



Android 15 SDK 移植指南

版本号: 1.0

发布日期: 2024.11.11

版本历史

版本号	日期	制/修订人	内容描述
1.0	2024.11.11	AWA0418	初始版本



目 录

1 前言	1
1.1 文档简介	1
1.2 目标读者	1
1.3 适用范围	1
1.4 名词解释	1
2 编译和发布	2
2.1 编译固件流程	2
2.1.1 GRF target file 来源	2
2.1.2 编译 target file	3
2.1.3 生成 debug 固件	3
2.2 编译模块	3
2.2.1 源码下编译	3
2.2.2 无源码打包	4
2.2.2.1 工作原理	5
2.2.2.2 使用	5
2.2.2.3 编译打包	5
2.3 发布	5
2.3.1 发布固件流程	5
2.3.2 OTA 包	6
3 方案定制	7
3.1 方案文件替换	7
3.2 overlay 说明	7
3.2.1 为产品添加 Overlay 目录	7
3.2.2 改变 mk 文件来添加 overlays 的编译项	8
3.2.3 在 overlay 目录下创建资源文件	8
3.3 预装 APK	8
3.3.1 预装到 system/app 目录	9
3.3.2 预装到 system/preinstall 目录	9
3.3.3 配置分区	9
3.4 修改启动 LOGO	10
3.5 修改开机动画	10
3.5.1 文件结构	10
3.5.2 desc.txt 配置文件	11
3.5.3 开机音乐	11
3.5.4 优化 png 图片	11
3.5.5 打包 bootanimation.zip	12
3.6 修改充电图标	12

3.7	定制 recovery 功能	12
3.7.1	键值的查看	12
3.7.2	按键选择	13
3.8	新增方案配置	14
4	模块配置	15
4.1	自定义按键配置	15
4.1.1	Android 按键功能的映射	15
4.2	LCD Panel 配置	15
4.3	Touch Panel 配置	15
4.3.1	配置文件的修改	15
4.3.2	Android 层的配置修改	16
4.4	G-Sensor 配置	16
4.4.1	Android 层配置修改	16
4.4.1.1	首先调试 Z 轴	17
4.4.1.2	其次查看 X, Y 轴是否互换	17
4.4.1.3	再次调试 X, Y 轴方向。	18
4.5	红外遥控器配置	18
4.5.1	内核配置	18
4.5.2	Device Tree 配置	19
4.5.3	Android multi_ir 配置	19
4.5.4	Android 按键功能的映射文件	19
4.5.5	新增遥控器适配	20
4.6	GPIO 配置	21
4.6.1	定义需要控制的 GPIO	21
4.7	显示配置	21
4.8	WiFi/BT 配置	21
4.9	Camera 配置	21
4.9.1	Camera 驱动加载顺序	22
4.9.2	Camera 参数配置	22
4.9.3	配置文件的修改	22
5	系统配置	23
5.1	SettingsProvider 设置	23
5.2	默认配置	23
5.3	Allwinner 平台 SettingsSetup	24
6	Launcher 及界面设置	25
6.1	默认壁纸设置	25
6.2	添加壁纸	25
6.3	Launcher 默认图标和快捷栏设置	25
7	后台服务管理配置	27
7.1	功能介绍	27

7.2	方案配置	27
7.3	用户设置	28
7.4	功能调试	28
8	调试 debug	29
8.1	将 logcat 和 dmesg 信息保存到文件系统	29
8.2	使用 fastboot	29



1 前言

1.1 文档简介

本文档介绍 Android 15 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 目标读者

SDK 平台负责人、版本集成人员、SDK 开发人员、模块开发人员。

1.3 适用范围

Allwinner Android 15 及以上平台。

1.4 名词解释

key	value
vendor-name	softwinner
platform-name	芯片平台名称
device-name	设备名称
product-name	产品名称
IC	芯片名称
BOARD	板卡名称

2 编译和发布

2.1 编译固件流程

编译 android 采用 GRF 的方式。当 system 与 vendor 所使用的版本不同时，需要 2 套 SDK 来进行开发和编译。

表 2-1: GRF 版本支持示意图

芯片	System 版本	Vendor 版本	支持新设备	支持老设备升级
A733	Android15	Android14	√	√
A523	Android15	Android13	√	√
A527	Android15	Android13	√	√

详细编译步骤见《Android_GRF_编译指南》文档。

```
$ cd <your-work-space>/android15
$ source build/envsetup.sh
$ lunch <PRODUCT><VARIANT> # ---选择对应平台方案配置
$ make -j16 -m [merge_param] # ---编译Android, j开启多核编译, 服务器开发一般为服务器cpu数量的一半。m是编译合并固件的意思, 编译GRF版本Android必须加-m参数
$ pack [-d|-v] # ---打包固件, -v参数用于启用安全系统校验。
```

2.1.1 GRF target file 来源

1. 编译参数 merge_param 指定，为 Vendor 所对应使用的 Android 环境。

```
编译方式为：
make -j32 -m ../android${Freezed_Vendor_Ver}
```

2. merge_param 不存在时，由环境变量 ANDROID_FROZEN_TARGET_FILE 指定。

```
ANDROID_FROZEN_TARGET_FILE指向Android${Freezed_Vendor_Ver} 冻结环境或者target file
```

```
编译方式为：
make -j32 -m
```

3. 如上两种方式都不存在时，从 sdk 默认路径 target_file 下匹配一个，如果有多个，则取第一个

```
编译方式为：  
make -j32 -m
```

```
匹配规则为：target_file/${target_product}-\{signed_\}\{0,1\}target_files.*zip
```

2.1.2 编译 target file

如果想要编译 target file 可以使用如下命令，更多 GRF 编译命令见《Android_GRF_编译指南》。

```
$ cd <frozen-sdk-path>  
$ source build/envsetup.sh  
$ lunch <PRODUCT>-<VARIANT>  
$ CONFIG_AW_BUILD_VENDOR_TARGET_FILE_ONLY=true build_grf_target_files . $TARGET_PRODUCT-  
$TARGET_BUILD_VARIANT -j16
```

最后会在 \$OUT/obj/PACKAGING/target_files_intermediates 下生成一个 target_file。

2.1.3 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。

2.2 编译模块

2.2.1 源码下编译

本节主要介绍各模块编译方法和更新方法。

1. 以 Settings 为例，编译：

- 快速编译模块：m + 任意模块名

```
示例：  
$ android$ m Settings -j40
```

2. 安装，对于 app 可以直接 install 替换：

```
$ adb install settings.apk绝对路径
```

3. 对于其它模块 so, bin 一类的在 \$OUT 目录下有具体文件的（或者说设备内 system/vendor 目录下），我们使用 push 更新：

示例：

```
首先需要解锁设备 // 非常重要
确认Settings 开发者选项中使能 OEM unlocking为 already unlocked状态
如果没有，则需要进行如下操作
Settings 开发者选项中使能 OEM unlocking
adb reboot bootloader
fastboot oem unlock
fastboot reboot

$ adb root && adb remount -R // 重启+开启remount
$ adb root && adb remount // remount设备
$ adb push test.so绝对路径 + /system/lib
$ adb reboot
```

4. 针对非 vendor 分区且大范围模块

建议生成固定 target file，然后放到 SDK 中的 target_file 目录，然后使用 make -j -m 进行编译。

5. 编译 longan 仓库下的模块

GRF 中 longan 使用 Android{Freezed_Vendor_Ver} 冻结版本，需要在 Android{Freezed_Vendor_Ver} 环境中进行编译。对于只需要编译 longan，而不需要编译 Android 的情况，我司提供了一种无源码打包方式，详情见下一小节。

(1) 编译 uboot

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

(2) 编译 dts

```
./build.sh dts
```

2.2.2 无源码打包

在 BSP 开发过程中，为保证所提交代码能够在 Android 系统上正常运行，一般需要编译 Android 固件进行验证。Android10 以后的 SDK，源码量（包含 git 记录）保守估计在 100G 以上，且 Android 没有改动，只编译 BSP 再生成固件的过程大约耗时 2~10 分钟（视机器性能与当前负载情况）。为此，根据 Android 打包过程，提出一种无需下载 Android 代码、无需编译 Android 的打包 Android 固件的实现。

2.2.2.1 工作原理

只编译内核时，实际改动只有 blmage。Android 会利用编译好的 ramdisk.img 与 blmage 重新生成 boot.img，该过程仅依赖几个参数及工具。这些参数在同一个 Android 版本上基本是固定的。从某一稳定版本完整 Android 固件中提取所有打包依赖的 Android image 镜像，直接 shell 调用相关工具，根据编译逻辑重新生成 boot.img，并替换 longan 编译所生成的 ko 文件利用生成好的 boot.img 及其他预编译好的提取得到的 image 文件，调用 longan 的打包工具，传递合适的参数，生成完整的 Android 固件。

2.2.2.2 使用

1. 先找到 pack hook，pack hook 所在目录根据由当前 sdk 的配置决定，具体值请查看 ./buildconfig 文件 LICHEE_PACK_HOOK 的值。
2. 拷贝一份稳定版本的 Android 完整固件到 pack hook 所在的目录，如 ./buildconfig 中 LICHEE_PACK_HOOK=build/hook/pack/hook.sh，则固件拷贝目录为 /build/hook/pack/

2.2.2.3 编译打包

```
$. /build.sh config  
$. /build.sh  
$. /build.sh pack<_debug><_secure>
```

⚠ 注意

注意：由于 linux 固件和 android 固件在内核编译时需要使用不同的编译工具链，所以在 config 阶段请选择 android 固件，并配置对应的版型。

2.3 发布

2.3.1 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。

如果使用了双SDK GRF编译，相对应的OTA打包的命令也进行了修改：

```
# make -dist -d -v -m ../android{Freezed_Vendor_Ver}/
```

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 vendor/security 目录。签名文件可参考 build/target/product/README 文档。

2.3.2 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：

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。

详细步骤参考 OTA 相关文档进行配置升级。

3 方案定制

可以通过 `cdevice` 命令或者 `cd $android_device_path` 进入 Android 方案配置目录 (`device/{vendor-name}/platform-name`)。

各项配置文件路径请参考 `device/{vendor-name}/{platform-name}/README.md` 说明。

3.1 方案文件替换

Android 方案中有时需要定制化一些需求，如替换开机 LOGO、`env.cfg`、分区配置等，除了可在 `longan` 目录下对应的路径下替换外，android 方案也提供了接口进行替换。其原理是在 `pack` 操作时，将需要替换的文件拷贝到 `longan/out/pack_out` 目录下。

在 android 方案中通过 `BOARD_ADD_PACK_CONFIG` 配置来指定需要替换的文件。

```
# BoardConfig.mk
BOARD_ADD_PACK_CONFIG += $(TARGET_DEVICE_DIR)/system/sys_partition.fex
BOARD_ADD_PACK_CONFIG += $(PRODUCT_PLATFORM_PATH)/common/system/b1_env.cfg:env.cfg
```

通过这种方式替换的文件只能是会被拷贝到 `longan/out/pack_out` 目录下的文件。格式为：

```
local_filepath[:copy_filename]
```

3.2 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](#)

3.2.1 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 的内容将覆盖 DEVICE_PACKAGE_OVERLAYS 中的内容。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 common/overlay 和 {device-name}/overlay 目录，分别用于 device 通用及 product 使用的 overlay 文件夹。

3.2.2 改变 mk 文件来添加 overlays 的编译项

在文件 common/overlay/config.mk 和 {device-name}/overlay/config.mk 中分别添加。

```
DEVICE_PACKAGE_OVERLAYS := \
$(LOCAL_MODULE_PATH)/overlay \
$(DEVICE_PACKAGE_OVERLAYS)

PRODUCT_PACKAGE_OVERLAYS := \
$(LOCAL_MODULE_PATH)/overlay \
$(PRODUCT_PACKAGE_OVERLAYS)
```

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

3.2.3 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 并添加要修改的一项配置，如。

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

3.3 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。

3.3.1 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}/{platform-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)
```

3.3.2 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
```

3.3.3 配置分区

在 sys_partition.fex 进行分区配置，配置文件默认在 device/{vendor-name}/{platform-name}/{device-name}/system/sys_partition.fex。

super 分区为系统分区大小，包括 system、vendor、product 等镜像包含在动态 super 分区当中，调整 super 分区即可调整系统分区大小。

```
[partition]
name = super
size = 2G
downloadfile = "super.fex"
user_type = 0x8000
```

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

📖 说明

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。

3.4 修改启动 LOGO

启动 LOGO 为初始引导阶段的 LOGO。

将默认的启动 logo 放入位置：`longan/device/config/chips/{IC}/configs/{BOARD}/bootlogo.bmp`。

如有多种用途的 logo，可按照使用用途等放到该目录下，如适用于 go 版本的 logo，可将其保存在 `longan/device/config/chips/{IC}/configs/{BOARD}/bootlogo-go.bmp`

同时，Android 对应方案配置应添加如下配置，以表明使用该 bootlogo：
`BOARD_ADD_PACK_CONFIG += longan/device/config/chips/${TARGET_BOARD_IC}/boot-resource/boot-resource/bootlogo-go.bmp:boot-resource/bootlogo.bmp`

3.5 修改开机动画

开机动画的目录通过如下 Makefile 变量定义：

```
device/softwinner/{PRODUCT}/common/media/config.mk:  
BOOTANIMATION_CONFIG_PATH := $(LOCAL_MODULE_PATH)/bootanimation
```

默认路径为 `device/softwinner/{PRODUCT}/common/media/bootanimation/`

可修改 `BOOTANIMATION_CONFIG_PATH` 的值切换 bootanimation 系统的找寻位置。

将动画放入：`bootanimation.zip` 放入以上定义的路径中。

📖 说明

Android13 开始支持在 OTA 自动升级系统时，夜晚系统自动重启过程中降低对用户的干扰，要求这时的开机画面应该以暗色调为主。如果原始开机动画文件以亮色调为主，可另设暗色调开机动画，文件命名为 `bootanimation_dark.zip`，放入以上定义的路径中。

3.5.1 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
```

3.5.2 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。

3.5.3 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。

3.5.4 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
```

3.5.5 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*
```

⚠ 注意

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。

3.6 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。

3.7 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。

3.7.1 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。打印信息如下代码所示。

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

由于 AD 采样的速度非常快，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。

3.7.2 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 `longan/device/config/chips/{IC}/configs/{BOARD}/uboot-board.dts` 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。

```
&fastboot_key {
    device_type = "fastboot_key";
    key_max = <42>;
    key_min = <38>;
};

&recovery_key {
    device_type = "recovery_key";
    key_max = <31>;
    key_min = <28>;
};
```

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， $b1 > c1$ ；
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。

3.8 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 (选择对应平台公版方案配置)
# 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}/{platform-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。

4 模块配置

4.1 自定义按键配置

4.1.1 Android 按键功能的映射

详细配置参考《Linux_LRADC_开发指南》，需要修改方案对应的 board.dts 中，主要修改 key_cnt 及对应电压值。

```
keyboard {
    compatible = "allwinner,keyboard_1350mv";
    status = "okay";
    key_cnt = <3>;
    key0 = <475 0x7372>;
    key1 = <646 0x73>;
    key2 = <900 0x72>;
};
```

4.2 LCD Panel 配置

详细配置请参考文档《Linux_LCD_开发指南》。

4.3 Touch Panel 配置

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

4.3.1 配置文件的修改

配置文件目录：longan/device/config/chips/{IC}/configs/{BOARD}/board.dts。

详细配置请参考相关模块发布文档。

4.3.2 Android 层的配置修改

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

```
insmod /vendor/lib/modules/gslX680new.ko
```

4.4 G-Sensor 配置

配置文件目录：`longan/device/config/chips/{IC}/configs/{BOARD}/board.dts`。

详细配置请参考相关模块发布文档。

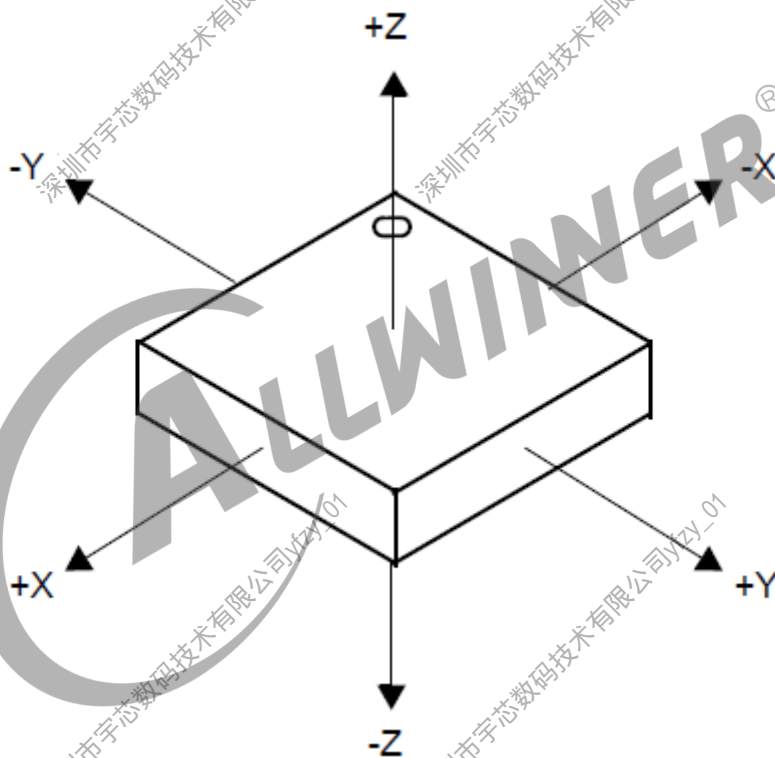


图 4-1: gsensor

4.4.1 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}/{platform-name}/{device-name}/input/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。

4.4.1.1 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 `gsnesor.cfg` 文件 `push` 到 `system/usr` 下，重启机器，按第一步观察现象。

4.4.1.2 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 `gsnesor.cfg` 文件 `push` 到 `system/usr` 下，重启机器，按第一步观察现象。

4.4.1.3 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（gsnesor_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。

4.5 红外遥控器配置

说明

不需要遥控器则跳过该章节

4.5.1 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

4.5.2 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>
};
```

4.5.3 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 \
```

4.5.4 Android 按键功能的映射文件

multi_ir 的按键映射文件主要放在 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。

```
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 \
```

4.5.5 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。

4.6 GPIO 配置



不需要自定义 GPIO 控制则跳过该章节

4.6.1 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/bsp/configs/linux-5.15/sunxxxxx.dtsi` 文件中，类似如下的配置信息。

```
gpio_para:gpio_para {
    device_type = "gpio_para";
    compatible = "allwinner,sunxi-direct-gpio";
    status = "disabled";
};
```

在 `board.dts` 中的配置如下：

```
&gpio_para {
    gpio-pins = <&pio PH 6 GPIO_ACTIVE_LOW>, <&pio PH 9 GPIO_ACTIVE_LOW>; //设备使用的pin脚配置<控制器 引脚号 有效电平>
    init-status = <GPIO_ACTIVE_LOW>, <GPIO_ACTIVE_LOW>; //初始状态
    pin-names = "normal_led", "standby_led"; //设备使用的pin脚名称
    gpio-sdcard-reused; //是否用于卡打印灯的控制，与sdcard复用gpio
    status = "okay"; //设备是否使用
};
```

4.7 显示配置

详细配置请参考相关模块发布文档。

4.8 WiFi/BT 配置

详细配置请参考相关模块发布文档。

4.9 Camera 配置

详细配置请参考相关模块发布文档。

4.9.1 Camera 驱动加载顺序

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

```
on boot
### csi module
insmod /vendor/lib/modules/videobuf2-memops.ko
insmod /vendor/lib/modules/videobuf2-dma-contig.ko
insmod /vendor/lib/modules/vin_io.ko
insmod /vendor/lib/modules/gc5035_mipi.ko
insmod /vendor/lib/modules/gc02m2_mipi.ko
insmod /vendor/lib/modules/ov2680_mipi.ko
insmod /vendor/lib/modules/actuator.ko
insmod /vendor/lib/modules/dw9714_act.ko
insmod /vendor/lib/modules/vin_v4l2.ko
```

4.9.2 Camera 参数配置

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

`media_profiles.xml` 的路径：

`device/{vendor-name}/{platform-name}/{device-name}/camera/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。

4.9.3 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。

5 系统配置

5.1 SettingsProvider 设置

配置文件 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

5.2 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

5.3 Allwinner 平台 SettingsSetup

Allwinner 平台提供了 SettingsSetup apk 用于在第一次开机启动时进行部分 Settings 字段的设置。一些启动不依赖且没有现成配置方式的设置字段，可以通过 SettingsSetup 来进行设置，如手势截屏录屏功能。

配置文件在 vendor/aw/public/package/apk/SettingsSetup/res/values/configs.xml，可通过 overlay 方式进行修改。

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
  <string-array name="secure_list">
  </string-array>
  <string-array name="system_list">
    <item>gesture_screenrecord_enable=1</item>
    <item>gesture_screenshot_enable=1</item>
  </string-array>
  <string-array name="global_list">
  </string-array>
</resources>
```

说明

secure_list、**system_list**、**global_list** 是指 settings 3 个命名空间。

6 Launcher 及界面设置

6.1 默认壁纸设置

替换文件 frameworks/base/core/res/res/drawable-swxxx-dp-nodpi/default_wallpaper.png。

说明

可通过 overlay 方式替换。

6.2 添加壁纸

准备壁纸及壁纸的缩略图放进壁纸存放目录 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。

6.3 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。



7 后台服务管理配置

7.1 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。

7.2 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
stopTaskTime: 60
skipService: true
blockBroadcast: false
lmk: false
lmk_level: 100,150,250

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。
- stopTaskTime: 熄屏多长时间后，对运行任务做清理，单位：秒；负值禁用
- 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。

7.3 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。

7.4 功能调试

通过删除 awbms_config 配置文件可以关闭后台管理功能，通过以下命令可开启/关闭相关调试打印：

```
service call background 1 i32 1/0
```

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。

8 调试 debug

8.1 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 分区文件系统中，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 AOSP 计算器中输入，AOSP 计算器集成了动态开关设置。

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

弹出开关设置界面，根据 UI 进行配置 log 保存选项。

📖 说明

GMS 方案中主界面显示的是 Google 计算器，通过以上操作无法打开动态开关。

AOSP 计算器默认隐藏，打开方式为 Settings->About tablet, 点击 5 次 Version number。

8.2 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
```

• boot loader 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(adb reboot fastboot)`，专门用于烧写 system、vendor、product 等逻辑分区。






著作权声明

版权所有 © 2024 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。