



Android 10 SDK 移植指南

版本号: 1.0
发布日期: 2020.08.05

版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
|-----|------------|--------|--------|
| 1.0 | 2020.08.05 | AW0385 | 初始版本文档 |



目 录

| | |
|---------------------------------|----------|
| 1 概述 | 1 |
| 1.1 编写目的 | 1 |
| 1.2 SDK 概述 | 1 |
| 1.3 名词解释 | 1 |
| 2 方案定制 | 2 |
| 2.1 overlay 说明 | 2 |
| 2.1.1 为产品添加 Overlay 目录 | 2 |
| 2.1.2 改变 mk 文件来添加 overlays 的编译项 | 3 |
| 2.1.3 在 overlay 目录下创建资源文件 | 3 |
| 2.2 预装 APK | 3 |
| 2.2.1 预装到 system/app 目录 | 3 |
| 2.2.2 预装到 system/preinstall 目录 | 4 |
| 2.2.3 配置分区 | 4 |
| 2.3 修改启动 LOGO | 4 |
| 2.4 修改开机动画 | 5 |
| 2.4.1 文件结构 | 5 |
| 2.4.2 desc.txt 配置文件 | 5 |
| 2.4.3 开机音乐 | 6 |
| 2.4.4 优化 png 图片 | 6 |
| 2.4.5 打包 bootanimation.zip | 6 |
| 2.5 修改充电图标 | 6 |
| 2.6 定制 recovery 功能 | 6 |
| 2.6.1 键值的查看 | 7 |
| 2.6.2 按键选择 | 7 |
| 2.7 新增方案配置 | 8 |
| 3 模块配置 | 9 |
| 3.1 自定义按键配置 | 9 |
| 3.1.1 KEY 的硬件原理 | 9 |
| 3.1.2 驱动与硬件对应的关系 | 9 |
| 3.1.3 Android 按键功能的映射 | 10 |
| 3.2 LCD Panel 配置 | 10 |
| 3.3 Touch Panel 配置 | 10 |
| 3.3.1 配置文件的修改 | 11 |
| 3.3.2 Android 层的配置修改 | 11 |
| 3.4 G-Sensor 配置 | 11 |
| 3.4.1 Android 层配置修改 | 11 |
| 3.5 红外遥控器配置 | 13 |
| 3.5.1 内核配置 | 13 |
| 3.5.2 Device Tree 配置 | 13 |
| 3.5.3 Android multi_ir 配置 | 14 |

| | | |
|----------|---------------------------------|-----------|
| 3.5.4 | Android 按键功能的映射文件 | 14 |
| 3.5.5 | 新增遥控器适配 | 14 |
| 3.6 | GPIO 配置 | 15 |
| 3.6.1 | 定义需要控制的 GPIO | 15 |
| 3.6.2 | 配置 boot 阶段初始化的 gpio 功能 | 16 |
| 3.6.3 | 控制 GPIO 的接口 | 16 |
| 3.6.4 | java 层的接口 | 16 |
| 3.6.5 | c++ 层的接口 | 17 |
| 3.7 | 显示配置 | 17 |
| 3.8 | WiFi/BT 配置 | 18 |
| 3.9 | Camera 配置 | 18 |
| 3.9.1 | Camera 驱动加载顺序 | 18 |
| 3.9.2 | Camera 参数配置 | 19 |
| 3.10 | SD 卡配置 | 19 |
| 3.10.1 | 配置文件的修改 | 19 |
| 4 | 系统配置 | 20 |
| 4.1 | SettingsProvider 设置 | 20 |
| 4.2 | 默认配置 | 20 |
| 4.3 | Allwinner 平台设置 | 21 |
| 5 | Launcher 及界面设置 | 22 |
| 5.1 | 默认壁纸设置 | 22 |
| 5.2 | 添加壁纸 | 22 |
| 5.3 | Launcher 默认图标和快捷栏设置 | 22 |
| 6 | 后台服务管理配置 | 24 |
| 6.1 | 功能介绍 | 24 |
| 6.2 | 方案配置 | 24 |
| 6.3 | 用户设置 | 25 |
| 6.4 | 功能调试 | 25 |
| 7 | 打包发布 | 26 |
| 7.1 | 编译固件 | 26 |
| 7.2 | 调试 debug | 26 |
| 7.2.1 | 将 logcat 和 dmesg 信息保存到文件系统 | 26 |
| 7.2.2 | 生成 debug 固件 | 26 |
| 7.2.3 | 使用 fastboot | 27 |
| 7.3 | 发布 | 27 |
| 7.3.1 | 发布固件流程 | 27 |
| 7.3.2 | OTA 包 | 28 |
| 7.4 | 使用 OTA 包升级 | 28 |
| 7.4.1 | Apply update from ADB | 28 |
| 7.4.2 | Apply update from TFcard or USB | 29 |

1 概述

1.1 编写目的

本文档介绍 Android 10 系统常见的定制开发问题，以帮助客户快速熟悉开发环境，实现快速移植方案。

1.2 SDK 概述

参考《SDK Quick Start Guide》搭建开发环境

1.3 名词解释

Android 10 平台快速移植文档，本文适用于 Android 10 方案。

| key | value |
|--------------|------------|
| vendor-name | softwinner |
| device-name | 设备名称 |
| product-name | 产品名称 |
| IC | 芯片名称 |
| BOARD | 板卡名称 |

2 方案定制

方案目录 device/{vendor-name}/{device-name}/

2.1 overlay 说明

Android overlay 机制允许在不修改 apk 或者 framework 源代码的情况下，实现资源的定制。以下几类能够通过 overlay 机制定义：

1. Configurations (string, bool, bool-array)
2. Localization (string, string-array)
3. UI Appearance (color, drawable, layout, style, theme, animation)
4. Raw resources (audio, video, xml)

更详细的资源文件可浏览[android 网站](#)

2.1.1 为产品添加 Overlay 目录

有两种不同的 overlay 目录定义：

1. PRODUCT_PACKAGE_OVERLAYS
用于指定产品
2. DEVICE_PACKAGE_OVERLAYS
用于同一设备模型的一系列产品

如果包含同一资源，那么 PRODUCT_PACKAGE_OVERLAYS 将覆盖 DEVICE_PACKAGE_OVERLAYS。如果要定义多个 overlays 目录，需要用空格隔开，同一资源的定义，将使用先定义的目录中的资源。

在方案目录下创建 overlay 和 {product-name}/overlay 目录，分别用于 device 通用及 product 使用的 overlay 文件夹。

2.1.2 改变 mk 文件来添加 overlays 的编译项

在文件device/{vendor-name}/{device-name}/{product-name}.mk中添加:

```
PRODUCT_PACKAGE_OVERLAYS := \
    device/{vendor-name}/{device-name}/{product-name}/overlay \
    $(PRODUCT_PACKAGE_OVERLAYS)
DEVICE_PACKAGE_OVERLAYS := \
    device/{vendor-name}/{device-name}/overlay \
    $(DEVICE_PACKAGE_OVERLAYS)
```

注: 必须加上 \$(PRODUCT_PACKAGE_OVERLAYS) 变量否则将找不到默认资源。

2.1.3 在 overlay 目录下创建资源文件

在overlay目录下创建和要替换资源所在文件相同的路径的文件, 此路径是相对于android platform目录。如替换 framework-res路径为: platform/framework/base/core/res/res/values/config.xml中的某一项, 则在 overlay中创建对应的路径: overlay/framework/base/core/res/res/values/config.xml并添加要修改的一向配置, 如:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="config_showNavigationBar">true</bool>
</resources>
```

2.2 预装 APK

预装 apk 安装有两种方法, 可以安装到 system/app 目录下, 也可以安装到 system/preinstall 目录下。

注: apk 名字不能含有中文、空格等特殊字符。

由于涉及版权问题, 建议不安装GAPP应用。若通过GMS认证需安装Google提供的正版GAPP应用。

2.2.1 预装到 system/app 目录

1. 在目录 vendor/aw/public/prebuild/apk/中创建一个目录存放对应 APK。
2. 将 apk 放入该目录中。
3. 在该目录中创建 Android.mk 文件, 并编辑:

```
# Example
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := APK_MODULE_NAME (模块的唯一名字)
LOCAL_MODULE_CLASS := APPS
LOCAL_MODULE_TAGS := optional
```

```
LOCAL_BUILT_MODULE_STEM := package.apk
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := PRESIGNED (签名方式)
#LOCAL_OVERRIDES_PACKAGES := OVERRIDES_MODULE (要替代的模块)
LOCAL_SRC_FILES := name.apk (apk的文件名, 一般与MODULE同名)
include $(BUILD_PREBUILT)
```

4. 在方案 mk 文件 (device/{vendor-name}/{device-name}/{product-name}.mk) 中 PRODUCT_PACKAGES 项中加入:

```
PRODUCT_PACKAGES += APK_MODULE_NAME (apk模块名字, 预装多个apk用空格隔开)
```

2.2.2 预装到 system/preinstall 目录

1. 同预装到 system/app 目录, 完成所有步骤。
2. 修改 apk 目录下的 Android.mk, 加入一行:

```
LOCAL_MODULE_PATH := $(TARGET_OUT)/preinstall
```

2.2.3 配置分区

系统分区大小, Android 10 之后引入动态分区, 因此系统分区包括 system、vendor、product 等分区是包含在动态分区 super 中, 调整 super 分区即可调整系统分区大小:

```
[partition]
name      = super
size      = 3145728
downloadfile = "super.fex"
user_type = 0x8000
```

其中 size 为分区扇区数量, 每个扇区大小为 512bytes。

data 分区大小可以由 BoardConfig.mk 文件的 BOARD_USERDATAIMAGE_PARTITION_SIZE 指定, 单位是字节。

注: 一般将最后一个分区作为 data 分区, 该分区大小是 Nand 或者 eMMC 总容量减去其他分区大小。如果需要烧写 data 分区镜像, 分区大小需要预留一定预度, 防止超出 Nand 或者 eMMC 容量。

2.3 修改启动 LOGO

启动 LOGO 为初始引导阶段的 LOGO。

将启动 logo 放入位置: longan/device/config/chips/{IC}/configs/{BOARD}/bootlogo.bmp

2.4 修改开机动画

将动画放入：device/{vendor-name}/{device-name}/media/bootanimation.zip

2.4.1 文件结构

bootanimation.zip 包含 part0 part1 文件夹和 desc.txt 文件，part0，part1 等文件夹里面放的是动画拆分的图片，格式为 png 或 jpg。

```
| - desc.txt
| - audio_conf.txt
| - part0
|   | - part000.png
|   | - part001.png
|   | - part0...png
|   | - audio.wav(可选)
|   | - trim.txt(可选)
| - part1
|   | - part100.png
|   | - part101.png
|   | - part1...png
|   | - audio.wav(可选)
|   | - trim.txt(可选)
| - part...N
```

2.4.2 desc.txt 配置文件

第一行：
WIDTH HEIGHT FPS
后面每行，表示部分part动画：
TYPE：类型（p：播放直到开机完成，c：播放完整动画）
COUNT：循环次数，0表示无限循环直到开机结束
PAUSE：part结束后暂停帧数
PATH：文件加路径（如：part0）
RGBHEX：（可选）背景颜色：#RRGGBB
CLOCK：（可选）画当前时间的y坐标(for watches)

例如：

```
800 480 15
p 1 0 part0
p 0 0 part1
```

说明：第一行：800 为宽度，480 为高度，15 为帧数。第二行开始 p 为标志符，接下来第二列为循环次数（0 为无限循环），第三项为两次循环之间间隔的帧数，第四项为对应的目录名。播放动画时会按照图片文件名顺序自动播放。

2.4.3 开机音乐

如需开机音乐，将开机音乐放入 part0 目录中，命名为 audio.wav。在根目录中加入 audio_conf.txt，复制原有动画配置即可。

2.4.4 优化 png 图片

由于图片占用内存较大，需要做一些优化来减少图片资源占用及加快读取时间，可参考源码中 frameworks/base/cmds/bootanimation/FORMAT.md 文件进行优化，如下命令可以无损压缩图片：

```
for fn in *.png ; do zopflipng -m ${fn} ${fn}.new && mv -f ${fn}.new ${fn}; done
```

2.4.5 打包 bootanimation.zip

windows 使用 winrar 打包，选择 ZIP 格式，压缩标准要选“储存”；linux 系统下使用命令：\$ zip -0qry -i *.txt *.png *.wav @ bootanimation.zip .txt part。linux 命令使用 -0 指定压缩等级为最低等级 stored，即只归档不压缩，否则可能由于包格式问题引起动画显示为黑屏。

2.5 修改充电图标

在 android 目录下执行：

```
python bootable/recovery/interlace-frames.py battery1.png battery2.png ... batteryn.png  
battery_scale.png
```

然后将生成的 battery_scale.png 替换 device/softwinner/common/health/images/目录下 battery_scale.png

其中 [battery1.png battery2.png ... batteryn.png] 为充电动画的图标。

如图片数量有变化则需修改配置 device/softwinner/common/health/animation.txt

2.6 定制 recovery 功能

Recovery 是 Android 的专用升级模式，用于对 android 自身进行更新；进入 recovery 模式的方法是，在 android 系统开机时，按住一个特定按键，则会自动进入 android 的 recovery 模式。

2.6.1 键值的查看

按键是通过 AD 转换的原理制成。当用户按下某个按键的时候，会得到这个按键对应的 AD 转换的值。同时，所有的按键的键值都不相同，并且，键值之间都有一定的间隔，没有相邻。比如，键值可能是 5,10,15,20，但是不可能是 5,11,12,13。

为了方便用户查看不同按键的键值，这种方法要求连接上串口使用，因此适合于开发阶段使用。具体步骤是：

把小机和 PC 通过串口线连接起来，小机开机时按住对应按键，此时会串口屏幕上打印对应按键的键值。如下的打印信息：

```
key value = 8
key value = 8
key value = 8
key value = 63
```

由于 AD 采用的速度非常快，所以同一个按键按下，屏幕上会出现多个值。用户可以看出，这个按键的键值是 8。最后出现的 63 是松开按键的时候的采用，是需要去掉的干扰数据。因此，用户查看按键键值的时候只要关注前面打印出的数值，后面出现的应该忽略不计。

2.6.2 按键选择

通常情况下，一块方案板上的按键个数不同，或者排列不同，这都导致了方案商在选择作为开机阶段 recovery 功能的按键有所不同。因此，系统中提供了一种方法用于选择进入 recovery 模式的按键：

在 sys_config.fex 配置脚本中，提供了一项配置，用于选择按键的键值，如下所示：

```
[recovery_key]
key_mix      = 0x3
key_man      = 0x5
```

它表示，所选择用于作为 recovery 功能的按键的键值范围落在 key_min 到 key_max 之间，即 4 到 6 之间。由于所有按键的选择都可以通过前面介绍的方法查看，因此，假设用户要选的按键是 a，用户这里选择配置的方法是：

1. 按照前面介绍的方法，读出所有按键的键值；
2. 读出 a 的键值 a1，同时取出两个相邻于 a 的键值，记为 b1 和 c1， $b1 < a1 < c1$ ；
3. 计算出 $(a1 + b1)/2$ ， $(a1 + c1)/2$ ，分别填写到 key_max 和 key_min 处；
4. 如果 a1 刚好是所有按键的最小值，则取 key_min 为 0；如果 a1 刚好是所有按键的最大值，则取 key_max 为 63；

经过以上的步骤，就可以选择一个特定的按键进入 recovery 模式。取了一个平均值的原因是考虑到长时间的使用，电阻的阻值可能会略有变化导致键值变化，取范围值就可以兼容这种阻值变化带来的键值变化。

在系统启动时，按住设定的特定按键进入 recovery 模式，进入该模式后，可以选择升级文件升级。

2.7 新增方案配置

1. 新增 bsp 板级方案：进入 `longan/device/config/chips/{IC}/configs` 目录，基于基础板级目录，如 `{BOARD}` 目录复制一份为 `newboard`，得到新增板级方案 `longan/device/config/chips/{IC}/configs/newboard` 根据实际硬件修改其中 `board.dts`、`sys_config.fex` 等信息。



说明

如硬件无修改，可不需要新增板级方案

2. 选择参考方案，按照以下脚本输入命令：

```
# cd android
# source ./build/envsetup.sh
# lunch (选择对应平台方案配置)
# clone
```

根据提示输入新增方案信息：

```
please enter PRODUCT_DEVICE({device-name}): newdevice
please enter PRODUCT_NAME({product-name}): newproduct
please enter PRODUCT_BOARD({BOARD}): newboard
please enter PRODUCT_MODEL({MODEL_NAME}): NEWMODEL
please enter DENSITY(160): 320
```

完成以上操作后，将会在 `device/{vendor-name}/newdevice` 目录新增方案配置。根据实际需求，修改方案目录中相关的 `mk` 文件，进行应用及参数的定制化。

3 模块配置

3.1 自定义按键配置

3.1.1 KEY 的硬件原理

目前 KEY 检测使用了 ADC 转换的原理实现的，由于该原理的限制，所以不能区分组合键（功能键，不包括电源键）；使用 GPADC 实现了两个键，硬件原理如下：

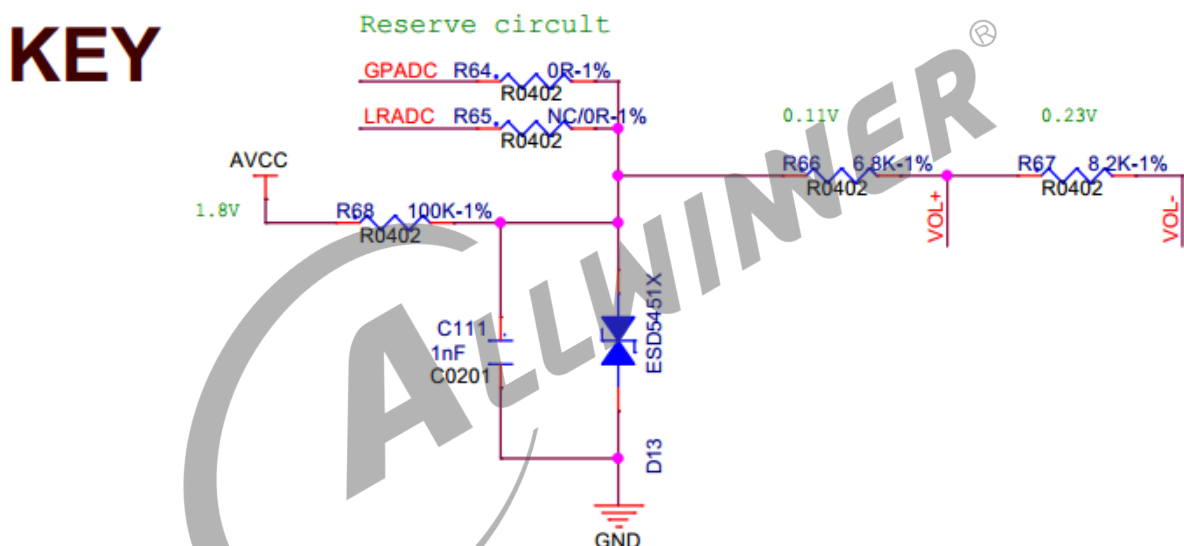


图 3-1: key

3.1.2 驱动与硬件对应的关系

Key 的驱动实现文件位置：longan/kernel/linux-4.9/drivers/input/sensor/sunxi_gpadc.c;

驱动从 sys_config 获取按键对应的设置电压值与上报值：

```
Key_cnt    = 2
Key0_vol   = 115
Key0_val   = 115
Key0_vol   = 235
Key0_val   = 114
```

驱动将按键电路最大输入电压分为 128 份，根据上面获取的 keyx_vol 划分 keypad_mapindex，

keypad_mapindex 元素为上报数组序号，通过 adc 测得当前电压值，按比例算得 keypad_mapindex 匹配值，最终按上报数组的序号上报 keyx_val 按键值。

通过检测电压值的数字量来区分当前是第几个按键，驱动中会根据 keyx_vol 值将其中元素设置正确，以下是初始值：

```
//0.2V mode
static unsigned char keypad_mapindex[128] =
{
    0,0,0,0,0,0,0,0,          //key 1, 8个, 0-7
    1,1,1,1,1,1,1,1,          //key 2, 7个, 8-14
    2,2,2,2,2,2,2,2,          //key 3, 7个, 15-21
    3,3,3,3,3,3,3,3,          //key 4, 6个, 22-27
    4,4,4,4,4,4,4,4,          //key 5, 6个, 28-33
    5,5,5,5,5,5,5,5,          //key 6, 6个, 34-39
    6,6,6,6,6,6,6,6,6,6,      //key 7, 10个, 40-49
    7,7,7,7,7,7,7,7,7,7,7,7, //key 8, 17个, 50-63
};
```

当有按键事件时，通过两次映射将最终的键值上报：

```
scancode = keypad_mapindex[key_val&0x3f];
input_report_key(swkbd_dev, sw_scankeycodes[scancode], 1);
input_sync(swkbd_dev);
```

3.1.3 Android 按键功能的映射

映射文件为 device/{vendor-name}/{device-name}/configs/sunxi-keyboard.kl:

```
key 114      VOLUME_UP
key 139      VOLUME_DOWN
```

key 后面的数字为驱动中上报的键值，后面为对应的功能，自定义按键时仅需要将前面的映射值和后面的功能对应起来即可。

3.2 LCD Panel 配置

详细配置请参考文档《LCD 驱动开发说明书》。

3.3 Touch Panel 配置

发布的 SDK 中，默认有对 FT5x 系列（敦泰）、GT81x/GT82x/GT9xx/GT9xxf（汇顶）、GSLx680(思立微) 等 ctp 的支持。

3.3.1 配置文件的修改

配置文件目录：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts 详细配置请参考文档《Sensor 自适应驱动开发说明书》。

3.3.2 Android 层的配置修改

在 device/{vendor-name}/{device-name}/init.device.rc 文件中加入装载驱动模块的语句：

```
insmod /vendor/modules/gt9xx_ts.ko
```

3.4 G-Sensor 配置

配置文件目录：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts 详细配置请参考文档《Sensor 自适应驱动开发说明书》。

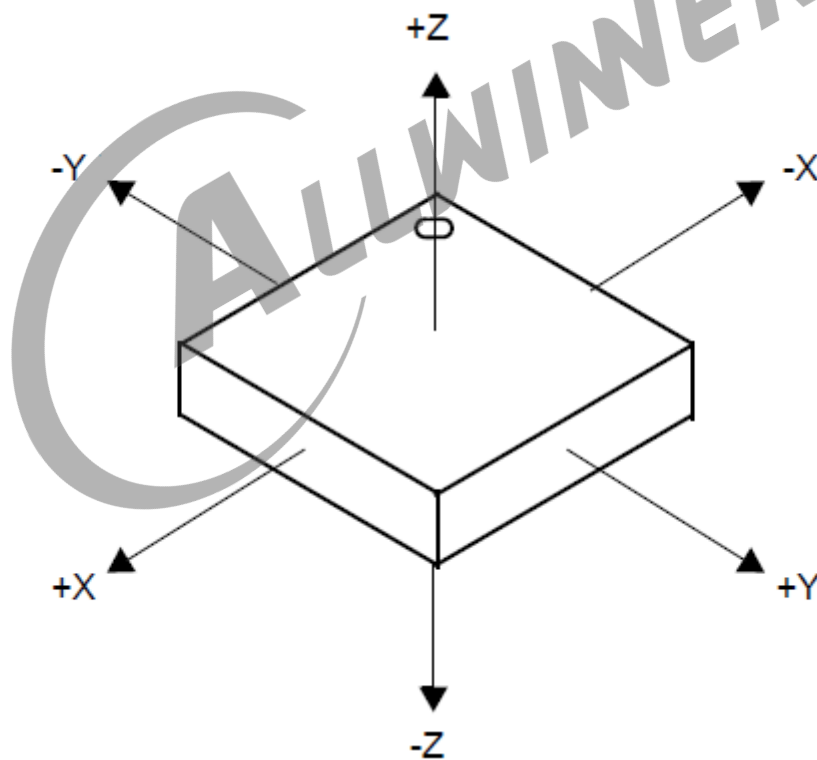


图 3-2: gsensor

3.4.1 Android 层配置修改

以 bma250 为例：

Sensor 驱动是自动加载的，自动加载部分配置好后，不需要手动 insmod。

1) 方向的调整：

在 device/{vendor-name}/{device-name}/configs/gsensor.cfg 中，以 bma250 方向为例进行说明。

```
;name: bma250
gsensor_name = bma250 //标示用bma250c gsensor
gsensor_direct_x = false //如果x轴反向，则置false
gsensor_direct_y = false //如果y轴反向，则置false
gsensor_direct_z = false //如果z轴反向，则置false
gsensor_xy_revert = true //如果x轴当y轴用，y轴当x轴，则置true
```

Gsensor 方向调试说明：

假定机器的长轴为 X 轴，短轴为 Y 轴，垂直方向为 Z 轴。

首先调试 Z 轴：

第一步观察现象：

旋转机器，发现当只有垂直 90° 时或者是在旋转后需要抖动一下，方向才会发生变化，则说明 Z 轴反了。若当机器大概 45° 拿着的时候也可以旋转，说明 Z 轴方向正确。无需修改 Z 轴方向。

第二步修改 Z 轴为正确方向。

此时需要找到当前使用模组的方向向量（根据模组的名称）。如果此时该方向 Z 轴向量（gsensor_direct_z）的值为 false，则需要修改为 true；当为 true，则需要修改为 false。通过 adb shell 将修改后的 gsensor.cfg 文件 push 到 system/usr 下，重启机器，按第一步观察现象。

其次查看 X, Y 轴是否互换：

第一步观察现象：

首先假定长轴为 X 轴，短轴为 Y 轴，以 X 轴为底边将机器立起来。查看机器的 X, Y 方向是否正好互换，若此时机器的 X, Y 方向正好互换，在说明需要将 X, Y 方向交换。若此时 X, Y 方向没有反置，则进入 X, Y 方向的调试。

第二步交换 X, Y 方向

当需要 X, Y 方向交换时，此时需要找到当前使用模组的方向向量（根据模组的名称）。如果此时该 X, Y 轴互换向量（gsensor_xy_revert）的值为 false，则需要修改为 true，当为 true，则需要修改为 false。通过 adb shell 将修改后的 gsensor.cfg 文件 push 到 system/usr 下，重启机器，按第一步观察现象。

再次调试 X, Y 轴方向：

第一步观察现象：

首先假定长轴为 X 轴，短轴为 Y 轴，以 X 轴为底边将机器立起来，查看机器的方向是否正确，如果正确，说明长轴配置正确，如果方向正好相反，说明长轴配置错误。将机器旋转到短轴，查看机器方向是否正确，如果正确，说明短轴配置正确，如果方向正好相反，说明短轴配置错误。

第二步修改 X, Y 轴方向：

当需要修改 X, Y 轴方向时，当只有长轴方向相反或者是只有短轴方向相反时，则只修改方向不正确的一个轴，当两个方向都相反时，则同时修改 X 与 Y 轴方向向量。找到当前使用模组的方向向量（根据模组的名称）。

若长轴方向相反，如果此时该方向 X 轴向量（gsensor_direct_x）的值为 false，则需要修改为 true，当为 true，则需要修改为 false。

若短轴方向相反，如果此时该方向 Y 轴向量 (gsnesor_direct_y) 的值为 false，则需要修改为 true，当为 true，则需要修改为 false。

通过 adb shell 将修改后的 gsnesor.cfg 文件 push 到 system/usr 下，重启机器，按第一步观察现象。若发现还是反向 X 轴或者 Y 轴的方向仍然相反，则说明 X 轴为短轴，Y 轴为长轴。此时：

若长轴方向相反，如果此时该方向 Y 轴向量 (gsnesor_direct_y) 的值为 false，则需要修改为 true，当为 true，则需要修改为 false。

若短轴方向相反，如果此时该方向 X 轴向量 (gsnesor_direct_x) 的值为 false，则需要修改为 true，当为 true，则需要修改为 false。

3.5 红外遥控器配置



说明
不需要遥控器则跳过该章节

3.5.1 内核配置

要支持红外遥控器 (多遥控器适配)，需要打开下面的配置：

```
1.Device Drivers -> <*>Multimedia support
2.Device Drivers -> <*>Multimedia support ->[*]Remote controller decoders
3.Device Drivers -> <*>Multimedia support ->[*]Remote controller decoders -> <*> Enable
   IR raw decoder for the NEC protocol
4.Device Drivers -> <*>Multimedia support ->[*]Remote controller decoders -> <*> Enable
   IR raw decoder for the RC-5 protocol
5.Device Drivers -> <*>Multimedia support ->[*]Remote Controller devices
6.Device Drivers -> <*>Multimedia support ->[*]Remote Controller devices -> <*> SUNXI IR
   remote control
7.Device Drivers -> <*>Multimedia support ->[*]Remote Controller devices -> <*> SUNXI IR
   Legacy feature
8.Device Drivers -> <*>Multimedia support ->[*]Remote Controller devices -> <*> sunxi
   multi support
```

3.5.2 Device Tree 配置

在 soc 节点下配置 s_cir 节点属性，其中 ir_protocol_used 属性配置红外协议，主要是 NEC(0x0) 和 RC5(0x1) 两种协议，这个属性可以不配置，不配置则默认使用 NEC 协议。

```
s_cir0: s_cir@07040000 {
    compatible = "allwinner,s_cir";
    reg = <0x0 0x07040000 0x0 0x400>;
    interrupts = <GIC_SPI 109 IRQ_TYPE_LEVEL_HIGH>;
    pinctrl-names = "default";
    pinctrl-0 = <&s_cir0_pins_a>;
    clocks = <&clk_hosc>,<&clk_cpurcir>;
```

```
supply = "vcc-pl";
supply_vol = "3300000";
status = "okay";
ir_protocol_used = <0>
};
```

3.5.3 Android multi_ir 配置

multi_ir 是 android 的一个服务，用于适配多遥控器，如果需要添加此功能，需要在方案下添加以下配置：

```
# utils, add multi_ir to recovery
PRODUCT_PACKAGES += \
    multi_ir \
    multi_ir.recovery \
    libmultiir_jni \
    libmultiirservice \
```

3.5.4 Android 按键功能的映射文件

multi_ir 的按键映射文件主要放在 vendor/aw/homlet/hardware/input/multi_ir/keylayout 目录下，以 customer_ir_xxxx.kl 命名的文件是不同遥控器的映射文件，xxxx 是底层驱动识别到的遥控器 id，随着事件上报。sunxi-ir.kl 则是 multi_ir 映射底层上报的键值为统一的 scancode。sunxi-ir-uinput.kl 是 inputflinger 所读取的映射文件。使用此功能时需要将这些映射文件放入到机器内部，应如下配置：

```
SUNXI_VENDOR_KL_DIR := vendor/aw/homlet/hardware/input/multi_ir/keylayout
PRODUCT_COPY_FILES += \
    $(SUNXI_VENDOR_KL_DIR)/virtual-remote.kl:system/usr/keylayout/virtual-remote.kl \
    $(SUNXI_VENDOR_KL_DIR)/sunxi-ir.kl:system/usr/keylayout/sunxi-ir.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_9f00.kl:system/usr/keylayout/customer_ir_9f00.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_dd22.kl:system/usr/keylayout/customer_ir_dd22.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_fb04.kl:system/usr/keylayout/customer_ir_fb04.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_ff00.kl:system/usr/keylayout/customer_ir_ff00.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_4cb3.kl:system/usr/keylayout/customer_ir_4cb3.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_bc00.kl:system/usr/keylayout/customer_ir_bc00.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_fc00.kl:system/usr/keylayout/customer_ir_fc00.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_ir_2992.kl:system/usr/keylayout/customer_ir_2992.kl \
    $(SUNXI_VENDOR_KL_DIR)/customer_rc5_ir_04.kl:system/usr/keylayout/customer_rc5_ir_04.kl \
    \
    $(SUNXI_VENDOR_KL_DIR)/sunxi-ir-uinput.kl:system/usr/keylayout/sunxi-ir-uinput.kl \
```

3.5.5 新增遥控器适配

当需要兼容新的遥控器，只要新增一个新的 customer_ir_xxxx.kl 文件，而文件主要内容应如下：

```
key 25  BACK
key 0   MENU
key 19  DPAD_CENTER
key 26  DPAD_DOWN
key 22  DPAD_UP
key 17  HOME
key 81  DPAD_LEFT
key 80  DPAD_RIGHT
key 24  VOLUME_UP
key 16  VOLUME_DOWN
key 15  APPS
key 67  CONTACTS
key 64  POWER
.....
key 34  ZOOM_OUT
key 35  INFO
```

其中这三列字符串分别表示事件类型 (KeyEvent)、scancode 和事件 lable。multi_ir 是根据 lable 来进行映射的，sunxi-ir.kl 中有所支持的所以事件 lable。新增遥控器主要修改 scancode。scancode 的获取方式可以通过机器执行 `getevent -l` (sunxi-ir 所对应的设备节点) 来获取，如下：

```
//getevent -l /dev/input/event1
EV_REP      REP_DELAY      00000000
EV_REP      REP_PERIOD      00000000
EV_MSC      MSC_SCAN        01fe0116
EV_SYN      SYN_REPORT      00000000
EV_MSC      MSC_SCAN        00fe0116
EV_SYN      SYN_REPORT      00000000
EV_MSC      MSC_SCAN        01fe0150
EV_SYN      SYN_REPORT      00000000
EV_MSC      MSC_SCAN        00fe0150
EV_SYN      SYN_REPORT      00000000
EV_MSC      MSC_SCAN        01fe011a
EV_SYN      SYN_REPORT      00000000
EV_MSC      MSC_SCAN        00fe011a
EV_SYN      SYN_REPORT      00000000
```

其中 MSC_SCAN 所上报的就是我们所需要的数据，由 8 位 16 进制数据组成 (32bit)，24 ~ 31bit 表示按下状态，0 表示松开，1 表示按下。8 ~ 23bit 表示设备 id，根据这个 id 生成新的 customer_ir_xxxx.kl，0 ~ 7bit 就是 scancode，对应 kl 文件的第二列数据。

3.6 GPIO 配置

3.6.1 定义需要控制的 GPIO

通常这一块不需要太多的修改，但如果需要进行修改的话，可以参考 `longan/device/config/chips/{IC}/configs/{BOARD}/sys_config.fex` 文件中，类似如下的配置信息：

```
-----
;userspace gpio interface for android
;-----
```

```
[gpio_para]
gpio_para_used      = 1
compatible          = "allwinner,sunxi-init-gpio"
gpio_num            = 2
gpio_pin_1          = port:PL10<1><default><default><0>
gpio_pin_2          = port:PA15<1><default><default><1>
normal_led          = "gpio_pin_2"
standby_led         = "gpio_pin_1"
easy_light_used     = 1
normal_led_light    = 1
standby_led_light   = 1
```

在这个范例中，变量 `gpio_para_used` 置为“1”表示此配置将起作用，其他的就是各个 GPIO 的配置信息。这些 GPIO 的编码必须从“1”开始依次递增。

通常盒子会有两个 gpio 控制两个颜色的灯，为了向 Android 框架提供统一路径控制 Led 灯，所以提供一个指定控制命名 Led 的 GPIO 的配置，包括 `normal` 和 `standby`，上面的内容就是将 `gpio_pin_2` 配置为 `normal_led`，`gpio_pin_1` 配置为 `standby_led`。

3.6.2 配置 boot 阶段初始化的 gpio 功能

系统上电的时候，能快速的初始化用户自定义的 GPIO 口，这里包括：上电亮灯等。配置在 `longan/device/config/chips/{IC}/configs/{BOARD}/sys_config.fex` 文件中，根据自己方案中要上电初始化 GPIO 来添加类似如下的配置信息：范例：

```
[boot_init_gpio]
boot_init_gpio_used = 1
gpio0 = port:PL10<1><default><default><0>
gpio1 = port:PA15<1><default><default><1>
```

以上配置表示：在 boot 阶段，设置 PA15 输出高电平，PL10 输出低电平。

3.6.3 控制 GPIO 的接口

添加了 GPIO 的配置后，会在 `sys` 文件系统下产生节点

```
# /sys/class/gpio_sw # ls
PA15 PL10 normal_led standby_led
# /sys/class/gpio_sw # echo 1 > normal_led
# /sys/class/gpio_sw # echo 0 > standby_led
```

对于目录下的 `normal_led/standby_led` 节点写入 0，将导致输出低电平，写入 1，将导致输出高电平，为了方便代码中进行操作，有提供 java 以及 C++ 的接口

3.6.4 java 层的接口

java 控制 GPIO 的接口定义在文件 `Gpio.java` 中，其路径为：`platform/frameworks/base/services/core/java/com/aw/server/Gpio.java` 在 java 代码中 `import com.softwinner.Gpio;`

的 setNormalLedOn(bool) 和 setStandbyLedOn(bool) 接口方便的操作 Led 的亮灭。提供的接口如下：

```
public static int setNormalLedOn(boolean on);
public static int setStandbyLedOn(boolean on);
public static int setNetworkLedOn(boolean on) ;
public static int writeGpio(char group, int num, int value);
public static int readGpio(char group, int num) ;
public static int setPull(char group, int num, int value);
public static int getPull(char group, int num);
public static int setDrvLevel(char group, int num, int value);
public static int getDrvLevel(char group, int num);
public static int setMulSel(char group, int num, int value);
public static int getMulSel(char group, int num);
private static String composePinPath(char group, int num);
```

3.6.5 c++ 层的接口

C++ 层的操作函数是对内核接口的简单封装，具体的接口如下

```
int readData(const char * filePath);
int writeData(const char *data, int count, const char *filePath);
```

cfg: 设置/读取gpio的功能
 0x00: input
 0x01: output
 pull: 设置/读取gpio电阻上拉或者下拉
 0x00: 关闭上拉/下拉
 0x01: 上拉
 0x02: 下拉
 0x03: 保留
 drv: 设置/读取gpio的驱动等级
 0x00: level 0
 0x01: level 1
 0x02: level 2
 0x03: level 3
 data: 设置/读取gpio的电平状态
 0x00: 低电平
 0x01: 高电平

在 C 语言中可以用 read 和 write 函数直接操作这 4 个文件。具体的范例可参考文件 vendor/aw/homlet/framework/gpio/libgpio/GpioService.cpp 中的代码。

3.7 显示配置



说明

不需要外接显示则跳过该章节

- 720 UI 分辨率设置

```
//路径: hardware/aw/hwc2/de2family/Display0pr.cpp
#define MAXUIWIDTH 1280
#define MAXUIHEIGHT 720
//路径: 方案目录{product-name}.mk
ro.sf.lcd_density=213
```

- 1080 UI 分辨率设置

```
//路径: hardware/aw/hwc2/de2family/Display0pr.cpp
#define MAXUIWIDTH 1920
#define MAXUIHEIGHT 1080
//路径: 方案目录{product-name}.mk
ro.sf.lcd_density=320
```

- 配置单双显

```
//路径: 方案目录{product-name}.mk
persist.display.policy=2
//默认是单显, 2是双显;
```

- 其余请参考《显示模块说明书》

3.8 WiFi/BT 配置

详细配置请参考文档《wifi 模块移植说明书》。

3.9 Camera 配置

详细配置请参考文档《Camera 模块使用说明书》。

3.9.1 Camera 驱动加载顺序

在文件 device/{vendor-name}/{device-name}/init.*.rc 添加 camera 驱动 ko 文件加载顺序如下:

```
on late-fs
insmod /vendor/modules/videobuf2-core.ko
insmod /vendor/modules/videobuf2-memops.ko
insmod /vendor/modules/videobuf2-dma-contig.ko
insmod /vendor/modules/videobuf2-v4l2.ko
insmod /vendor/modules/vin_io.ko
insmod /vendor/modules/gc2385_mipi.ko
insmod /vendor/modules/gc030a_mipi.ko
insmod /vendor/modules/gc2355_mipi.ko
```

```
insmod /vendor/modules/gc0310_mipi.ko
insmod /vendor/modules/vin_v4l2.ko
```

3.9.2 Camera 参数配置

配置文件路径：device/{vendor-name}/{device-name}/configs/camera.cfg。

media_profiles.xml 的路径：device/{vendor-name}/{device-name}/configs/media_profiles.xml

内容简介：该文件主要保存 Camera 支持的摄像相关参数，包括摄像质量，音视频编码格式、帧率、比特率等等，该参数主要有摄像头厂商提供。需要注意帧率配置，我们配置 frameRate="24" 为 24 帧，这个是多媒体要求的 camera 帧率最低的要求，这样配置我们可以满足低性能的 sensor，适用的 sensor 范围广一些。

3.10 SD 卡配置

发布的 SDK 中支持 SD 卡和 Mirco SD (TF) 卡及其兼容性卡。

3.10.1 配置文件的修改

配置文件路径：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts 根据原理图进行相关配置参数的修改，参考《device tree 使用文档》

4 系统配置

4.1 SettingsProvider 设置

配置文件 `platform/frameworks/base/packages/SettingsProvider/res/values/defaults.xml` 中各项为 Android 原生属性，可通过 overlay 修改进行配置，下面列出一些常用修改

| name | value | description |
|--------------------------------------|---------|-------------|
| def_screen_off_timeout | Int(毫秒) | 默认 LCD 关闭时间 |
| def_screen_brightness | 0~255 | 默认亮度设置 |
| def_screen_brightness_automatic_mode | Boolean | 是否默认打开自动亮度 |
| def_wifi_on | Boolean | 是否默认打开 WIFI |
| def_bluetooth_on | Boolean | 是否默认打开蓝牙 |

4.2 默认配置

通过系统配置文件 `platform/frameworks/base/core/res/res/values/config.xml` 中各项修改系统的配置，可通过 overlay 方式进行修改。

| name | value | description |
|---------------------------------------|---------|-----------------------|
| config_showNavigationBar | Boolean | 默认显示导航栏 |
| config_multiuserMaximumUsers | 1~8 | 最大用户数量 |
| config_enableMultiUserUI | Boolean | 是否支持多用户 UI（多用户时不需要设置） |
| config_unplugTurnsOnScreen | Boolean | 拔出 usb 或电源时唤醒屏幕 |
| config_automatic_brightness_available | Boolean | 是否支持自动亮度调节 |
| config_enableWifiDisplay | Boolean | 是否支持 Miracast |
| config_allowAllRotations | Boolean | 是否支持 4 个方向旋转 |
| config_enableLockScreenRotation | Boolean | 锁屏时是否支持旋转 |

4.3 Allwinner 平台设置

配置文件 `platform/frameworks/base/packages/SettingsProvider/res/values/custom_config.xml` 中各项为平台自定义属性，可通过 `overlay` 修改进行配置。

| name | value | description |
|--|-------|-------------|
| <code>def_gesture_screenshot_enable</code> | 1/0 | 默认打开手势截屏功能 |
| <code>def_gesture_screenrecord_enable</code> | 1/0 | 默认打开手势录屏功能 |



5 Launcher 及界面设置

5.1 默认壁纸设置

替换文件 `platform/frameworks/base/core/res/res/drawable-swxxxdp-nodpi/default_wallpaper.png`

说明

可通过 **overlay** 方式将文件放在 `device/{vendor-name}/{device-name}/overlay/frameworks/base/core/res/res/drawable-swxxxdp-nodpi/default_wallpaper.png`。

5.2 添加壁纸

准备壁纸及壁纸的缩略图放进壁纸存放目录 `platform/packages/apps/WallpaperPicker2/res/drawable-nodpi`，并按照文件夹内文件命名，分别为 `wallpaper_xxx.png` 与 `wallpaper_xxx_small.png`

在 `platform/packages/apps/WallpaperPicker2/res/values-nodpi/wallpapers.xml` 中添加该壁纸索引，如下：

```
<resources>
  <string-array name="wallpapers" translatable="false">
    <item>wallpaper_00</item>
    <item>wallpaper_01</item>
    .....
    <item>wallpaper_xxx</item>
  </string-array>
</resources>
```

注：可通过 **overlay** 方式修改，具体请参照 2.1overlay 说明。

5.3 Launcher 默认图标和快捷栏设置

修改文件 `platform/packages/apps/Launcher3/res/xml/default_workspace_5x6.xml`，文件中各配置项含义如下：

| 设置名 | 意义 |
|-------------|---|
| packageName | 所运行的 APP 的 package 名。 |
| className | 该 APP 的 Activity 的 class 名 (packageName.ActivityName) |
| screen | 表示在第几个，根据显示的个数决定 |
| x | 放在该屏的第几行 |

| 设置名 | 意义 |
|-----|----------|
| y | 放在该屏的第几列 |

注：可通过 overlay 方式修改，具体请参照 2.1overlay 说明。



6 后台服务管理配置

6.1 功能介绍

限制后台服务，使得多应用运行场景下还保留足够的空闲内存，当系统资源内存紧张时，通过清除后台服务，让系统保持运行流畅。

6.2 方案配置

说明

通过 **device/softwinner/common/config/awbms_config** 复制到 **\$(TARGET_COPY_OUT_VENDOR)/etc** 开启后台管理服务。通过配置文件中各项修改后台服务管理的配置。

```
debug: false
limit: 12
threadCheck: true
memoryCheck: true
memoryLimit: 450,0
memoryTrim: false
skipService: true
blockBroadcast: false
lmk: false
lmk_level: 100,150,250
lmk_adj: 99,200,600

whitelist:
android
com.android
com.android.phone
```

- debug 是否打开调试打印，取值 true/false，默认为 false。
- limit 限制后台个数，清理多余的后台，-1 则不限制，默认为 0。
- threadCheck 检查优化关键进程优先级，取值 true/false，默认为 false。
- memoryCheck 根据当前内存情况清理后台应用，取值 true/false，默认为 false。
- skipService 是否跳过非白名单服务自动启动，取值 true/false，默认为 true。
- whitelist 配置包名前缀白名单，系统白名单的进程不会被上述机制清理。
- 当前输入法、正在播放音乐进程等自动系统白名单，无须单独配置。

6.3 用户设置

通过：设置-》系统-》自动运行可以进入设置后台清理列表。列表中只显示不在系统配置白名单中的应用。勾选后的应用添加到用户白名单中。

6.4 功能调试

通过删除 awbms_config 配置文件可以关闭后台管理功能，通过命令 `# service call background 1 i32 1/0` 可开启/关闭相关调试打印。

logcat 中关于 BackgroundManagerService 相关的打印：

```
D bms: skipped service Intent(xxx)
I bms: forceStopPackage com.xxx.xxx
D bms: killBackgroundProcesses com.xxx.xxx
```

当看到则 forceStopPackage 或 killBackgroundProcesses 表示后台应用被后台服务管理清除，如有必要则需要将应用加入白名单或者在设置中取消勾选清除列表。

使用命令：dumpsys background 可以查看后台服务当前的状态。

7 打包发布

7.1 编译固件

编译内核及芯片相关：

```
# cd longan
# ./build.sh config (选择对应平台内核)
# ./build.sh
```

编译 android：

```
# cd android
# source ./build/envsetup.sh
# lunch (选择对应平台方案配置)
# extract-bsp
# make -j16
```

7.2 调试 debug

7.2.1 将 logcat 和 dmesg 信息保存到文件系统

为了调试方便，可以在开发调试阶段将系统的 logcat 和内核 log 自动保存到 data 分区文件系统上，这种方法可以方便调试偶发问题，log 保存在/data/media/awlog 目录

在文件中加入 `PRODUCT_DEBUG := true` 即可打开该功能，eng 及 userdebug 固件默认开启

或者在计算器集成了动态开关设置，计算器中输入

```
log(666+!)++
```

弹出开关设置界面，根据 GUI 进行配置 log 保存选项。

7.2.2 生成 debug 固件

编译 android 后，`pack -d` 即可生成 debug 固件，该固件将串口引入卡口打印出来，配合配套的工具即可实时查看 log 信息。

7.2.3 使用 fastboot

获取 fastboot 工具：

1. 建议更新最新版本 Android SDK tools 中的 fastboot 工具
2. 在 android 源代码编译过后的生成文件获得 (platform/out/host/linux-x86/bin/fastboot)

- fastboot 常用命令

```
usage: fastboot [ <option> ] <command>
commands:
  update <filename>                reflash device from update.zip
  flash <partition> [ <filename> ]  write a file to a flash partition
  erase <partition>                 erase a flash partition
  format <partition>                format a flash partition
  help                             show this help message
```

- bootloader 模式

用于升级传统的物理分区，在 adb shell 中，使用命令 `reboot bootloader` 即可进入 bootloader 模式。

安装驱动后，在 PC 端执行 fastboot 命令即可进行 fastboot 操作。

- fastbootd 模式

Android Q 之后引入动态分区，由于在传统 bootloader 模式中无法识别到逻辑分区，因此基于 recovery 系统启动的应用进程 fastbootd，专门用于烧写 system、vendor、product 等逻辑分区。

7.3 发布

7.3.1 发布固件流程

发布固件即可用作量产使用的固件，同时也支持 OTA 升级功能。发布时需要使用该流程进行发布。

编译固件流程后，使用命令：

```
# pack4dist [-v]
```

即可生成固件及对应版本的 OTA 包。（注：上述 -v 参数用于启用安全系统校验）

品牌签名：如果品牌商有自己的系统签名文件，把相关签名文件放入 platform/vendor/security 目录。签名文件可参考 platform/build/target/product/README 文档

7.3.2 OTA 包

OTA 包包含差分包和完整包，以下是各个名词定义：

- 目标文件包 (target-files-package)：包含固件完整的编译后目标文件的包。
- 差分包 (incremental-package)：将基础版本与新版本固件之间的差别制作的补丁包。
- 完整包 (full-package)：将新版本固件的完整补丁包。

OTA 包生成过程：

使用 pack4dist 后会自动生成目标文件包 (target-files-package) 路径为：

```
$OUT/obj/PACKAGING/target_files_intermediates/$TARGET_PRODUCT-target_files-$DATE.zip
```

若包含签名目标文件包，则路径为：

```
$OUT/signed_target_files-$DATE.zip
```

注：生成的 target_files.zip 文件需要与固件一同保存，用于后续生成 OTA 包。

完整包路径为：

```
$OUT/$TARGET_PRODUCT-full_ota-$DATE.zip
```

完整包生成命令：

```
# ./build/tools/releasetools/ota_from_target_files [ -k vendor/security/releasekey ] target  
.zip ota.zip
```

差分包生成命令：

```
# ./build/tools/releasetools/ota_from_target_files [ -k vendor/security/releasekey ] -i  
origin.zip target.zip inc.zip
```

注：其中 vendor/security 为签名 key 放置路径，origin.zip 为基础版本（即需要升级的版本）的目标文件包，target.zip 为当前版本的目标文件包。

7.4 使用 OTA 包升级

选择“设置 > 备份和重置 > Recovery 模式”重启进入 Recovery。

或 PC 端通过 adb reboot recovery 命令，重启进入 recovery。

7.4.1 Apply update from ADB

1. 将固件放在 PC 端，如：E:/update.zip。

2. 进入 Recovery。
3. 选择 Apply update from ADB。
4. 打开 cmd，并输入 adb sideload E:/update.zip。
5. 等待打印 Install from ADB complete. 升级完成。
6. 选择 reboot system now 重启并进入 android。

7.4.2 Apply update from TFcard or USB

1. 将固件放入 TF 卡或 U 盘中。
2. 进入 recovery。
3. 插入 TF 卡或 U 盘。
4. 在 Recovery 菜单中选择 Apply update from SD card。
5. 找到升级包的路径并选择开始升级。
6. 等待打印 Install from SD card complete. 升级完成。
7. 选择 reboot system now 重启并进入 android。






著作权声明

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。