3.2.2 -d -u -n 选项

```
-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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

注：若由 misc-upgrade 自行下载镜像，当前实现暂不支持可选的 boot0/uboot 镜像，即不会自动从服务器下载升级 boot0/uboot。

4.4 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。

4.4.1 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。
}
```

ALLWINNER®

4.4.2 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
}
```

4.4.3 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
}
```

4.4.4 写分区完成

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

4.4.5 -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
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.4.6 -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
```

4.5 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，只能由 aw-upgrade shell 修改。
-s -v 自定义使用。
```

4.6 OTA 配置

4.6.1 recovery 系统生成

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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

4.6.2 recovery 系统切换

4.6.2.1 切换方式 1：基于 misc 分区

对于存在 misc 分区的方案，可以在 misc 分区中设置 boot-recovery 标志。启动时，uboot 会检测 misc 分区，如果为 boot-recovery，则执行 env 中配置的 boot_recovery 命令启动内核。否则执行 boot_normal 命令启动内核。

因此，env 中需要正确配置 boot_normal 和 boot_recovery。类似：

```
boot_normal=sunxi_flash read 40007800 boot;bootm 40007800
boot_recovery=sunxi_flash read 40007800 extend;bootm 40007800
```

对于大容量方案 boot_recovery 配置为 recovey 分区启动，对于小容量方案，boot_recovery 配置为从 extend 分区启动。

OTA 过程需要正确设置 misc 分区的值。

4.6.2.2 切换方式 2：基于 env 分区

可以去除 misc 分区，直接基于 env 进行系统切换。OTA 过程需修改 env 分区中的值。

一般是将 env 的 boot_normal 配置为从 $boot_partition 分区启动，即将要使用的分区抽离成一个变量。

类似：

```
boot_partition=boot
boot_normal=sunxi_flash read 40007800 ${boot_partition};bootm 40007800
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

env 中默认 boot_partition=boot。需要切换到 recovery 系统时，在用户空间直接设置 env，设置 boot_partition=recovery。需要切换回主系统时，设置 boot_recovery=boot。

OTA 过程需要正确设置 env 中的值。

4.7 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_data 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。

4.8 对 busybox-init 的处理

4.8.1 upgrade_etc 标志（不再使用）

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。若文件系统中已经有 rdinit 文件 (例如是一个到 pesudo_init 的软链接），则可使用 rdinit=/rdinit。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.8.2 etc_need_update 文件

rootfs_data 有两种可能的用法

1. 保存 rootfs /etc 的副本，挂载到/etc，以支持/etc 可写
2. 用作 overlayfs，以支持全目录可写，类似 procd 方案

对于早期版本，仅支持 1。对于当前最新版本，/pseudo_init 的最上方可以配置，当配置 MOUNT_ETC=1, MOUNT_OVERLAY=0 时，即为 1，跟早期版本相同，此处只讨论上述的 1。

其行为如下:

1. 烧录后第一次启动，rootfs_data 分区为空，则/pesudo_init 会格式化 rootfs_data，将/etc 文件拷贝一份到 rootfs_data 的文件系统，将 rootfs_data 挂载为/etc，对用户来说看到的/etc 就是 rootfs_data 中的
2. 第二次启动，rootfs_data 中存在有效的文件系统，直接挂载为/etc，对用户来说看到的/etc 就是 rootfs_data 中的
3. 此时若直接更新 rootfs 分区中的 rootfs，重启，用户看到的/etc 仍为 rootfs_data 中的，不会改变
4. 若创建/etc/etc_need_update 文件 (OTA 之后应该创建它)，则下次启动/pseudo_init 脚本会将 rootfs 分区中的/etc 进行一次同步
5. 同步/etc 的具体行为:
   5.1. 将 rootfs_data 挂载到/mnt
   5.2. 将 rootfs 分区的/etc 拷贝到/tmp/etc
   5.3. 将 rootfs_data 中不希望被覆盖的文件，拷贝到/tmp/etc。例如 wpa_supplocant.conf。如果有其他特殊文件不希望被 OTA 覆盖，可参考/pseudo_init 中的 wpa_supplicant 的处理方法，修改/pseudo_init。
   5.4. 将/tmp/etc 的内容，拷贝回/mnt/etc(即 rootfs_data 分区)
   5.5. 创建标志文件 etc_complete，删除标志文件 etc_need_update。若在此步骤完成前掉电，则下次启动会从步骤 1 重新同步 etc
   5.6 将 rootfs_data 的挂载点从/mnt 改为/etc, 继续启动

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.9 常见问题

4.9.1 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. 参照最新版本的做法，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。

4.9.2 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。）

4.9.3 是否可更新 boot0/uboot

misc-upgrade 原设计流程中未包含此功能

当前版本中，本地升级支持可选地更新 boot0/uboot(存在镜像则更新，不存在则跳过)，网络升级暂未支持。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

具体升级功能由 ota-burnboot 软件包支持，细节请参考相关章节。

判断是否支持的方式：搜索脚本中是否有地方调用了 ota-burnboot0 和 ota-burnuboot。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5 Tina upgrade app 介绍 (建议改用 swupdate)

5.1 功能简介

以前 Tina 只有 misc-upgrade，为了特殊需求，创建了本脚本。现在建议直接使用 swupdate。

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。

5.2 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
```

5.3 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。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.4 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
```

5.5 env 设置

对于 tina3.5.0 及之前版本，位于：

```
target/allwinner/${board}/configs/env-xxx.cfg
```

对于 tina3.5.1 及之后版本，位于：

```
device/config/chips/${chip}/configs/${board}/linux/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。

5.6 自动回退

applimit=0 时，没有任何作用。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

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。

5.7 OTA 步骤

5.7.1 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
```

5.7.2 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。

5.7.3 执行 OTA

执行脚本：

```
aw_upgrade_dual_app.sh
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.8 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
```

ALLWINNER®

6 其他功能

6.1 在用户空间操作 env

Tina 中支持了 uboot-envtools 软件包。

```
make menuconfig --> Utilities  ---> <*> uboot-envtools
```

选上后即可在用户空间使用 fw_printenv 和 fw_setenv 来读写 env 分区的变量。

6.2 AB 系统切换

6.2.1 uboot 原生启动计数机制

uboot 原生支持了启动计数功能。该功能主要涉及三个变量。

```
upgrade_available=0/1  #总开关，设置1才会进行bootcount++及检查切换，设为0则表示关闭此功能
bootcount=N            #启动计数，若upgrade_available=1，则每次启动bootcount++，并用于跟bootlimit比较
bootlimit=5            #配置判定启动失败的阈值
```

即编译支持该功能后，当 upgrade_available=1，则 uboot 会维护一个 bootcount 计数值，每次启动自动加一，并判断 bootcount 是否超过 bootlimit。如果未超过，则正常启动，执行 env 中配置的 bootcmd 命令。如果超过，则执行 env 中的 altbootcmd 命令。

支持此功能后，启动脚本或主应用需要在合适的时机清除 bootcount 计数值，表示已经正常启动。

配置：

```
使能：配置CONFIG_BOOTCOUNT_LIMIT
选择bootcount存放位置，
如存放在env分区(掉电不丢失)：配置CONFIG_BOOTCOUNT_ENV
如存放在RTC寄存器(掉电丢失)：配置CONFIG_BOOTCOUNT_RTC
如需存放在其他位置可自行拓展
```

在用户空间可配置 upgrade_available 的值对此功能进行动态开关。例如可在 OTA 之前打开，确认 OTA 成功后关闭。也可一直保持打开。在用户空间，需要清空 bootcount，否则多次重启就会导致 bootcount 超过 bootlimit。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6.2.2 全志定制系统切换

全志自己添加了 CONFIG_SUNXI_SWITCH_SYSTEM 功能进行系统切换。目前尚未大量使用，供参考。

对上接口：

在 flash 的 env 分区中设置了 2 个标志变量 systemAB_now,systemAB_next。systemAB_now：由 uboot 阶段进行设置, 该标志位系统应用只读不能写（通过工具 fw_printenv，或其他工具）。systemAB_next：无论是 uboot 还是系统应用都可读可写（一般 uboot 不会主动修改，除非系统已经损坏需要切换）初始值一般设置为

```
systemAB_next=A          #表示下次启动A系统
systemAB_now=A           #表示当前为A系统，不设置也可以，uboot会自动设置
```

当前系统应用可以通过 systemAB_now 标志识别当前系统是 A 系统还是 B 系统, 系统应用可以把标志变量 systemAB_next=A/B 写到 env 分区里, 然后重启. 重启后 uboot 会去检查 systemAB_next 这个标志, 如果标志是 systemAB_next=A,uboot 会启动 A 系统, 如果是 systemAB_next=B,uboot 会启动 B 系统。同时 uboot 会根据本次 systemAB_next 的值设置 systemAB_now 供系统读取。

底层设置：

对上接口定义的是 systemA/B，以系统为单位。目前的系统组成定义为包含 kernel 和 rootfs 分区。考虑不同方案，分区名可能不同，因此支持用环境变量指定具体分区名，而不是 hardcode 为某个分区名。

```
systemA=boot             A系统kernel分区名
rootfsA=rootfs           A系统rootfs分区名
systemB=boot_B           B系统kernel分区名
rootfsB=rootfs_B         B系统rootfs分区名
```

底层实现：

此机制会动态修改 boot_partition 和 root_partition 的值。具体 boot_partition 和 root_partition 变量的作用，请参考本文档其他章节的介绍。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

7 注意事项

7.1 Q & A

Q：系统的哪些部分是可以升级的?

A：kernel 和 rootfs 是可以升级的，但为了掉电安全，需要搭配一个 recovery 系统或者做 AB 系统。对于 nand 和 emmc 来说，boot0/uboot 存在备份，可以升级。对于 nor 来说，boot0/uboot 没有备份，不能升级。或者说升级有风险，中途掉电会导致无法启动。boot0/uboot 的升级具体可参考本文档中的 ota-burnboot 部分。对于 swupdate 升级方案，可以自行在 sw-description 中配置策略，升级自己的定制分区和文件，但务必考虑升级中途掉电的情况，必要的话需要做备份和恢复机制。

Q：系统的哪些部分是不能升级的?

A：分区表是不可升级的，因为改动分区表后，具体分区对应的数据也要迁移。建议在量产前规划好分区，为每个可能升级的分区预留部分空间，防止后续升级空间不足。nor 方案的 boot0/uboot 是不可升级的，因为没有备份。其余不确定是否能够升级的，请向开发人员确认。

Q：dts/sys_config 如何升级?

A：默认 dts 和 sys_config，会跟 uboot 绑定生成一个 bin 文件。因此升级 uboot 实质上是升级了 uboot+dts/sys_config。

Q：能否单独升级 dts?

A：目前默认跟 uboot 绑定，需要跟开发人员确认如何将 dts 独立出来，放到独立分区或者跟 kernele 绑定到一起。 如果是 dts 位于独立分区，那么就需要修改配置，将 dts 放置到 OTA 包中，OTA 时写入到对应分区。

Q：升级过程掉电重启后，是从断点继续升级还是从头升级?

A：从头开始升级，例如定义了在 recovery 系统中升级 boot 分区和 rootfs 分区，则在升级 boot 或 rootfs 过程中断电，重启后均是从 boot 重新开始升级。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

8 升级失败问题排查

凡是遇到升级失败问题先看串口 log，如果不行再看/mnt/UDISK/swupdate.log 文件

8.1 分区比镜像文件小引起的失败

log 大概如下。找 error 的地方，可以看到 recovery 分区比其镜像小，所以报错。

```
[ERROR] : SWUPDATE failed [0] ERROR handlers/ubivol_handler.c : update_volume : 171 : "
    recovery"  will not fit volume "recovery"
```

解决方法：增加对应方案 tina/device/config/chips/< 芯片编号 >/configs/< 方案名 >/sys_partition.fex 文件的对应分区的大小

8.2 校验失败

差分有严格的版本控制，当出现 checksum 有问题时，基本可以归类为这种问题。

解决方法：重新烧录固件，制作差分包

版权所有 © 珠海全志科技股份有限公司。保留一切权利

著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、、全志科技、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。