Android GRF
编译指南

版本号: 1.3
发布日期: 2024.10.30

版本历史

版本号	日期	制/修订人	内容描述
1.0	2023.10.10	AWA0989	初始版本文档
1.1	2024.01.3	AWA0681	增加 SDK 中如何快速识别应该修改 vendor 还是 framework 的说明
1.2	2024.01.22	AWA0989	test-key/release-key 说明，及部分章节结构整理
1.3	2024.10.30	AWA0989	调整 system_dlkm 为 vendor 分区；更新 hardware/aw/下文件说明

目　录

1 前言

Android13 全志开始加入 GRF 计划。GRF 要求 Vendor 部分使用冻结版本的 SDK 进行编译，Framework 部分新版本 SDK 编译，因此需要对整个 SDK 的组织存放、编译、打包及模块开发做出一定改变。本文主要针对这些变化进行说明。

2 GRF 介绍

2.1 什么是 GRF

GRF，即 Google Requirement Freeze。推出目的是为了降低**手持设备** OS 的升级难度，确保 Vendor Software 不新增特性的情况下可以支持 4 个版本（3 次 OS 升级），减少碎片化。换言之，Framework 对 Vendor 的需求没有新增时，芯片厂商无需关注 Vendor 的升级及适配，直接进行新版本系统 Framework 相关的开发。

2.2 加入 GRF 的意义

加入 GRF 计划后，Framework 对 Vendor 的需求没有新增时，芯片厂商无需关注 Vendor 的升级及适配，直接进行新版本系统 Framework 相关的开发。新系统升级时，Vendor 部分仍旧来自于已经稳定生产的旧版本 Android SDK。在此种模式下 Vendor 几乎无需做出改动即可适配新的 SDK，从而降低开发难度，缩短开发周期，提升底层的稳定性。

3 GRF 带来的变化

3.1 GRF 开发模式

以 Android13+Android14 为例，GRF 模式下，系统的开发方式如下：

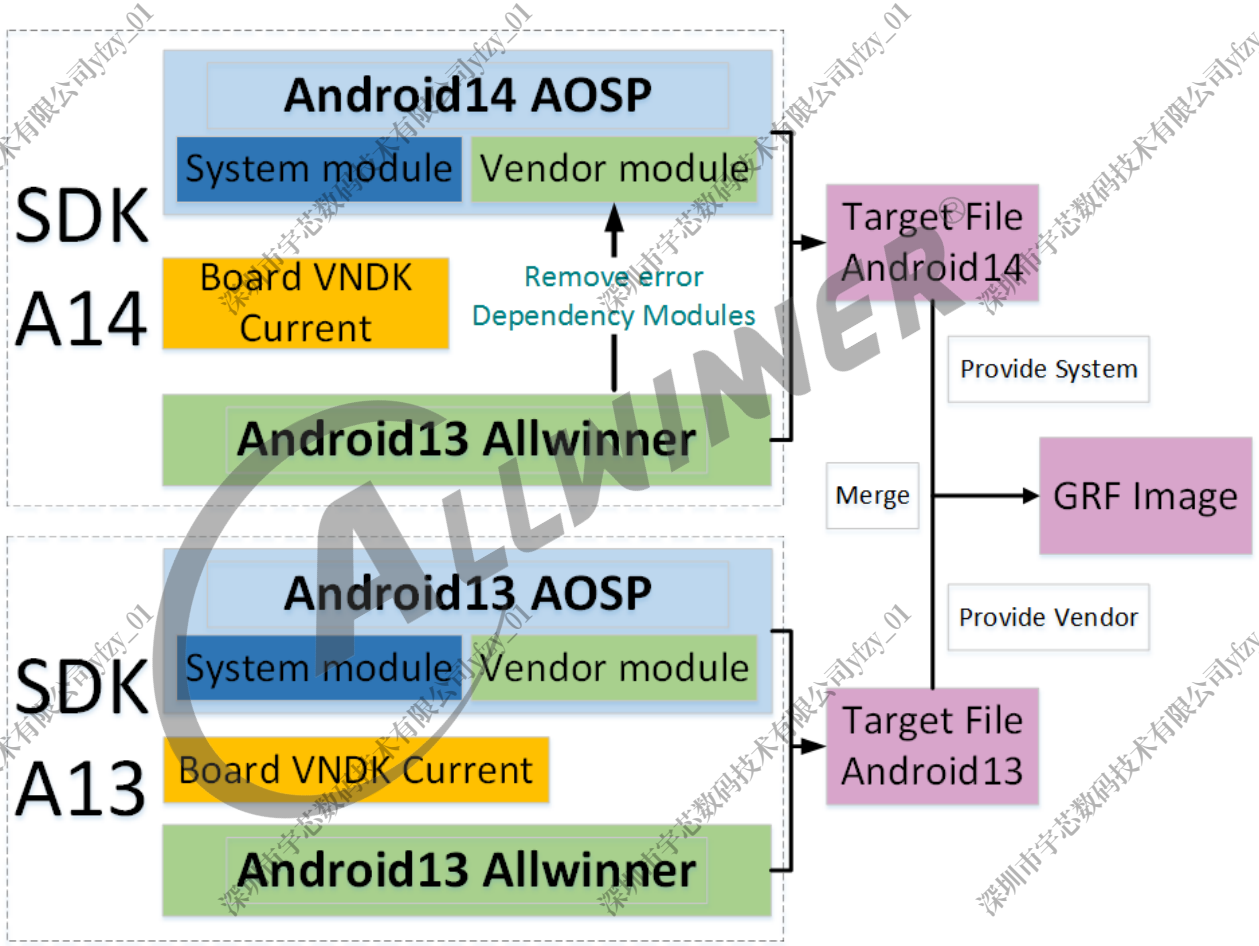

图 3-1: GRF 系统开发方式

由上图可知，新系统升级时，Vendor 部分仍旧来自于已经稳定生产的旧版本 Android13 SDK，在此种模式下 Vendor 几乎无需做出改动即可适配新的 SDK，从而降低开发难度，缩短开发周期，提升底层的稳定性。

3.2 GRF 下 SDK 组织结构

AW Android SDK 通常由 AOSP Android、BSP 两部分构成：

```
$ tree -L 1
.
├── Android.bp -> build/soong/root.bp
├── art
├── bionic
├── bootable
...
├── hardware
├── libcore
├── libnativehelper
├── longan
├── META
├── out
├── packages
├── pdk
....
```

在 GRF 开发模式下，通常需要维护两份 SDK。以 Android14 为例，其代码组织结构如下：

```
$ tree -L 1
.
├── android13
│   ├── Android.bp -> build/soong/root.bp
│   ├── art
│   ├── bionic
│   ├── bootable
│   ├── ...
│   ├── kernel
│   ├── libcore
│   ├── libnativehelper
│   ├── longan
│ ....
└── android14
    ├── Android.bp -> build/soong/root.bp
    ├── art
    ├── bionic
    ├── bootable
    ├── ...
    ├── kernel
    ├── libcore
    ├── libnativehelper
    ├── longan
....
```

其中，Android13 被称之为 frozen vendor sdk，一般来自于 AW 发布的、稳定版本的 SDK，用于 vendor 部分的支持，后续无重大问题时不会再频繁提交、发布更新。Android14 来自于当年 Google 发布的新版本 Android 版本，用于 framework 相关的开发。

代码存放路径无特殊需求，但建议按上述方式存放。

3.3 GRF 中的 vendor/framework

3.3.1 整体原则

在 GRF 开发模式中，vendor 一般指如下分区，及编译到这些分区中的内容：

- BOOT：来自 Google GKI，kernel image + ramdisk（空的）
- INIT_BOOT：kernel image + ramdisk，暂时均为空
- ROOT：boot 开始运行时用的 rootfs
- VENDOR_BOOT：bootconfig、dtb、vendor_ramdisk
- VENDOR：HAL、vendor bin、vendor lib、vendor selinux 等
- VENDOR_DLKM：vendor 驱动 ko 文件
- SYSTEM_DLKM：来自 Google GKI 的一部分，kernel 外需要用户主动加载的 ko

framework 则代指除如上分区外内容外，其余所有分区的内容。如：

- SYSTEM：来自 Google Framework、System 库，虚拟机，libcore、maninline 等
- PRODUCT：GMS、其他产品相关的 APK 等

3.3.2 快速识别 framework/vendor

常见目录

- longan 下所有仓库都是 vendor 的内容，需修改 Android13
- hardware 下所有 hal 层相关仓库都是 vendor 的内容，需修改 Android13，但 hardware/aw/media 除外
- 方案配置目录 device/softwinner/saturn 建议 Android13 和 Android14 都要修改
- frameworks/、packages/ 属于 system，需修改 Android14
- vendor/aw/public 下 bin 文件一般属于 vendor，需修改 Android13；其他一般属于 system 或 product，需要改 Android14
- device/softwinner/common/sepolicy/vendor 属于 vendor，需修改 Android13；device/softwinner/common/sepolicy/private 属于 system，需修改 Android14

方案配置内属性控制变量

- PRODUCT_DEFAULT_PROPERTY_OVERRIDES: framework
- PRODUCT_SYSTEM_DEFAULT_PROPERTIES: framework
- PRODUCT_PRODUCT_PROPERTIES: framework

- PRODUCT_PROPERTY_OVERRIDES: framework
- PRODUCT_VENDOR_PROPERTIES: vendor

方案配置内 COPY_FILES

- TARGET_COPY_OUT_VENDOR: vendor
- TARGET_COPY_OUT_SYSTEM: framework

方案配置内 PRODUCT_PACKAGES

- outmod {module_name} 输出目录在 vendor、root 的属于 vendor，其他属于 framework

说明

- **按照上述说明，仍无法确认到底是修改 Android13 还是 Android14，为避免遗漏，请 Android13 和 Android14 一并修改。**

4 GRF 下编译/打包流程

4.1 编译打包流程介绍

通常，Android 的所有目标可以从 target-files 中生成。因此，GRF 模式下，所有的编译操作都是基于 target-files 的。GRF 模式下，编译流程如下：

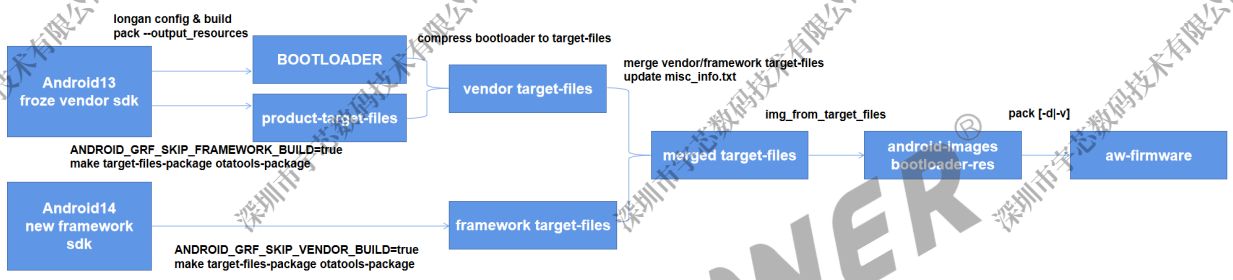

图 4-1: GRF 编译流程

由上图可知，整个编译流程较为复杂。为简化编译流程，AW 针对 make 命令进行了定制，使得可以直接使用 make -m [merge-param] 的方式进行编译。整个完整的编译流程如下 (按推荐路径存放 sdk 时)：

```
$ cd <your-work-space>/android14
$ source build/envsetup.sh
$ lunch <PRODUCT>-<VARIANT>
$ make -j16 -m [merge_param]
$ pack [-d|-v]
```

其中，-m 代指固件由 merge target-files 的方式生成，merge_param 则用来指定 frozen vendor sdk/vendor target-files 的位置。

- 当 merge_param 为一个 zip 文件时，将该文件作为 vendor target-files
- 当 merge_param 为一个未压缩的 target-files 路径时，将该路径作为 vendor target-files
- 当 merge_param 为一个 aw android sdk 路径时，将从该路径编译出 vendor target-files
- 当 merge_param 为空时，将使用 <your-work-space>/android14/target_file 下匹配该产品的 target-files 作为 vendor target-files

同时，当 merge_param 为空时，可以由 ANDROID_FROZEN_TARGET_FILE 环境变量来指定 merge_param。使用方式见上述描述。

4.2 GRF SDK 快速编译参数说明

GRF 编译时，还支持如下编译参数，以方便定制化编译内容来加速编译：

表 4-1: GRF 编译选项

编译参数	含义
--skip-bsp-clean	不执行 Vendor BSP distclean
--skip-bsp-build	不编译 vendor BSP
--skip-vendor-clean	不执行 vendor sdk make installclean
--skip-vendor-build	不编译 vendor，直接取其 target file
--skip-framework-build	不编译 framework，直接取其 target file
--dist	编译 release/OTA 版本固件
--include-dragonboard	编译包含 dragonboard 测试系统的二合一固件
--include-dragonabts	编译包含 dragonabts 测试系统的二合一固件

说明

- **bsp 代指 Allwinner brandy、kernel 等代码**
- **vendor 代指 Android vendor 相关的 package**
- **framework 代指 android 除 vendor 外的部分**

4.3 编译示例

⚠ 注意

- **不携带 --dist 参数时使用 test-key 进行签名，使用 --dist 参数则为 release-key 签名**
- **test-key 编译速度较快，但只可用于调试，不可用于 OTA 升级或 GMS 认证**
- **编译 release/OTA 版本必须携带 --dist 参数**

说明

- **编译之前需要先在 <android14-top-path> 下执行 source、lunch 动作**

- 编译完整系统

```
make -j16 -m ../android13
```

- 编译包含含 dragonboard 测试系统的完整系统

```
make -j16 -m ../android13 --include-dragonboard
```

- 编译 release/OTA

```
make -j16 -m ../android13 --dist  # --dist后可跟-d|-v参数，无需再单独pack
```

- 编译包含 dragonboard 测试系统 release/OTA 固件

```
make -j16 -m ../android13 --dist -t dragonboard # --dist后可跟-d|-v参数，无需再单独pack
```

- 开发 AW BSP，如 spl、uboot、内核驱动，使用 skip 参数加速编译

```
make -j16 -m ../android13 --skip-framework-build
```

- 开发 HAL，使用 skip 参数加速编译

```
make -j16 -m ../android13 --skip-bsp-build --skip-framework-build
```

- 开发 Framework，使用 skip 参数加速编译

```
make -j16 -m ../android13 --skip-vendor-build
```

说明

- **GRF 开发过程中无需再单独到 <android14-top-path>/longan 或 <android13-top-path>/longan 下执行./build.sh config、./build.sh**

4.4 打包 pack 说明

GRF 下如果要编译包含 dragonboard/dragonabts 测试系统的固件，必须在编译时指定--include-dragonboard/--include-dragonabts，且在 pack 时传入-t dragonboard/-t dragonabts。

编译参数	含义
-d	卡打印固件
-v	安全固件
-t dragonboard	包含 dragonboard 测试系统的二合一固件
-t dragonabts	包含 dragonabts 测试系统的二合一固件

说明

- **release/OTA 固件下无需单独执行 pack，而是由–dist 后的参数来代替 pack 参数，非 OTA 下参数含义与之前保持一致。**

- pack 卡打印安全固件

```
pack -d -v
```

- pack 带 dragonboard 测试系统的安全卡打印固件

```
pack -d -v -t dragonboard
```

- pack 带 dragonabts 测试系统的安全卡打印固件

```
pack -d -v -t dragonabts
```

著作权声明

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





（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。