D1 Tina Linux OTA
开发指南

版本号: 1.0
发布日期: 2021.04.02

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.02	AWA1615	初始版本

目　录

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：

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。

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	sys_config 会在打包阶段使用；linux5.4 中不再合并到 dtb。
dtb	设备树，由 dts 配置得到。
boot_package.fex	最终用到的 uboot 和其他文件，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，可能还有 bootlogo 等文件。

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 d1-nezha
    => 选择d1-nazha方案。
$ muboot
    => 编译 uboot。
$ mboot0
    => 编译 boot0。
```

编译后会自行拷贝 bin 文件到该平台的目录下, 即：

```
device/config/chips/d1/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
=> D1方案的打包命令。
```

2.3.2 关于更新 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。

2.3.3 Bin 文件路径

2.3.3.1 使用 uboot2018 的非安全方案

以 d1 的 nezha 方案为例。

根据对应存储介质选择 bin。

boot0：

```
out/d1-nezha/image/boot0_nand.fex          : nand 方案使用的 boot0,d1-nezha使用此文件。
```

uboot：

```
out/d1-nezha/image/boot_package.fex        : nand 方案使用的 uboot,d1-nezha使用此文件。
```

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。

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。

2.4.3.2 使用示例

```
root@TinaLinux:/# ota-burnuboot /tmp/boot_package.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)

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 UBI 方案实现

d1-nezha 使用此方法

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

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。

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。

- 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/d1-nezha/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/d1-nezha/defconfig
```

对于 recovery 系统，使用：

```
make ota_menuconfig
```

配置结果保存在：

```
target/allwinner/d1-nezha/defconfig_ota
```

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。

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.
```

```
    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/d1-nezha/defconfig_ota
```

编译得到：

```
out/d1-nezha/boot_initramfs_recovery.img
```

3.2.9 配置 env

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。

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 配置启动脚本

procd-init 是默认配置好的。

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/d1-nezha/swupdate/sw-description-ab
target/allwinner/d1-nezha/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/d1-nezha/swupdate/sw-description-ab-sign拷贝成sw-description，后续同理。
target/allwinner/d1-nezha/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/d1-nezha/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/d1-nezha/swupdate/sw-subimgs-ab.cfg            # 改为AB系统。
target/allwinner/d1-nezha/swupdate/sw-subimgs-secure.cfg        # 改为安全系统。
target/allwinner/d1-nezha/swupdate/sw-subimgs-ab-rdiff.cfg      # 改为AB方案，差分方案。
target/allwinner/d1-nezha/swupdate/sw-subimgs-readback.cfg      # 增加回读校验。
target/allwinner/d1-nezha/swupdate/sw-subimgs-sign.cfg          # 增加签名校验。
```

技巧

当前 *d1-nezha* 使用 *ubi* 介质，所以 *target/allwinner/d1-nezha/swupdate* 目录下的文件才是适合当前方案的。其余介质的 *OTA* 描述文件和配置文件参考 *target/allwinner/generic/swupdate/*目录的文件

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。

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

d1-nezha 方案这里我们直接使用：

```
target/allwinner/d1-nezha/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";

    /*
     * 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 */
                volume = "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，
    * 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";
                    volume = "boot";
                    installed-directly = true;
                },
                {
                    filename = "rootfs";
                    volume = "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 = {
            /*
             * 清除一些标志。
            */
            /* 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 = "";
                },
                {
                    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/d1-nezha/swupdate/sw-subimgs.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，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/d1-nezha/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
```

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 d1-nezha 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
```

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/d1-nezha/swupdate/tina-d1-nezha.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/tina-d1-nezha.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/tina-d1-nezha.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。

tina/target/allwinner/d1-nezha/swupdate 目录下的 env_ab.cfg 和 sys_partition_ab.fex 是 ab 系统配置文件的 demo，把这 2 个文件复制到 tina/device/config/chips/d1/configs/nezha 目录，并替换原本的 env.cfg 和 sys_partiton.fex。

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/d1-nezha/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 */
                    volume = "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 */
                volume = "rootfsB";
                installed-directly = true;
            },
            {
                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";
                volume = "bootA";
```

```
                    installed-directly = true;
                },
                {
                    filename = "rootfs";
                    volume = "rootfsA";
                    installed-directly = true;
                },
                {
                    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：

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/d1-nezha/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/d1-nezha/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。

```
pack
```

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 d1-nezha 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/d1-nezha/swupdate/tina-d1-nezha-ab.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/tina-d1-nezha-ab.swu -e stable,now_A_next_B
当前处于B系统：
swupdate -i /mnt/UDISK/tina-d1-nezha-ab.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/tina-d1-nezha-ab.swu -e stable,now_A_next_B
当前处于B系统：
swupdate_cmd.sh -i /mnt/UDISK/tina-d1-nezha-ab.swu -e stable,now_B_next_A
```

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。

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。

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。

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。

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/d1-nezha/swupdate/swupdate_priv.password
private key:tina/target/allwinner/d1-nezha/swupdate/swupdate_priv.pem
public key:tina/target/allwinner/d1-nezha/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/d1-nezha/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。

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/d1-nezha/swupdate/sw-description-sign b/allwinner/d1-nezha/swupdate/
    sw-description-ab
index ed04b64..467ac3b 100644
--- a/allwinner/d1-nezha/swupdate/sw-description-ab
+++ b/allwinner/d1-nezha/swupdate/sw-description-ab
@@ -9,14 +9,17 @@ software =
                {
                    filename = "recovery"
                    volume = "recovery";
+                   sha256 = "@recovery"
                },
                {
                    filename = "uboot"
                    type = "awuboot";
+                   sha256 = "@uboot"
                },
                {
                    filename = "boot0"
                    type = "awboot0";
+                   sha256 = "@boot0"
                }

            );
@@ -34,10 +37,12 @@ software =
                {
                    filename = "kernel";
                    volume = "boot";
+                   sha256 = "@kernel"
                },
                {
                    filename = "rootfs";
                    volume = "rootfs";
+                   sha256 = "@rootfs"
                }
            );
            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/d1-nezha/swupdate/sw-description-sign
target/allwinner/d1-nezha/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-
    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/
```

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 -i /mnt/UDISK/tina-d1-nezha-sign.swu -k /etc/swupdate_public.pem
```

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/d1-nezha/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
```

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。

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/d1-nezha/swupdate/tina-d1-nezha-sign.swu  /mnt/UDISK
```

小机端执行 (不带签名校验版本)：

```
swupdate_cmd.sh -i /mnt/UDISK/tina-d1-nezha-sign.swu -e stable,upgrade_recovery
```

小机端执行 (带签名校验版本)：

```
swupdate_cmd.sh -i /mnt/UDISK/tina-d1-nezha-sign.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/d1-nezha/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-d1-nezha-sign.swu -e stable,
    upgrade_recovery
```

例如 (带签名校验版本)：

```
swupdate_cmd.sh -d -uhttp://192.168.35.112/tina-d1-nezha-sign.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

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。

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。

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/d1-nezha/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/d1-nezha/tina_d1-nezha_uart0.img    tina_d1-nezha_uart0_OTA1.img
       cp   out/d1-nezha/swupdate/tina-d1-nezha.swu   tina-d1-nezha_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/d1-nezha/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/d1-nezha/tina_d1-nezha_uart0.img    tina_d1-nezha_uart0_OTA2.img
cp   out/d1-nezha/swupdate/tina-d1-nezha.swu   tina-d1-nezha_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/d1-nezha/swupdate/tina-d1-nezha_OTA1.swu /mnt/UDISK
```

小机端执行：

```
swupdate_cmd.sh  -i /mnt/UDISK/tina-d1-nezha_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-d1-nezha_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
```

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：

```
out/d1-nezha/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/d1-nezha/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/d1-nezha/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";
                    volume = "recovery";
                    installed-directly = true;
                },
        /* 更新mypart-r， 此时掉电，mypart分区是完整的 */
                {
                    filename = "mypart-r";
                    volume = "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";
                    volume = "boot";
                    installed-directly = true;
                },
```

```
                {
                    filename = "rootfs";
                    volume = "rootfs";
                    installed-directly = true;
                },
        /* 更新mypart，  此时掉电，mypart-r分区还是完整的 */
                {
                    filename = "mypart-r";
                    volume = "mypart";
                    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

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。

目前支持 nand，详见 ota-burnboot 章节。

使用方式：

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 */
}
```

3.15.2 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。

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

当前 d1-nezha 使用 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
```

target/allwinner/d1-nezha/swupdate/目录下的文件都是根据 target/allwinner/generic/swupdate/sw-description-ubi 和 sw-subimgs-ubi.cfg 适配成 ubi 的

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/d1-nezha/swupdate/sw-description-ab-rdiff
target/allwinner/d1-nezha/swupdate/sw-subimgs-ab-rdiff.cfg
```

差分文件的生成使用 rdiff：

```
$ 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/d1-nezha/swupdate/tina-d1-nezha-ab.swu
cp out/d1-nezha/swupdate/tina-d1-nezha-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/d1-nezha/swupdate/tina-d1-nezha-ab.swu
cp out/d1-nezha/swupdate/tina-d1-nezha-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
```

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。

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/d1-nezha/swupdate/sw-description-ab-rdiff-sign
target/allwinner/d1-nezha/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/d1-nezha/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/d1-nezha/swupdate/tina-d1-nezha-ab.swu tina-d1-nezha-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/d1-nezha/swupdate/tina-d1-nezha-ab.swu tina-d1-nezha-ab_V2.swu

# 此时 tina-d1-nezha-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-d1-nezha-ab_V1.swu ./tina-d1-nezha-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
```

4 注意事项

4.1 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 来说，boot0/uboot 存在备份，可以升级。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。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。

5 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

5.1 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 tina/device/config/chips/d1/configs/nezha/sys_partition.fex 文件的对应分区的大小

5.2 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：重新制作差分包

著作权声明

商标声明

免责声明