



A40i 系列 & T3 系列 Linux-5.10 OTA 开发指南

版本号: 1.0

发布日期: 2022.11.10

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.11.10	AWA1739	初始版本

目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
1.4 OTA 方案	1
1.4.1 recovery 系统方案	1
1.4.2 AB 系统方案	2
2 ota-burnboot 介绍	3
2.1 文档说明	3
2.2 概念说明	3
2.3 用于更新的 bin 文件	4
2.3.1 编译 boot0 uboot	4
2.3.2 关于更新 boot0	4
2.3.3 Bin 文件路径	5
2.3.3.1 使用 u-boot-2018 及更高版本的非安全方案	5
2.3.3.2 安全方案	5
2.4 OTA 升级命令	5
2.4.1 支持 OTA 升级命令	5
2.4.2 ota-burnboot0	6
2.4.2.1 命令说明	6
2.4.2.2 使用示例	6
2.4.3 ota-burnuboot	6
2.4.3.1 命令说明	6
2.4.3.2 使用示例	6
2.5 OTA 升级 C/C++ APIs	6
2.5.1 int OTA_burnboot0(const char *img_path)	7
2.5.2 int OTA_burnuboot(const char *img_path)	7
2.6 底层实现	7
2.6.1 如何保证安全更新 boot0/uboot	7
2.6.2 Nand Flash NFTL 方案实现	7
2.6.3 Nand Flash UBI 方案	8
2.6.4 MMC Flash 实现	8
2.6.5 NOR Flash 实现	8
3 Tina SWUpdate OTA 介绍	9
3.1 swupdate 介绍	9
3.1.1 简介	9
3.1.2 移植到 tina5.0 buildroot 文件系统下的改动	9

3.2 配置	10
3.2.1 recovery 系统介绍	10
3.2.2 系统配置命令	10
3.2.3 主系统和 recovery 都需要的 swupdate 包	10
3.2.4 配置主系统	11
3.2.5 编译主系统	11
3.2.6 配置 recovery 系统	11
3.2.7 编译 recovery 系统	12
3.2.8 配置 env	12
3.2.8.1 boot_partition 变量	12
3.2.8.2 root_partition 变量	13
3.3 OTA 包	13
3.3.1 OTA 策略描述文件: sw-description	13
3.3.2 OTA 包配置文件: sw-subimsgs.cfg	14
3.3.3 OTA 包生成: swupdate_pack_swu	15
3.4 recovery 系统方案举例	15
3.4.1 配置分区和 env	15
3.4.2 配置主系统	16
3.4.3 配置 recovery 系统	16
3.4.4 准备 sw-description-recovery	16
3.4.5 准备 sw-subimsgs-recovery.cfg	20
3.4.6 编译 OTA 包所需的子镜像	20
3.4.7 执行 OTA	21
3.4.7.1 准备 OTA 包	21
3.4.7.2 调用 swupdate	21
3.5 AB 系统方案举例	21
3.5.1 配置分区和 env	21
3.5.2 配置主系统	22
3.5.3 配置 recovery 系统	22
3.5.4 准备 sw-description	22
3.5.5 准备 sw-subimsgs-ab.cfg	25
3.5.6 编译 OTA 包所需的子镜像	26
3.5.7 执行 OTA	26
3.5.7.1 准备 OTA 包	26
3.5.7.2 判断 AB 系统	26
3.5.7.3 调用 swupdate	27
3.6 辅助脚本 swupdate_cmd.sh	27
3.7 版本号	28
3.7.1 使用方式	28
3.7.2 实现例子	28
3.8 调用 OTA	30
3.8.1 进度条	30
3.8.2 重启	30

3.8.3 本地升级示例	31
3.8.4 错误处理	31
3.9 裁剪	31
3.10 调试	32
3.10.1 直接调用 swupdate	32
3.10.2 手工切换系统	32
3.10.3 更新 boot0/uboot	32
3.10.4 解压 OTA 包	33
3.11 测试固件示例	33
3.11.1 生成方式	33
3.11.1.1 生成固件 1 和 OTA 包 1	33
3.11.1.2 生成固件 2 和 OTA 包 2	34
3.11.2 使用方式	34
3.11.2.1 本地升级方式	34
3.11.2.2 升级过程	34
3.11.2.3 判断升级	35
3.12 升级定制分区	35
3.13 handler 说明	35
3.13.1 awboot	36
3.13.1.1 nand/emmc	36
3.13.2 nand ubi 方案	36
3.13.2.1 示例	37
3.14 Emmc/Nand 自适应 OTA 升级	37
3.14.1 Emmc/Nand 自适应要求	37
3.14.2 Emmc/Nand 自适应 swu 制作方法步骤	38
4 其他功能	39
4.1 在用户空间操作 env	39
4.2 AB 系统切换	39
4.2.1 uboot 原生启动计数机制	39
5 注意事项	40
5.1 Q & A	40
6 升级失败问题排查	41
6.1 分区比镜像文件小引起的失败	41

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 Tina5.0 buildroot 文件系统软件平台的广大客户，以帮助客户使用 Tina5.0 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 Tina5.0。

1.3 相关人员

适用 Tina5.0 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 intiramps 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	全志特有的配置文件, 对于使用 linux3.4/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 linux3.4/uboot2011 的平台最终用到的 uboot, 其实是 uboot+sys_config。
boot_package.fex	使用 linux3.10/uboot2014 及更高版本的平台最终用到的 uboot, 其实包含的文件由配置文件 boot_package.cfg 决定, 一般至少包含了 uboot 和 dtb, 安全方案会包含一些安全所需文件文件, 可能还有 bootlogo 等文件。
toc0.fex	安全方案使用的 boot0。
toc1.fex	安全方案使用的 uboot, 类似 boot_package.fex 说明, 其中实际也包含了 dts 等多个文件。

即，本文介绍的升级 uboot，其实是升级 uboot+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。
$ ./build.sh config
=> 选择方案。
$ build.sh bootloader
=> 编译 uboot 和 编译 boot0。
```

编译后会自行拷贝 bin 文件到该平台的目录下，即：对于 tina5.0 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：

```
$ ./build.sh
=> 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。
```

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 build.sh bootloader 编译。

2.3.3 Bin 文件路径

2.3.3.1 使用 u-boot-2018 及更高版本的非安全方案

以 A40I 的 p3 方案为例。

根据对应存储介质选择 bin。

boot0:

out/a40i/p3/pack_out/boot0_nand.fex	: nand 方案使用的 boot0。
out/a40i/p3/pack_out/boot0_sdcard.fex	: mmc 方案使用的 boot0。
out/a40i/p3/pack_out/boot0_spinor.fex	: nor 方案使用的 boot0。

uboot:

out/a40i/p3/pack_out/boot_package.fex	: nand/mmc 方案使用的 uboot。
out/a40i/p3/pack_out/boot_package_nor.fex	: nor 方案使用的 uboot。

2.3.3.2 安全方案

以 A40I 的 P3 方案为例。

boot0:

out/a40i/p3/pack_out/toc0.fex	: 安全方案使用的 boot0。
-------------------------------	------------------

uboot:

out/a40i/p3/pack_out/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, 源码路径在: platform/allwinner/system/ota-burnboot 编译配置路径在: buildroot/config/buildroot/allwinner/system/ota, 即:

```
./build.sh buildroot_menuconfig ---> Target packages ---> allwinner platform private  
package select ---> system ---> <*>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。

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

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 中存有不只一份 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

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 ioctl 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。

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。

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 buildroot/buildroot-202205/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 tina5.0 buildroot 文件系统下的改动

移植到 tina5.0 主要做了以下修改：

- 位置在 buildroot/buildroot-202205/package/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 buildroot/buildroot-202205/configs/sun8iw11p1_recovery_ramfs_defconfig。

如果没有此文件，可以拷贝 xxx_defconfig 为 sun8iw11p1_recovery_ramfs_defconfig，再做配置裁剪。

3.2.2 系统配置命令

对于主系统，使用：

```
./build.sh buildroot_menuconfig  
./build.sh buildroot_saveconfig
```

配置结果保存在：

```
buildroot/buildroot-202205/configs/defconfig
```

对于 recovery 系统，进入 buildroot/buildroot-202205 目录后，使用：

```
make sun8iw11p1_recovery_ramfs_defconfig  
make menuconfig  
make savedefconfig
```

配置结果保存在：

```
buildroot/buildroot-202205/configs/sun8iw11p1_recovery_ramfs_defconfig
```

3.2.3 主系统和 recovery 都需要的 swupdate 包

选上 swupdate 包。


```
--> Target packages --> System tools --> [*]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 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。

3.2.5 编译主系统

正常./build.sh 即生成主系统。

```
./build.sh
```

3.2.6 配置 recovery 系统

对于 Buildroot recovery 系统，会编译出 cpio 文件，因此使用 gzip 压缩方式以节省 flash 空间，进入 buildroot/buildroot-202205 目录下执行：

```
make sun8iw11p1_recovery_ramfs_defconfig
make menuconfig
---> Filesystem images
----> [*] cpio the root filesystem (for use as an initial RAM filesystem)
    Compression method (gzip) ---->
        (X) gzip
```

编译 recovery 独立内核

#说明：

默认recovery系统跟主系统使用不同的内核配置文件，优点是支持定制化裁减recovery内核，缺点是需要注意维护两份kernel配置。

#准备独立内核配置(以linux-5.10为例)

```
cd device/config/chips/${CHIP}/configs/${BOARD}/linux-5.10
```

```
cp buildroot_bsp_defconfig buildroot_bsp_recovery_defconfig
```

#选择独立内核配置，在device/config/chips/\${CHIP}/configs/\${BOARD}/buildroot 添加如下配置：

```
LICHEE_KERN_DEFCONF_RECOVERY=buildroot_bsp_recovery_defconfig
```

#裁减独立内核配置

```
./build.sh recovery_menuconfig
```

#注意

1. ./build.sh recovery_menuconfig命令生成的配置先跟./build.sh menuconfig生成的配置比较下，如果除了

- 主动裁减部分，其他也有差异，则需联系全志协助修改。
2. 使用 `./build.sh recovery_menuconfig` 配置好后，使用 `./build.sh recovery_saveconfig` 保存配置。

3.2.7 编译 recovery 系统

要编译生成 recovery 系统，可使用：

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

编译得到：

```
out/a40i/p3/buildroot/recovery.img
```

说明

注意：当执行 `./build.sh recovery` 不会编译文件系统，只会拷贝 `cpio` 文件到 `img` 中。`recovery` 系统需要单独编译 `cpio` 文件，默认已经提供编译好的 `cpio` 文件，如有需要修改，请咨询全志工程师进行协助。

3.2.8 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。

3.2.8.1 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=sunxi_flash read 45000000 boot;bootm 45000000
```

改成：

```
boot_partition=boot
boot_normal=sunxi_flash read 45000000 ${boot_partition};bootm 45000000
```

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。

3.2.8.2 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。

3.3 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
```

也可以自行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${LICHEE_BOARD_CONFIG_DIR}/${LICHEE_LINUX_DEV}/swupdate/sw-description-ab拷贝成sw-  
description，后续同理。  
${LICHEE_BOARD_CONFIG_DIR}/${LICHEE_LINUX_DEV}/swupdate/sw-description-ab:sw-description  
${LICHEE_PLAT_OUT}/uboot.img:uboot  
${LICHEE_PACK_OUT_DIR}/boot0_nand.fex:boot0  
${LICHEE_PACK_OUT_DIR}/boot.fex:kernel  
${LICHEE_PACK_OUT_DIR}/rootfs-ubifs.fex:rootfs  
)
```

tina5.0 提供了几个示例：

```
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs.cfg  
# 默认系统，AB系统，整包升级。这是其余demo的基础版本。  
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs-ab.cfg  
# 改为AB系统。  
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs-ab-emmc-nand.cfg  
# 改为nand和emmc自适应AB系统。  
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs-recovery.cfg  
# 改为recovery方案。  
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs-recovery-emmc-nand.  
cfg # 改为nand和emmc自适应recovery方案。
```

也可以自行具体的方案编写描述文件。

```
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs.cfg
```

本文件在 SDK 中的路径需位于 device/config/chips/a40i/configs/buildroot/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-description 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 -ab，则会寻找 sw-subimgs-ab.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，swupdate_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 recovery 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

./build.sh config 选择方案后，./build.sh buildroot_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。./build.sh config 选择方案后，拷贝配置文件。

```
cd device/config/chips/a40i/configs/<board>/linux-5.10
cp xxx_defconfig_sun8iw1lp1_recovery_ramfs_defconfig
```

根据上文介绍，./build.sh buildroot_menuconfig 选上 swupdate, ramdisk, recovery 后缀等必要的配置。

recovery 系统整个运行在 ram 中，如果系统过大无法启动，所以需要进行裁剪。./build.sh recovery_menuconfig，将不必要的包尽量从 recovery 系统中去掉。

3.4.4 准备 sw-description-recovery

这里我们直接使用：

```
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-description-recovery
```

内容如下，该配置是 emmc 版型配置，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 */
                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 = "awuboot"; /* 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";
            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 = "";
    },
    {
        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。

注意：对于 spinand/raw NAND ubi 方案，sw-description-recovery 配置需要注意的是有如下三点修改：

```
#device = "/dev/by-name/recovery"; ==> volume = "recovery";
#device = "/dev/by-name/boot";      ==> volume = "boot";
#device = "/dev/by-name/rootfs";    ==> volume = "rootfs";
```


3.4.5 准备 sw-subimgs-recovery.cfg

我们直接看下 tina5.0 版型包的：

```
device/config/chips/a40i/configs/<board>/buildroot/swupdate/sw-subimgs-recovery.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  
${LICHEE_BOARD_CONFIG_DIR}/${LICHEE_LINUX_DEV}/swupdate/sw-description-recovery:sw-  
description  
#取得boot_initramfs_recovery.img, 重命名为recovery, 放到OTA包中。以下雷同  
${LICHEE_PLAT_OUT}/recovery.img:recovery  
#uboot.img和boot0_sdcard.fex是执行swupdate_pack_swu时自动拷贝得到的, 需配置sys_config.fex中的  
storage_type  
out/${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  
${LICHEE_PACK_OUT_DIR}/boot0_sdcard.fex:boot0  
${LICHEE_PACK_OUT_DIR}/boot.fex:kernel  
${LICHEE_PACK_OUT_DIR}/rootfs.fex:rootfs  
#下面这行是给小容量方案预留的, 目前注释掉  
#out/${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。

注意：对于 spinand/rawnand ubi 方案，sw-subimgs-recovery.cfg 配置需要注意的是有如下三点修改：

```
${LICHEE_PACK_OUT_DIR}/boot0_sdcard.fex:boot0 ==> ${LICHEE_PACK_OUT_DIR}/boot0_nand.fex;  
boot0  
${LICHEE_PACK_OUT_DIR}/rootfs.fex ==> ${LICHEE_PACK_OUT_DIR}/rootfs-ubifs.  
fex
```

3.4.6 编译 OTA 包所需的子镜像

编译 kernel 和 rootfs。

```
./build.sh
```

编译 recovery 系统。

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

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。

```
./build.sh pack / ./build.sh pack_secure
```

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

注意，如果方案目录下存在 sw-subimgs-recovery.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 -ab / swupdate_pack_swu -recovery
```

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/a40i/<board>/buildroot/swupdate/buildroot_a40i_<board>-recovery.swu /mnt/  
UDISK
```

3.4.7.2 调用 swupdate

若使用原生的 swupdate，则调用：

```
swupdate -i /mnt/UDISK/buildroot_a40i_<board>-recovery.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/buildroot_a40i_<board>-recovery.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。

3.5.2 配置主系统

./build.sh config 选择方案后, ./build.sh buildroot 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:

```
device/config/chips/a40i/configs/<board>/buildroot/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tina5.0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;
},
{
    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 = "awuboot0"; /* 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;
        },
        {
            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。

注意：对于 spinand/rawnand ubi 方案，sw-description-ab 配置如下所示，需要注意的是有如下三点修改：

```
#device = "/dev/by-name/bootA";    ==> volume = "bootA";  
#device = "/dev/by-name/rootfsA";  ==> volume = "rootfsA";  
#device = "/dev/by-name/bootB";    ==> volume = "bootB";  
#device = "/dev/by-name/rootfsB";  ==> volume = "rootfsB";
```

3.5.5 准备 sw-subimgs-ab.cfg

我们直接看下 tina5.0 版型包下的：

```
device/config/chips/a40i/configs/<board>/buildroot/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  
${LICHEE_BOARD_CONFIG_DIR}/${LICHEE_LINUX_DEV}/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_sdcard.fex是执行swupdate_pack_swu时自动拷贝得到的，需配置sys_config.fex中的  
storage_type  
${LICHEE_PLAT_OUT}/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  
${LICHEE_PACK_OUT_DIR}/boot0_sdcard.fex:boot0  
${LICHEE_PACK_OUT_DIR}/boot.fex:kernel  
${LICHEE_PACK_OUT_DIR}/rootfs.fex:rootfs  
#${LICHEE_PLAT_OUT}/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。

注意：对于 spinand/rawnand ubi 方案，sw-subimsgs-recovery.cfg 配置需要注意的是有如下三点修改：

```
{LICHEE_PACK_OUT_DIR}/boot0_sdcard.fex:boot0 ==> ${LICHEE_PACK_OUT_DIR}/boot0_nand.fex:boot0
${LICHEE_PACK_OUT_DIR}/rootfs.fex ==> ${LICHEE_PACK_OUT_DIR}/rootfs-ubifs.fex
```

3.5.6 编译 OTA 包所需的子镜像

编译 kernel 和 rootfs。

```
./build.sh
```

打包

```
./build.sh pack / ./build.sh pack_secure
```

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

注意，如果方案目录下存在 sw-subimsgs-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/a40i/<board>/buildroot/swupdate/buildroot_a40i_<board>-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/buildroot_a40i_<board>-ab.swu -e stable,now_A_next_B
当前处于B系统：
swupdate -i /mnt/UDISK/buildroot_a40i_<board>-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/buildroot_a40i_<board>-ab.swu -e stable,now_A_next_B
当前处于B系统：
swupdate_cmd.sh -i /mnt/UDISK/buildroot_a40i_<board>-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 时也无法传入获得并版本号，则第一次升级时不会检查版本。

注：这是 tina5.0 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 tina5.0 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。  
            Tina5.0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 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。

3.8.1 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。

3.8.2 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。

3.8.3 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。

3.8.4 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"。

3.9 裁剪

swupdate 本身是可配置的，不需要某些功能时，可将其裁剪掉。

```
./build.sh buildroot_menuconfig --> Target packages --> System tools --> [*]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

```
./build.sh buildroot_menuconfig
Target packages --->
System tools --->
  <*> swupdate --->
    Image Handlers -->
      [*] allwinner boot0/uboot
```

取消掉

3.10 调试

3.10.1 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
```

3.10.2 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
```

3.10.3 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。

3.10.4 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
```

3.11 测试固件示例

3.11.1 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。

3.11.1.1 生成固件 1 和 OTA 包 1

重新编译 boot，使得编译时间更新：

```
./build.sh bootloader
```

重新编译打包，使得编译时间更新：

```
./build.sh
```

重新编译 recovery 系统：

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

生成 OTA 包 1：

```
swupdate_pack_swu -recovery
```

得到产物：

```
cp out/a40i_linux_p3_uart0.img a40i_linux_p3_uart0_OTA1.img
cp out/a40i/p3/buildroot/swupdate/buildroot_a40i_p3-recovery.swu
buildroot_a40i_p3-recovery_OTA1.swu
```

3.11.1.2 生成固件 2 和 OTA 包 2

更新代码后重新编译 boot，使得编译时间更新：

```
./build.sh bootloader
```

更新代码后重新编译打包，使得编译时间更新：

```
./build.sh
```

重新编译 recovery 系统：

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

生成 OTA 包 2：

```
swupdate_pack_swu -recovery
```

得到产物：

```
cp out/a40i_linux_p3_uart0.img a40i_linux_p3_uart0_OTA2.img
cp out/a40i/p3/buildroot/swupdate/buildroot_a40i_p3-recovery.swu
buildroot_a40i_p3-recovery_OTA2.swu
```

3.11.2 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。

3.11.2.1 本地升级方式

PC 端执行：

```
adb push out/a40i/p3/buildroot/swupdate/buildroot_a40i_p3-recovery_OTA1.swu /mnt/UDISK
```

小机端执行：

```
swupdate_cmd.sh -i /mnt/UDISK/buildroot_a40i_p3-recovery_OTA1.swu
```

3.11.2.2 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。

3.11.2.3 判断升级

uboot 和内核从 log 中的时间可以判断当前运行的版本。

例如：

OTA_1:

U-Boot 2018.07-00022-g302cd3e5bf-dirty (Nov 08 2022 - 16:46:23 +0800) Allwinner Technology

[0.000000] Linux version 5.10.149 (liuzhaoqin@GZExdroid02) (arm-linux-gnueabi-gcc (Linaro GCC 5.3-2016.05) 5.3.1 20160412, GNU ld (Linaro Binutils-2016.05) 2.25.0 Linaro 2016_02) #3 SMP PREEMPT Tue Nov 8 17:15:46 CST 2022 (注, 进入控制台cat /proc/version 也可看到)

OTA_2:

U-Boot 2018.07-00022-g302cd3e5bf-dirty (Nov 08 2022 - 16:56:51 +0800) Allwinner Technology

[0.000000] Linux version 5.10.149 (liuzhaoqin@GZExdroid02) (arm-linux-gnueabi-gcc (Linaro GCC 5.3-2016.05) 5.3.1 20160412, GNU ld (Linaro Binutils-2016.05) 2.25.0 Linaro 2016_02) #3 SMP PREEMPT Tue Nov 8 17:25:38 CST 2022 (注, 进入控制台cat /proc/version 也可看到)

3.12 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。

3.13 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。

3.13.1 awboot

3.13.1.1 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：

选上 handler 支持：

```
./build.sh buildroot_menuconfig
Target packages --->
System tools --->
  <*> swupdate --->
    Image Handlers -->
      [*] allwinner boot0/uboot
```

对于 recovery 系统，进入 buildroot/buildroot-202205 目录后，使用：

```
make sun8iw1lp1_recovery_ramfs_defconfig
make menuconfig //加入如上配置
```

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 = "awuboot0"; /* type为awuboot，则swupdate会调用对应的handler做处理 */
}
```

3.13.2 nand ubi 方案

用于 p3-spinand ubi 和 p3-rawnand ubi 方案。需要选择上 MTD 方案，在 buildroot 和 buildroot_recovery 系统下的 defconfig 配置都需要进行修改：

```
--> Target packages
--> Filesystem and flash utilities
---> [*] mtd, jffs2 and ubi/ubifs tools
```

需先选上 swupdate MTD 支持，在 buildroot 和 buildroot_recovery 系统下的 defconfig 配置都需要进行修改：

```
--> Target packages
--> System tools
--> swupdate
--> Swupdate Settings
--> General Configuration
--> [*] MTD support
```

再选上对应 swupdate handler，在 buildroot 和 buildroot_recovery 系统下的 defconfig 配置都需要进行修改：

```
--> Target packages
--> System tools
--> swupdate
--> Image Handlers
--> [*] ubivol
```

3.13.2.1 示例

请参考 p3-rawnand 和 p3-spinand 版型下面配置文件：

```
device/config/chips/a40i/configs/p3-rawnand-ubi/buildroot/swupdate/sw-subimgs-recovery.cfg
device/config/chips/a40i/configs/p3-rawnand-ubi/buildroot/swupdate/sw-description-recovery
```

最后使用 swupdate_pack_swu -recovery 命令打包 OTA 包

3.14 Emmc/Nand 自适应 OTA 升级

Emmc/Nand 自适应 OTA 升级功能是指除了存储器之外，开发板其他模块一样的情况下，Emmc 样机和 Nand 样机可以都使用同一个 swu 实现 ota 正常升级，并且可以用 recovery 方式和 ab 方式两种方式升级。

3.14.1 Emmc/Nand 自适应要求

因为 emmc 和 nand 要保证编译的 rootfs, kernel, uboot 一致。所以用 recovery 和 ota 两种升级方式实现自适应需要客户自行修改如下 5 点：（1）emmc 和 nand 版型中的 uboot, kernel 和 buildroot 下的三份 defconfig 文件都需要一致

（2）emmc 和 nand 版型中的 uboot 和 kernel 下的两份 dts 文件都需要一致

（3）fw_env.config 这个需要根据 nand 还是 emmc 来动态加载，目前是根据编译选项判断的，客户可以自行修改，不过这个只会影响到 fw_printenv 命令执行，其他功能不影响。

Emmc样机中fw_env.config如下配置

```
$ cat fw_env.config  
/dev/mmcblk0p2 0x00000 0x20000
```

Nand样机中fw_env.config如下配置

```
$ cat fw_env.config  
/dev/ubi0:env 0x0 0x20000 0x20000
```

(4) 需要确保 emmc+rawnand 搭配或者 emmc + spinand 搭配来实现 nand 和 emmc 自适应。

(5) 需要在 buildroot/buildroot-202205/package/swupdate/swupdate_cmd.sh 进行如下修改，实现 swupdate 能够自适应功能：具体修改如下：

```
diff --git a/package/swupdate/swupdate_cmd.sh b/package/swupdate/swupdate_cmd.sh  
index 352a18e8..832215d9 100755  
--- a/package/swupdate/swupdate_cmd.sh  
+++ b/package/swupdate/swupdate_cmd.sh  
@@ -85,8 +85,8 @@ mkdir -p /var/lock  
# echo "swu_software: ##$swu_software##"  
echo "swu_software $swu_software" >> /tmp/swupdate_param_file  
swu_mode=$(echo "$swu_param_e" | awk -F ' ' '{print $2}')-# echo "swu_mode: ##$swu_mode##"  
-# swu_mode=${swu_mode}_${get_flash_type}  
+ echo "swu_mode: ##$swu_mode##"  
+ swu_mode=${swu_mode}_${get_flash_type}  
# echo "swu_mode after fix to emmc/nand: ##$swu_mode##"  
echo "swu_mode $swu_mode" >> /tmp/swupdate_param_file  
fw_setenv -s /tmp/swupdate_param_file
```

3.14.2 Emmc/Nand 自适应 swu 制作方法步骤

(1) 达成如上 5 点自适应要求 (2) 对于 ab 分区，编译固件后，执行 swupdate_pack_swu-ab-emmc-nand 编译升级包；(3) 对于 recovery 分区，编译固件后，执行 swupdate_pack_swu-recovery-emmc-nand 编译升级包

注意：Emmc/Nand 自适应 OTA 升级和 Emmc/Nand 单独升级命令是一样的，无需修改。

4 其他功能

4.1 在用户空间操作 env

Tina 中支持了 uboot-envtools 软件包。

```
./build.sh buildroot_menuconfig ---> Target packages ---> Hardware handling ---> [*] u-boot tools
```

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。

4.2 AB 系统切换

4.2.1 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。

5 注意事项

5.1 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。

说明

注意：对于 ab 系统，不能单独升级 kernel 或者 rootfs 分区，需要一起升级。

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 kernel 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 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。

6 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

6.1 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




著作权声明

版权所有 © 2022 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。