[野火] 嵌入式 Linux 镜像构建与部署——基于 LubanCat-RK 系列板卡

EmbedFire
野火电子

2025 年 12 月 11 日

Contents

关于本项目

鲁班猫（LubanCat）

1、鲁班猫是野火推出的运行 Linux、Android 的卡片电脑品牌。该系列卡片电脑硬件型号丰富，操作系统适配度高，开源教材资料众多，应用极其简单；

2、鲁班猫卓越的性能及其丰富的硬件型号，覆盖了教育、商业应用、工业控制等领域，具备广泛的应用场景：卡片电脑、Linux 服务器、家庭智能化中枢、工业板卡；

3、鲁班猫支持 Ubuntu、Debian、Android 等系统，提供多套教材，覆盖纯应用层用户以及系统开发用户，即使初入行业的嵌入式小白，也能根据我们的教程完成开发，而对资深的嵌入式老鸟，则能加速产品二次开发过程。

文档说明

本文档主要面向有 Linux 系统使用经验的用户，讲解如何在野火 LubanCat 系列板卡上开发、部署以及备份量产自己想要的应用。

本文档以野火的鲁班猫系统（LubanCat OS）作为例子，讲解如何构建出适用于 LubanCat-RK 系列板卡的镜像，并且包含了镜像与应用部署的详细说明，给快速开发基于 Linux 的工程应用提供了非常便捷明了的参考。

点击右侧链接可在线阅读本项目文档：《嵌入式 Linux 镜像构建与部署——基于 LubanCat-RK 系列板卡 》

0.0.1 开发环境

本教程使用的开发环境说明如下：

- PC 系统 Windows：默认使用 Win10 64 位，兼容 Win7 等系统。在使用 RockChip 的烧录工具时需要使用到 Windows 系统。
- PC 系统 Linux：**Ubuntu20.04** ，强烈建议使用相同的版本。
- 鲁班猫系统：主要基于野火定制的鲁班猫系统镜像进行讲解。

板卡配套在线文档及教程

《LubanCat-RK 系列板卡在线教材与源码仓库 》

关于野火

开源共享，共同进步

野火在发布第一块 STM32 开发板之初，就喊出 **开源共享，共同进步**的口号，把代码和文档教程都免费提供给用户下载，而我们也一直把这个理念贯穿至今。

目前我们的产品已经包括 *STM32*、*i.MX RT 系列*、*GD32V*、*FPGA*、*Linux*、*emXGUI*、操作系统、网络、下载器等分支，覆盖电子工程应用领域的各种常用技术，其中教学类产品的代码和文档一直保持着开源的姿态发布到网络上，为电子工程师排忧解难，让嵌入式没有难用的技术是我们最大的愿望。

项目定制

野火接受针对 STM32、i.MX RT、i.MX6、FPGA 主控芯片的各种硬件、软件项目定制，目前已有丰富的成功案例，有需要可以直接发送邮件咨询：embedfire@embedfire.com 。

联系方式

- 官网：http://www.embedfire.com
- 论坛：http://www.firebbs.cn
- github 主页：https://github.com/Embedfire
- gitee 主页：https://gitee.com/Embedfire
- 淘宝：https://yehuosm.tmall.com
- 邮箱：embedfire@embedfire.com

- 电话：0769-33894118

第 1 章　Linux 系统构成简单介绍

一个完整的 linux 系统，通常包含了 Uboot、kernel、设备树以及根文件系统。

1.1 Uboot

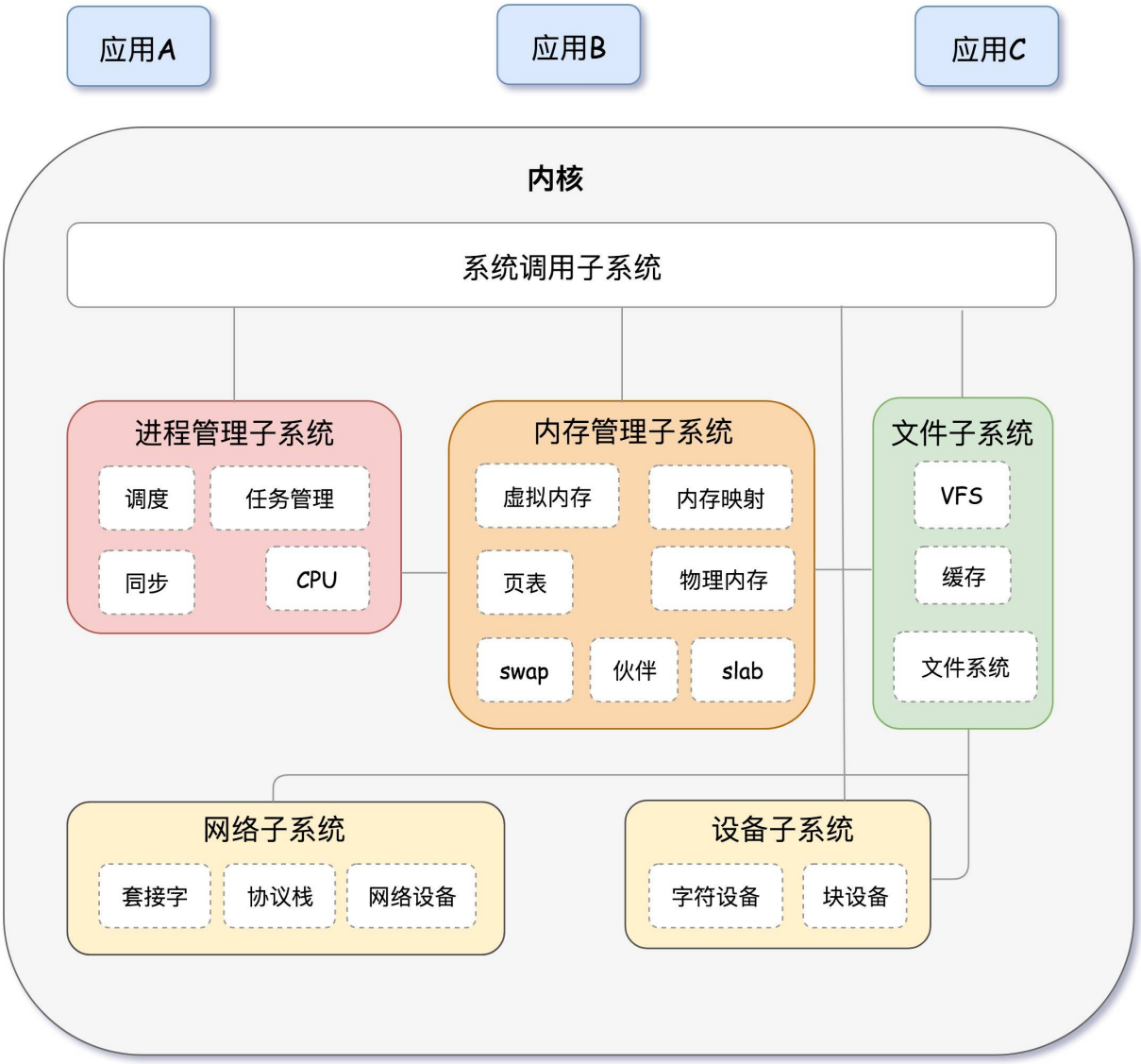
U-Boot 是一个主要用于嵌入式系统的引导加载程序，可以支持多种不同的计算机系统结构，包括 PPC、ARM、AVR32、MIPS、x86、68k、Nios 与 MicroBlaze。这也是一套在 GNU 通用公共许可证之下发布的自由软件。Uboot 的全称 Universal Boot Loader，是遵循 GPL 条款的开源项目，U-Boot 的主要作用是用来启动操作系统内核，它分为两个阶段，即 boot + loader，boot 阶段启动系统，初始化硬件设备，建立内存空间映射图，将系统的软硬件带到一个合适的状态，loader 阶段将操作系统内核文件加载至内存，之后跳转到内核所在地址运行。

另外，某些 BootLoader 可能含有一些高级特性，如校验操作系统镜像，从多个操作系统镜像中选择引导合适的操作系统，或者添加网络功能，让系统自主从网上寻找合适的镜像并且进行引导等等。

Bootloader	Monitor	描　述	x86	ARM	PowerPC
LILO	否	Linux磁盘引导程序	是	否	否
GRUB	否	GNU的LILO替代程序	是	否	否
Loadlin	否	从DOS引导Linux	是	否	否
ROLO	否	从ROM引导Linux而不需要BIOS	是	否	否
Etherboot	否	通过以太网卡启动Linux系统的固件	是	否	否
LinuxBIOS	否	完全替代BUIS的Linux引导程序	是	否	否
BLOB	否	LART等硬件平台的引导程序	否	是	否
U-boot	是	通用引导程序	是	是	是
RedBoot	是	基于eCos的引导程序	是	是	是

1.2 Linux 内核

Linux 是一种开源电脑操作系统内核。它是一个用 C 语言写成，符合 POSIX 标准的类 Unix 操作系统。Linux 内核是一个用来和硬件打交道并为用户程序提供一个有限服务集的低级支撑软件。一个计算机系统是一个硬件和软件的共生体，它们互相依赖，不可分割。计算机的硬件，含有外围设备、处理器、内存、硬盘和其他的电子设备组成计算机的发动机。但是没有软件来操作和控制它，自身是不能工作的。完成这个控制工作的软件就称为操作系统，在 Linux 的术语中被称为“内核”，也可以称为“核心”。Linux 内核的主要模块（或组件）分以下几个部分：进程管理子系统、内存管理子系统、文件子系统、网络子系统、设备子系统等。



1.3 设备树

设备树是一种描述硬件的数据结构，它把这些硬件设备的信息，而这个文件，就是 Device Tree（设备树），设备树包括设备树源码（Device Tree Source，DTS）文件、设备树编译工具（Device Tree Compiler，DTC）与二进制格式设备树（Device Tree Blob，DTB），DTS 包含的头文件格式为

DTSI。

列表 1: 设备树描述

```
node1 {
    a-string-property = "A string";
    a-string-list-property = "first string", "second string";
    a-byte-data-property = [0x01 0x23 0x34 0x56];

    child-node1 {
        first-child-property;
        second-child-property = <1>;
        a-string-property = "Hello, world";
    };
};
```

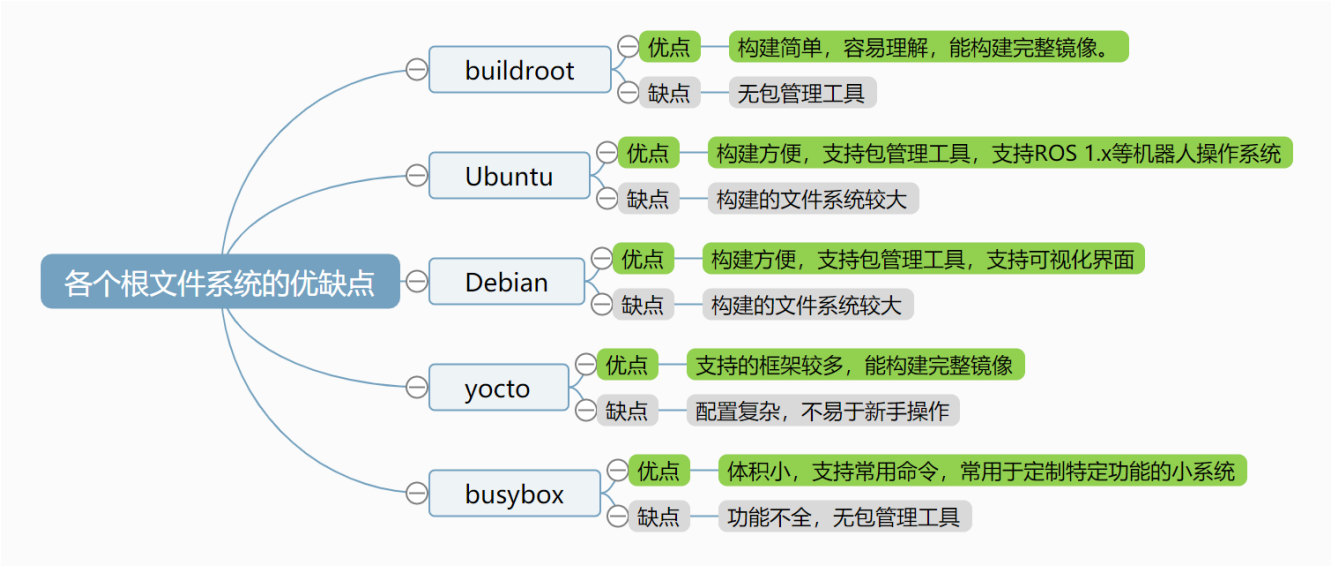
Uboot 和 Linux 不能直接识别 DTS 文件，而 DTB 可以被内核与 BootLoader 识别解析，通常在制作 NAND Flash、SD Card 启动镜像时，通常会为 DTB 文件留下一部分存储区域以存储 DTB，在 BootLoader 启动内核时，会先读取 DTB 到内存，再提供给内核使用。

1.4 根文件系统

根文件系统（rootfs）是 linux 在初始化时加载的第一个文件系统，根文件系统包括根目录和真实文件系统，它包含系统引导和使其他文件系统得以挂载（mount）所必要的文件。根文件系统包函 Linux 启动时所必须的目录和关键性的文件，例如 Linux 启动时必要的初始化文件，它在 init 目录下。此外根文件系统中还包括了许多的应用程序 bin 目录等，任何包括这些 Linux 系统启动所必须的文件都可以成为根文件系统。

在 Linux 内核启动的初始阶段，首先内核会初始化一个基于内存的文件系统，如 initramfs，initrd 等，然后以只读的方式去加载根文件系统（load rootfs），读取并且运行/sbin/init 初始化文件，根据/etc/inittab 配置文件完成系统的初始化工作（提示：/sbin/init 是一个二进制可执行文件，为系统的初始化程序，而/etc/inittab 是它的配置文件），在初始化的过程中，还会以读写的方式重新挂载根文件系统，在系统启动后，根文件系统就可用于存储数据了，存在根文件系统是 Linux 启动时的必要条件。

我们常见的根文件系统制作工具有 buildroot、Ubuntu、Debian、yocto、busybox，这些工具的优缺点列出如下



在后文中，我们会详细讲解 Ubuntu 和 Debian 根文件系统的制作过程，这两个根文件系统也是我们主要支持的版本。

第 2 章　不同 LubanCat_SDK 对比

注解： LubanCat_Chip_SDK 将于 2024 年 9 月 10 日起逐步停止新功能的添加，只进行 Bug 修正。RK 系列板卡的后续开发支持已经转移到 LubanCat_Linux_Generic_SDK。各版本 SDK 独有内容将在标题添加 **(Gen)** 或 **(Chip)** 标识，未添加此标识则表示内容通用。

2.1 不同版本 SDK 对比

为了满足用户的需求，目前野火基于 LubanCat-RK 系列板卡共推出 3 个 SDK 包，分别是 LubanCat_Linux_rk356x_SDK、LubanCat_Linux_rk3588_SDK 和 LubanCat_Linux_Generic_SDK。

2.1.1 区分 SDK 版本

本文档同时包含了多个 SDK 的使用说明，除了通用内容以外，还包含根据 SDK 差异编写的文档，为了更好的将本文档内容与不同的 SDK 包对应，我们先要区分 SDK 版本。

通过网盘下载 SDK 源码的用户，可以通过源码压缩包的名称来区分，分别是：

- LubanCat_Linux_Generic_SDK
- LubanCat_Linux_rk356x_SDK
- LubanCat_Linux_rk3588_SDK

通过 Github 在线下载或已经将 SDK 源码包解压的用户，可以通过 repo 工具的配置文件来查看。

查看.repo/manifest.xml 文件中的 include 这一行，name 后面的字段是实际的 xml 配置文件的名称

- lubancat_linux_generic.xml，对应 LubanCat_Linux_Generic_SDK
- rk356x_linux_release.xml，对应 LubanCat_Linux_rk356x_SDK

- rk3588_linux_release.xml，对应 LubanCat_Linux_rk3588_SDK

2.1.2 SDK 对比

在本文档中，将 LubanCat_Linux_Generic_SDK 简称为 LubanCat_Gen_SDK；由于 LubanCat_Linux_rk356x_SDK 和 LubanCat_Linux_rk3588_SDK 基于同一框架，差异较小，所以统称为 LubanCat_Chip_SDK。如果文档中的内容通用于所有版本 SDK，则以 LubanCat-SDK 代指

差异	LubanCat_Gen_SDK	LubanCat_Chip_SDK
支持板卡	所有使用 RK 芯片的 LubanCat 板卡	对应 RK 芯片的 LubanCat 板卡
更新状态	持续更新中	已进入稳定版本, 只修复 BUG
便利性	框架较新, 功能更多, 开发更便利	配置简单，冗余功能少
功能	使用较新的内核、根文件系统等组件	使用稳定版的内核、根文件系统等组件

2.2 LubanCat_Gen_SDK 板卡支持情况

2.2.1 rk312x

- 鲁班猫 0H 系列, 使用 RK3128 主芯片

2.2.2 rk3528

- 鲁班猫 Q1 系列, 使用 RK3528A 主芯片

2.2.3 rk3562

- 鲁班猫 1HS 系列, 使用 RK3562/RK3562J(工业级) 主芯片

2.2.4 rk3566_rk3568

- 鲁班猫 0 系列, 使用 RK3566 主芯片
- 鲁班猫 1 系列, 使用 RK3566 主芯片
- 鲁班猫 2 系列, 使用 RK3568/RK3568J(工业级) 主芯片

2.2.5 rk3576

- 鲁班猫 3 系列, 使用 RK3576 主芯片

2.2.6 rk3588

- 鲁班猫 4 系列, 使用 RK3588s 主芯片
- 鲁班猫 5 系列, 使用 RK3588 主芯片

2.3 LubanCat_Chip_SDK 板卡支持情况

2.3.1 LubanCat_Linux_rk356x_SDK

- 鲁班猫 0 系列, 使用 RK3566 主芯片
- 鲁班猫 1 系列, 使用 RK3566 主芯片
- 鲁班猫 2 系列, 使用 RK3568/RK3568J(工业级) 主芯片

警告： 此 SDK 不支持使用 rk3562/RK3562J 主芯片的 LubanCat-1HS 板卡

2.3.2 LubanCat_Linux_rk3588_SDK

- 鲁班猫 4 系列, 使用 RK3588s 主芯片
- 鲁班猫 5 系列, 使用 RK3588 主芯片

第 3 章　LubanCat_Gen_SDK

3.1 简介

LubanCat_Gen_SDK 是基于瑞芯微通用 Linux SDK 工程的深度定制版本，适用于以 Rockchip 处理器为主芯片的 LubanCat-RK 系列板卡。

整个 SDK 工程，目录包含有 debian、kernel、u-boot、device、rkbin、ubuntu 等目录。每个目录或其子目录会对应一个 git 工程，提交需要在各自的目录下进行。

- debian11：debian11 根文件系统构建脚本, 搭配 Kernel-5.10 使用。
- debian12：debian12 根文件系统构建脚本, 搭配 Kernel-6.1 使用。
- device/rockchip：存放各芯片板级配置，以及构建进行相关的脚本等。
- kernel-5.10：存放 kernel 5.10 版本源码。
- kernel-6.1：存放 kernel 6.1 版本源码。
- lubancat-bin：存放鲁班猫板卡使用的二进制文件。
- prebuilts：存放交叉编译工具链。
- rkbin：存放 Rockchip 相关的 Binary 和工具。
- rockdev：存放编译输出的镜像文件 (软链接到 output/firmware)。
- output：存放编译过程中生成的过程文件、日志、编译出的镜像等。
- output-release：存放编译完成并归档后的系统镜像
- tools：存放 Linux 和 Windows 操作系统环境下常用工具。
- u-boot：存放基于 v2017.09 版本进行开发的 uboot 代码。
- ubuntu20.04: Ubuntu 20.04 根文件系统构建脚本, 搭配 Kernel-5.10 使用。
- ubuntu22.04: Ubuntu 22.04 根文件系统构建脚本, 搭配 Kernel-6.1 使用。

3.2 extboot 分区介绍

extboot 分区系统是野火基于瑞芯微 Linux_SDK 框架搭建的一种 LubanCat-RK 系列板卡通用镜像实现方式。可以实现一个镜像烧录到 LubanCat 使用同一型号处理器的所有板卡，解决了默认 rkboot 分区方式设备树固定，导致一个镜像只能适配一款板卡的问题，大大降低了由于型号众多导致的后期维护的复杂性。

extboot 分区使用 ext4 文件系统格式，在编译过程中将所有 LubanCat-RK 系列板卡设备树都编译并打包到分区内，并借助 SDRADC 读取板卡硬件 ID，来实现设备树自动切换。同时支持设备树插件，自动更新内核 deb 包，在线更新内核和驱动模块等功能。

还对系统镜像的分区做了调整，删减一些冗余功能，实现了对系统存储的高效利用。

3.3 SDK 开发环境搭建

LubanCat_Gen_SDK 是基于 Ubuntu LTS 系统开发测试的，在开发过程中，主要是用 Ubuntu 20.04 版本，推荐用户使用 Ubuntu20.04 或 Ubuntu22.04，不支持 Ubuntu20.04 以下版本开发。

硬件配置推荐：64 位系统，硬盘空间大于 80G。如果您进行多个构建，将需要更大的硬盘空间。

安装 SDK 依赖的软件包

```
# 安装 SDK 构建所需要的软件包
# 整体复制下面内容到终端中安装
sudo apt-get update && sudo apt-get install git ssh make gcc libssl-dev \
liblz4-tool expect expect-dev g++ patchelf chrpath gawk texinfo chrpath \
diffstat binfmt-support qemu-user-static live-build bison flex fakeroot \
cmake gcc-multilib g++-multilib unzip device-tree-compiler ncurses-dev \
libgucharmap-2-90-dev bzip2 expat gpgv2 cpp-aarch64-linux-gnu libgmp-dev \
libmpc-dev bc python-is-python3 python3-pip python2 u-boot-tools curl \
python3-pyelftools dpkg-dev
```

3.4 安装 repo

repo 是 google 用 Python 脚本写的调用 git 的一个脚本，主要是用来下载、管理项目的软件仓库。

```
mkdir ~/bin
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
# 如果上面的地址无法访问，可以用下面的：
# curl -sSL  'https://gerrit-googlesource.proxy.ustclug.org/git-repo/+/master/repo?format=TEXT' |base64 -d > ~/bin/repo
chmod a+x ~/bin/repo
echo PATH=~/bin:$PATH >> ~/.bashrc
source ~/.bashrc
```

执行完上面的命令后来验证 repo 是否安装成功能正常运行。

```
repo --version

# 返回以下信息
# 返回的信息根据 Ubuntu 版本的不同略有差异
<repo not installed>
repo launcher version 2.32
    (from /home/he/bin/repo)
git 2.25.1
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0]
OS Linux 5.15.0-60-generic (#66~20.04.1-Ubuntu SMP Wed Jan 25 09:41:30 UTC 2023)
CPU x86_64 (x86_64)
Bug reports: https://bugs.chromium.org/p/gerrit/issues/entry?template=Repo+tool+issue
```

3.5 SDK 源码获取

LubanCat_Gen_SDK 的代码被划分为了若干 git 仓库分别进行版本管理，可以使用 repo 工具对这些 git 仓库进行统一的下载，提交，切换分支等操作。

运行以下命令，将在当前用户的家目录下创建一个名为 LubanCat_SDK 的目录，用来放入 SDK 源码。

```
mkdir ~/LubanCat_SDK
```

3.5.1 切换 Python 3 版本

```
# 查看当前 Python 版本
python -V
```

若返回的版本号为 Python3 版本，则无需再切换 Python 版本。若为 Python2 版本或未发现 python，则可以用以下方式切换：

```
# 查看当前系统安装的 Python 版本有哪些
ls /usr/bin/python*

# 将 python 链接到 python3
sudo ln -sf /usr/bin/python3 /usr/bin/python

# 重新查看默认 Python 版本
python -V
```

此时系统默认 Python 版本切换为 python3

```
dev@dev_152:~/LINUX$ python -V
Python 2.7.17
dev@dev_152:~/LINUX$ ls /usr/bin/python*
/usr/bin/python             /usr/bin/python3.6             /usr/bin/python3m
/usr/bin/python2            /usr/bin/python3.6m            /usr/bin/python-config
/usr/bin/python2.7          /usr/bin/python3-jsondiff      /usr/bin/pythontex
/usr/bin/python2.7-config   /usr/bin/python3-jsonpatch     /usr/bin/pythontex3
/usr/bin/python2-config     /usr/bin/python3-jsonpointer
/usr/bin/python3            /usr/bin/python3-jsonschema
dev@dev_152:~/LINUX$ sudo ln -sf /usr/bin/python3 /usr/bin/python
dev@dev_152:~/LINUX$ python -V
Python 3.6.9
dev@dev_152:~/LINUX$
```

3.5.2 Git 配置

设置自己的 git 信息，以确保后续拉取代码时正常进行，如果不需要提交代码的话可以随意设置用户名和邮箱地址。

```
git config --global user.name "your name"
git config --global user.email "your mail"
```

3.5.3 SDK 在线下载并同步

LubanCat_Gen_SDK 源码可以在线获取，源码托管在 Github。由于在线下载的方式要从 Github 拉取大量的仓库，体积很大，对于不能快速访问 Github 的用户不建议使用这种方式。

```
cd ~/LubanCat_SDK

# 拉取 LubanCat_Linux_Generic_SDK
repo init -u https://github.com/LubanCat/manifests.git -b linux -m lubancat_
↪linux_generic.xml

# 如果运行以上命令失败，提示：fatal: Cannot get https://gerrit.googlesource.com/
↪git-repo/clone.bundle
# 则可以在以上命令中添加选项 --repo-url https://mirrors.tuna.tsinghua.edu.cn/
↪git/git-repo
```

(下页继续)

(续上页)

```
8
9 .repo/repo/repo sync -c -j4
```

```
dev@dev_152:~/LubanCat_SDK$ repo init -u https://github.com/LubanCat/manifests.git -b linux -m rk356
x_linux_release.xml
warning: Python 3 support is currently experimental. YMMV.
Please use Python 2.6 - 2.7 instead.

... A new version of repo (2.29) is available.
... New version is available at: /home/dev/LubanCat_SDK/.repo/repo/repo
... The launcher is run from: /usr/bin/repo
!!! The launcher is not writable.  Please talk to your sysadmin or distro
!!! to get an update installed.


Your identity is: 贺嘉文 <hjw0415@outlook.com>
If you want to change this, please re-run 'repo init' with --config-name

repo has been initialized in /home/dev/LubanCat_SDK
```

如果同步失败可以重新运行 sync 命令来同步

```
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c -j4
Fetching: 58% (17/29) u-boot
remote: Enumerating objects: 1708, done.
remote: Counting objects:   0% (1/300)
remote: Counting objects:   1% (3/300)
remote: Counting objects:   2% (6/300)
remote: Counting objects:   3% (9/300)
remote: Counting objects:   4% (12/300)
remote: Counting objects:   5% (15/300)
remote: Counting objects:   6% (18/300)
remote: Counting objects:   7% (21/300)
remote: Counting objects:   8% (24/300)
remote: Counting objects:   9% (27/300)
remote: Counting objects:  10% (30/300)
remote: Counting objects:  11% (33/300)
remote: Counting objects:  12% (36/300)
remote: Counting objects:  13% (39/300)
remote: Counting objects:  14% (42/300)
```

```
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c -j4
Fetching: 86% (25/29) camera_engine_rkaiq
Fetching: 100% (29/29), done in 28m49.416s
NOT Garbage collecting:   0% (0/29), done in 0.010s
Checking out files: 100% (17367/17367), done.
Checking out files: 100% (696/696), done.
Checking out files: 100% (1196/1196), done.
Checking out files: 100% (1487/1487), done.
Checking out files: 100% (141/141), done.
Checking out files: 100% (960/960), done.
Checking out files: 100% (75371/75371), done.
Checking out files: 100% (17900/17900), done.
Checking out files: 100% (7165/7165), done.
Checking out files: 100% (237/237), done.
Checking out files: 100% (13583/13583), done.
Checking out: 100% (29/29), done in 44.538s
repo sync has finished successfully.
```

3.5.4 SDK 离线安装下载

由于 Github 服务器在国外，拉取这么多的仓库需要很多时间，还可能因为网络不畅通而导致下载失败。为此，我们将需要使用的仓库整体打包，使用网盘下载的方式，以减少连接 Github 导致的问题。

3.5.4.1 下载地址

访问百度网盘资源介绍页面获取 SDK 源码压缩包: 8-SDK 源码压缩包

下载最新日期的 LubanCat_Linux_Generic_SDK 压缩包即可。

注解： 在开发过程中，当 SDK 源码进入 Release 版本或有重大 Bug 时，将会更新源码压缩包。源码压缩包解压到本地后，当有 Release 版本更新时，可以借助 Github 更新到最新版本。

3.5.4.2 解压源码

以下过程以 LubanCat_Linux_Generic_SDK 进行演示，实际文件名称以自己下载的 SDK 为准

```
# 安装 tar 压缩工具，一般来说系统默认安装了
sudo apt install tar

# 在用户家目录创建 LubanCat_SDK 目录
mkdir ~/LubanCat_SDK

# 将下载的 SDK 源码移动到 LubanCat_SDK 目录下，xxx 为日期
mv LubanCat_Linux_Generic_SDK_xxx.tgz ~/LubanCat_SDK

# 进入 LubanCat_SDK 目录
cd ~/LubanCat_SDK

# 解压 SDK 压缩包
tar -xzvf LubanCat_Linux_Generic_SDK_xxx.tgz

# 查看解压后的文件，可以看到解压出.repo 文件夹
ls -al

# 检出各个 git 子仓库
# 注意：下面的命令一点要在 SDK 顶层文件夹中执行，且 repo 路径一定为.repo/repo/repo
.repo/repo/repo sync -l

# 将所有的源码仓库同步到最新版本
# 如果使用 LubanCat_Linux_Gen_Full_SDK，则无需使用下面的命令更新 SDK
.repo/repo/repo sync -c
```

如果.repo/repo/repo sync -c 执行时提示网络连接超时，请检查并能否通畅访问 github。确认可以正常访问 github 的话，可以重复多次执行.repo/repo/repo sync -c 命令来进行同步。若无法访问 github，可以忽略同步源码仓库到最新版本这一步骤。

```
jiawen@dev120:~$ sudo apt install tar
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
tar 已经是最新版 (1.30+dfsg-7ubuntu0.20.04.4)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 27 个软件包未被升级。
jiawen@dev120:~$ mkdir ~/LubanCat_SDK
jiawen@dev120:~$ mv LubanCat_Linux_Generic_SDK_xxx.tgz ~/LubanCat_SDK
mv: 无法获取'LubanCat_Linux_Generic_SDK_xxx.tgz' 的文件状态(stat): 没有那个文件或目录
jiawen@dev120:~$ mv LubanCat_Linux_Generic_SDK_20240913.tgz ~/LubanCat_SDK
jiawen@dev120:~$ cd ~/LubanCat_SDK
jiawen@dev120:~/LubanCat_SDK$ ls
LubanCat_Linux_Generic_SDK_20240913.tgz
jiawen@dev120:~/LubanCat_SDK$ tar -xzvf LubanCat_Linux_Generic_SDK_20240913.tgz
.repo/
.repo/repo/
.repo/repo/.git/
.repo/repo/.git/branches/
.repo/repo/.git/hooks/
.repo/repo/.git/hooks/applypatch-msg.sample
.repo/repo/.git/hooks/commit-msg.sample
.repo/repo/.git/hooks/fsmonitor-watchman.sample
.repo/repo/.git/hooks/post-update.sample
.repo/repo/.git/hooks/pre-applypatch.sample
.repo/repo/.git/hooks/pre-commit.sample
.repo/repo/.git/hooks/pre-push.sample
.repo/repo/.git/hooks/pre-rebase.sample
.repo/projects/lubancat-bin.git/.repo_config.json
.repo/.repo_fetchtimes.json
.repo/.repo_localsyncstate.json
.repo/project.list
.repo/copy-link-files.json
jiawen@dev120:~/LubanCat_SDK$ ls
LubanCat_Linux_Generic_SDK_20240913.tgz
jiawen@dev120:~/LubanCat_SDK$ ls -al
总用量 6618272
drwxrwxr-x  3 jiawen jiawen         66 Sep 13 17:24 .
drwxr-xr-x 46 jiawen jiawen       4096 Sep 13 17:24 ..
-rw-rw-r--  1 jiawen jiawen 6777101462 Sep 13 11:38 LubanCat_Linux_Generic_SDK_20240913.tgz
drwxrwxr-x  7 jiawen jiawen        243 Sep 13 11:13 .repo
jiawen@dev120:~/LubanCat_SDK$
```

解压完成后 checkout 到指定的提交。

```
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -l
Checking out files: 100% (17367/17367), done.
Checking out files: 100% (696/696), done.
Checking out files: 100% (1198/1198), done.
Checking out files: 100% (1487/1487), done.
Checking out files: 100% (73/73), done.
Checking out files: 100% (960/960), done.
Checking out files: 100% (75373/75373), done.
Checking out files: 100% (17900/17900), done.
Checking out files: 100% (7165/7165), done.
Checking out files: 100% (237/237), done.
Checking out: 100% (29/29), done in 31.111s
repo sync has finished successfully.
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c
remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
Fetching: 100% (29/29), done in 0.824s
NOT Garbage collecting:   0% (0/29), done in 0.001s
repo sync has finished successfully.
dev@dev_152:~/LubanCat_SDK$
```

一般情况下网盘保存的离线源码包已经是最新版本，如果距离离线源码包下载时间不久，可以忽略从 Github 在线更新这一步。

3.5.5 SDK 更新

我们会对 LubanCat_Gen_SDK 不断更新，并将修改的内容实时同步到 Github，如果需要在本地 LubanCat_Gen_SDK 同步更新内容，则可以借助 repo 或 git 来实现。

3.5.5.1 使用 repo 更新整个 SDK

> **警告：** 此小节内容 LubanCat_Linux_Gen_Full_SDK 不适用

使用 repo 可以将 SDK 整体更新到提供的最新版本。

> **警告：** 此操作会把各个子 Git 仓库更新到此 SDK 各组件互相匹配的最新版本，虽然单个仓库可能不是最新，但各组件匹配性好，工作稳定

首先要更新.repo/manifests，里面保存了 repo 的配置文件，记录了仓库的版本信息。

```
# 进入.repo/manifests 目录
cd .repo/manifests

# 切换分支到 Linux
git checkout linux

# 拉取最新的 manifests
git pull

# 进入 SDK 顶层文件夹
cd ~/LubanCat_SDK

# 同步远端仓库
.repo/repo/repo sync -c
```

```
dev@dev_152:~/LubanCat_SDK$ cd .repo/manifests
dev@dev_152:~/LubanCat_SDK/.repo/manifests$ git pull
Updating eb35cbe..e3f0107
Fast-forward
 README.md                          | 11 +++++------
 rk356x_linux/rk356x_linux_dev.xml | 10 +++++-----
 2 files changed, 10 insertions(+), 11 deletions(-)
dev@dev_152:~/LubanCat_SDK/.repo/manifests$ cd ~/LubanCat_SDK
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c --no-tags
Fetching: 100% (29/29), done in 0.662s
NOT Garbage collecting:   0% (0/29), done in 0.001s
kernel/: manifest switched refs/heads/stable-4.19-rk356x...27fc0130643a8be83f86c80c5d4fced232c7a694
kernel/: discarding 4 commits removed from upstream
project kernel/
First, rewinding head to replay your work on top of it...
Applying: kernel-android-confifg:refresh kernel configuration
Applying: add JL2101 rx/tx delayline
Applying: kernel:lubancat2 fix ir infrared and headphone unplugging problem
Applying: dts 添加LubanCat2N mipi和双屏显示设备树

project kernel/

Checking out: 100% (29/29), done in 1.520s
repo sync has finished successfully.
dev@dev_152:~/LubanCat_SDK$
```

3.5.5.2 使用 Git 更新单独的源码仓库

有时只想更新某个仓库，而不是去更新整个 SDK。或者已经对 SDK 的某些仓库做出了修改，使用 repo 同步的话就会失败。此时就需要对单个仓库进行更新了。

警告： LubanCat_Linux_Gen_Full_SDK 如果要进行更新，则只能依照此小节内容进行单独 Git 仓库更新

警告： 此操作会单独更新 SDK 子 Git 仓库，会将指定的仓库更新到最新版本，但可能打破各组件之间的依赖关系，降低 SDK 稳定性

为了减少 SDK 传输的文件大小，默认在 SDK 下载时只同步 Git 仓库指定的单次提交，默认处于无分支状态，如果要使用 Git 更新单独的源码仓库，则需要先将对应仓库切换到指定分支。

以下是 LubanCat_Gen_SDK 常用仓库的分支说明：

仓库路径	默认分支
debian11	debian11
debian12	debian12
kernel-5.10	lbc-develop-5.10
kernel-6.1	lbc-develop-6.1
rkbin	main
tools	main
u-boot	main
ubuntu20.04	ubuntu20.04
ubuntu22.04	ubuntu22.04

如果要更新的仓库分支在此处没有列出，可以查看 **.repo/manifests/lubancat_linux/lubancat_linux_generic_rele** 文件中各条目中的 dest-branch 就是要切换的分支

这里以 Kernel-5.10 仓库为例

```
# 进入 kernel-5.10 目录下
cd kernel-5.10

# 检出到对应仓库的默认分支
git checkout lbc-develop-5.10

# 拉取 git 仓库
git pull
```

3.6 LubanCat_Gen_SDK 自动构建

LubanCat_Gen_SDK 的构建脚本可以实现自动构建，具体操作方式如下：

以下文档以使用 rk3588 主芯片的板卡为例，使用其他型号芯片的板卡操作方法类似。

在 SDK 顶层文件夹下，执行以下命令，以选择要构建的板卡主芯片型号和 SDK 的配置文件。

```
# 选择 SDK 配置文件
./build.sh chip

1. rk312x
2. rk3528
3. rk3562
4. rk3566_rk3568
5. rk3576
6. rk3588

# 输入想要构建的板卡主芯片型号的编号，并确认，这里选择 rk3588。
Which would you like? [1]: 5

# 选择要构建的根文件系统配置文件
1. rockchip_defconfig
2. LubanCat_rk3588_debian_gnome_defconfig
```

(下页继续)

(续上页)

```
3. LubanCat_rk3588_debian_lite_defconfig
4. LubanCat_rk3588_ubuntu_gnome_defconfig
5. LubanCat_rk3588_ubuntu_lite_defconfig
6. LubanCat_rk3588_ubuntu_xfce_defconfig
Which would you like? [1]: 2
```

注解： 芯片型号编号和配置文件编号顺序可能发生变化，以实际为准。

```
jiawen@dev120:~/LINUX_SDK_RELEASE$ ./build.sh chip

############### LubanCat Linux SDK ###############

Manifest: lubancat_linux_generic_20250115.xml

Log colors: message notice warning error fatal

ls: cannot access '/home/jiawen/LINUX_SDK_RELEASE/device/rockchip/.chip/': No such file or direc
tory
Log saved at /home/jiawen/LINUX_SDK_RELEASE/output/sessions/2025-01-16_14-55-26
Pick a chip:

1. rk312x
2. rk3528
3. rk3562
4. rk3566_rk3568
5. rk3576
6. rk3588
Which would you like? [1]: 6
Switching to chip: rk3588
Pick a defconfig:

1. rockchip_defconfig
2. LubanCat_rk3588_debian_gnome_defconfig
3. LubanCat_rk3588_debian_lite_defconfig
4. LubanCat_rk3588_ubuntu_gnome_defconfig
5. LubanCat_rk3588_ubuntu_lite_defconfig
6. LubanCat_rk3588_ubuntu_xfce_defconfig
7. rockchip_rk3588_evb1_lp4_v10_amp_defconfig
8. rockchip_rk3588_evb1_lp4_v10_defconfig
9. rockchip_rk3588_evb1_lp4_v10_mcu_defconfig
10. rockchip_rk3588_evb7_v11_defconfig
11. rockchip_rk3588_ipc_evb1_v10_defconfig
12. rockchip_rk3588_ipc_evb7_lp4_v11_defconfig
13. rockchip_rk3588_multi_ipc_evb1_v10_defconfig
14. rockchip_rk3588s_evb1_lp4x_v10_defconfig
Which would you like? [1]: 1
Switching to defconfig: /home/jiawen/LINUX_SDK_RELEASE/device/rockchip/.chip/rockchip_defconfig
mkdir -p /home/jiawen/LINUX_SDK_RELEASE/output/kconf/lxdialog
make CC="gcc" HOSTCC="gcc" \
    obj=/home/jiawen/LINUX_SDK_RELEASE/output/kconf -C /home/jiawen/LINUX_SDK_RELEASE/device/roc
kchip/common/kconfig -f Makefile.br conf
make[1]: Entering directory '/home/jiawen/LINUX_SDK_RELEASE/device/rockchip/common/kconfig'
gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCA
LE  -I/home/jiawen/LINUX_SDK_RELEASE/output/kconf -DCONFIG_=\"\"  -MM *.c > /home/jiawen/LINUX_S
DK_RELEASE/output/kconf/.depend 2>/dev/null || :
gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCA
LE  -I/home/jiawen/LINUX_SDK_RELEASE/output/kconf -DCONFIG_=\"\"   -c conf.c -o /home/jiawen/LIN
```

如果在编译完一个主芯片的工程后需要切换编译其他的主芯片，要先用以下命令清理 SDK，防止由缓存或编译环境引起的编译错误。

```
# 清除整个 SDK 的
./build.sh cleanall

# 选择 SDK 配置文件
./build.sh chip
```

如果已经选择过了主芯片并且不需要切换主芯片，而是要切换同一主芯片的其他板卡或文件系统类型，则不需要清理 SDK。

```
# 选择 SDK 配置文件
./build.sh lunch

# 输入想要构建的板卡及文件系统配置文件编号，并确认，这里选择配置文件 LubanCat_rk3588_
↪debian_gnome_defconfig。
Which would you like? [0]: 2
```

```
jiawen@dev120:~/LINUX_SDK_RELEASE$ ./build.sh lunch

################ LubanCat Linux SDK ###############

Manifest: lubancat_linux_generic_20250115.xml

Log colors: message notice warning error fatal

Log saved at /home/jiawen/LINUX_SDK_RELEASE/output/sessions/2025-01-16_14-56-42
Pick a defconfig:

1. rockchip_defconfig
2. LubanCat_rk3588_debian_gnome_defconfig
3. LubanCat_rk3588_debian_lite_defconfig
4. LubanCat_rk3588_ubuntu_gnome_defconfig
5. LubanCat_rk3588_ubuntu_lite_defconfig
6. LubanCat_rk3588_ubuntu_xfce_defconfig
7. rockchip_rk3588_evb1_lp4_v10_amp_defconfig
8. rockchip_rk3588_evb1_lp4_v10_defconfig
9. rockchip_rk3588_evb1_lp4_v10_mcu_defconfig
10. rockchip_rk3588_evb7_v11_defconfig
11. rockchip_rk3588_ipc_evb1_v10_defconfig
12. rockchip_rk3588_ipc_evb7_lp4_v11_defconfig
13. rockchip_rk3588_multi_ipc_evb1_v10_defconfig
14. rockchip_rk3588s_evb1_lp4x_v10_defconfig
Which would you like? [1]: 2
Switching to defconfig: /home/jiawen/LINUX_SDK_RELEASE/device/rockchip/.chip/LubanCat_rk3588_deb
ian_gnome_defconfig
#
# configuration written to /home/jiawen/LINUX_SDK_RELEASE/output/.config
#
Using preferred kernel version(5.10)
```

也可以直接设置 SDK 配置文件，方法如下

```
# 选择构建 LubanCat-RK3588 系列板卡 debian 系统 gnome 桌面镜像
./build.sh LubanCat_rk3588_debian_gnome_defconfig
```

```
jiawen@dev120:~/LINUX_SDK_RELEASE$ ./build.sh LubanCat_rk3588_debian_gnome_defconfig

################ LubanCat Linux SDK ###############

Manifest: lubancat_linux_generic_20250115.xml

Log colors: message notice warning error fatal

Log saved at /home/jiawen/LINUX_SDK_RELEASE/output/sessions/2025-01-16_14-57-34
Switching to defconfig: /home/jiawen/LINUX_SDK_RELEASE/device/rockchip/.chip/LubanCat_rk3588_deb
ian_gnome_defconfig
#
# configuration written to /home/jiawen/LINUX_SDK_RELEASE/output/.config
#
Using preferred kernel version(5.10)
```

当选择完配置文件以后，会使用粉色提示字体提示当前配置文件所用的内核版本，如果配置文件中指定的内核版本与当前使用的内核 (kernel 目录) 版本不一致，则会自动切换内核版本。

```
# 括号中显示配置文件中指定的内核版本
# 指定使用 Kernel-5.10
Using preferred kernel version(5.10)
# 指定使用 Kernel-6.1
Using preferred kernel version(5.10)

# 当配置文件指定的内核版本和 SDK 中的不一致时自动切换内核版本。
# 将 SDK 使用的内核切换为 5.10
Switching to kernel-5.10
# 将 SDK 使用的内核切换为 6.1
Switching to kernel-6.1
```

```
jiawen@dev120:~/LINUX_SDK_RELEASE$ ./build.sh chip

################ LubanCat Linux SDK ################

Manifest: lubancat_linux_generic_20250115.xml

Log colors: message notice warning error fatal

Log saved at /home/jiawen/LINUX_SDK_RELEASE/output/sessions/2025-01-16_14-58-12
Pick a chip:

1. rk312x
2. rk3528
3. rk3562
4. rk3566_rk3568
5. rk3576
6. rk3588
Which would you like? [1]: 5
Switching to chip: rk3576
Pick a defconfig:

1. rockchip_defconfig
2. LubanCat_rk3576_debian_gnome_defconfig
3. LubanCat_rk3576_debian_lite_defconfig
4. LubanCat_rk3576_debian_xfce_defconfig
5. LubanCat_rk3576_ubuntu_gnome_defconfig
6. LubanCat_rk3576_ubuntu_lite_defconfig
7. rockchip_rk3576_evb1_v10_defconfig
8. rockchip_rk3576_industry_evb_v10_defconfig
9. rockchip_rk3576_iotest_v10_defconfig
10. rockchip_rk3576_ipc_evb1_v10_defconfig
11. rockchip_rk3576_multi_ipc_evb1_v10_defconfig
12. rockchip_rk3576_test1_v10_defconfig
13. rockchip_rk3576_test2_v10_defconfig
Which would you like? [1]: 1
Switching to defconfig: /home/jiawen/LINUX_SDK_RELEASE/device/rockchip/.chip/rockchip_defconfig
#
# configuration written to /home/jiawen/LINUX_SDK_RELEASE/output/.config
#
Using preferred kernel version(6.1)

Switching to kernel-6.1
```

选择 SDK 配置文件以后，还要安装 debian 根文件系统构建依赖的软件包，操作如下。

```
1 # 安装本地软件包
2 # 如果 SDK 中 debian 目录是具体的 debian11 或者 debian12，需要将下面命令中的 debian 目录路径替换为实际的路径。都有则选择较高版本的安装。
```

(下页继续)

(续上页)

```
3 sudo dpkg -i debian/ubuntu-build-service/packages/*
4 sudo apt-get install -f
```

```
dev@dev_152:~/LubanCat_SDK$     sudo dpkg -i debian/ubuntu-build-service/packages/*
(Reading database ... 160958 files and directories currently installed.)
Preparing to unpack .../debootstrap_1.0.87_all.deb ...
Unpacking debootstrap (1.0.87) over (1.0.87) ...
Preparing to unpack .../linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb ...
Unpacking linaro-image-tools (2012.12-0ubuntu1~linaro1) over (2012.12-0ubuntu1~linaro1) ...
Preparing to unpack .../live-build_3.0.5-1linaro1_all.deb ...
Unpacking live-build (3.0.5-1linaro1) over (3.0.5-1linaro1) ...
Preparing to unpack .../python-linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb ...
Unpacking python-linaro-image-tools (2012.12-0ubuntu1~linaro1) over (2012.12-0ubuntu1~linaro1) ...
Setting up debootstrap (1.0.87) ...
dpkg: dependency problems prevent configuration of linaro-image-tools:
 linaro-image-tools depends on python-debian (>= 0.1.16ubuntu1~); however:
  Package python-debian is not installed.
 linaro-image-tools depends on python-parted; however:
  Package python-parted is not installed.
 linaro-image-tools depends on python-yaml; however:
  Package python-yaml is not installed.

dpkg: error processing package linaro-image-tools (--install):
 dependency problems - leaving unconfigured
Setting up live-build (3.0.5-1linaro1) ...
dpkg: dependency problems prevent configuration of python-linaro-image-tools:
 python-linaro-image-tools depends on python-support (>= 0.90.0); however:
  Package python-support is not installed.

dpkg: error processing package python-linaro-image-tools (--install):
 dependency problems - leaving unconfigured
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Errors were encountered while processing:
 linaro-image-tools
 python-linaro-image-tools
dev@dev_152:~/LubanCat_SDK$     sudo apt-get install -f
Reading package lists... Done
Building dependency tree
Reading state information... Done
Correcting dependencies... Done
The following additional packages will be installed:
  python-chardet python-debian python-linaro-image-tools python-parted python-yaml
The following NEW packages will be installed:
```

注意： 由于 debian 和 ubuntu 根文件系统不同版本构建所需的依赖包版本不同，当切换了根

文件系统版本的选择以后，就需要根据自己所选的根文件系统版本安装不同的依赖包。

如果选择了 ubuntu 根文件系统的 SDK 配置文件，就需要安装 ubuntu 根文件系统构建的依赖软件包，操作如下。

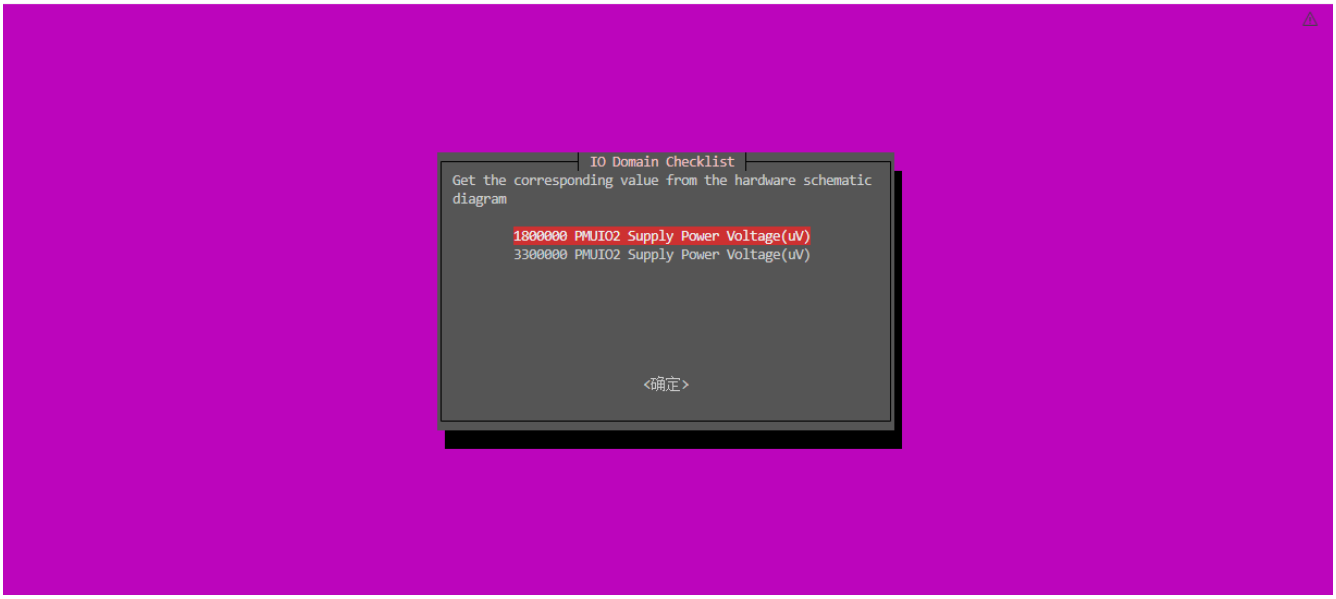
```
# 安装本地软件包
# 如果 SDK 中 ubuntu 目录是具体的 ubuntu20.04 或者 ubuntu22.04，需要将下面命令中的 ubuntu 目录路径替换为实际的路径。都有则选择较高版本的安装。
sudo dpkg -i ubuntu/ubuntu-build-service/packages/*
sudo apt-get install -f
```

安装过程中可能会报错，这是正常现象，忽略即可。待软件包安装完成后，就可以进行一键构建了。

```
# 一键编译 u-Boot，kernel，Rootfs 并打包为 update.img 镜像
./build.sh
```

```
jiawen@dev120:~/LubanCat_SDK$ ./build.sh

############### LubanCat Linux SDK ###############

Manifest: lubancat_linux_generic_20240910.xml

Log colors: message notice warning error fatal

Log saved at /home/jiawen/LubanCat_SDK/output/sessions/2024-09-13_17-35-33

=========================================
          Final configs
=========================================
RK_BOOT_FIT_ITS=/home/jiawen/LubanCat_SDK/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_NUMBER=debian11
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=/home/jiawen/LubanCat_SDK/device/rockchip/.chips/rk3588/LubanCat_rk3588_debian_gn
ome_defconfig
RK_EXTRA_PARTITION_NUM=0
RK_EXTRA_PARTITION_STR=@@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=lubancat_linux_rk3588_defconfig
RK_KERNEL_DTB=kernel/arch/arm64/boot/dts/rockchip/rk3588-lubancat-generic.dtb
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588-lubancat-generic.dts
RK_KERNEL_EXTBOOT=y
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_KBUILD_ARCH=host
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_REAL=5.10
RK_LOADER=y
RK_OWNER=jiawen
RK_OWNER_UID=1001
```

警告：如果在编译过程中出现如下图所示的提示，请在 SDK 中的 output/final.env 文件中搜索 **RK_KERNEL_DTS** 获取指定的设备树路径，并根据对应设备树文件中的 &pmu_io_domains 节点中的电压值进行选择。顺序从 pmuio2-supply 开始，其中 vccio_acodec 为 3v3、vccio_sd 为 3v3。



构建好的镜像保存在 rockdev/目录下，可以继续使用 **./build.sh release** 命令在 output-release/目录下进行镜像压缩归档。

3.7 LubanCat_Gen_SDK 分步构建

在进行固件开发时，一键构建就显得过于耗时了，每一个改动都要构建整个镜像并重新打包，这无疑是巨大的时间浪费。此时可以使用 SDK 单个模块构建的功能。

3.7.1 选择 SDK 配置文件

首先，还是要选择 SDK 的配置文件，如果已经选择过了则这一步可以跳过，这里以 LubanCat-RK3588 系列板卡的 debian 系统 gnome 桌面镜像为例

```
# 选择 SDK 配置文件
./build.sh LubanCat_rk3588_debian_gnome_defconfig
```

3.7.2 U-Boot 构建

```
./build.sh uboot
```

构建生成的 U-boot 镜像为 u-boot/uboot.img

3.7.3 Kernel 构建

extboot 分区内核镜像，要先生成内核 deb 包，然后再编译内核并将生成的 deb 包打包进 extboot 分区。

按顺序执行以下命令，将自动完成 kernel 的构建及打包。

```
./build.sh kerneldeb

./build.sh extboot
```

构建生成的 kernel 镜像为 kernel/extboot.img

3.7.4 rootfs 构建

LubanCat 主要支持 Ubuntu、Debian、OpenWrt 这几种 rootfs，不同 rootfs 的构建过程不同，这里分开说明。

注解： 由于 OpenWrt 系统的构建不使用 LubanCat_Gen_SDK，所以不在此处说明，具体构建过程查看 LubanCatWRT 构建说明

3.7.4.1 Debian

目前提供 debian11/debian12 的根文件系统构建脚本，Kernel-5.10 搭配 debian11，Kernel-6.1 则搭配 debian12。具体的对应情况在 SDK 配置文件中已经配置好，用户无需特殊关心。

首先要确保 SDK 的配置文件与要构建的 rootfs 一致，如果当前配置文件与要构建的 rootfs 不一致，需要先切换配置文件。

```
# 选择 SDK 配置文件
./build.sh LubanCat_rk3588_debian_gnome_defconfig

# 构建 Debian
./build.sh debian
```

生成的根文件系统镜像的命名规则是 linaro-(SOC 型号)-(桌面版本)-rootfs.img，保存在对应的 debian11 或 debian12 目录下。

刚刚构建生成的 rk3588 的 rootfs 镜像为 debian11/linaro-rk3588-gnome-rootfs.img，这个镜像还会被软链接到 rockdev/rootfs.img 用于系统镜像打包。

注意： 为了加速 SDK 构建效率，只有不存在已经构建好的根文件系统镜像时才会重新编译根文件系统。如果要重新构建根文件系统，需要先手动删除根文件系统镜像文件。以 rk3588 为例，我们需要删除 debian/linaro-rk3588-gnome-rootfs.img 后重新构建镜像时才会重新构建 Debian 根文件系统。

由于从头构建根文件系统会从网络获取很多文件，并且耗费很多时间。不需要自行构建根文件系统的用户可以将发布的系统镜像中的 rootfs 分区提取出来直接使用。

rootfs 分区提取需要使用解包工具，使用方法请查看本文档 **完整镜像的解包和打包**章节。

提取出 rootfs.img 镜像以后，将 rootfs.img 镜像移动到 SDK 目录下与配置文件对应的 debian11 或 debian12 目录中，并将解包的系统镜像的名称重命名为 linaro-(SOC 型号)-(桌面版本)-rootfs.img，如 linaro-rk3588-gnome-rootfs.img。

注意： rk3566 和 rk3568 共用同一个根文件系统镜像，SOC 型号需要设置为 rk356x，如 linaro-rk356x-xfce-rootfs.img

重命名完成后，重新选择与根文件系统镜像一致的配置文件进行编译

```
# 选择 SDK 配置文件
./build.sh LubanCat_rk3588_debian_gnome_defconfig

# 构建 Debian
./build.sh debian

# 返回的提示信息：跳过镜像构建，删除 rootfs 镜像后以便重新构建 Debian 根文件系统镜像
[    Already Exists IMG,  Skip Make Debian Scripts    ]
[ Delate linaro-rk3588-gnome-rootfs.img To Rebuild Debian IMG ]
```

如果返回的信息与文中一致，则使用解包后的根文件系统镜像成功

3.7.4.2 Ubuntu

目前提供 20.04/22.04 的根文件系统构建脚本，Kernel-5.10 搭配 Ubuntu20.04，Kernel-6.1 则搭配 Ubuntu22.04。具体的对应情况在 SDK 配置文件中已经配置好，用户无需特殊关心。

首先要确保 SDK 的配置文件与要构建的 rootfs 一致，如果当前配置文件与要构建的 rootfs 不一致，需要先切换配置文件。

```
# 选择 SDK 配置文件
./build.sh LubanCat_rk3588_ubuntu_gnome_defconfig

# 构建 Ubuntu
./build.sh ubuntu
```

生成的根文件系统镜像的命名规则是 ubuntu-(SOC 型号)-(桌面版本)-rootfs.img，保存在对应的 ubuntu20.04 或 ubuntu22.04 目录下。

刚刚构建生成的 rk3588 的 rootfs 镜像为 ubuntu/ubuntu-rk3588-gnome-rootfs.img，这个镜像还会被软链接到 rockdev/rootfs.img 用于系统镜像打包。

注意： 为了加速 SDK 构建效率，只有不存在已经构建好的根文件系统镜像时才会重新编译根文件系统。如果要重新构建根文件系统，需要先手动删除根文件系统镜像文件。以 rk3588 为例，我们需要删除 ubuntu20.04/ubuntu-rk3588-gnome-rootfs.img 时后重新构建镜像时才会重新构建 Debian 根文件系统。

由于从头构建根文件系统会从网络获取很多文件，并且耗费很多时间。不需要自行构建根文件系统的用户可以将发布的系统镜像中的 rootfs 分区提取出来直接使用。

rootfs 分区提取需要使用解包工具，使用方法请查看本文档 **完整镜像的解包和打包**章节。

提取出 rootfs.img 镜像以后，将 rootfs.img 镜像移动到 SDK 目录下与配置文件对应的 ubuntu20.04 或 ubuntu22.04 目录中，并将解包的系统镜像的名称重命名为 ubuntu-(SOC 型号)-(桌面版本)-rootfs.img，如 ubuntu-rk3588-gnome-rootfs.img。

注意： rk3566 和 rk3568 共用同一个根文件系统镜像，SOC 型号需要设置为 rk356x，如 ubuntu-rk356x-xfce-rootfs.img

重命名完成后，重新选择与根文件系统镜像一致的配置文件进行编译

```
# 选择 SDK 配置文件
./build.sh LubanCat_rk3588_ubuntu_gnome_defconfig

# 构建 Debian
./build.sh ubuntu

# 返回的提示信息：跳过镜像构建，删除 rootfs 镜像后以便重新构建 Ubuntu 根文件系统镜像
[    Already Exists IMG,  Skip Make Ubuntu Scripts    ]
[ Delate ubuntu-rk3588-gnome-rootfs.img To Rebuild Ubuntu IMG ]
```

如果返回的信息与文中一致，则使用解包后的根文件系统镜像成功

3.7.5 镜像打包

当 u-boot，kernel，Rootfs 都构建完成以后，需要再执行./build.sh firmware 进行固件打包，主要是检查分区表文件是否存在，各个分区是否与分区表配置对应，并根据配置文件将所有的文件复制或链接到 rockdev/内。

为了方便镜像的发布，还可以将各个分立的分区打包成一个文件，打包好的文件就能用于烧录了。

```
# 固件打包
./build.sh firmware

# 生成 update.img
./build.sh updateimg
```

以 debian 系统为例，rockdev/目录下的文件如下所示

rockdev/	链接文件
boot.img	kernel/boot.img
MiniLoaderAll.bin	u-boot/rk3588_spl_loader_v1.17.113.bin
parameter.txt	device/rockchip/.chips/rk3588/parameter.txt
rootfs.img	debian/linaro-rk3588-gnome-rootfs.img
uboot.img	u-boot/uboot.img

3.8 ./build.sh 构建脚本

./build.sh 脚本是整个 SDK 功能的入口，SDK 配置文件的选择、修改；系统各部分配置文件的修改、编译；以及镜像打包等，都离不开./build.sh 脚本

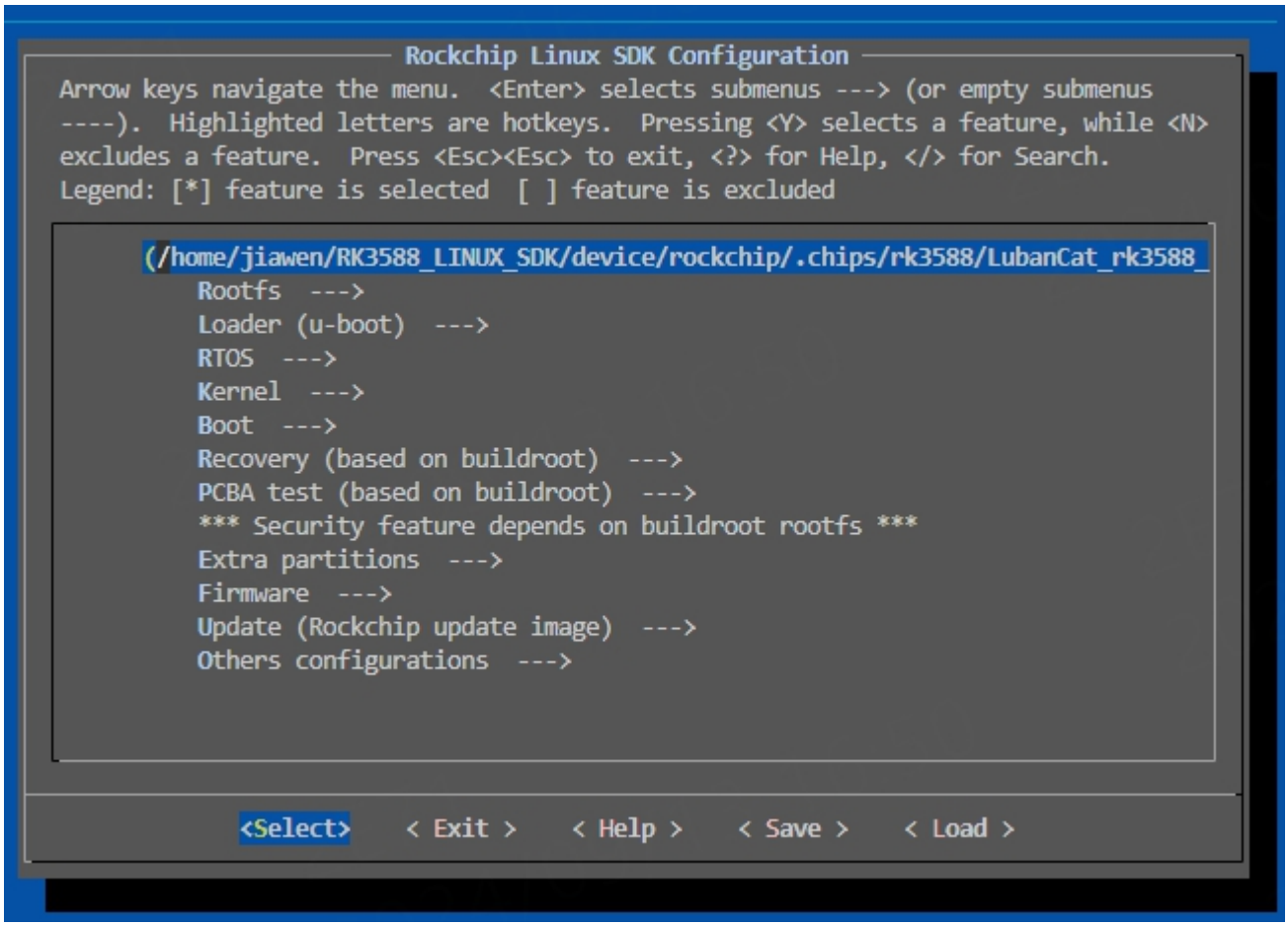
可以使用./build.sh help 命令查看所有支持的功能。

注意： 由于 SDK 是经二次开发，增删了部分功能，help 返回命令中，有些是不生效的，以实际情况为准。

```
jiawen@dev120:~/LubanCat_SDK$ ./build.sh help

############### LubanCat Linux SDK ###############

Manifest: lubancat_linux_generic_20240910.xml

Log colors: message notice warning error fatal

Usage: build.sh [OPTIONS]
Available options:
chip[:<chip>[:<config>]]                choose chip
defconfig[:<config>]                    choose defconfig
 *_defconfig                            switch to specified defconfig
    available defconfigs:
        LubanCat_rk3588_debian_gnome_defconfig
        LubanCat_rk3588_debian_lite_defconfig
        LubanCat_rk3588_ubuntu_gnome_defconfig
        LubanCat_rk3588_ubuntu_lite_defconfig
        rockchip_defconfig
        rockchip_rk3588_evb1_lp4_v10_defconfig
        rockchip_rk3588_evb7_v11_defconfig
        rockchip_rk3588s_evb1_lp4x_v10_defconfig
 olddefconfig                           resolve any unresolved symbols in .config
 savedefconfig                          save current config to defconfig
 menuconfig                             interactive curses-based configurator
config                                  modify SDK defconfig
print-parts                             print partitions
list-parts                              alias of print-parts
mod-parts                               interactive partition table modify
edit-parts                              edit raw partitions
new-parts:<offset>:<name>:<size>...     re-create partitions
insert-part:<idx>:<name>[:<size>]       insert partition
del-part:(<idx>|<name>)                 delete partition
move-part:(<idx>|<name>):<idx>          move partition
rename-part:(<idx>|<name>):<name>       rename partition
resize-part:(<idx>|<name>):<size>       resize partition
misc                                    pack misc image
kernel[:cmds]                           build kernel
kerneldeb                               build kernel debian package
extboot                                 build kernel extboot image
recovery-kernel[:cmds]                  build kernel for recovery
modules[:cmds]                          build kernel modules
linux-headers[:cmds]                    build linux-headers
kernel-config[:cmds]                    modify kernel defconfig
kconfig[:cmds]                          alias of kernel-config
kernel-make[:<arg1>:<arg2>]             run kernel make
kmake[:<arg1>:<arg2>]                   alias of kernel-make
```

3.8.1 SDK 配置文件修改

可以使用./build.sh config 命令来修改 SDK 配置文件



./build.sh savedefconfig 命令可以将最小 SDK 配置文件保存在选择的配置文件里

在 SDK 配置中，可以对 rootfs、loader、kernel、分区表、扩展分区、镜像打包等各部分做详细配置，不过一般情况下直接使用已经保存好的配置文件即可，修改可能会导致编译失败。

3.8.2 内核相关命令

使用./build.sh kernel-config 或./build.sh kconfig 可以直接打开内核配置界面。

当我们退出配置页面后，又会将对内核的修改同步到我们在 SDK 配置文件中选中的内核配置文件。

第 4 章　LubanCat_Chip_SDK

注解： LubanCat_Chip_SDK 将于 2024 年 9 月 10 日起逐步停止支持，rk356x 板卡和 rk3588 板卡后续开发支持已经转移到 LubanCat_Gen_SDK。

注解： LubanCat 专用镜像已于 2023 年 4 月 11 日停止支持，部分与专用镜像相关的内容并未删除，这些内容仅供参考，同时本文档在相关位置也会添加 **已停止支持**的标记

注解： LubanCat Buildroot 相关内容已于 2023 年 8 月 3 日停止支持，部分与 Buildroot 构建相关的内容并未删除，这些内容仅供参考，同时本文档在相关位置也会添加 **已停止支持**的标记

4.1 简介

LubanCat_Chip_SDK 是基于瑞芯微通用 Linux SDK 工程的深度定制版本，适用于以 Rockchip 处理器为主芯片的 LubanCat-RK 系列板卡。

整个 SDK 工程，目录包含有 buildroot、debian、app、kernel、u-boot、device、external 等目录。每个目录或其子目录会对应一个 git 工程，提交需要在各自的目录下进行。

- app：(**已停止支持**) 存放上层应用 app，主要是 qcamera/qfm/qplayer/settings 等一些应用程序，在 buildroot 构建时会使用到这些程序。
- buildroot：(**已停止支持**) 基于 buildroot (2018.02-rc3) 开发的根文件系统。
- debian：Debian 根文件系统构建脚本。

- device/rockchip：存放各芯片板级配置和 Parameter 文件，以及一些构建与打包固件的脚本和预备文件。
- external：(**已停止支持**) 存放第三方相关仓库, 包括音频、视频、网络、recovery 等，在 buildroot 构建时将会用到。
- kernel：存放 kernel 源码。
- prebuilts：存放交叉编译工具链。
- rkbin：存放 Rockchip 相关的 Binary 和工具。
- rockdev：存放编译输出固件。
- tools：存放 Linux 和 Windows 操作系统环境下常用工具。
- u-boot：存放基于 v2017.09 版本进行开发的 uboot 代码。
- ubuntu: Ubuntu 根文件系统构建脚本

4.2 extboot 与 rkboot 分区对比

注解： 由于基于 rkboot 分区的 LubanCat 专用镜像已停止支持，相关 SDK 配置文件已移动至 device/rockchip/rk356x/.others 目录下，以下相关内容仅做参考。

4.2.1 extboot 分区系统镜像特点

extboot 分区系统是野火基于瑞芯微 Linux_SDK 框架搭建的一种 LubanCat-RK 系列板卡通用镜像实现方式。可以实现一个镜像烧录到 LubanCat 使用同一型号处理器的所有板卡，解决了默认 rkboot 分区方式设备树固定，导致一个镜像只能适配一款板卡的问题，大大降低了由于型号众多导致的后期维护的复杂性。

extboot 分区使用 ext4 文件系统格式，在编译过程中将所有 LubanCat-RK 系列板卡设备树都编译并打包到分区内，并借助 SDRADC 读取板卡硬件 ID，来实现设备树自动切换。同时支持设备树插件，自动更新内核 deb 包，在线更新内核和驱动模块等功能。

还对系统镜像的分区做了调整，删减一些冗余功能，仅包含 uboot、boot、rootfs 分区，实现了对系统存储的高效利用。

此版本也是主要支持的版本。

4.2.2 rkboot 分区系统镜像特点（已停止支持）

rkboot 分区是使用瑞芯微 Linux_SDK 自带的 boot 分区打包脚本所构建的分区，仅修改了 kernel 配置文件和适配板卡设备树，系统更加原汁原味。使用此类型 boot 分区的优点是启动快，冗余功能强，进一步配置可以实现 OTA，多系统，镜像校验等功能 (需二次开发自行适配)，但是像设备树插件这些功能是缺失的。

rkboot 分区的系统镜像，一个型号的板卡要对应一个型号的系统镜像，如果硬件发生改变要修改设备树，就要重新构建镜像。

4.2.3 对比

功能	extboot	rkboot
启动速度	稍慢	快
使用相同处理器板卡镜像通用	支持	不支持
系统内切换设备树	支持	不支持
设备树插件	支持	不支持
存储空间利用率	较高	较低
在线升级内核	支持	不支持

对于两种类型分区的系统镜像，其 uboot 做了通用处理，rootfs 也做了通用处理，所以 uboot.img 和 rootfs.img 是可以通用的。

4.3 SDK 开发环境搭建

LubanCat_Chip_SDK 是基于 Ubuntu LTS 系统开发测试的，在开发过程中，主要是用 Ubuntu 20.04 版本，为了不必要的麻烦，推荐用户使用 Ubuntu20.04 及以上版本。

注解： LubanCat-RK3588 系列 SDK 不支持 Ubuntu20.04 以下版本环境进行镜像构建

安装 SDK 依赖的软件包

```
# 安装 SDK 构建所需要的软件包
# 整体复制下面内容到终端中安装
sudo apt install git ssh make gcc libssl-dev liblz4-tool u-boot-tools curl \
expect g++ patchelf chrpath gawk texinfo chrpath diffstat binfmt-support \
qemu-user-static live-build bison flex fakeroot cmake gcc-multilib g++-
↪multilib \
unzip device-tree-compiler python3-pip libncurses5-dev python3-pyelftools␣
↪dpkg-dev
```

4.4 安装 repo

```
mkdir ~/bin
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
# 如果上面的地址无法访问，可以用下面的：
# curl -sSL  'https://gerrit-googlesource.proxy.ustclug.org/git-repo/+/
↪master/repo?format=TEXT' |base64 -d > ~/bin/repo
chmod a+x ~/bin/repo
echo PATH=~/bin:$PATH >> ~/.bashrc
source ~/.bashrc
```

执行完上面的命令后来验证 repo 是否安装成功能正常运行。

```
repo --version

# 返回以下信息
# 返回的信息根据 Ubuntu 版本的不同略有差异
<repo not installed>
repo launcher version 2.32
    (from /home/he/bin/repo)
git 2.25.1
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0]
OS Linux 5.15.0-60-generic (#66~20.04.1-Ubuntu SMP Wed Jan 25 09:41:30 UTC␣
↪2023)
CPU x86_64 (x86_64)
Bug reports: https://bugs.chromium.org/p/gerrit/issues/entry?
↪template=Repo+tool+issue
```

4.5 SDK 源码获取

LubanCat_Chip_SDK 的代码被划分为了若干 git 仓库分别进行版本管理，可以使用 repo 工具对这些 git 仓库进行统一的下载，提交，切换分支等操作。

运行以下命令，将在当前用户的家目录下创建一个名为 LubanCat_SDK 的目录，用来放入 SDK 源码。

```
mkdir ~/LubanCat_SDK
```

4.5.1 切换 Python 3 版本

```
# 查看当前 Python 版本
python -V
```

若返回的版本号为 Python3 版本，则无需再切换 Python 版本。若为 Python2 版本或未发现 python，则可以用以下方式切换：

```
# 查看当前系统安装的 Python 版本有哪些
ls /usr/bin/python*

# 将 python 链接到 python3
sudo ln -sf /usr/bin/python3 /usr/bin/python

# 重新查看默认 Python 版本
python -V
```

此时系统默认 Python 版本切换为 python3

```
dev@dev_152:~/LINUX$ python -V
Python 2.7.17
dev@dev_152:~/LINUX$ ls /usr/bin/python*
/usr/bin/python            /usr/bin/python3.6            /usr/bin/python3m
/usr/bin/python2           /usr/bin/python3.6m           /usr/bin/python-config
/usr/bin/python2.7         /usr/bin/python3-jsondiff     /usr/bin/pythontex
/usr/bin/python2.7-config  /usr/bin/python3-jsonpatch    /usr/bin/pythontex3
/usr/bin/python2-config    /usr/bin/python3-jsonpointer
/usr/bin/python3           /usr/bin/python3-jsonschema
dev@dev_152:~/LINUX$ sudo ln -sf /usr/bin/python3 /usr/bin/python
dev@dev_152:~/LINUX$ python -V
Python 3.6.9
dev@dev_152:~/LINUX$
```

4.5.2 Git 配置

设置自己的 git 信息，以确保后续拉取代码时正常进行，如果不需要提交代码的话可以随意设置用户名和邮箱地址。

```
git config --global user.name "your name"
git config --global user.email "your mail"
```

4.5.3 SDK 在线下载并同步

LubanCat_Chip_SDK 可以适配不同的板卡。由于使用这种方式要从 Github 拉取大量的仓库，体积很大，对于不能快速访问 Github 的用户不建议使用这种方式。

```
cd ~/LubanCat_SDK

# 拉取 LubanCat-RK356x 系列 Linux_SDK
repo init -u https://github.com/LubanCat/manifests.git -b linux -m rk356x_
↪linux_release.xml


# 拉取 LubanCat-RK3588 系列 Linux_SDK
repo init -u https://github.com/LubanCat/manifests.git -b linux -m rk3588_
↪linux_release.xml

# 如果运行以上命令失败，提示：fatal: Cannot get https://gerrit.googlesource.com/
↪git-repo/clone.bundle
# 则可以在以上命令中添加选项 --repo-url https://mirrors.tuna.tsinghua.edu.cn/
↪git/git-repo

.repo/repo/repo sync -c -j4
```

```
dev@dev_152:~/LubanCat_SDK$ repo init -u https://github.com/LubanCat/manifests.git -b linux -m rk356
x_linux_release.xml
warning: Python 3 support is currently experimental. YMMV.
Please use Python 2.6 - 2.7 instead.

... A new version of repo (2.29) is available.
... New version is available at: /home/dev/LubanCat_SDK/.repo/repo/repo
... The launcher is run from: /usr/bin/repo
!!! The launcher is not writable.  Please talk to your sysadmin or distro
!!! to get an update installed.


Your identity is: 贺嘉文 <hjw0415@outlook.com>
If you want to change this, please re-run 'repo init' with --config-name

repo has been initialized in /home/dev/LubanCat_SDK
```

如果同步失败可以重新运行 sync 命令来同步

```
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c -j4
Fetching: 58% (17/29) u-boot
remote: Enumerating objects: 1708, done.
remote: Counting objects:   0% (1/300)
remote: Counting objects:   1% (3/300)
remote: Counting objects:   2% (6/300)
remote: Counting objects:   3% (9/300)
remote: Counting objects:   4% (12/300)
remote: Counting objects:   5% (15/300)
remote: Counting objects:   6% (18/300)
remote: Counting objects:   7% (21/300)
remote: Counting objects:   8% (24/300)
remote: Counting objects:   9% (27/300)
remote: Counting objects:  10% (30/300)
remote: Counting objects:  11% (33/300)
remote: Counting objects:  12% (36/300)
remote: Counting objects:  13% (39/300)
remote: Counting objects:  14% (42/300)
```

```
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c -j4
Fetching: 86% (25/29) camera_engine_rkaiq
Fetching: 100% (29/29), done in 28m49.416s
NOT Garbage collecting:   0% (0/29), done in 0.010s
Checking out files: 100% (17367/17367), done.
Checking out files: 100% (696/696), done.
Checking out files: 100% (1196/1196), done.
Checking out files: 100% (1487/1487), done.
Checking out files: 100% (141/141), done.
Checking out files: 100% (960/960), done.
Checking out files: 100% (75371/75371), done.
Checking out files: 100% (17900/17900), done.
Checking out files: 100% (7165/7165), done.
Checking out files: 100% (237/237), done.
Checking out files: 100% (13583/13583), done.
Checking out: 100% (29/29), done in 44.538s
repo sync has finished successfully.
```

4.5.4 SDK 离线安装下载

由于 Github 服务器在国外，拉取这么多的仓库需要很多时间，还可能因为网络不畅通而导致下载失败。为此，将需要使用的仓库整体打包，使用网盘下载的方式，以减少连接 Github 导致的问题。

4.5.4.1 下载地址

访问百度网盘资源介绍页面获取 SDK 源码压缩包: 8-SDK 源码压缩包

根据鲁班猫板卡的型号下载对应版本的最近日期的压缩包即可。

注解： 在开发过程中，当 SDK 源码进入 Release 版本或有重大 Bug 时，将会更新源码压缩包。源码压缩包解压到本地后，当有 Release 版本更新时，可以借助 Github 更新到最新版本。

4.5.4.2 解压源码

以下过程以 LubanCat_Linux_rk356x_SDK 进行演示，实际文件名称以自己下载的 SDK 为准

```
# 安装 tar 压缩工具，一般来说系统默认安装了
sudo apt install tar

# 在用户家目录创建 LubanCat_SDK 目录
mkdir ~/LubanCat_SDK

# 将下载的 SDK 源码移动到 LubanCat_SDK 目录下，xxx 为日期
mv LubanCat_Linux_rk356x_SDK_xxx.tgz ~/LubanCat_SDK

# 进入 LubanCat_SDK 目录
cd ~/LubanCat_SDK

# 解压 SDK 压缩包
tar -xzvf LubanCat_Linux_rk356x_SDK_xxx.tgz

# 查看解压后的文件，可以看到解压出.repo 文件夹
ls -al

# 检出.repo 目录下的 git 仓库
# 注意：下面的命令一点要在 SDK 顶层文件夹中执行，且 repo 路径一定为.repo/repo/repo
.repo/repo/repo sync -l

# 将所有的源码仓库同步到最新版本
.repo/repo/repo sync -c
```

如果.repo/repo/repo sync -c 执行时提示网络连接超时，请检查并能否通畅访问 github。确认可以正常访问 github 的话，可以重复多次执行.repo/repo/repo sync -c 命令来进行同步。若无法访问 github，可以忽略同步源码仓库到最新版本这一步骤。

```
dev@dev_152:~$ sudo apt install p7zip-full
Reading package lists... Done
Building dependency tree
Reading state information... Done
p7zip-full is already the newest version (16.02+dfsg-6).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
dev@dev_152:~$ mkdir ~/LubanCat_SDK
dev@dev_152:~$ mv LubanCat_Linux_SDK_20221130.7z ~/LubanCat_SDK
dev@dev_152:~$ cd ~/LubanCat_SDK
dev@dev_152:~/LubanCat_SDK$ ls
LubanCat_Linux_SDK_20221130.7z
dev@dev_152:~/LubanCat_SDK$ 7z x LubanCat_Linux_SDK_20221130.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,88 CPUs Intel(R) Xeo
n(R) CPU E5-2696 v4 @ 2.20GHz (406F1),ASM,AES-NI)

Scanning the drive for archives:
1 file, 4837528980 bytes (4614 MiB)

Extracting archive: LubanCat_Linux_SDK_20221130.7z
--
Path = LubanCat_Linux_SDK_20221130.7z
Type = 7z
Physical Size = 4837528980
Headers Size = 23593
Method = LZMA2:24
Solid = +
Blocks = 3

Everything is Ok

Folders: 1025
Files: 1446
Size:       5105741187
Compressed: 4837528980
dev@dev_152:~/LubanCat_SDK$ ls -a
.  ..  LubanCat_Linux_SDK_20221130.7z  .repo
dev@dev_152:~/LubanCat_SDK$
```

解压完成后 checkout 到指定的提交。

```
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -l
Checking out files: 100% (17367/17367), done.
Checking out files: 100% (696/696), done.
Checking out files: 100% (1198/1198), done.
Checking out files: 100% (1487/1487), done.
Checking out files: 100% (73/73), done.
Checking out files: 100% (960/960), done.
Checking out files: 100% (75373/75373), done.
Checking out files: 100% (17900/17900), done.
Checking out files: 100% (7165/7165), done.
Checking out files: 100% (237/237), done.
Checking out: 100% (29/29), done in 31.111s
repo sync has finished successfully.
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c
remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
Fetching: 100% (29/29), done in 0.824s
NOT Garbage collecting:   0% (0/29), done in 0.001s
repo sync has finished successfully.
dev@dev_152:~/LubanCat_SDK$
```

一般情况下网盘保存的离线源码包已经是最新版本，如果距离离线源码包下载时间不久，可以忽略从 Github 在线更新这一步。

4.5.5 SDK 更新

我们会对 LubanCat_Chip_SDK 不断更新，并将修改的内容实时同步到 Github，如果需要在本地 LubanCat_Chip_SDK 同步更新内容，则可以借助 repo 或 git 来实现。

4.5.5.1 使用 repo 更新整个 SDK

使用 repo 可以将 SDK 整体更新到提供的最新版本。

> **警告：** 此操作会把各个子 Git 仓库更新到此 SDK 各组件互相匹配的最新版本，虽然单个仓库可能不是最新，但各组件匹配性好，工作稳定

首先要更新.repo/manifests，里面保存了 repo 的配置文件，记录了仓库的版本信息。

```
# 进入.repo/manifests 目录
cd .repo/manifests

# 切换分支到 Linux
git checkout linux

# 拉取最新的 manifests
git pull

# 返回 SDK 顶层文件夹
cd ~/LubanCat_SDK

# 同步远端仓库
.repo/repo/repo sync -c
```

```
dev@dev_152:~/LubanCat_SDK$ cd .repo/manifests
dev@dev_152:~/LubanCat_SDK/.repo/manifests$ git pull
Updating eb35cbe..e3f0107
Fast-forward
 README.md                          | 11 +++++------
 rk356x_linux/rk356x_linux_dev.xml | 10 +++++-----
 2 files changed, 10 insertions(+), 11 deletions(-)
dev@dev_152:~/LubanCat_SDK/.repo/manifests$ cd ~/LubanCat_SDK
dev@dev_152:~/LubanCat_SDK$ .repo/repo/repo sync -c --no-tags
Fetching: 100% (29/29), done in 0.662s
NOT Garbage collecting:   0% (0/29), done in 0.001s
kernel/: manifest switched refs/heads/stable-4.19-rk356x...27fc0130643a8be83f86c80c5d4fced232c7a694
kernel/: discarding 4 commits removed from upstream
project kernel/
First, rewinding head to replay your work on top of it...
Applying: kernel-android-confifg:refresh kernel configuration
Applying: add JL2101 rx/tx delayline
Applying: kernel:lubancat2 fix ir infrared and headphone unplugging problem
Applying: dts 添加LubanCat2N mipi和双屏显示设备树

project kernel/

Checking out: 100% (29/29), done in 1.520s
repo sync has finished successfully.
dev@dev_152:~/LubanCat_SDK$
```

4.5.5.2 使用 Git 更新单独的源码仓库

有时只想更新某个仓库，而不是去更新整个 SDK。或者已经对 SDK 的某些仓库做出了修改，使用 repo 同步的话就会失败。此时就需要对单个仓库进行更新了。

> **警告：** 此操作会单独更新 SDK 子 Git 仓库，会将指定的仓库更新到最新版本，但可能打破各组件之间的依赖关系，降低 SDK 稳定性

为了减少 SDK 传输的文件大小，默认在 SDK 下载时只同步 Git 仓库指定的单次提交，默认处于无分支状态，如果要使用 Git 更新单独的源码仓库，则需要先将对应仓库切换到指定分支。

以下是 LubanCat_Linux_rk356x_SDK 常用仓库的分支说明：

仓库路径	默认分支
debian	stable-4.19-rk356x
kernel	stable-4.19-rk356x
ubuntu	stable-4.19-rk356x

以下是 LubanCat_Linux_rk3588_SDK 常用仓库的分支说明：

仓库路径	默认分支
debian	stable-5.10-rk3588
kernel	stable-5.10-rk3588
ubuntu	stable-5.10-rk3588

如果要更新的仓库分支在此处没有列出，可以查看 **.repo/manifests/rk356x_linux_release.xml** 文件中各条目中的 dest-branch 就是要切换的分支

这里以 rk356x Kernel 仓库为例

```
# 进入 kernel 目录下
cd kernel
```

(下页继续)

(续上页)

```
3
4  # 检出到当前提交所在的分支
5  git checkout stable-4.19-rk356x
6
7  # 拉取 git 仓库
8  git pull
```

4.6 LubanCat_Chip_SDK 自动构建

LubanCat 板卡对应的配置文件列表:

镜像说明	boot 分区类似	文件系统	对应 BoardConfig 配置文件
LubanCat-RK3566 系列通用镜像	extboot 分区	debian	BoardConfig-LubanCat-RK3566-debian-(版本).mk
LubanCat-RK3566 系列通用镜像	extboot 分区	ubuntu	BoardConfig-LubanCat-RK3566-ubuntu-(版本).mk
LubanCat-RK3568 系列通用镜像	extboot 分区	debian	BoardConfig-LubanCat-RK3568-debian-(版本).mk
LubanCat-RK3568 系列通用镜像	extboot 分区	ubuntu	BoardConfig-LubanCat-RK3568-ubuntu-(版本).mk

- LubanCat-RK3566 通用镜像适用于 LubanCat-1 系列和 LubanCat-0 系列全部板卡
- LubanCat-RK3568 通用镜像适用于 LubanCat-2 系列全部板卡

LubanCat_Chip_SDK 的构建脚本可以实现自动构建，具体操作方式如下:

在 SDK 顶层文件夹下，执行以下命令，以选择 SDK 配置文件，来选择要构建的板卡和文件系统类型。

```
# 选择 SDK 配置文件
./build.sh lunch

# 输入想要构建的板卡及文件系统配置文件编号，并确认，这里选择配置文件 BoardConfig-LubanCat-RK3566-debian-xfce.mk。
Which would you like? [0]: 4
```

也可以直接设置 SDK 配置文件，方法如下

```
# 选择构建 LubanCat-RK3566 系列板卡通用 debian 镜像
./build.sh BoardConfig-LubanCat-RK3566-debian-xfce.mk
```

```
jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh lunch
processing option: lunch

You're building on Linux
Lunch menu...pick a combo:

0. default BoardConfig.mk
1. BoardConfig-LubanCat-Backup.mk
2. BoardConfig-LubanCat-RK3566-debian-lite.mk
3. BoardConfig-LubanCat-RK3566-debian-xfce-full.mk
4. BoardConfig-LubanCat-RK3566-debian-xfce.mk
5. BoardConfig-LubanCat-RK3566-ubuntu-lite.mk
6. BoardConfig-LubanCat-RK3566-ubuntu-xfce-full.mk
7. BoardConfig-LubanCat-RK3566-ubuntu-xfce.mk
8. BoardConfig-LubanCat-RK3568-debian-lite.mk
9. BoardConfig-LubanCat-RK3568-debian-xfce-full.mk
10. BoardConfig-LubanCat-RK3568-debian-xfce.mk
11. BoardConfig-LubanCat-RK3568-ubuntu-lite.mk
12. BoardConfig-LubanCat-RK3568-ubuntu-xfce-full.mk
13. BoardConfig-LubanCat-RK3568-ubuntu-xfce.mk
14. BoardConfig.mk
Which would you like? [0]: 4
switching to board: /home/jiawen/RK356X_LINUX_SDK/device/rockchip/rk356x/BoardConfig-LubanCat-RK3566-debian-xfce.mk
jiawen@dev120:~/RK356X_LINUX_SDK$ ./build.sh BoardConfig-LubanCat-RK3566-debian-xfce.mk
processing option: BoardConfig-LubanCat-RK3566-debian-xfce.mk
switching to board: /home/jiawen/RK356X_LINUX_SDK/device/rockchip/rk356x/BoardConfig-LubanCat-RK3566-debian-xfce.mk
jiawen@dev120:~/RK356X_LINUX_SDK$
```

选择 SDK 配置文件以后，还要安装 debian 根文件系统构建依赖的软件包，操作如下。

```
# 安装本地软件包
sudo dpkg -i debian/ubuntu-build-service/packages/*
sudo apt-get install -f
```

```
dev@dev_152:~/LubanCat_SDK$     sudo dpkg -i debian/ubuntu-build-service/packages/*
(Reading database ... 160958 files and directories currently installed.)
Preparing to unpack .../debootstrap_1.0.87_all.deb ...
Unpacking debootstrap (1.0.87) over (1.0.87) ...
Preparing to unpack .../linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb ...
Unpacking linaro-image-tools (2012.12-0ubuntu1~linaro1) over (2012.12-0ubuntu1~linaro1) ...
Preparing to unpack .../live-build_3.0.5-1linaro1_all.deb ...
Unpacking live-build (3.0.5-1linaro1) over (3.0.5-1linaro1) ...
Preparing to unpack .../python-linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb ...
Unpacking python-linaro-image-tools (2012.12-0ubuntu1~linaro1) over (2012.12-0ubuntu1~linaro1) ...
Setting up debootstrap (1.0.87) ...
dpkg: dependency problems prevent configuration of linaro-image-tools:
 linaro-image-tools depends on python-debian (>= 0.1.16ubuntu1~); however:
  Package python-debian is not installed.
 linaro-image-tools depends on python-parted; however:
  Package python-parted is not installed.
 linaro-image-tools depends on python-yaml; however:
  Package python-yaml is not installed.

dpkg: error processing package linaro-image-tools (--install):
 dependency problems - leaving unconfigured
Setting up live-build (3.0.5-1linaro1) ...
dpkg: dependency problems prevent configuration of python-linaro-image-tools:
 python-linaro-image-tools depends on python-support (>= 0.90.0); however:
  Package python-support is not installed.

dpkg: error processing package python-linaro-image-tools (--install):
 dependency problems - leaving unconfigured
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Errors were encountered while processing:
 linaro-image-tools
 python-linaro-image-tools
dev@dev_152:~/LubanCat_SDK$     sudo apt-get install -f
Reading package lists... Done
Building dependency tree
Reading state information... Done
Correcting dependencies... Done
The following additional packages will be installed:
  python-chardet python-debian python-linaro-image-tools python-parted python-yaml
The following NEW packages will be installed:
```

注意： 由于 debian 和 ubuntu 根文件系统不同版本构建所需的依赖包版本不同，当切换了根文件系统版本的选择以后，就需要根据自己所选的根文件系统版本安装不同的依赖包。

如果选择了 ubuntu 根文件系统的 SDK 配置文件，就需要安装 ubuntu 根文件系统构建的依赖软件包，操作如下。

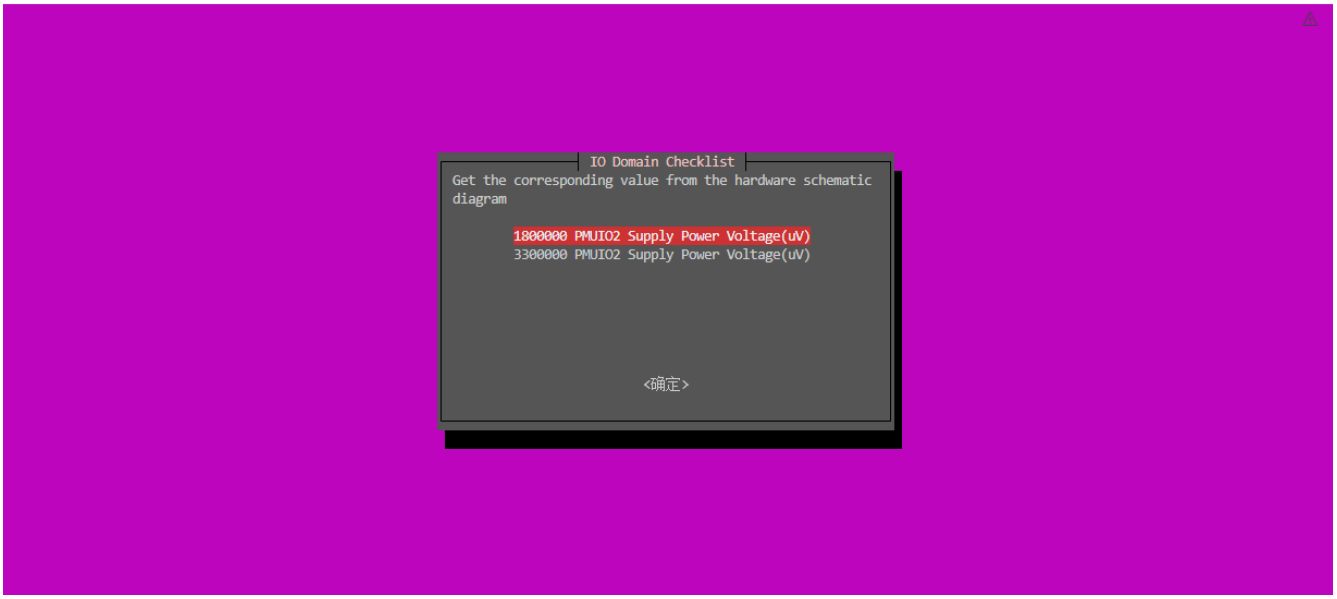
```
# 安装本地软件包
sudo dpkg -i ubuntu/ubuntu-build-service/packages/*
sudo apt-get install -f
```

安装过程中可能会报错，这是正常现象，忽略即可。待软件包安装完成后，就可以进行一键构建了。

```
# 一键编译 u-Boot, kernel, Rootfs 并打包为 update.img 镜像
./build.sh
```

```
dev@dev_152:~/LubanCat_SDK$  ./build.sh
processing option: allsave
============================================
TARGET_ARCH=arm64
TARGET_PLATFORM=rk356x
TARGET_UBOOT_CONFIG=rk3566
TARGET_SPL_CONFIG=
TARGET_KERNEL_CONFIG=lubancat2_defconfig
TARGET_KERNEL_DTS=rk356x-lubancat-rk_series
TARGET_TOOLCHAIN_CONFIG=
TARGET_BUILDROOT_CONFIG=rockchip_rk3566
TARGET_RECOVERY_CONFIG=
TARGET_PCBA_CONFIG=
TARGET_RAMBOOT_CONFIG=
============================================
============Start building uboot============
TARGET_UBOOT_CONFIG=rk3566
========================================
grep: .config: No such file or directory
## make  rk3568_defconfig rk3566.config -j176
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
```

警告： 如果在编译过程中出现如下图所示的提示，可以全部选择 3v3，如果选择完成后报错，可以根据报错信息找到对应设备树文件中的 &pmu_io_domains 节点中的电压值选择，并注意顺序从 pmuio2-supply 开始。其中 vccio_acodec 为 3v3、vccio_sd 为 3v3。



下图是选择错误后的报错信息，“**检查内核文件**”后是使用的设备树文件地址。

```
PLEASE CHECK BOARD GPIO POWER DOMAIN CONFIGURATION !!!!!
<<< ESPECIALLY Wi-Fi/Flash/Ethernet IO power domain >>> !!!!!
Check Node [pmu_io_domains] in the file: /home/jiawen/RK356X_LINUX_SDK/kernel/arch/arm64/boot/dts/rockchip/rk356x-lubancat-generic.dts

请再次确认板级的电源域配置！！！！！！
<<< 特别是Wi-Fi，FLASH，以太网这几路IO电源的配置 >>> ！！！！！
检查内核文件 /home/jiawen/RK356X_LINUX_SDK/kernel/arch/arm64/boot/dts/rockchip/rk356x-lubancat-generic.dts 的节点 [pmu_io_domains]
```

构建好的镜像保存在 rockdev/目录下，同时在也可以使用 **./build.sh save** 命令在 IMAGE/目录下进行备份。

4.7 LubanCat_Chip_SDK 分步构建

在进行固件开发时，一键构建就显得过于耗时了，每一个改动都要构建整个镜像并重新打包，这无疑是巨大的时间浪费。此时可以使用 SDK 单个模块构建的功能。

4.7.1 选择 SDK 配置文件

首先，还是要选择 SDK 的配置文件，这里以 LubanCat2 系列板卡的通用 debian 镜像为例

```
# 选择 SDK 配置文件
./build.sh BoardConfig-LubanCat-RK3568-debian-xfce.mk
```

4.7.2 U-Boot 构建

```
./build.sh uboot
```

构建生成的 U-boot 镜像为 u-boot/uboot.img

4.7.3 Kernel 构建

4.7.3.1 extboot 分区

extboot 分区内核镜像，使用的配置文件为 BoardConfig-LubanCat-RK356x-xxx.mk

要先生成内核 deb 包，然后再编译内核并将生成的 deb 包打包进 extboot 分区。

执行以下命令自动完成 kernel 的构建及打包。

```
./build.sh kerneldeb

./build.sh extboot
```

构建生成的 kernel 镜像为 kernel/extboot.img

4.7.4 rootfs 构建

LubanCat 主要支持 Ubuntu、Debian、OpenWrt 这几种 rootfs，不同 rootfs 的构建过程不同，这里分开说明。

注解： 由于 OpenWrt 系统的构建不使用 LubanCat_Chip_SDK，所以不在此处说明，具体构建过程查看 LubanCatWRT 构建说明

4.7.4.1 Debian

首先要确保 SDK 的配置文件与要构建的 rootfs 一致，如果当前配置文件与要构建的 rootfs 不一致，需要先切换配置文件。

```
# 选择 SDK 配置文件
./build.sh BoardConfig-LubanCat-RK3568-debian-xfce.mk

# 构建 Debian
./build.sh debian
```

构建生成的 rootfs 镜像为 debian/linaro-xfce-rootfs.img，同时被软链接到 rockdev/rootfs.ext4

注意： 只有不存在 debian/linaro-xfce-rootfs.img 时，才会重新构建 Debian 根文件系统。如果要重新构建 linaro-xfce-rootfs.img，需要先手动删除。

提示： 由于构建不同版本的 rootfs 很容易遇到环境依赖问题，推荐不需要重新构建 Debian 根文件系统的用户直接下载提供的定制根文件系统镜像或使用 Docker 构建，详情请查看 Debian 根文件系统构建独立章节

由于从头构建根文件系统会从网络获取很多文件，并且耗费很多时间。不需要自行构建根文件系

统的用户可以将发布的系统镜像中的 rootfs 分区提取出来直接使用。

rootfs 分区提取需要使用解包工具，使用方法请查看本文档 **完整镜像的解包和打包**章节。

提取出 rootfs.img 镜像以后，将 rootfs.img 镜像移动到 SDK 目录下的 debian 目录中，并根据解包的系统镜像的名称，重命名为 linaro-(桌面版本)-rootfs.img，如 linaro-xfce-rootfs.img。

重命名完成后，重新选择与根文件系统镜像一致的配置文件进行编译

```
# 选择 SDK 配置文件
./build.sh BoardConfig-LubanCat-RK3568-debian-xfce.mk

# 构建 Debian
./build.sh debian

# 返回的提示信息：跳过镜像构建，删除 rootfs 镜像后以便重新构建 Debian 根文件系统镜像
[     Already Exists IMG,  Skip Make Debian Scripts     ]
[ Delate linaro-xfce-rootfs.img To Rebuild Debian IMG ]
```

如果返回的信息与文中一致，则使用解包后的根文件系统镜像成功

4.7.4.2 Ubuntu

目前提供 Ubuntu18.04/20.04/22.04 的根文件系统构建脚本，默认使用 20.04 版本分支，如要构建其他版本，在构建前要先拉取构建源码仓库并切换到对应分支，具体操作请查看 Ubuntu 根文件系统构建独立章节。

注解： 推荐使用 Ubuntu20.04 版本，这是主要支持版本。

首先要确保 SDK 的配置文件与要构建的 rootfs 一致，如果当前配置文件与要构建的 rootfs 不一致，需要先切换配置文件。

```
# 选择 SDK 配置文件
./build.sh BoardConfig-LubanCat-RK3568-ubuntu-xfce.mk
```

(下页继续)

(续上页)

```

# 构建 Ubuntu
./build.sh ubuntu
```

构建生成的 rootfs 镜像为 ubuntu/ubuntu-xfce-rootfs.img，同时被软链接到 rockdev/rootfs.ext4

注意： 只有不存在 ubuntu/ubuntu-xfce-rootfs.img 时，才会重新构建 Ubuntu 根文件系统。如果要重新构建 ubuntu-xfce-rootfs.img，需要先手动删除。

提示： 由于构建不同版本的 rootfs 很容易遇到环境依赖问题，推荐不需要重新构建 Ubuntu 根文件系统的用户直接下载提供的定制根文件系统镜像或使用 Docker 构建，详情请查看 Ubuntu 根文件系统构建独立章节

由于从头构建根文件系统会从网络获取很多文件，并且耗费很多时间。不需要自行构建根文件系统的用户可以将发布的系统镜像中的 rootfs 分区提取出来直接使用。

rootfs 分区提取需要使用解包工具，使用方法请查看本文档 **完整镜像的解包和打包**章节。

提取出 rootfs.img 镜像以后，将 rootfs.img 镜像移动到 SDK 目录下的 ubuntu 目录中，并根据解包的系统镜像的名称，重命名为 ubuntu-(桌面版本)-rootfs.img，如 ubuntu-xfce-rootfs.img。

重命名完成后，重新选择与根文件系统镜像一致的配置文件进行编译

```
# 选择 SDK 配置文件
./build.sh BoardConfig-LubanCat-RK3568-ubuntu-xfce.mk

# 构建 Ubuntu
./build.sh ubuntu

# 返回的提示信息：跳过镜像构建，删除 rootfs 镜像后以便重新构建 Ubuntu 根文件系统镜像
[     Already Exists IMG,  Skip Make Ubuntu Scripts     ]
```

(下页继续)

(续上页)

```
[ Delate ubuntu-xfce-rootfs.img To Rebuild Ubuntu IMG ]
```

如果返回的信息与文中一致，则使用解包后的根文件系统镜像成功

4.7.5 镜像打包

当 u-Boot，kernel，Rootfs 都构建完成以后，需要再执行./mkfirmware.sh 进行固件打包，主要是检查分区表文件是否存在，各个分区是否与分区表配置对应，并根据配置文件将所有的文件复制或链接到 rockdev/内。

为了方便镜像的发布，还可以将各个分立的分区打包成一个文件，打包好的文件就能用于烧录了。

```
# 固件打包
./mkfirmware.sh

# 生成 update.img
./build.sh updateimg
```

以 debian 系统为例，rockdev/目录下的文件如下所示

rockdev/	链接文件
boot.img	kernel/boot.img
MiniLoaderAll.bin	u-boot/rk356x_spl_loader_v1.15.112.bin
parameter.txt	device/rockchip/rk356x/parameter-ubuntu-fit.txt
rootfs.ext4	debian/debian-xfce-rootfs.img
rootfs.img	debian/debian-xfce-rootfs.img
uboot.img	u-boot/uboot.img

4.8 SDK 配置文件说明

如果说./build.sh 脚本是肌肉，完成 SDK 各部分的构建工作，那 SDK 的配置文件则是大脑，控制着肌肉的运作方式。

虽然看起来 SDK 的配置文件条目繁多，但在使用中，需要去修改的并不多。

这里以 LubanCat-RK 系列配置文件为例，其他板卡的配置文件也类似

```
# Target arch
export RK_ARCH=arm64
# Uboot defconfig
export RK_UBOOT_DEFCONFIG=rk3566
# Uboot image format type: fit(flattened image tree)
export RK_UBOOT_FORMAT_TYPE=fit
# Kernel defconfig
export RK_KERNEL_DEFCONFIG=lubancat2_defconfig
# Kernel defconfig fragment
export RK_KERNEL_DEFCONFIG_FRAGMENT=
# Kernel dts
export RK_KERNEL_DTS=rk356x-lubancat-rk_series
# boot image type
export RK_BOOT_IMG=boot.img
# kernel image path
export RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
# kernel image format type: fit(flattened image tree)
export RK_KERNEL_FIT_ITS=boot.its
# parameter for GPT table
export RK_PARAMETER=parameter-ubuntu-fit.txt
# 分区表对应的打包文件
export RK_PACKAGE_FILE=rk356x-package-file-ubuntu
# Buildroot config
export RK_CFG_BUILDROOT=
# Recovery config
```

(下页继续)

(续上页)

```
export RK_CFG_RECOVERY=
# Recovery image format type: fit(flattened image tree)
export RK_RECOVERY_FIT_ITS=boot4recovery.its
# ramboot config
export RK_CFG_RAMBOOT=
# Pcba config
export RK_CFG_PCBA=
# Build jobs
export RK_JOBS=24
# target chip
export RK_TARGET_PRODUCT=rk356x
# Set rootfs type, including ext2 ext4 squashfs
export RK_ROOTFS_TYPE=ext4
# yocto machine
export RK_YOCTO_MACHINE=rockchip-rk3568-evb
# rootfs image path
export RK_ROOTFS_IMG=rockdev/rootfs.${RK_ROOTFS_TYPE}
# Set ramboot image type
export RK_RAMBOOT_TYPE=
# Set oem partition type, including ext2 squashfs
export RK_OEM_FS_TYPE=ext2
# Set userdata partition type, including ext2, fat
export RK_USERDATA_FS_TYPE=ext2
#OEM config
export RK_OEM_DIR=
# OEM build on buildroot
#export RK_OEM_BUILDIN_BUILDROOT=YES
#userdata config
export RK_USERDATA_DIR=
#misc image
export RK_MISC=
#choose enable distro module
```

(下页继续)

(续上页)

```
export RK_DISTRO_MODULE=
# Define pre-build script for this board
export RK_BOARD_PRE_BUILD_SCRIPT=app-build.sh

# SOC
export RK_SOC=rk356x
# build.sh save 打包时名称
export RK_PKG_NAME=lubancat-${RK_UBOOT_DEFCONFIG}

# 定义默认 rootfs 为 debian
export RK_ROOTFS_SYSTEM=debian
# 设置 debian 版本 (debian10: buster)
export RK_DEBIAN_VERSION=10
# 定义默认 rootfs 是否为桌面版  xfce : 桌面版       lite : 控制台版 xfce-full : 桌面版 + 推荐软件包
export RK_ROOTFS_TARGET=xfce
# 定义默认 rootfs 是否添加 DEBUG 工具  debug : 添加  none : 不添加
export RK_ROOTFS_DEBUG=debug

# 定义默认 rootfs 为 ubuntu
export RK_ROOTFS_SYSTEM=ubuntu
# 默认 Ubuntu 版本
export RK_UBUNTU_VERSION=20.04
# 定义默认 rootfs 是否为桌面版  xfce : 桌面版       lite : 控制台版 xfce-full : 桌面版 + 推荐软件包
export RK_ROOTFS_TARGET=xfce
# 定义默认 rootfs 是否添加 DEBUG 工具  debug : 添加  none : 不添加
export RK_ROOTFS_DEBUG=none

# 使用 exboot 内核分区
export RK_EXTBOOT=true
```

- RK_ARCH：定义处理器架构，无需修改

- RK_UBOOT_DEFCONFIG：Uboot 配置文件
- RK_KERNEL_DEFCONFIG：Kernel 配置文件
- RK_KERNEL_DEFCONFIG_FRAGMENT：Kernel 配置文件的叠加配置
- RK_KERNEL_DTS：内核设备树文件名
- RK_BOOT_IMG：boot 分区镜像名称，无需修改
- RK_KERNEL_IMG：内核镜像所在位置：无需修改
- RK_PARAMETER：分区表名称，无需修改
- RK_CFG_BUILDROOT：Buildroot 配置文件，无需修改
- RK_CFG_RECOVERY：revcovery 分区构建的配置文件，无需修改
- RK_JOBS：编译线程数，可根据电脑配置酌情修改
- RK_TARGET_PRODUCT：目标芯片，无需修改
- RK_ROOTFS_TYPE：rootfs 的格式，无需修改
- RK_ROOTFS_IMG：rootfs 镜像的位置，无需修改
- RK_ROOTFS_SYSTEM：rootfs 的类型，目前支持 ubuntu、debian，无需修改
- RK_DEBIAN_VERSION：debian 发行版本，暂不支持修改
- RK_UBUNTU_VERSION：Ubuntu 发行版本，暂不支持修改
- RK_ROOTFS_TARGET：rootfs 是否支持桌面显示，xfce: 桌面版 lite：控制台版 xfce-full: 桌面版 + 推荐软件包
- RK_ROOTFS_DEBUG：rootfs 是否添加 Debug 工具，debug 添加、none 不添加
- RK_EXTBOOT：是否使用 exboot 内核分区

第 5 章　U-boot 的介绍

5.1 U-boot 简介

U-boot 是从 FADSROM、8xxROM、PPCBOOT 逐步发展演化而来的。U-boot 发展至今，已经可以实现非常多的功能，在操作系统方面，它不仅支持嵌入式 Linux 系统的引导，还支持 NetBSD,VxWorks, QNX, RTEMS, ARTOS, LynxOS, Android 等嵌入式操作系统的引导。在 CPU 架构方面，U-boot 支持 PowerPC、MIPS、x86、ARM、NIOS、XScale 等诸多常用系列的处理器。

一般来说 BootLoader 必须提供系统上电时的初始化代码，在系统上电时初始化相关环境后，BootLoader 需要引导完整的操作系统，然后将控制器交给操作系统。简单来说 BootLoader 是一段小程序，它在系统上电时执行，通过这段小程序可以将硬件设备进行初始化，如 CPU、SDRAM、Flash、串口、网络等，初始化完毕后调用操作系统内核。

5.2 启动 U-boot

不同 LubanCat 板卡使用相同版本的 U-Boot 源码进行构建，仅少量配置不同，下述文档以使用 RK3568 处理器的鲁班猫板卡为例进行说明，使用其他型号处理器的鲁班猫板卡内容基本相同，仅在必要处指出不同点。

接下来我们以野火 LubanCat 板卡为例，使用 RockChip 提供的基于 U-boot v2017 深度定制的版本（称为 next-dev），介绍 U-boot 的使用，

将板卡 Debug 串口与电脑连接，在上电后，U-boot 启动 kernel 之前按下键盘的 ctrl+c 键：进入 U-Boot 命令行模式。如下所示

```
1 U-Boot 2017.09-gdde42ec6a2-211221 #jiawen (Jun 01 2022 - 14:07:06 +0800)
2
3 Model: Rockchip RK3568 Evaluation Board
4 PreSerial: 2, raw, 0xfe660000
```

（下页继续）

(续上页)

```
DRAM:  4 GiB
Sysmem: init
Relocation Offset: ed34b000
Relocation fdt: eb9f87e8 - eb9fecd0
CR: M/C/I
Using default environment

Hotkey: ctrl+c
dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
DM: v1
boot mode: None
FIT: no signed, no conf required
DTB: rk-kernel.dtb
HASH(c): OK
I2c0 speed: 100000Hz
vsel-gpios- not found! Error: -2
vdd_cpu 1025000 uV
PMIC:  RK8090 (on=0x40, off=0x00)
vdd_logic init 900000 uV
vdd_gpu init 900000 uV
vdd_npu init 900000 uV
io-domain: OK
Could not find baseparameter partition
Model: EmbedFire LubanCat2
Rockchip UBOOT DRM driver version: v1.0.1
VOP have 2 active VP
vp0 have layer nr:3[0 2 4 ], primary plane: 4
vp1 have layer nr:3[1 3 5 ], primary plane: 5
vp2 have layer nr:0[], primary plane: 0
```

(下页继续)

(续上页)

```
Using display timing dts
dsi@fe060000:  detailed mode clock 59603 kHz, flags[8000000a]
    H: 0720 0730 0736 0756
    V: 1280 1290 1294 1314
bus_format: 100e
VOP update mode to: 720x1280p0, type: MIPI0 for VP0
VOP VP0 enable Smart0[654x270->654x270@33x505] fmt[2] addr[0xedf04000]
final DSI-Link bandwidth: 396 Mbps x 4
xfer: num: 2, addr: 0x50
xfer: num: 2, addr: 0x50
Monitor has basic audio support
Could not find baseparameter partition
mode:1920x1080
hdmi@fe0a0000:  detailed mode clock 148500 kHz, flags[5]
    H: 1920 2008 2052 2200
    V: 1080 1084 1089 1125
bus_format: 2025
VOP update mode to: 1920x1080p0, type: HDMI0 for VP1
VOP VP1 enable Smart1[654x270->654x270@633x405] fmt[2] addr[0xedf04000]
CEA mode used vic=16
final pixclk = 148500000 tmdsclk = 148500000
PHY powered down in 0 iterations
PHY PLL locked 1 iterations
PHY powered down in 1 iterations
PHY PLL locked 1 iterations
sink has audio support
hdmi_set_clk_regenerator: fs=48000Hz ftdms=148.500MHz N=6144 cts=148500
CLK: (sync kernel. arm: enter 816000 KHz, init 816000 KHz, kernel 0N/A)
    apll 1416000 KHz
    dpll 780000 KHz
    gpll 1188000 KHz
    cpll 1000000 KHz
```

(下页继续)

(续上页)

```
    npll 1200000 KHz
    vpll 742000 KHz
    hpll 59000 KHz
    ppll 200000 KHz
    armclk 1416000 KHz
    aclk_bus 150000 KHz
    pclk_bus 100000 KHz
    aclk_top_high 500000 KHz
    aclk_top_low 400000 KHz
    hclk_top 150000 KHz
    pclk_top 100000 KHz
    aclk_perimid 300000 KHz
    hclk_perimid 150000 KHz
    pclk_pmu 100000 KHz
Net:   eth0: ethernet@fe2a0000, eth1: ethernet@fe010000
Hit key to stop autoboot('CTRL+C'):  0
=> <INTERRUPT>
```

可以看出 U-boot 打印出了板子的一些基本信息，包括 CPU、内存等信息。还包括了在板卡初始化过程中的一些提示信息，如屏幕显示和时钟频率等。

5.3 U-boot 快捷键

我们除了使用快捷按键进入 U-Boot 命令行模式，next-dev 还提供了其他的串口组合快捷键，常用的如下

- ctrl+c：进入 U-Boot 命令行模式；
- ctrl+d：进入 loader 烧写模式；
- ctrl+b：进入 maskrom 烧写模式；

5.4 U-boot 命令

当不清楚 U-boot 支持什么命令时，可输入 **help** 或 **?** 可查看 U-boot 支持的命令列表，如下所示

```
?       - alias for 'help'
android_print_hdr- print android image header
atags   - Dump all atags
base    - print or set address offset
bdinfo  - print Board Info structure
bidram_dump- Dump bidram layout
boot    - boot default, i.e., run 'bootcmd'
boot_android- Execute the Android Bootloader flow.
boot_fit- Boot FIT Image from memory or boot/recovery partition
bootavb - Execute the Android avb a/b boot flow.
bootd   - boot default, i.e., run 'bootcmd'
booti   - boot arm64 Linux Image image from memory
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
bootrkp - Boot Linux Image from rockchip image type
bootz   - boot Linux zImage image from memory
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
crypto_sum- crypto checksum engine
dhcp    - boot image via network using DHCP/TFTP protocol
dm      - Driver model low level access
download- enter rockusb/bootrom download mode
dtimg   - manipulate dtb/dtbo Android image
dump_irqs- Dump IRQs
dump_resource- dump resource list
echo    - echo args to console
editenv - edit environment variable
```

(下页继续)

(续上页)

```
env     - environment handling commands
exit    - exit script
ext2load- load binary file from a Ext2 filesystem
ext2ls  - list files in a directory (default /)
ext4load- load binary file from a Ext4 filesystem
ext4ls  - list files in a directory (default /)
ext4size- determine a file's size
false   - do nothing, unsuccessfully
fastboot- use USB or UDP Fastboot protocol
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
fatsize - determine a file's size
fatwrite- write file into a dos filesystem
fdt     - flattened device tree utility commands
fstype  - Look up a filesystem type
go      - start application at address 'addr'
gpt     - GUID Partition Table
help    - print command description/usage
iomem   - Show iomem data by device compatible(high priority) or node name
lcdputs - print string on video framebuffer
load    - load binary file from a filesystem
loop    - infinite loop on address range
ls      - list files in a directory (default /)
md      - memory display
mdio    - MDIO utility commands
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - display MMC info
mtd     - MTD utils
mtd_blk - MTD Block device sub-system
```

(下页继续)

(续上页)

```
mw      - memory write (fill)
nand    - NAND sub-system
nboot   - boot from NAND device
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
part    - disk partition related commands
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
pxe     - commands to get and boot from pxe files
rbrom   - Perform RESET of the CPU
reboot  - Perform RESET of the CPU, alias of 'reset'
reset   - Perform RESET of the CPU
rkimgtest- Test if storage media have rockchip image
rockchip_show_bmp- load and display bmp from resource partition
rockchip_show_logo- load and display log from resource partition
rockusb - Use the rockusb Protocol
run     - run commands in an environment variable
save    - save file to a filesystem
saveenv - save environment variables to persistent storage
setcurs - set cursor position within screen
setenv  - set environment variables
showvar - print local hushshell variables
size    - determine a file's size
source  - run script from memory
sysboot - command to get and boot from syslinux files
sysmem_dump- Dump sysmem layout
sysmem_search- Search a available sysmem region
test    - minimal test like /bin/sh
tftp    - download image via network using TFTP protocol
tftpbootm- tftpbootm aosp/uImage/FIT image via network using TFTP protocol
tftpflash- flash image via network using TFTP protocol
tftpput - TFTP put command, for uploading files to a server
```

(下页继续)

(续上页)

```
true    - do nothing, successfully
ums     - Use the UMS [USB Mass Storage]
usb     - USB sub-system
usbboot - boot from USB device
version - print monitor, compiler and linker version
```

可看到 U-boot 支持很多的命令，功能十分强大，与 linux 类似，在执行某条 U-boot 命令时，可使用 *tab* 自动补全命令，在没有命令名冲突的情况下可以使用命令的前几个字母作为命令的输入，例如想要执行 **reset** 命令，输入 *res* 或 *re* 即可。

错误： 默认不支持使用 saveenv 命令保存环境变量，因为可能会覆盖原有的默认 env 配置导致系统无法启动，如果要修改环境变量，请修改 uboot 源码。

当需要具体使用哪个命令时，可使用 **“help 命令”** 或 **“? 命令”** 的方式查看具体命令的使用说明。以 **“help printenv”** 为例，

```
=> help printenv
printenv - print environment variables

Usage:
printenv [-a]
    - print [all] values of all environment variables
printenv name ...
    - print value of environment variable 'name'
```

可以看到 printenv 命令的说明以及使用方法。

关于 U-boot 命令的使用可参考 U-boot 官方链接：http://www.denx.de/wiki/DULG/Manual **5.9. uboot Command Line Interface** 部分。

5.4.1 U-boot 常见命令

U-boot 命令众多，下面介绍常用的 U-boot 命令, 详细的 U-boot 命令使用方式请使用 **help [命令]** 查看。

表 1: 常见命令

命令	说明	举例
help	列出当前 U-boot 所有支持的命令	
help [命令]	查看指定命令的帮助	help printenv
reset	重启 U-boot	
printenv	打印所有环境参数的值	
printenv [参数名]	查看指定的环境参数值	printenv bootdelay
setenv	设置/修改/删除环境参数的值	setenv bootdelay 3
saveenv	保存环境参数	
ping	检测网络是否连通	ping 192.168.0.1
md	查看内存地址上的值	md.b 0x80000000 10 以字节查看 0x80000000 后 0x10 个数据
mw	用于修改内存地址上的值	mw.b 0x80000000 ff 10 以字节修改 0x80000000 后 0x10 个数据为 ff
echo	打印信息，与 linux 下的 echo 类似	
run	在执行某条环境参数命令	run bootcmd，执行 bootcmd
bootz	在内存中引导内核启动	
ls	查看文件系统中目录下的文件	
load	从文件系统中加载二进制文件到内存	

以上为用户较为常用使用的部分命令，具体的使用方式可使用 **help [命令]** 查看。

5.4.2 mmc 命令

mmc 命令能够对如 sd 卡以及 emmc 类的存储介质进行操作，以下进行简单说明，对于 mmc 命令不熟悉可使用 **help mmc** 查看相关命令的帮助，常用功能如下所示

表 2: mmc 命令功能

命令	说明
mmc list	查看板子上 mmc 设备
mmc dev	查看/切换当前默认 mmc 设备
mmc info	查看当前 mmc 设备信息
mmc part	查看当前 mmc 设备分区
mmc read	读取当前 mmc 设备数据
mmc write	写入当前 mmc 设备数据
mmc erase	擦除当前 mmc 设备数据

5.4.2.1 查看 mmc 设备

使用 **mmc list** 查看板卡相关设备，示例板卡板载 emmc，可看到打印信息如下。

```
dwmmc@fe2b0000: 1
dwmmc@fe2c0000: 2
sdhci@fe310000: 0 (eMMC)
```

使用 **mmc dev index** 切换当前 mmc 设备，index 为设备号，根据 **mmc list** 的结果，可知 eMMC 的设备号为 0

```
=> mmc dev 0
switch to partitions #0, OK
mmc0(part 0) is current device
```

使用 **mmc info** 查看当前 mmc 设备的信息。

```
=> mmc info
Device: sdhci@fe310000
Manufacturer ID: e9
OEM: 10f
Name: K93SK
Timing Interface: HS200
Tran Speed: 200000000
Rd Block Len: 512
MMC version 5.1
High Capacity: Yes
Capacity: 7.3 GiB
Bus Width: 8-bit
Erase Group Size: 512 KiB
HC WP Group Size: 8 MiB
User Capacity: 7.3 GiB
Boot Capacity: 4 MiB ENH
RPMB Capacity: 4 MiB ENH
```

5.4.2.2 查看分区信息

使用 **mmc part** 列出当前 mmc 设备分区

```
=> mmc part

Partition Map for MMC device 0  --   Partition Type: EFI

Part    Start LBA       End LBA         Name
        Attributes
        Type GUID
        Partition GUID
  1     0x00004000      0x00005fff      "uboot"
        attrs:  0x0000000000000000
```

(下页继续)

(续上页)

```
        type:    030f0000-0000-440a-8000-5a0500003c68
        guid:    54040000-0000-4235-8000-6d5c00004fb3
  2     0x00006000      0x00045fff      "boot"
        attrs:   0x0000000000000004
        type:    b0440000-0000-4d43-8000-788400007a88
        guid:    346f0000-0000-4136-8000-17bb00007f36
  3     0x00046000      0x00e9ffbf      "rootfs"
        attrs:   0x0000000000000000
        type:    3d410000-0000-4b71-8000-4d7e00004948
        guid:    614e0000-0000-4b53-8000-1d28000054a9
```

5.4.3 文件系统操作命令

U-boot 能够对 ext2/3/4 以及 fat 文件系统设备进行访问，可使用 fstype 命令判断存储介质分区使用的是什么类型的文件系统。以 mmc 介质为例，判断后两个分区的文件系统类型

```
=> fstype mmc 0:2
ext4
=> fstype mmc 0:3
ext4
```

文件系统为 ext4 的第三个分区对应 rootfs 根文件系统。

知道了文件系统的类型即可使用相对应的命令对分区内容进行操作了。

5.4.3.1 ext4 格式文件系统

ext4 文件系统的命令使用方式和 FAT 使用方式相似，仅命令名不同，U-boot 提供的 ext 文件系统命令如下

表 3: ext4 格式文件系统命令

命令	说明
ext4ls	查看存储设备的 ext4 分区里的内容
ext4load	从 ext4 分区里读出文件到指定的内存地址
ext4write	把内存上的数据存储到 ext4 分区的一个文件里

5.4.3.1.1 ext4 文件系统操作

下面以将/etc/apt/sources.list 的内容读取到内存实例，简单说明 U-boot 对 ext4 文件系统操作。

1. 查看/etc/apt 目录中的文件内容，

```
=> ext4ls mmc 0:3 /etc/apt/
<DIR>       4096 .
<DIR>       4096 ..
            464 sources.list
<DIR>       4096 apt.conf.d
<DIR>       4096 auth.conf.d
<DIR>       4096 preferences.d
<DIR>       4096 sources.list.d
<DIR>       4096 trusted.gpg.d
```

2. 将/etc/apt/sources.list 文件读取到内存地址 0x8000 0000 处

```
=> ext4load mmc 0:3  0x80000000  /etc/apt/sources.list
464 bytes read in 701 ms (0 Bytes/s)
```

3. 查看内存 0x8000 0000 的部分数据内存

```
=> md.b 0x80000000 0x80
80000000: 64 65 62 20 68 74 74 70 3a 2f 2f 6d 69 72 72 6f    deb http://
 ↪mirro
```

(下页继续)

(续上页)

```
3 80000010: 72 73 2e 75 73 74 63 2e 65 64 75 2e 63 6e 2f 64    rs.ustc.edu.cn/
  ↪d
4 80000020: 65 62 69 61 6e 20 62 75 73 74 65 72 20 6d 61 69    ebian buster␣
  ↪mai
5 80000030: 6e 20 63 6f 6e 74 72 69 62 20 6e 6f 6e 2d 66 72    n contrib non-
  ↪fr
6 80000040: 65 65 0a 64 65 62 2d 73 72 63 20 68 74 74 70 3a    ee.deb-src␣
  ↪http:
7 80000050: 2f 2f 6d 69 72 72 6f 72 73 2e 75 73 74 63 2e 65    //mirrors.ustc.
  ↪e
8 80000060: 64 75 2e 63 6e 2f 64 65 62 69 61 6e 20 62 75 73    du.cn/debian␣
  ↪bus
9 80000070: 74 65 72 20 6d 61 69 6e 20 63 6f 6e 74 72 69 62    ter main␣
  ↪contrib
```

5.5 U-boot 启动内核过程

bootcmd 与 bootargs 可以说是 U-boot 最重要的两个环境参数，U-boot 执行完毕之后，如果没有按下回车，则会自动执行 bootcmd 命环境参数里的内容，而 bootargs 则是传递给内核的启动参数。

使用 **printenv bootcmd** 可查看 bootcmd 的内容。

```
1 => printenv bootcmd
2 bootcmd=boot_android ${devtype} ${devnum};boot_fit;bootrkp;run distro_
  ↪bootcmd;
```

如果烧录了 rkboot 分区的镜像，会执行 bootrkp，通过 rk 定义的启动流程来启动内核。具体内容可参考 RK 官方文档 《U-Boot v2017(next-dev) 开发指南》 2.4 启动流程

而如果烧录了 extboot 分区镜像，bootcmd 最终会执行 distro_bootcmd，同样可以使用 printenv distro_bootcmd 查看 distro_bootcmd 的内容如下

```
=> printenv distro_bootcmd
distro_bootcmd=for target in ${boot_targets}; do run bootcmd_${target}; done

=> printenv boot_targets
boot_targets=mmc1 mmc0 usb0 pxe dhcp
```

也就是说 distro_bootcmd 会执行 **bootcmd_mmc1**、**bootcmd_mmc0 bootcmd_usb0 bootcmd_pxe bootcmd_dhcp** 这五个环境参数，

在前面我们知道，mmc1 表示的 sd 卡的存储设备，mmc0 表示的 emmc 设备，也就是说当 sd 卡插在板子时，若 sd 卡装有系统则会优先从 sd 卡内启动。

后面三个启动参数，我们的系统暂不支持。

```
=> printenv bootcmd_mmc0
bootcmd_mmc0=setenv devnum 0; run mmc_boot
=> printenv bootcmd_mmc1
bootcmd_mmc1=setenv devnum 1; run mmc_boot
```

bootcmd_mmc0 与 bootcmd_mmc1 均设置各自 **devnum** 环境参数的值，最后运行 mmc_boot 环境参数，mmc_boot 内容如下

```
=> printenv mmc_boot
mmc_boot=if mmc dev ${devnum}; then setenv devtype mmc; run scan_dev_for_
↪boot_part; fi
```

在 mmc_boot 中设置 **devtype** 环境参数的值，最后运行 scan_dev_for_boot_part 环境参数，其作用是扫描设备中带有启动标志的分区

```
=> printenv scan_dev_for_boot_part
scan_dev_for_boot_part=part list ${devtype} ${devnum} -bootable devplist;␣
↪env exists devplist || setenv devplist 1; for distro_bootpart in $
↪{devplist}; do if fstype ${devtype} ${devnum}:${distro_bootpart}␣
↪bootfstype; then run scan_dev_for_boot; fi; done
```

提示： 若从 U-boot 中直接使用 printenv 查看会显得格式很乱，我们在 U-boot 源码中查看。

- include/configs/rk3568_common.h
- include/configs/rockchip-common.h
- include/config_distro_bootcmd.h

在源文件 config_distro_bootcmd.h 中我们可以看到 scan_dev_for_boot_part 的定义

```
"scan_dev_for_boot_part="   \
  "part list ${devtype} ${devnum} -bootable devplist; "   \
  "env exists devplist || setenv devplist 1; "    \
  "for distro_bootpart in ${devplist}; do "   \
    "if fstype ${devtype} "          \
        "${devnum}:${distro_bootpart} "  \
        "bootfstype; then "      \
      "run scan_dev_for_boot; "   \
    "fi; "    \
  "done\0"     \
```

上述脚本先判断是否存在带 **-bootable** 标志的分区，如果存在就记录分区号，然后运行 scan_dev_for_boot 去扫描 boot 分区是否存在启动所需的文件。

```
"scan_dev_for_boot="     \
  "echo Scanning ${devtype} "   \
      "${devnum}:${distro_bootpart}...; "  \
  "for prefix in ${boot_prefixes}; do "   \
    "run scan_dev_for_scripts; "   \
    "run scan_dev_for_extlinux; "    \
  "done;"  \
  SCAN_DEV_FOR_EFI  \
  "\0"
```

scan_dev_for_boot 的定义这是依次运行 scan_dev_for_scripts、scan_dev_for_extlinux。如果要改变不

同启动方式的顺序，可以修改这一部分。

scan_dev_for_extlinux 用于扫描 boot 分区是否存在 extlinux.conf 配置文件，存在的话会读取配置文件进行启动。配置文件里定义了内核、设备树等，还可以图形化选取不同版本内核启动，感兴趣可以去了解一下。

scan_dev_for_scripts 用于扫描 boot 分区是否存在 boot.scr 文件，存在的话就会按照 boot.scr 脚本中的启动流程进行启动。

```
"scan_dev_for_scripts="   \
  "for script in ${boot_scripts}; do "   \
    "if test -e ${devtype} "   \
        "${devnum}:${distro_bootpart} "   \
        "${prefix}${script}; then "   \
      "echo Found U-Boot script "   \
        "${prefix}${script}; "   \
      "run boot_a_script; "  \
      "echo SCRIPT FAILED: continuing...; " \
    "fi; "    \
  "done\0"    \

"boot_scripts=boot.scr.uimg boot.scr\0" \

"boot_a_script="                                          \
  "load ${devtype} ${devnum}:${distro_bootpart} "          \
    "${scriptaddr} ${prefix}${script}; "              \
  "source ${scriptaddr}\0"                                \
```

上述脚本在 scan_dev_for_scripts 中先去检测我们的 boot 分区中有没有名称为 boot.scr.uimg 或 boot.scr 的启动脚本，如果检测到的话先输出”Found U-Boot script 脚本路径”这个提示信息，然后运行 boot_a_script。如果检测不到的话就跳出。

在我们的鲁班猫板卡 boot 分区打包时已经编译了 boot.scr 的源文件并进行打包，所以这里会继续执行 boot_a_script

在 boot_a_script 中读取 boot 分区中的 boot.scr 文件并加载到内存中，并使用 source 运行这个脚本。

boot.scr 的源文件储存在 SDK 目录下的 kernel/arch/arm64/boot/dts/rockchip/uEnv/boot.cmd，下面是 boot.cmd 中的内容

```
echo [boot.cmd] run boot.cmd scripts ...;

if test -e ${devtype} ${devnum}:${distro_bootpart} /uEnv/uEnv.txt; then

    echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_r} /uEnv/uEnv.txt ...;
    load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_r} /uEnv/uEnv.txt;

    echo [boot.cmd] Importing environment from ${devtype} ...
    env import -t ${env_addr_r} 0x8000

    setenv bootargs ${bootargs} root=/dev/mmcblk${devnum}p3 ${cmdline}
    printenv bootargs

    echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r} /initrd-${uname_r} ...
    load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r} /initrd-${uname_r}

    echo [boot.cmd] loading ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} /Image-${uname_r} ...
    load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} /Image-${uname_r}

    echo [boot.cmd] loading default rk-kernel.dtb
    load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} /rk-kernel.dtb

    fdt addr  ${fdt_addr_r}
    fdt set /chosen bootargs
```

(下页继续)

(续上页)

```
25
26      echo [boot.cmd] dtoverlay from /uEnv/uEnv.txt
27      setenv dev_bootpart ${devnum}:${distro_bootpart}
28      dtfile ${fdt_addr_r} ${fdt_over_addr}  /uEnv/uEnv.txt ${env_addr_r}
29
30      echo [boot.cmd] [${devtype} ${devnum}:${distro_bootpart}] ...
31      echo [boot.cmd] [booti] ...
32      booti ${kernel_addr_r} ${ramdisk_addr_r} ${fdt_addr_r}
33  fi
34
35  echo [boot.cmd] run boot.cmd scripts failed ...;
36
37  # Recompile with:
38  # mkimage -C none -A arm -T script -d /boot/boot.cmd /boot/boot.scr
```

1. 打印提示信息，提示运行到了 boot.scr 脚本
2. 判读 boot 分区中是否存在 uEnv/uEnv.txt 这个文件，如果存在的话就继续执行下面的脚本
3. load 命令读取 uEnv 文件到内存，里面保存了自定义的环境变量，如内核，设备树插件等信息
4. env import 命令导入读取到的环境变量
5. setenv 命令设置 bootargs 环境变量，以便启动内核时向内核传递参数。具体内容是设置了 rootfs 所在的分区，还将 uEnv.txt 文件中的 cmdline 追加到了 bootargs
6. printenv 命令打印 bootargs 环境变量到控制台，便于调试
7. 从 boot 分区加载 initrd 镜像到内存
8. 从 boot 分区加载 kernel 镜像到内存
9. 从 boot 分区加载主设备树 rk-kernel.dt 到内存
10. fdt 命令读取加载的主设备树，并清空/chosen 节点中的 bootargs 的内容，否则会覆盖步骤 5 设置的 bootargs

11. dtfile 命令读取 uEnv 文件中定义的设备树插件，并合并到主设备树中
12. 使用 booti 命令启动内核

经过上面的流程，我们就离开了 uboot 的工作区域，进入到了内核中。如果以上流程失败，将会继续运行 scan_dev_for_extlinux，解决由于有时设备树插件加载失败等原因导致无法启动的问题。

5.6 U-boot 环境参数介绍

U-boot 中环境参数为我们提供一种不修改 U-boot 源码的情况下，能够修改 kernel 启动倒计时、ip 地址、以及向内核传递不同的参数等。

在板子上使用 **printenv** 可查看板子上所有的环境参数，使用 **setenv** 添加/修改/删除环境参数，具体说明如下所示

```
# 设置新的环境参数名为 abc，值为 100
=> setenv abc 100
=> echo $abc
= 100

# 将值修改为 200
=> setenv abc 200
=> echo $abc
200

# 删除 abc 环境参数
=> setenv abc
=> echo $abc
```

默认情况下使用 setenv 命令修改环境参数重启后就会消失，若想要掉电保存需要执行 **saveenv** 将环境参数保存到存储介质。

警告： 不建议在 uboot 中使用此操作，会覆盖默认的环境变量。

U-boot 上有一些官方规定的环境变量，这些环境变量在 U-boot 有着特殊的作用，可通过以下链接查看：https://www.denx.de/wiki/view/DULG/UBootEnvVariables

第 6 章　U-boot 的修改与编译

借助 LubanCat-SDK，我们可以很方便的进行 U-boot 的编译，但是有时候我们还要根据板卡的实际情况来修改 U-boot，实现一些定制功能。为此，我们必须对 U-boot 进行一个深入的了解。

下面的内容，我们将会了解 LubanCat-SDK 是如何实现对 U-boot 的一键构建，构建参数又与哪些文件有关

不同 LubanCat 板卡使用相同版本的 U-Boot 源码进行构建，仅少量配置不同，下述文档以使用 RK3568 处理器的鲁班猫板卡为例进行说明，使用其他型号处理器的鲁班猫板卡内容基本相同，仅在必要处指出不同点。

6.1 获取 U-boot

由于主线版 U-boot 对大部分比较新的处理器支持十分有限，所以我们选择使用 RK 版本的 U-Boot：next-dev，而 RK 版本的 U-Boot 又分为两部分，一个是 U-boot 本身的代码仓库，也就是 u-boot 目录下的内容，还有一部分是工具包仓库，存放 RK 不开源的二进制文件、脚本、打包工具，用于打包生成 loader、trust、uboot 固件，在 rkbin 目录下。

> **注意：** rkbin 和 U-Boot 工程必须保持同级目录关系。

在 LubanCat-SDK 中已经有了这两部分内容，我们可以直接使用，也可用从官方获取。

6.1.1 下载源代码

官方 GitHub：

https://github.com/Caesar-github/u-boot next-dev 分支

https://github.com/Caesar-github/rkbin master 分支

野火源码仓库下载链接：

https://github.com/LubanCat

6.1.2 下载指定分支

通常一个 U-boot 仓库往往维护着不同分支的 U-boot，进入仓库目录下可通过命令查看及切换 U-boot 分支，

```
git clone --branch=main https://github.com/LubanCat/u-boot.git
git clone --branch=master https://github.com/LubanCat/rkbin.git
```

6.2 U-boot 编译 (Chip)

在 LubanCat-SDK 中，自动编译脚本基本上都存放在 build.sh 中，这是 SDK 的主要功能入口

注意： 以下对 build.sh 脚本中内容的描述仅适用于 LubanCat_Chip_SDK。LubanCat_Gen_SDK 编译流程相同，但具体内容有所变化，具体可查看 device/rockchip/common/scripts/mk-loader.sh

在 build.sh 中，U-boot 的构建函数是 function build_uboot() 具体内容如下

```
function build_uboot(){
    check_config RK_UBOOT_DEFCONFIG || return 0
    build_check_cross_compile
```

(下页继续)

(续上页)

```
    prebuild_uboot

    echo "============Start building uboot============"
    echo "TARGET_UBOOT_CONFIG=$RK_UBOOT_DEFCONFIG"
    echo "============================================"

    if [ "$RK_RAMDISK_SECURITY_BOOTUP" = "true" ];then
        if [ -n "$RK_CFG_RAMBOOT" ];then
            build_ramboot
        else
            build_kernel
        fi

        if [ -n "$RK_CFG_RECOVERY" ]; then
            build_recovery
        fi
        cp -f $TOP_DIR/rockdev/boot.img $TOP_DIR/u-boot/boot.img
        cp -f $TOP_DIR/rockdev/recovery.img $TOP_DIR/u-boot/recovery.img ||␣
↪true
    fi

    cd u-boot
    rm -f *_loader_*.bin
    if [ "$RK_LOADER_UPDATE_SPL" = "true" ]; then
        rm -f *spl.bin
    fi

    if [ -n "$RK_UBOOT_DEFCONFIG_FRAGMENT" ]; then
        if [ -f "configs/${RK_UBOOT_DEFCONFIG}_defconfig" ]; then
            make ${RK_UBOOT_DEFCONFIG}_defconfig $RK_UBOOT_DEFCONFIG_
↪FRAGMENT
        else
```

(下页继续)

(续上页)

```
34              make ${RK_UBOOT_DEFCONFIG}.config $RK_UBOOT_DEFCONFIG_FRAGMENT
35          fi
36          ./make.sh $UBOOT_COMPILE_COMMANDS
37      elif [ -d "$TOP_DIR/prebuilts/gcc/linux-x86/arm/gcc-arm-10.3-2021.07-
    ↪x86_64-arm-none-linux-gnueabihf" ]; then
38          ./make.sh $RK_UBOOT_DEFCONFIG \
39              $UBOOT_COMPILE_COMMANDS CROSS_COMPILE=$CROSS_COMPILE
40      elif [ -d "$TOP_DIR/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.
    ↪07-x86_64-aarch64-none-linux-gnu" ]; then
41          ./make.sh $RK_UBOOT_DEFCONFIG \
42              $UBOOT_COMPILE_COMMANDS CROSS_COMPILE=$CROSS_COMPILE
43      else
44          ./make.sh $RK_UBOOT_DEFCONFIG \
45              $UBOOT_COMPILE_COMMANDS
46      fi
47
48      if [ "$RK_IDBLOCK_UPDATE_SPL" = "true" ]; then
49          ./make.sh --idblock --spl
50      fi
51
52      if [ "$RK_RAMDISK_SECURITY_BOOTUP" = "true" ];then
53          ln -rsf $TOP_DIR/u-boot/boot.img $TOP_DIR/rockdev/
54          ln -rsf $TOP_DIR/u-boot/recovery.img $TOP_DIR/rockdev/ || true
55      fi
56
57      finish_build
58  }
```

- check_config 检查是否有 RK_UBOOT_DEFCONFIG 这个配置变量，定义了 U-boot 编译用到的配置文件，这个变量定义在 device/rockchip/rk356x/BoardConfig-*.mk 中
- build_check_cross_compilep 判断是否存在 arm-none-linux-gnueabihf 或 aarch64-none-linux-gnu 交叉编译工具链，我们没有使用，这一部分可以忽略。

- prebuild_uboot 根据 BoardConfig-*.mk 中的定义来设定编译参数 UBOOT_COMPILE_COMMANDS
- 判断 RK_RAMDISK_SECURITY_BOOTUP 值是否为 true，是否使用 secure boot 来对镜像进行签名认证，我们没有定义，直接跳过
- cd 进入 u-boot 目录，先清除以前生成的 miniloader 文件，再判断 uboot 前一阶段是否为 SPL，如果是的话还要清除 spl 文件
- RK_UBOOT_DEFCONFIG_FRAGMENT：由于 RK 版本 U-boot 支持配置文件的 overlay 功能，这里判断是否开启了该功能，如果开启的话先将配置文件合并，我们没有使用到
- 判断有没有 arm-none-linux-gnueabihf 或 aarch64-none-linux-gnu 来进行编译, 没有使用，可以忽略。
- 没有找到上面两个编译器，向./make.sh 传递构建参数进行构建 U-boot 操作
- RK_IDBLOCK_UPDATE_SPL 和 RK_RAMDISK_SECURITY_BOOTUP 都没有定义，直接跳过
- finish_build 编译完成的一个提示函数。

6.3 U-boot 修改

一般来说，要适配一块新的板卡，在 U-boot 中主要设计三部分的修改和添加，分别是 board 目录下涉及板卡初始化的部分、configs 目录下对应板卡的配置文件、arch 目录下与板级外设相关的文件。

得益于 Rockchip U-boot 对 rk 系列处理器的完善支持，在 U-boot 阶段，我们直接根据使用的处理器型号选择 Rockchip 官方参考板的配置即可。例如 rk356x 处理器使用 evb_rk3568，rk3588 则使用 evb_rk3588。

野火仅在 U-boot 中添加了设备树插件功能的实现。并使用 boot_scripts 方式进行启动引导，用于实现 extboot 分区的启动。

boot_scripts 启动方式的启动命令脚本在 SDK 目录下的 kernel/arch/arm64/boot/dts/rockchip/uEnv/boot.cmd，在构建 boot 分区时进行编译，并打包到 extboot 分

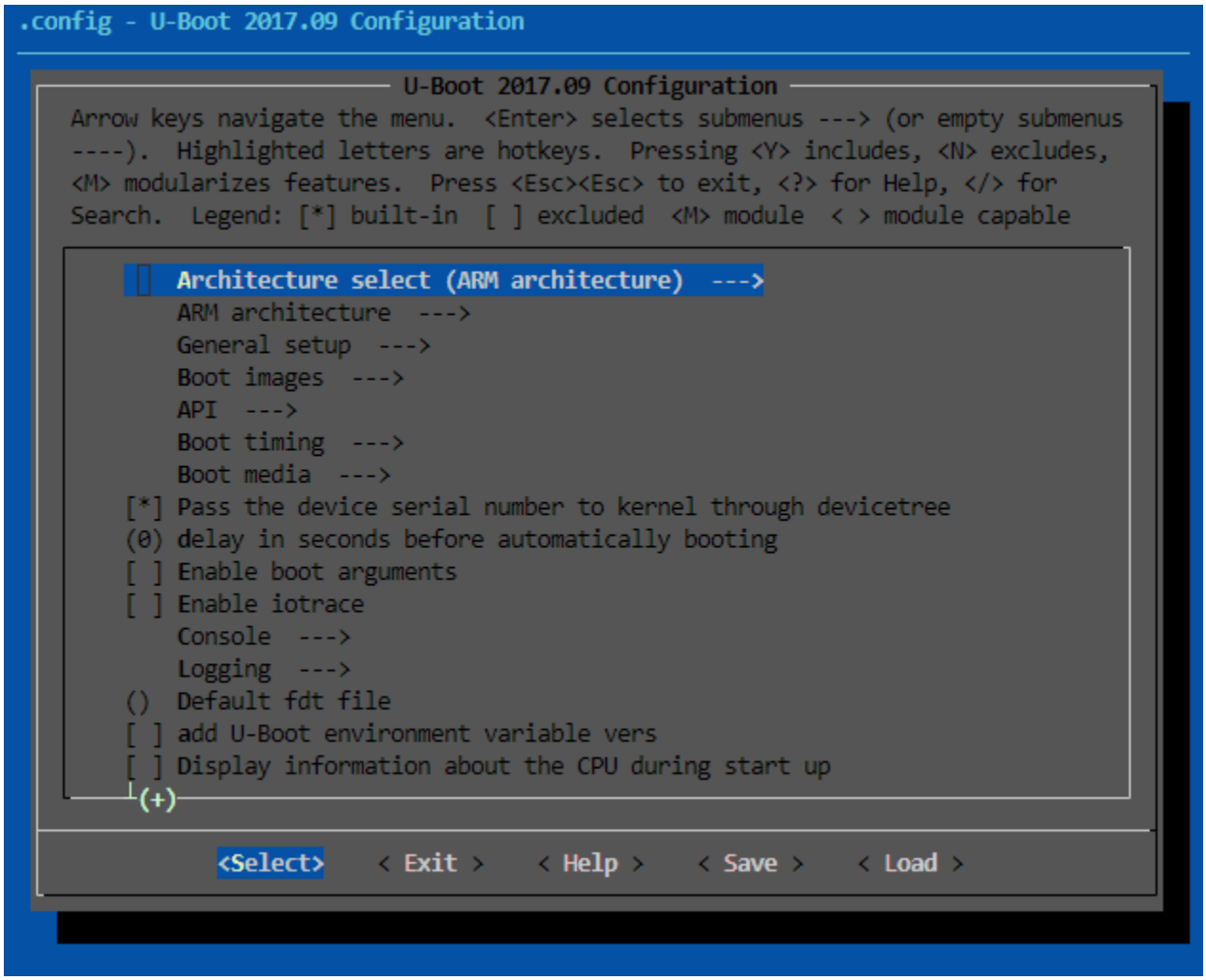
区内，命名为 boot.scr，在系统启动前被 U-boot 读取并加载。具体流程请查看上一章节《U-boot 启动内核过程》

6.3.1 U-boot 配置文件

我们编译使用的配置文件由 BoardConfig-*.mk 的 RK_UBOOT_DEFCONFIG 变量定义，在 LubanCat-RK 系列板卡上使用的具体文件是 u-boot/configs/rk3568_defconfig 和 rk3566.config，这里以 LubanCat-2 为例，如果我们要修改的话，可以借助 menuconfig 工具。

首先我们要来到 U-boot 顶层文件夹下，然后执行以下操作:

```
# 应用配置文件
make rk3568_defconfig

# 使用 menuconfig 来管理修改配置文件
make menuconfig
```



修改完成之后按 ESC 按键退出，并保存

```
# 保存 defconfig 文件
make savedefconfig

# 覆盖原来的配置文件
cp defconfig configs/rk3568_defconfig
```

6.3.2 设备树文件

设备树文件是板级设备的描述文件，系统通过设备树文件得知板卡上有哪些外设，从而加载相应的驱动使外设正常工作。

原生的 U-Boot 只支持使用 U-Boot 自己的 DTB，RK 平台增加了 kernel DTB 机制的支持，即使用 kernel DTB 去初始化外设。主要目的是为了兼容外设板级差异，如：power、clock、display 等。

二者的作用：

- U-Boot DTB：负责初始化存储、打印串口等设备；
- Kernel DTB：负责初始化存储、打印串口以外的设备；

U-Boot 初始化时先用 U-Boot DTB 完成存储、打印串口初始化，然后从存储上加载 Kernel DTB 并转而使用这份 DTB 继续初始化其余外设。

开发者一般不需要修改 U-Boot DTB（除非更换打印串口），RK 系列处理器使用的 defconfig 都已启用 kernel DTB 机制。所以通常对于外设的 DTS 修改，用户应该修改 kernel DTB。

关于 U-Boot DTB：

DTS 目录：

- u-boot/arch/arm/dts/rk3568-evb.dts
- u-boot/arch/arm/dts/rk3588-evb.dts

启用 kernel DTB 机制后：编译阶段会把 U-Boot DTS 里带 u-boot,dm-pre-reloc 和 u-boot,dm-spl 属性的节点过滤出来，在此基础上再剔除 defconfig 中 CONFIG_OF_SPL_REMOVE_PROPS 指定的 property，最终生成 u-boot.dtb 文件并且追加在 u-boot.bin 的末尾。

6.4 参考资料

RK U-boot 添加了很多自定义的功能，但其主体还是基于官方 U-boot 的，所以关于 U-boot 的操作都是基本相同的，我们可以参考以下 wiki。

官方 uboot 下载链接: http://www.denx.de/wiki/uboot/WebHome

官方 uboot GitHub：https://github.com/uboot/uboot

uboot wiki: http://www.denx.de/wiki/uboot/WebHome

RK U-boot 的自定义功能则可以详细查看 Rockchip 文档 《U-Boot v2017(next-dev) 开发指南》

第 7 章　Linux 的介绍

7.1 Unix 简介

Unix 系统源自贝尔实验室，提供源码。在 Unix 中几乎所有东西都被当成文件对待。

7.2 Linux 简介

芬兰人 Linus 最早开发，是类 Unix 系统，但不是 Unix，实现了 Unix 的 API（具体实现和 Unix 可能不同）

而 Linux 使用 GPLv2 开源协议

7.3 Linux 的发展史

1991

8 月 25 号: 21 岁的芬兰学生 Linus Benedict Torvalds 在 comp.os.minix 新闻组上宣布了它正在编写一个免费的操作系统。

9 月 1 号: Linux 0.01 在网上发布。

2007

6 月 6 号: 华硕在 2007 的台北电脑展上展出了两款 “易 PC”（Eee PC）: 701 和 1001。第 1 批易 PC 预装的是 Xandros Linux，这是一个基于 Debian，轻量级的为适应小屏幕进行过优化的 Linux 发行版。

8 月 8 号: 2007 年 Linux 基金会由开源发展实验室 (OSDL) 和自由标准组织 (FSG) 联合成立。这个基金会目的是赞助 Linux 创始人 Linus 的工作。基金会得到了主要的 Linux 和开源公司，包括富士通，HP，IBM，Intel，NEC，Oracle，Qualcomm，三星和来自世界各地的开发者的支持。

11 月 5 号: 与之前大家推测的发布 Gphone 不同，Google 宣布组建开放手机联盟 (Open Handset Alliance) 和发布 Android，它被称为“第一个真正开放的综合移动设备平台”。

2011

5 月 11 号: 2011 年 Google I/O 大会发布了 Chrombook。这是一款运行着所谓云操作系统 Chrome OS 的笔记本。Chome OS 是基于 Linux 内核的。

6 月 21 号: Linus Torvalds 发布了 Linux3.0 版本。

对 linux 发展史感兴趣的也可以到网上自行搜索 linux 发展史

7.4 单内核与微内核区别

单内核：所有内核从整体上作为一个单独大过程实现，运行于内核地址空间，内核通信简单。内核通常以单个静态二进制文件存放于磁盘上。简单性能高，大多数 Unix 都是单内核。

微内核：微内核功能划分为多个过程，各个过程运行在单独地址空间，需要通过进程间通信 IPC 处理微内核通信，只有请求强烈的过程才运行在内核态，其他过程在用户态。IPC 开销大，且涉及用户态和内核态上下文切换。所以多数的微内核实现（Windows NT、OS X）将所有微内核过程都运行于内核态

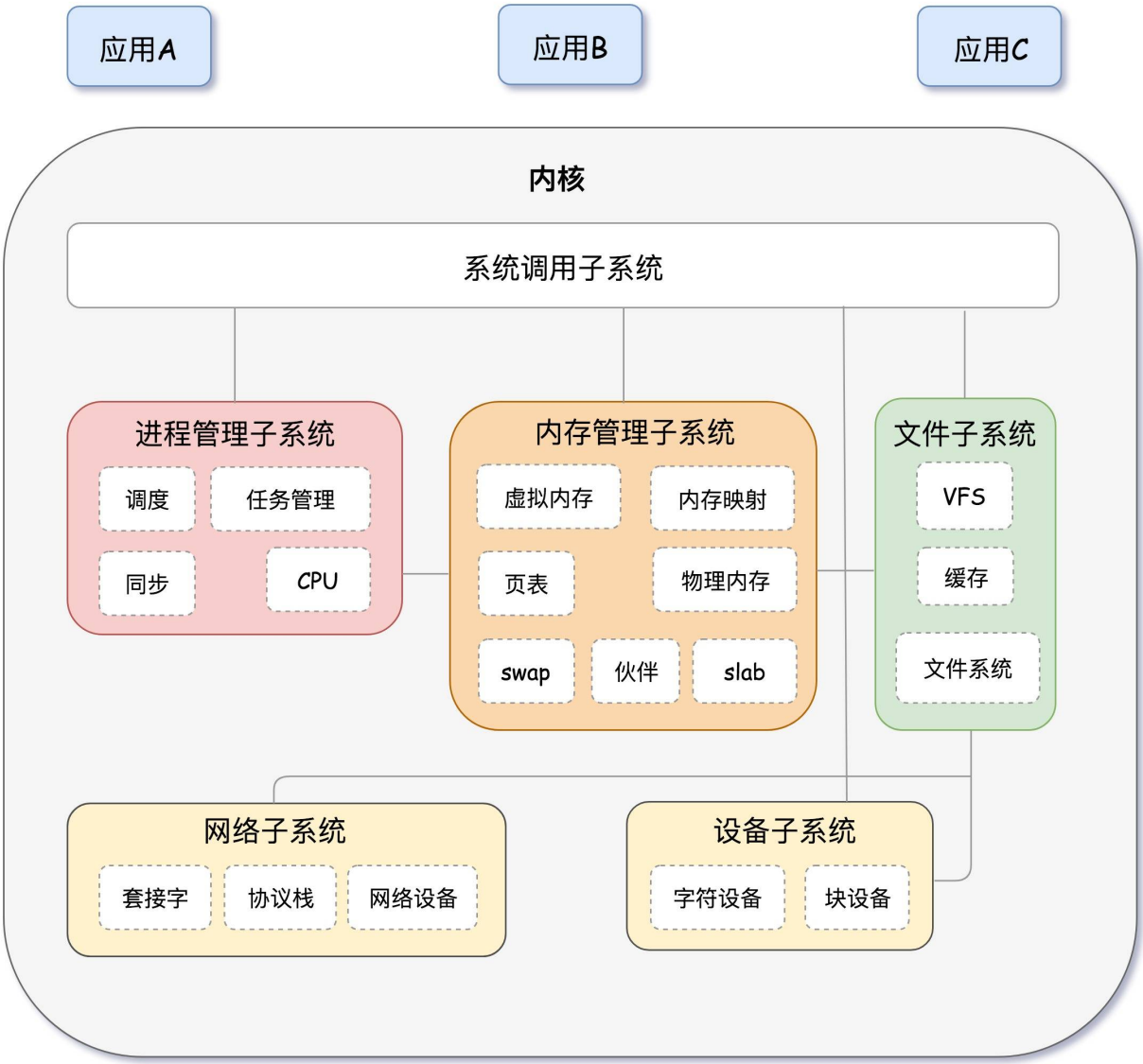
7.5 Linux 内核

Linux 是单内核，但吸取微内核精华：模块化涉及，抢占式内核，支持内核线程，可动态装载内核模块

Kernel 即是 Linux 内核，Linux 内核采用宏内核架构，即 Linux 大部分功能都会在内核中实现，如进程管理、内存管理、设备管理、文件管理以及网络管理等功能，Linux 在发展的过程中，引入了内核模块（Loadable Kernel Module，LKM）机制，内核模块全称为动态可加载内核模块，就是在内核运行时可以动态加载一组目标代码来实现某些特定的功能，在这过程中不需要重新编译内核就可以实现动态扩展。

7.6 Linux 内核组成

Linux 内核主要由 5 部分组成，分别为：进程管理子系统，内存管理子系统，文件子系统，网络子系统，设备子系统，如下图所示。



1、进程管理

负责进程的创建和销毁，进程的调度。

2、内存管理

负责内存的分配和回收，记录哪些内存被哪些进程使用，管理虚拟内存，将内存的物理地址和逻辑地址做一个映射，主要由 MMU 进行转换，页表的方式。

3、文件系统

这里的文件系统不仅仅只是硬盘的抽象管理，它也可以是某些 io 口的抽象；文件系统屏蔽了底层的细节，为上层提供统一的接口；linux 中一切皆文件。

4、设备管理

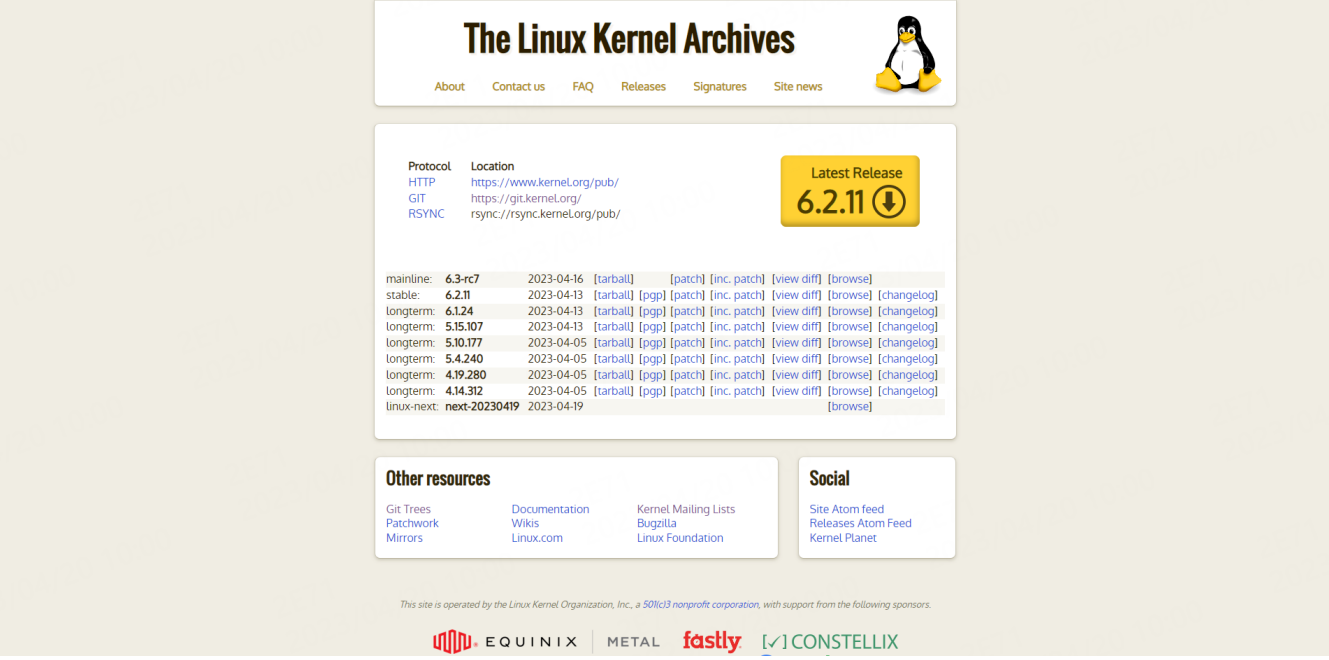
设备管理功能主要由驱动程序提供，主要任务是控制设备完成输入或输出操作；linux 把设备看作是特殊的文件，系统通过处理文件的接口（虚拟文件系统 VFS）来管理和控制各种设备。

5、网络功能

网络功能值的是除了驱动程序提供的基本硬件操作外，还有系统提供的机制和功能，比如 TCP 协议，地址解析等。

7.7 Linux 官方站点

官方地址为：https://www.kernel.org/



在这可以下载最新版本的内核文件，但是大部分是没适配相应芯片的，适配芯片是芯片原厂干的事，我们一般不去为一款芯片做适配，我们也没这个能力去适配，我们作为嵌入式开发人员主要职责是进行内核的配置与使用，不要把大量的精力用于去适配一款芯片，这对初学者非常劝退，所以本书主要讲解内核的配置和使用。

目前 LubanCat-2 和 LubanCat-1 板卡已加入 Linux Kernel，自 mainline 6.3-rc1 获得支持，相关内容可访问 https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/arch/arm64/boot/dts/rockchip 查看或访问 https://github.com/torvalds/linux/tree/master/arch/arm64/boot/dts/rockchip 查看

-rw-r--r--	rk3566-anbernic-rg503.dts	4996	log	stats	plain
-rw-r--r--	rk3566-anbernic-rgxx3.dtsi	16446	log	stats	plain
-rw-r--r--	rk3566-box-demo.dts	10666	log	stats	plain
-rw-r--r--	rk3566-lubancat-1.dts	12107	log	stats	plain
-rw-r--r--	rk3566-pinenote-v1.1.dts	302	log	stats	plain
-rw-r--r--	rk3566-pinenote-v1.2.dts	299	log	stats	plain
-rw-r--r--	rk3566-pinenote.dtsi	15292	log	stats	plain
-rw-r--r--	rk3566-quartz64-a.dts	17700	log	stats	plain
-rw-r--r--	rk3566-quartz64-b.dts	15489	log	stats	plain
-rw-r--r--	rk3566-radxa-cm3-io.dts	5684	log	stats	plain
-rw-r--r--	rk3566-radxa-cm3.dtsi	9392	log	stats	plain
-rw-r--r--	rk3566-roc-pc.dts	14730	log	stats	plain
-rw-r--r--	rk3566-soquartz-blade.dts	3958	log	stats	plain
-rw-r--r--	rk3566-soquartz-cm4.dts	3583	log	stats	plain
-rw-r--r--	rk3566-soquartz-model-a.dts	4771	log	stats	plain
-rw-r--r--	rk3566-soquartz.dtsi	14827	log	stats	plain
-rw-r--r--	rk3566.dtsi	585	log	stats	plain
-rw-r--r--	rk3568-bpi-r2-pro.dts	17481	log	stats	plain
-rw-r--r--	rk3568-evb1-v10.dts	14288	log	stats	plain
-rw-r--r--	rk3568-lubancat-2.dts	14856	log	stats	plain
-rw-r--r--	rk3568-odroid-m1.dts	15270	log	stats	plain
-rw-r--r--	rk3568-pinctrl.dtsi	64535	log	stats	plain
-rw-r--r--	rk3568-radxa-cm3i.dtsi	8533	log	stats	plain
-rw-r--r--	rk3568-radxa-e25.dts	4741	log	stats	plain
-rw-r--r--	rk3568-rock-3a.dts	18080	log	stats	plain
-rw-r--r--	rk3568.dtsi	7602	log	stats	plain
-rw-r--r--	rk356x.dtsi	50005	log	stats	plain

第 8 章　Linux 内核的编译

这里的 Linux 内核指的是 Kernel，或者称内核也是指的 Kernel

8.1 为什么要自己编译 Kernel

虽然我们提供的内核已经支持绝大多数功能了，但是对于一些需要定制功能的客户来说不一定有他们需要的功能配置，所以本章将讲解如何配置与编译内核

8.2 获取 kernel

8.2.1 下载源代码

有三个方法获取 Kernel 源码，一个是 Kernel 官方内核源码，一个是 RockChip 官方的 kernel 源码，一个是经过我们修改适配我们板卡的 kernel 源码，我们这里使用我们提供的源码为例，对官方源码感兴趣的小伙伴也可以下载来学习配置。

我们的 kernel 是根据 Rockchip 官方提供的 kernel 定制的，Rockchip 的 kernel 是根据 kernel 官方 LTS(长期支持) 版本进行芯片适配的。我们作为嵌入式开发者，一般只需要使用芯片厂商适配好的 kernel 进行开发即可，对于从 kernel 官方下载新版本来适配这是芯片厂商做的事情。

目前 LubanCat_Linux_rk356x_SDK 使用 kernel-4.19 版本。LubanCat_Linux_rk3588_SDK 使用 kernel-5.10 版本，LubanCat_Linux_Generic_SDK 根据不同板卡使用 kernel-5.10 或 kernel-6.1

由于内核主线对特定芯片的开发时慢于芯片厂商的，并且芯片厂商很少提交到内核主线，就导致芯片的很多功能并没有被实现，所以目前我们只能使用 Rockchip 较旧的 kernel 版本。

Kernel 官方内核源码：https://www.kernel.org/

Rockchip 内核源码: https://github.com/rockchip-linux/kernel/

LubanCat 内核源码：https://github.com/LubanCat/kernel

LubanCat 内核源码分支说明：

- lbc-develop-5.10: 适用于 LubanCat_Linux_Generic_SDK 的 5.10 内核，版本较新
- lbc-develop-6.1: 适用于 LubanCat_Linux_Generic_SDK 的 6.1 内核，版本最新，板卡逐步支持中
- stable-4.19-rk356x：适用于 LubanCat_Linux_rk356x_SDK 的稳定版内核
- stable-4.19-rk356x-rt104：适用于 LubanCat_Linux_rk356x_SDK 的实时内核
- stable-5.10-rk3588：适用于 LubanCat_Linux_rk3588_SDK 的稳定版内核
- stable-5.10-rk3588-rt89：适用于 LubanCat_Linux_rk3588_SDK 的实时内核

使用我们提供的内核源码

```
# LubanCat_Gen_SDK
git clone -b lbc-develop-5.10 https://github.com/LubanCat/kernel
git clone -b lbc-develop-6.1 https://github.com/LubanCat/kernel

# LubanCat_Chip_SDK
git clone -b stable-4.19-rk356x https://github.com/LubanCat/kernel
git clone -b stable-5.10-rk3588 https://github.com/LubanCat/kernel
```

8.3 kernel 工程结构分析

学习一个软件，尤其是开源软件，首先应该从分析软件的工程结构开始。一个好的软件有良好的工程结构，对于读者学习和理解软件的架构以及工作流程都有很好的帮助。

内核源码目录如下：

```
arch          COPYING         drivers    init      kernel        make_deb.sh␣
↪README     sound
block         CREDITS         firmware   ipc       lib           Makefile     ␣
↪samples    tools
```

(下页继续)

(续上页)

```
build_image  crypto         fs        Kbuild   LICENSES     mm          ␣
↪scripts   usr
certs        Documentation  include   Kconfig  MAINTAINERS  net         ␣
↪security  virt
```

我们可以看到 Linux 内核源码目录下是有非常多的目录，且目录下也有非常多的文件，下面我们简单分析一下这些目录的主要作用。

arch： 主要包含和硬件体系结构相关的代码，如 arm、x86、MIPS，PPC，每种 CPU 平台占一个相应的目录。arch 中的目录下存放的是各个平台以及各个平台的芯片对 Linux 内核进程调度、内存管理、中断等的支持，以及每个具体的 SoC 和电路板的板级支持代码。

block： 在 Linux 中 block 表示块设备（以块（多个字节组成的整体，类似于扇区）为单位来整体访问），譬如说 SD 卡、Nand、硬盘等都是块设备，block 目录下放的是一些 Linux 存储体系中关于块设备管理的代码。

crypto： 这个目录下存放的是常用加密和散列算法（如 md5、AES、SHA 等），还有一些压缩和 CRC 校验算法。

Documentation：内核各部分的文档描述。

drivers： 设备驱动程序，里面列出了 linux 内核支持的所有硬件设备的驱动源代码，每个不同的驱动占用一个子目录，如 char、block、net、mtd、i2c 等。

fs： fs 就是 file system，里面包含 Linux 所支持的各种文件系统，如 EXT、FAT、NTFS、JFFS2 等。

include： 目录包括编译核心所需要的大部分头文件，例如与平台无关的头文件在 `include/linux` 子目录下，与 cpu 架构相关的头文件在 include 目录下对应的子目录中。

init： 内核初始化代码，这个目录下的代码就是 linux 内核启动时初始化内核的代码。

ipc： ipc 就是 `inter process commuication`，进程间通信，该目录下都是 linux 进程间通信的代码。

kernel： kernel 就是 Linux 内核，是 Linux 中最核心的部分，包括进程调度、定时器等，而和平台相关的一部分代码放在 arch/*/kernel 目录下。

lib： lib 是库的意思，lib 目录下存放的都是一些公用的有用的库函数，注意这里的库函数和 C 语

言的库函数不一样的，因为在内核编程中是不能用 C 语言标准库函数的，所以需要使用 lib 中的库函数，除此之外与处理器结构相关的库函数代码被放在 `arch/*/lib/` 目录下。

mm： 目录包含了所有独立于 cpu 体系结构的内存管理代码，如页式存储管理内存的分配和释放等，而与具体硬件体系结构相关的内存管理代码位于 `arch/*/mm` 目录下，例如 `arch/arm/mm/fault.c` 。

net： 网络协议栈相关代码，net 目录下实现各种常见的网络协议。

scripts： 这个目录下全部是脚本文件，这些脚本文件不是 linux 内核工作时使用的，而是用了配置编译 linux 内核的。

security： 内核安全模型相关的代码，例如最有名的 SELINUX。

sound： ALSA、OSS 音频设备的驱动核心代码和常用设备驱动。

usr： 实现用于打包和压缩的 cpio 等。

此处仅列出一些常见的目录。

8.4 内核配置选项

在不同版本的内核和不同系列的鲁班猫板卡使用的配置文件都保存在内核目录的 arch/arm64/configs 中，具体使用的配置文件如下：

- Kernel 5.10.198/209：根据主芯片型号区分 lubancat_linux_xxxx_defconfig，xxxx 为芯片型号
- Kernel 6.1：根据主芯片型号区分 lubancat_linux_xxxx_defconfig，xxxx 为芯片型号
- Kernel 4.19.232：LubanCat-RK356x 系列板卡都使用 lubancat2_defconfig
- Kernel 5.10.160：LubanCat-RK3588 系列板卡都使用 lubancat_linux_rk3588_defconfig

8.4.1 修改内核配置 (LubanCat_Chip_SDK)

提示： 本节需要输入命令的操作，需在 kernel 目录下进行

以下以 LubanCat-2 板卡为例进行说明

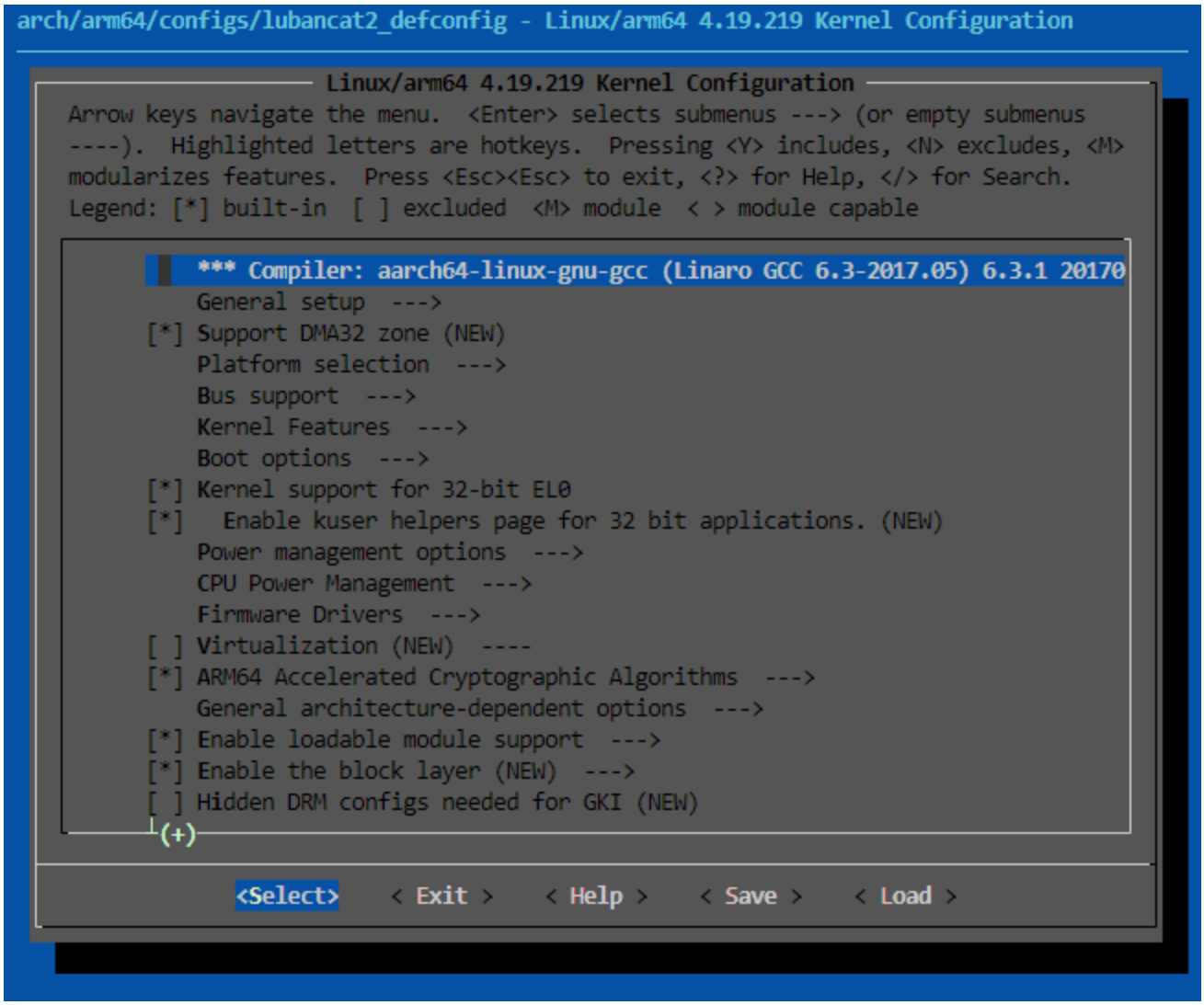
Linux 内核的配置系统由三个部分组成，分别是：

- Makefile：分布在 Linux 内核源代码顶层目录及各层目录中，定义 Linux 内核的编译规则；
- 配置文件：给用户提供配置选择的功能，如 Kconfig 文件定义了配置项，在编译时，使用 `arch/arm64/configs/lubancat2_defconfig` 文件对配置项进行赋值；
- 配置工具：包括配置命令解释器（对配置脚本中使用的配置命令进行解释）和配置用户界面（linux 提供基于字符界面、基于 Ncurses 图形界面以及基于 Xwindows 图形界面的用户配置界面，各自对应于 make config、make menuconfig 和 make xconfig）。

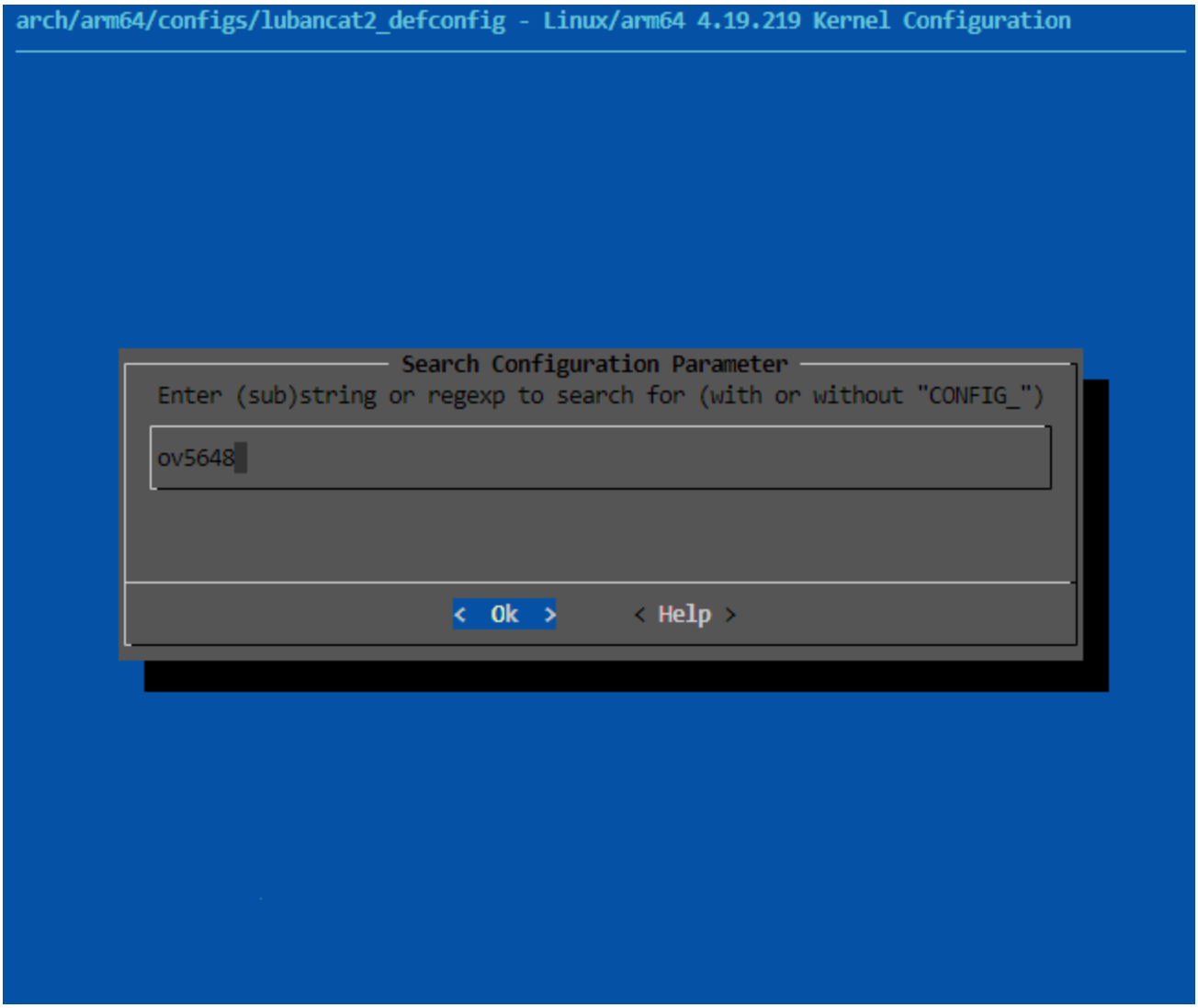
注意： 如果自定义配置文件，编译时要在板卡对应的 device/rockchip/rk356x/BoardConfig.mk 文件中修改 RK_KERNEL_DEFCONFIG 的定义。

我们可以通过 `make menuconfig KCONFIG_CONFIG=arch/arm64/configs/lubancat2_defconfig ARCH=arm64` 命令来查看我们的配置，make menuconfig 是一个基于文本选择的配置界面，推荐在字符终端下使用，而这个配置文件为 `lubancat2_defconfig` 此时就可以看到在 lubancat2_defconfig 的配置选择，可以通过键盘的”上”、”下”、”左”、”右”、”回车”、”空格”、”?”、”ESC”等按键进行选择配置，具体见：

```
# 使用图形界面配置需要额外安装 libncurses-dev
sudo apt install libncurses-dev

# 执行命令
make menuconfig KCONFIG_CONFIG=arch/arm64/configs/lubancat2_defconfig␣
↪ARCH=arm64
```

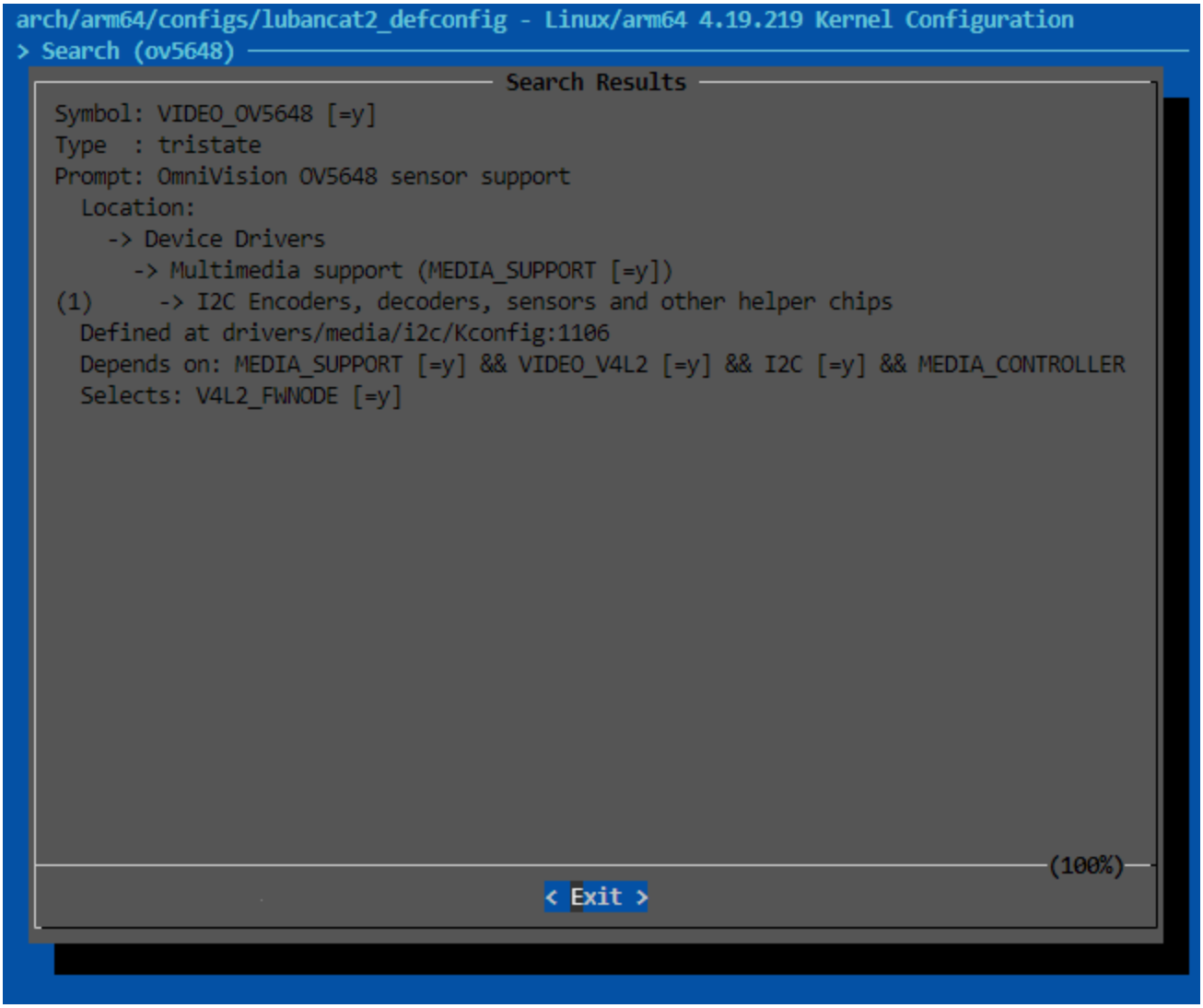

arch/arm64/configs/lubancat2_defconfig - Linux/arm64 4.19.219 Kernel Configuration
Linux/arm64 4.19.219 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [ ] excluded <M> module < > module capable
*** Compiler: aarch64-linux-gnu-gcc (Linaro GCC 6.3-2017.05) 6.3.1 20170
General setup --->
[*] Support DMA32 zone (NEW)
Platform selection --->
Bus support --->
Kernel Features --->
Boot options --->
[*] Kernel support for 32-bit EL0
[*] Enable kuser helpers page for 32 bit applications. (NEW)
Power management options --->
CPU Power Management --->
Firmware Drivers --->
[ ] Virtualization (NEW) ----
[*] ARM64 Accelerated Cryptographic Algorithms --->
General architecture-dependent options --->
[*] Enable loadable module support --->
[*] Enable the block layer (NEW) --->
[ ] Hidden DRM configs needed for GKI (NEW)
(+)
<Select> < Exit > < Help > < Save > < Load >

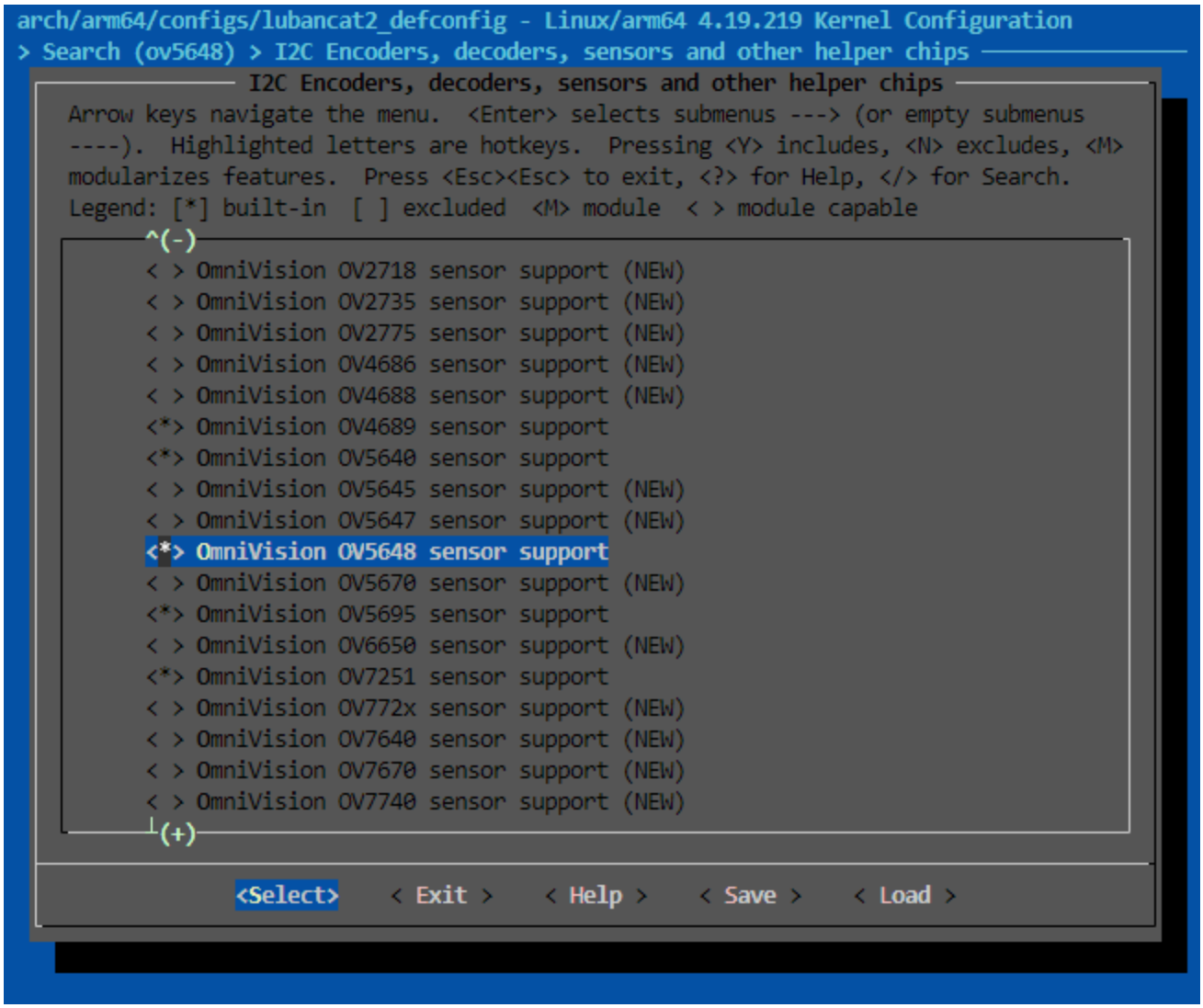
比如我们选择配置板卡的 ov5648 摄像头驱动：`ov5648`，如果找不到这个配置选项在哪里，可以使用 `make menuconfig` 中的搜索功能，在英文输入法状态下按下”/”则可以进行搜索，输入”ov5648”找到改配置选项的位置，当输入错误时，可使用 *Ctrl*+ 退格键删除输入。具体见：



从图中可以很明显看出 `ov5648` 配置选项位于 `-> Device Drivers` 选项下的 `-> Multimedia support (MEDIA_SUPPORT [=y])` 下的 `-> I2C Encoders, decoders, sensors and other helper chips` 的选项中，其实也可以按下 `"1"` 直接可以定位到对应的选项，然后选中以下内容即可，具体见图：

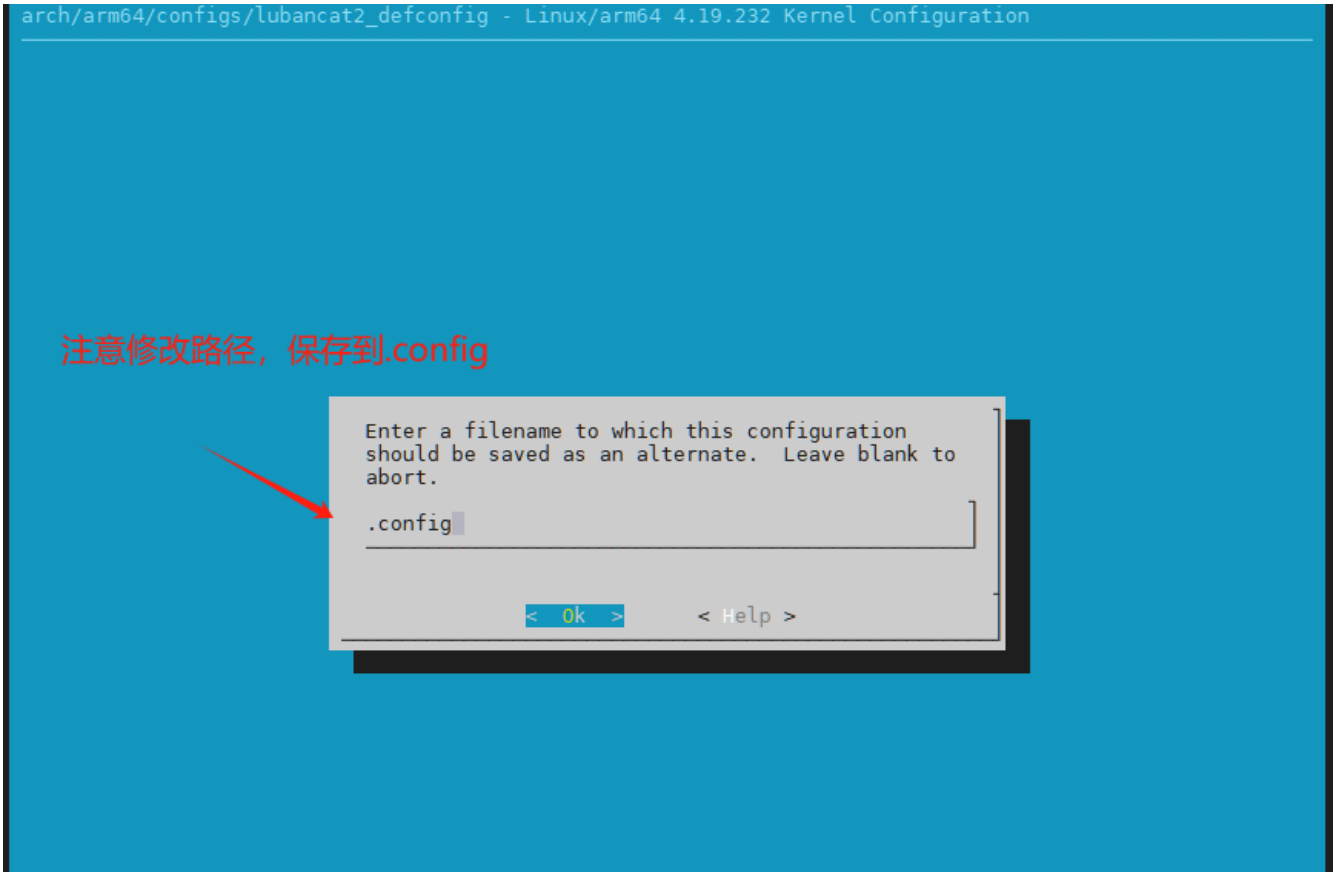
可使用 `y`、`n`、`m` 键更改 ov5648 驱动的配置时，其中 y 表示编译进内核中，m 表示编译成模块，n 表示不编译。也可使用空格选择 ov5648 驱动的配置选项。


arch/arm64/configs/lubancat2_defconfig - Linux/arm64 4.19.219 Kernel Configuration
> Search (ov5648)
Search Results
Symbol: VIDEO_OV5648 [=y]
Type : tristate
Prompt: OmniVision OV5648 sensor support
Location:
-> Device Drivers
-> Multimedia support (MEDIA_SUPPORT [=y])
(1) -> I2C Encoders, decoders, sensors and other helper chips
Defined at drivers/media/i2c/Kconfig:1106
Depends on: MEDIA_SUPPORT [=y] && VIDEO_V4L2 [=y] && I2C [=y] && MEDIA_CONTROLLER
Selects: V4L2_FWNODE [=y]
(100%)
< Exit >


arch/arm64/configs/lubancat2_defconfig - Linux/arm64 4.19.219 Kernel Configuration
> Search (ov5648) > I2C Encoders, decoders, sensors and other helper chips
I2C Encoders, decoders, sensors and other helper chips
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [ ] excluded <M> module < > module capable
^(-)
< > OmniVision OV2718 sensor support (NEW)
< > OmniVision OV2735 sensor support (NEW)
< > OmniVision OV2775 sensor support (NEW)
< > OmniVision OV4686 sensor support (NEW)
< > OmniVision OV4688 sensor support (NEW)
<*> OmniVision OV4689 sensor support
<*> OmniVision OV5640 sensor support
< > OmniVision OV5645 sensor support (NEW)
< > OmniVision OV5647 sensor support (NEW)
<*> OmniVision OV5648 sensor support
< > OmniVision OV5670 sensor support (NEW)
<*> OmniVision OV5695 sensor support
< > OmniVision OV6650 sensor support (NEW)
<*> OmniVision OV7251 sensor support
< > OmniVision OV772x sensor support (NEW)
< > OmniVision OV7640 sensor support (NEW)
< > OmniVision OV7670 sensor support (NEW)
< > OmniVision OV7740 sensor support (NEW)
v(+)
<Select> < Exit > < Help > < Save > < Load >

想要配置其他的驱动也是如此。

修改完成后，选择右下角 Save 进行保存，**注意不要保存到原路径，而是保存到.config**，然后使用以下命令来保存 defconfig 文件并覆盖原来的配置文件。




```
# 保存 defconfig 文件
make savedefconfig ARCH=arm64

# 覆盖原来的配置文件
cp defconfig arch/arm64/configs/lubancat2_defconfig
```

这样保存的原因是配置文件默认是精简版本的，编译使用时会和默认的配置文件进行比较从而得到完整的配置，如果直接保存则是完整版本的，会比精简版多几千行配置，不利于观察、修改。

8.4.2 修改内核配置 (LubanCat_Gen_SDK)

在 LubanCat_Gen_SDK 中，除了使用上一小节 **修改内核配置 (LubanCat_Chip_SDK)** 提供的在 kernel 目录下使用 `make menuconfig` 来打开配置界面的方法以外，还新增了在 SDK 顶层目录中直接使用 `./build.sh kconfig` 命令来进入 menuconfig 配置界面，当退出配置界面后，还可以自动生成 defconfig 并保存到对应的配置文件中。

8.5 kernel 编译

提示： 本节需要输入命令的操作，需在 SDK 顶层文件夹下进行

在 LubanCat-SDK 中，自动编译脚本基本上都存放在 build.sh 中，这是 SDK 的主要功能入口

对于 kernel 的编译，有两种方式。一种是 rk 标准的 boot 分区，称为 rkboot 分区 (已弃用)，另一种是野火修改的 extboot 分区。

rkboot 分区是以二进制形式打包的，各个部分存放的地址经过了严格定义，读取文件时通过二进制文件头去判断文件的种类，目前 LubanCat 板卡已停止支持。

而我们修改的 extboot 分区，是以 ext4 格式存储文件的，在系统内是可读可修改的，大大增加了便利性。我们以此为基础，为 extboot 分区系统增加了在线更新内核版本，修改设备树插件，切换主设备树的功能，最终实现了一个镜像通用所有使用同一型号处理器的 LubanCat 板卡的功能。

8.5.1 extboot 分区编译 (Chip)

注意： 以下内容中对 function build_extboot() 的描述仅适用于 LubanCat_Chip_SDK。在 LubanCat_Gen_SDK 中编译流程基本相同，具体内容可以查看 device/rockchip/common/scripts/mk-kernel.sh

由于在打包 extboot 分区时，会打包内核 deb 包，所以在构建 extboot 分区镜像前，我们先要进行下一步骤，构建内核 deb 包。

```
function build_extboot(){
    check_config RK_KERNEL_DTS RK_KERNEL_DEFCONFIG || return 0

    echo "============Start building kernel============"
    echo "TARGET_ARCH           =$RK_ARCH"
    echo "TARGET_KERNEL_CONFIG =$RK_KERNEL_DEFCONFIG"
    echo "TARGET_KERNEL_DTS     =$RK_KERNEL_DTS"
    echo "TARGET_KERNEL_CONFIG_FRAGMENT =$RK_KERNEL_DEFCONFIG_FRAGMENT"
    echo "=========================================="
    pwd

    build_check_cross_compile

    cd kernel
    make ARCH=$RK_ARCH $RK_KERNEL_DEFCONFIG $RK_KERNEL_DEFCONFIG_FRAGMENT
    make ARCH=$RK_ARCH $RK_KERNEL_DTS.img -j$RK_JOBS
    make ARCH=$RK_ARCH dtbs -j$RK_JOBS

    echo -e "\e[36m Generate extLinuxBoot image start\e[0m"

    KERNEL_VERSION=$(cat $TOP_DIR/kernel/include/config/kernel.release)

    EXTBOOT_IMG=${TOP_DIR}/kernel/extboot.img
    EXTBOOT_DIR=${TOP_DIR}/kernel/extboot
    EXTBOOT_DTB=${EXTBOOT_DIR}/dtb/

    rm -rf $EXTBOOT_DIR
    mkdir -p $EXTBOOT_DTB/overlay
    mkdir -p $EXTBOOT_DIR/uEnv
    mkdir -p $EXTBOOT_DIR/kerneldeb

```

(下页继续)

(续上页)

```
32      cp ${TOP_DIR}/$RK_KERNEL_IMG $EXTBOOT_DIR/Image-$KERNEL_VERSION
33
34      if [ "$RK_ARCH" == "arm64" ];then
35          cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/rockchip/*.dtb
    ↪$EXTBOOT_DTB
36          cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/rockchip/overlay/*.
    ↪dtbo $EXTBOOT_DTB/overlay
37          cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/rockchip/uEnv/uEnv*.
    ↪txt $EXTBOOT_DIR/uEnv
38      else
39          cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/*.dtb $EXTBOOT_DTB
40          cp ${TOP_DIR}/kernel/arch/${RK_ARCH}/boot/dts/overlay/*.dtbo
    ↪$EXTBOOT_DTB/overlay
41      fi
42      cp -f $EXTBOOT_DTB/${RK_KERNEL_DTS}.dtb $EXTBOOT_DIR/rk-kernel.dtb
43
44      if [[ -e ${TOP_DIR}/kernel/ramdisk.img ]]; then
45          cp ${TOP_DIR}/kernel/ramdisk.img $EXTBOOT_DIR/initrd-$KERNEL_VERSION
46          echo -e "\tinitrd /initrd-$KERNEL_VERSION" >> $EXTBOOT_DIR/extlinux/
    ↪extlinux.conf
47      fi
48
49      cp ${TOP_DIR}/kernel/.config $EXTBOOT_DIR/config-$KERNEL_VERSION
50      cp ${TOP_DIR}/kernel/System.map $EXTBOOT_DIR/System.map-$KERNEL_VERSION
51      cp ${TOP_DIR}/kernel/logo.bmp $EXTBOOT_DIR/
52
53      cp ${TOP_DIR}/linux-headers-"$KERNEL_VERSION"_"$KERNEL_VERSION"-*.deb
    ↪$EXTBOOT_DIR/kerneldeb
54      cp ${TOP_DIR}/linux-image-"$KERNEL_VERSION"_"$KERNEL_VERSION"-*.deb
    ↪$EXTBOOT_DIR/kerneldeb
55
56      rm -rf $EXTBOOT_IMG && truncate -s 128M $EXTBOOT_IMG
```

(下页继续)

(续上页)

```
57      fakeroot mkfs.ext4 -F -L "boot" -d $EXTBOOT_DIR $EXTBOOT_IMG
58
59      finish_build
60  }
```

1. check_config 检查配置文件是否存在
2. build_check_cross_compile 设置交叉编译参数
3. 应用定义的内核配置文件
4. 编译内核镜像
5. 编译设备树
6. 创建 EXTBOOT_DIR 临时文件夹，存放用于生成 boot 分区内的文件
7. 复制内核镜像、设备树文件、设备树插件、uEnv 环境变量文件
8. 判断是否添加 ramdisk 镜像
9. 复制启动 logo、内核配置文件、System.map、内核 deb 包
10. 以 ext4 格式将 EXTBOOT_DIR 文件夹中的文件打包到 extboot.img

在后续的步骤中，会将 extboot.img 软链接到 rockdev/boot.img

8.6 构建内核 deb 包

我们在内核运行 make bindeb-pkg 会生成至多 5 个 Debian 软件包，在 LubanCat-SDK 中我们可以直接使用以下命令构建。

```
1  ./build.sh kerneldeb
```

其构建脚本如下

注意： 以下内容中对 function build_kerneldeb() 的描述仅适用于 LubanCat_Chip_SDK。在 LubanCat_Gen_SDK 中编译流程基本相同，具体内容可以查看 device/rockchip/common/scripts/mk-kernel.sh

```
./build.sh kerneldeb

function build_kerneldeb(){
    check_config RK_KERNEL_DTS RK_KERNEL_DEFCONFIG || return 0

    build_check_cross_compile

    echo "============Start building kernel deb============"
    echo "TARGET_ARCH           =$RK_ARCH"
    echo "TARGET_KERNEL_CONFIG =$RK_KERNEL_DEFCONFIG"
    echo "TARGET_KERNEL_DTS     =$RK_KERNEL_DTS"
    echo "TARGET_KERNEL_CONFIG_FRAGMENT =$RK_KERNEL_DEFCONFIG_FRAGMENT"
    echo "========================================="
    pwd
    cd kernel
    make ARCH=$RK_ARCH $RK_KERNEL_DEFCONFIG $RK_KERNEL_DEFCONFIG_FRAGMENT
    make ARCH=$RK_ARCH bindeb-pkg RK_KERNEL_DTS=$RK_KERNEL_DTS -j$RK_JOBS
    finish_build
}
```

1. check_config 检查配置文件是否存在
2. build_check_cross_compile 设置交叉编译参数
3. 应用定义的内核配置文件
4. 编译内核 deb 包

生成的软件包如下：

- linux-image-version：一般包括内核镜像和内核模块，我们还添加了设备树及设备树插件。

- linux-headers-version：包括创建外部模块所需的头文件
- linux-firmware-image-version：包括某些驱动程序所需的固件 (这里不生成)
- linux-image-version-dbg：在 linux-image-version 基础上多了 debug 符号
- linux-libc-dev：包括 GNU glibc 之类与用户程序库相关的标头

提示： version 为内核版本

其中我们最主要用到的就是 linux-image 和 linux-headers，而 linux-firmware 我们则可以通过网络进行安装。

通过安装 linux-headers，我们可以直接在板卡上安装 gcc 编译器进行本地编译，具体的使用方法详见应用部署章节

通过安装 linux-image 则可以更新内核镜像、设备树和设备树插件、内核模块。

8.6.1 内核 deb 包的安装

我们将生成的 deb 包通过 USB 存储设备或网络复制到运行 Ubuntu 或 Debian 镜像的板卡中，使用以下命令安装软件包

错误： 注意在更新时不要断电，否则可能会导致系统损坏无法启动。

```
# 内核为 4.19.232 版本
sudo dpkg -i linux-headers-4.19.xxx_4.19.xxx-xxx_arm64.deb
sudo dpkg -i linux-image-4.19.xxx_4.19.xxx-xxx_arm64.deb

# 内核为 5.10.160 版本
sudo dpkg -i linux-headers-5.10.xxx_5.10.xxx-xxx_arm64.deb
sudo dpkg -i linux-image-5.10.xxx_5.10.xxx-xxx_arm64.deb

```

(下页继续)

(续上页)

```
# 内核为 5.10.198 及以上版本
sudo dpkg -i linux-headers-5.10.xxx-芯片型号_5.10.xxx-芯片型号-xxx_arm64.deb
sudo dpkg -i linux-image-5.10.xxx-芯片型号_5.10.xxx-芯片型号-xxx_arm64.deb

# 内核为 6.1.xxx 版本
sudo dpkg -i linux-headers-6.1.xxx-芯片型号_6.1.xxx-芯片型号-xxx_arm64.deb
sudo dpkg -i linux-image-6.1.xxx-芯片型号_6.1.xxx-芯片型号-xxx_arm64.deb
```

注解： 在安装本地 deb 包时，要在 deb 包所在的目录下运行上面命令，xxx 为 deb 包实际对应的数字。

等待 deb 包安装完成后重启，即可完成内核更新。

除了直接使用 deb 包进行本地更新以外，还可以使用野火软件源进行在线更新

```
sudo apt update

# 更新全部要更新的软件包
sudo apt upgrade

# 查看内核版本
uname -a

# 只更新 linux-image 和 linux-headers
# 内核为 4.19.232
sudo apt install linux-image-4.19.232
sudo apt install linux-headers-4.19.232

# 内核为 5.10.160
sudo apt install linux-image-5.10.160
sudo apt install linux-headers-5.10.160

```

(下页继续)

(续上页)

```
18 # 内核为 5.10.198 及以上版本
19 sudo apt install linux-image-5.10.xxx-芯片型号
20 sudo apt install linux-headers-5.10.xxx-芯片型号
21
22 # 内核为 6.1.xxx 版本
23 sudo apt install linux-image-6.1.xxx-芯片型号
24 sudo apt install linux-headers-6.1.xxx-芯片型号
```

升级完成后重启板卡，即可完成内核更新。

第 9 章　修改启动 logo

在板卡开机时，若有连接屏幕，会在屏幕上显示 LubanCat 的 logo，若想要修改启动 logo，可以按照以下步骤操作。

9.1 从 SDK 源码修改启动 logo

9.1.1 准备一张图片

选择自己想要的修改的 logo 图片，例如这里以 LubanCat 的 logo 为例。

将图片底色设置为透明底色，并将格式转换为 8bit 或者 24bit 的 bmp 格式，修改图片分辨率和屏幕的显示比例，并控制图片大小在 500KB 以内。

注解： 如果图片分辨率大于屏幕显示分辨率，将不改变长宽比自动压缩图片。

如果需要进一步缩小 bmp 文件的体积大小，可以使用以下命令将图片由 24 位真彩色模式调整为 8 位调色板模式并进行压缩，对于内容简单的图片可以有效缩小体积。

```
convert logo.bmp -density 72 -units PixelsPerInch -depth 8 -colors 256 -
↪compress RLE BMP3:logo.bmp
```

9.1.2 替换原本的 logo 文件

1、在转换完成后，我们将得到的 bmp 格式的文件分别重命名为 logo.bmp 和 logo_kernel.bmp，替换 SDK 工程 kernel 目录下原有的 logo.bmp 和 logo_kernel.bmp 文件，这就完成了启动 logo 的替换。

2、重新编译整个镜像，烧录到板卡上即可。

9.2 从板卡系统修改启动 logo

如果不从 SDK 源码改启动 logo 可以在烧录好 Debian 镜像或者 Ubuntu 镜像的板卡直接替换/boot 目录下的 logo.bmp 和 logo_kernel.bmp 来更新启动 logo。

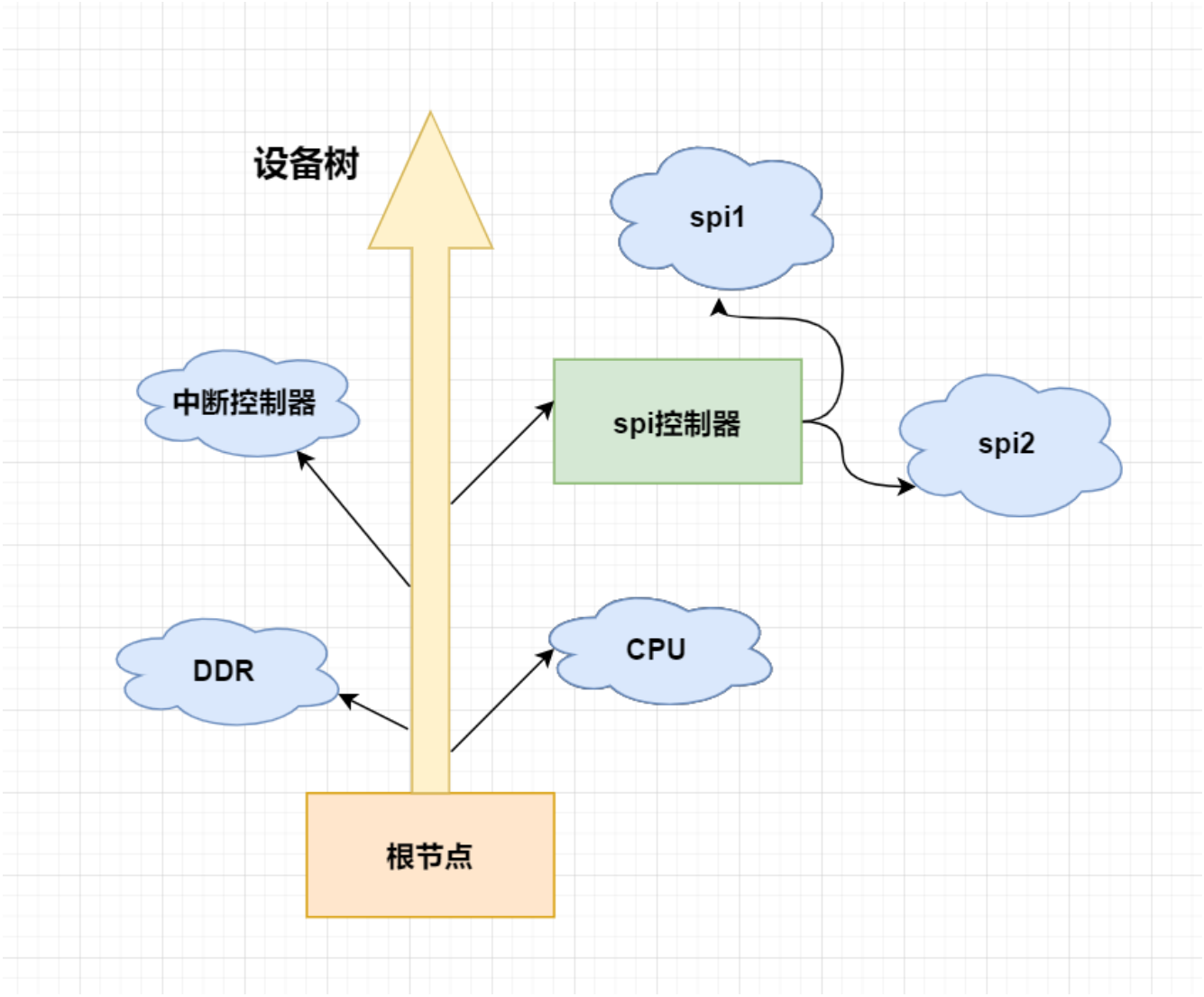
9.3 注意

- 默认开启了启动信息输出到 tty1，会将启动信息显示在屏幕上，如果要关闭输出，修改/boot/uEnv.txt 中的 bootargs 变量，删除 console=tty1。
- 使用设备树插件开启的屏幕，由于 logo 显示在设备树插件加载之前，在启动过程中不会有 logo 显示，而野火系统 mipi 屏幕均使用设备树插件来开启，因此不能显示 logo 在 mipi 屏幕上，要想正常显示需要修改主设备树，将 mipi 屏幕配置添加到主设备树，同时需要注意原来的 HDMI 是否冲突。

第 10 章　设备树的简介

10.1 设备树的介绍

设备树的作用就是描述一个硬件平台的硬件资源。这个“设备树”可以被 bootloader(uboot) 传递到内核，内核可以从设备树中获取硬件信息。



设备树描述硬件资源时有两个特点。

- 第一，以“树状”结构描述硬件资源。例如本地总线为树的“主干”在设备树里面称为“根节点”，挂载到本地总线的 IIC 总线、SPI 总线、UART 总线为树的“枝干”在设备树里称为“根节点的子节点”，IIC 总线下的 IIC 设备不止一个，这些“枝干”又可以再分。
- 第二，设备树可以像头文件（.h 文件）那样，一个设备树文件引用另外一个设备树文件，这样可以实现“代码”的重用。例如多个硬件平台都使用 RK3568 作为主控芯片，那么我们可以将 RK3568 芯片的硬件资源写到一个单独的设备树文件里面一般使用“.dtsi”后缀，其他设备树文件直接使用“#include xxx.dtsi”引用即可。

如 arch/arm64/boot/dts/rockchip/rk3568.dtsi，这个文件一般由芯片厂商提供，其中几乎包含了 rk3568 芯片中所有设备及外设接口。在我们使用时，我们只需要在我们板卡的设备树源文件中 #include “rk3568.dtsi”就可以导入 rk3568 这可芯片所有的设备，然后我们再根据板卡上的外设来修改即可。

10.2 常见的几个 DT

- **DTS** 是指.dts 格式的文件，是一种 ASII 文本格式的设备树描述，也是我们要编写的设备树源码，一般一个.dts 文件对应一个硬件平台。在 Linux 源码的“arch/arm64/boot/dts/”目录下又根据芯片厂商进行分类。
- **DTSI** 是指由芯片厂商提供，是同一芯片平台“共用”的设备树文件。
- **DTC** 是指编译设备树源码的工具，一般情况下我们需要手动安装这个编译工具。
- **DTB** 是设备树源码编译生成的文件，类似于我们 C 语言中“.C”文件编译生成“.bin”文件。
- **DTBO** 是设备树叠加层编译生成的文件，可以对 DTB 进行叠加补充。

10.3 设备树的节点编写

具体详细的编写教程可以参考 Linux 设备树

第 11 章　设备树的编译

在 U-boot 和 kernel 的源码中都有设备树文件。按照 U-Boot 的原生架构要求一块板卡必须对应一份 U-Boot dts，并且 U-Boot dts 生成的 dtb 是打包到 U-Boot 自己的镜像中的。这样就会出现各 SoC 平台上，N 块板卡需要 N 份 U-Boot 镜像。

我们不难发现，其实一个 SoC 平台不同的板卡之间主要是外设的差异，SoC 核心部分是一致的。RK 平台为了实现一个 SoC 平台仅需要一份 U-Boot 镜像，因此增加了 kernel DTB 机制。本质就是在较早的阶段切到 kernel DTB，用它的配置信息初始化外设。

RK 平台通过支持 kernel DTB 可以达到兼容板卡差异，如：display、pmic/regulator、pinctrl、clk 等。

我们可以简单的理解为，一些基础的通用外设使用 U-Boot 的设备树来进行初始化，而不通用的部分通过在 U-Boot 阶段加载内核的设备树来初始化。所以我们在修改设备树时，主要也是去修改内核的设备树。例如在 LubanCat-RK 系列的开发过程中，就没有修改过 U-Boot 的设备树，直接使用了 rk3568-evb 板卡的设备树。

11.1 编译设备树文件

借助 LubanCat-SDK 提供的编译环境，我们在编译 kernel 镜像的同时，也会编译对应的设备树。

在 LubanCat-SDK 顶层文件夹下运行以下命令开编译内核：

```
# 编译 kerneldeb 文件
./build.sh kerneldeb

# 编译 extboot 分区
./build.sh extboot
```

编译生成的设备树文件和设备树源文件在同一目录下，如 LubanCat-2 板卡，其设备树源文件是 arch/arm64/boot/dts/rockchip/rk3568-lubancat-2.dts，编译后生成的设备树文件是 arch/arm64/boot/dts/rockchip/rk3568-lubancat-2.dtb。

11.2 同一板卡更换设备树文件

11.2.1 extboot 分区更换设备树文件

对于使用 extboot 分区的镜像而言，在生成 extboot 镜像时就已经将所有 Makefile 中指定的设备树都打包了进去，板卡第一次启动时使用一个基础设备树，并在第一次系统启动以后根据板卡硬件 ID 自动切换为对应板卡的设备树。

系统的 boot 分区默认挂载在/boot 目录下，如果我们要切换显示设备的话，可以使用配置工具或直接修改/boot/rk-kernel.dtb 软链接到的实际设备树的文件地址。

具体的切换方法请查看：《设备树和设备树插件》

如果想要自己添加设备树的话，可以按以下流程进行：

1. 在 kernel/arch/arm64/boot/dts/rockchip 下新建一个设备树源文件，如名称为 rk3568-lubancat-2-xxx.dts
2. 修改同一目录下的 Makefile，在 **dtb-$(CONFIG_CPU_RK3568) +=** 后添加 rk3568-lubancat-2-xxx.dtb \
3. 使用./build.sh kerneldeb 命令构建内核 deb 包
4. 使用./build.sh extboot 命令构建 extboot 分区
5. 使用./mkfirmware.sh 命令重新打包固件
6. 在 rockdev 目录下查看 boot.img 分区镜像文件
7. 使用分区烧录工具进行烧录

第 12 章　根文件系统的介绍

12.1 根文件系统简介

根文件系统首先是内核启动时所 mount 的第一个文件系统，内核代码映像文件保存在根文件系统中，而系统引导启动程序会在根文件系统挂载之后从中把一些基本的初始化脚本和服务等加载到内存中去运行。相当于 windows 下的 C 盘，保存了系统启动后的应用程序和系统配置

12.2 根文件系统目录简介

/bin 目录

该目录下的命令可以被 root 与一般账号所使用，由于这些命令在挂接其它文件系统之前就可以使用，所以/bin 目录必须和根文件系统在同一个分区中。/bin 目录下常用的命令有：cat、chgrp、chmod、cp、ls、sh、kill、mount、umount、mkdir 等。我们之后在利用 Busybox 制作根文件系统时，在生成的 bin 目录下，可以看到一些可执行的文件，也就是可用的一些命令。

/sbin 目录

该目录下存放系统命令，即只有系统管理员（俗称最高权限的 root）能够使用的命令，系统命令还可以存放在/usr/sbin,/usr/local/sbin 目录下，/sbin 目录中存放的是基本的系统命令，它们用于启动系统和修复系统等，与/bin 目录相似，在挂接其他文件系统之前就可以使用/sbin，所以/sbin 目录必须和根文件系统在同一个分区中。/sbin 目录下常用的命令有：shutdown、reboot、fdisk、fsck、init 等，本地用户自己安装的系统命令放在/usr/local/sbin 目录下。

/dev 目录

该目录下存放的是设备与设备接口的文件，设备文件是 Linux 中特有的文件类型，在 Linux 系统下，以文件的方式访问各种设备，即通过读写某个设备文件操作某个具体硬件。比如通过”dev/ttySAC0” 文件可以操作串口 0，通过” /dev/mtdblock1” 可以访问 MTD 设备的第 2 个分区。比较重要的文件有/dev/null, /dev/zero, /dev/tty, /dev/lp* 等。

/etc 目录

该目录下存放着系统主要的配置文件，例如人员的账号密码文件、各种服务的其实文件等。一般来说，此目录的各文件属性是可以让一般用户查阅的，但是只有 root 有权限修改。对于 PC 上的 Linux 系统，/etc 目录下的文件和目录非常多，这些目录文件是可选的，它们依赖于系统中所拥有的应用程序，依赖于这些程序是否需要配置文件。在嵌入式系统中，这些内容可以大为精减。

/lib 目录

该目录下存放共享库和可加载（驱动程序），共享库用于启动系统。运行根文件系统中的可执行程序，比如：/bin /sbin 目录下的程序。

/home 目录

系统默认的用户文件夹，它是可选的，对于每个普通用户，在/home 目录下都有一个以用户名命名的子目录，里面存放用户相关的配置文件。

/root 目录

系统管理员（root）的主文件夹，即是根用户的目录，与此对应，普通用户的目录是/home 下的某个子目录。

/usr 目录

/usr 目录的内容可以存在另一个分区中，在系统启动后再挂接到根文件系统中的/usr 目录下。里面存放的是共享、只读的程序和数据，这表明/usr 目录下的内容可以在多个主机间共享，这些主要也符合 FHS 标准的。/usr 中的文件应该是只读的，其他主机相关的，可变的文件应该保存在其他目录下，比如/var。/usr 目录在嵌入式中可以精减。

/var 目录

与/usr 目录相反，/var 目录中存放可变的数据，比如 spool 目录（mail,news），log 文件，临时文件。

/proc 目录

这是一个空目录，常作为 proc 文件系统的挂接点，proc 文件系统是个虚拟的文件系统，它没有实际的存储设备，里面的目录，文件都是由内核临时生成的，用来表示系统的运行状态，也可以操作其中的文件控制系统。

/mnt 目录

用于临时挂载某个文件系统的挂接点，通常是空目录，也可以在里面创建一引起空的子目录，比如/mnt/cdram /mnt/hda1 。用来临时挂载光盘、移动存储设备等。

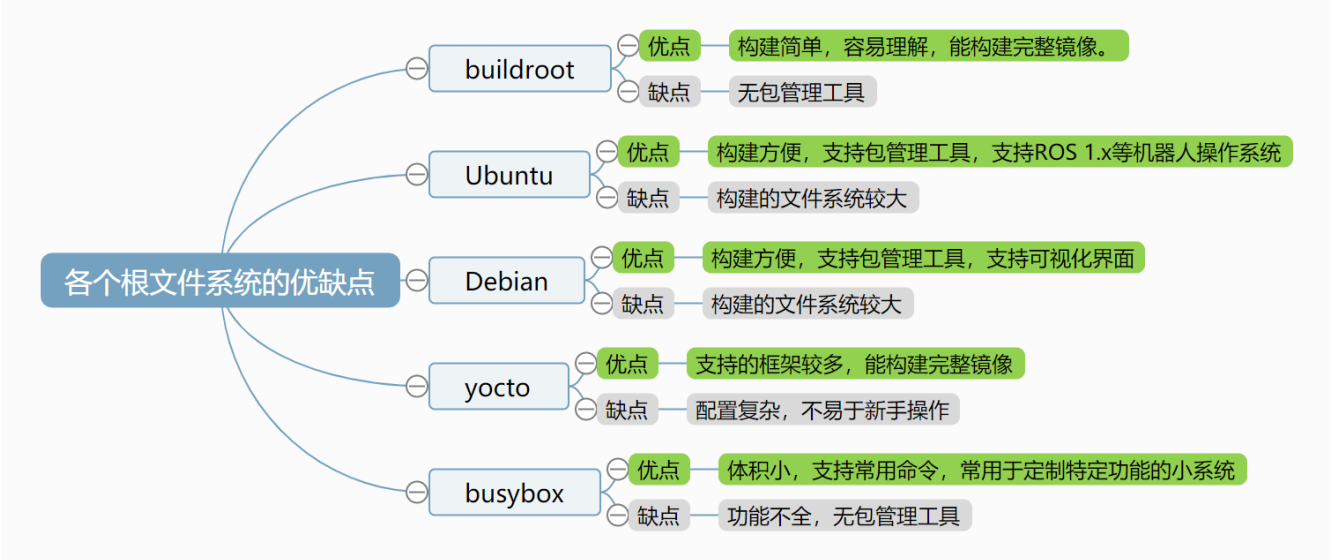
/tmp 目录

用于存放临时文件，通常是空目录，一些需要生成临时文件的程序用到的/tmp 目录下，所以/tmp 目录必须存在并可以访问。

对于嵌入式 Linux 系统的根文件系统来说，一般可能没有上面所列出的那么复杂，比如嵌入式系统通常都不是针对多用户的，所以/home 这个目录在一般嵌入式 Linux 中可能就很少用到，而/boot 这个目录则取决于你所使用的 BootLoader 是否能够重新获得内核映象从你的根文件系统在内核启动之前。一般说来，只有/bin，/dev，/etc，/lib，/proc，/var，/usr 这些需要的，而其他都是可选的。

12.3 常见的根文件系统

根文件系统本质都是一些文件和配置信息组成，之间的界限没有很明确的界定，以下大致列出一些根文件系统的特点



接下来的章节我们将会讲解各种根文件系统的构建方法

第 13 章　Debian 根文件系统构建

借助 LubanCat-SDK 我们可以方便的一键构建 Debian 镜像，但是 Debian 根文件系统的构建过程是相对独立的，不依赖 SDK 的其他部分。

Debian 根文件系统借助 live build 来进行构建，live build 是一组用于构建实时系统映像的脚本。live build 是一个工具套件，它使用一个配置目录来完全自动化和定制构建 live 镜像的所有方面。

除了 live build 外，我们的 Debian 构建仓库还有很多构建脚本，可以最大化的减少人工操作，使构建的根文件系统具有一致性。

13.1 Debian 系统支持情况

13.1.1 LubanCat_Linux_Generic_SDK

根据内核版本，Kernel-5.10 支持 Debian 11 “Bullseye”，Kernel-6.1 支持 Debian 12 “Bookworm”

- lite 无桌面版本
- gnome 桌面版本 (默认桌面)
- xfce 桌面版本 (默认桌面)

13.1.2 LubanCat_Linux_rk356x_SDK

仅支持 Debian 10 “Buster”。

- lite 无桌面版本
- xfce4 桌面版本
- xfce4-full 桌面版本

13.1.3 LubanCat_Linux_rk3588_SDK

仅支持 Debian 11 “Bullseye”。

- lite 无桌面版本
- gnome 桌面版本 (默认桌面)
- xfce 桌面版本 (无技术支持)
- lxde 桌面版本 (无技术支持)

13.1.4 Debian 源码仓库说明

Debian10 https://github.com/LubanCat/debian

- stable-4.19-rk356x 和 debian10 为同一分支，适用于 LubanCat_Linux_rk356x_SDK

Debian11 https://github.com/LubanCat/debian11

- stable-5.10-rk3588：适用于 LubanCat_Linux_rk3588_SDK
- debian11：适用于 LubanCat_Linux_Generic_SDK 适用 5.10 内核的板卡

Debian12 https://github.com/LubanCat/debian12

- debian12：适用于 LubanCat_Linux_Generic_SDK 适用 6.1 内核的板卡

13.2 什么是 Debian

以下文档以 Debian10 为例进行演示，Debian11 内容类似。如使用 Debian11，将相关内容或脚本中的”debian10“和”buster“替换为“debian11”和“bullseye”即可

Debian GNU/Linux(简称 Debian), 是目前世界最大的非商业性 Linux 发行版之一, 是一种完全自由开放并广泛用于各种设备的 Linux 操作系统。

Debian 的特点:

- 面向用户的 Debian
  - Debian 是自由软件，每個人都能自由使用、修改，以及分发。
  - 稳定且安全。
  - 广泛的硬件支持。
  - 提供平滑的更新。
  - 是许多其他发行版的基础。
  - Debian 项目是一個社区。
- 面向开发者的 Debian
  - 支持多种硬件架构，包括 AMD64、i386，ARM 和 MIPS 等
  - 支持物联网和嵌入式设备
  - 拥有大量的套件，使用 deb 格式
  - 不同的发布版本
  - 公开的错误跟踪系统

Debian 官网：https://www.debian.org/

13.3 Debian 根文件系统构建仓库

在我们发布的 LubanCat-SDK 中就已经包含了完整的 Debian 根文件系统构建项目，保存在 SDK 的 debian 目录下。

进入 debian 目录下，有以下文件

```
ls -hgG

-rwxrwxr-x  1 1.1K 3 月  10 11:04 mk-base-debian.sh
-rwxrwxr-x  1 8.1K 3 月  10 11:25 mk-buster-rootfs.sh
-rwxrwxr-x  1  477 3 月  10 10:49 mk-image.sh
-rwxrwxr-x  1  441 12 月 26 15:18 mk-rootfs.sh
drwxrwxr-x  7   67 11 月 30 11:19 overlay
drwxrwxr-x  5   49 11 月 30 11:19 overlay-debug
drwxrwxr-x  4   28 11 月 30 11:19 overlay-firmware
drwxrwxr-x  3   19 11 月 30 11:19 packages
drwxrwxr-x  5   47 11 月 30 11:19 packages-patches
-rwxrwxr-x  1 3.0K 11 月 30 11:19 post-build.sh
-rw-rw-r--  1 2.6K 3 月  10 11:17 readme.md
drwxrwxr-x  7  164 11 月 30 11:19 ubuntu-build-service
```

- mk-base-debian.sh: 清理构建目录并调用 live build 开始构建。
- mk-buster-rootfs.sh：添加 Rockchip overlay 层。
- mk-image.sh：将根文件系统打包成 img 镜像文件
- overlay：Rockchip overlay 层，主要是 rootfs 中的配置文件
- overlay-debug：Rockchip overlay 层，主要是 debug 脚本和工具
- overlay-firmware：Rockchip overlay 层，主要是 wifi/bt/npu 的固件
- packages：硬件加速包
- ubuntu-build-service：用于搭建构建环境的依赖文件和 live build 配置文件

目前构建脚本支持三种版本的镜像构建

- lite：无桌面, 终端版
- xfce：使用 xfce 套件的桌面版
- xfce-full：使用 xfce 套件 + 更多推荐软件包的桌面版

13.4 Debian 根文件系统构建流程

Debian 根文件系统的构建主要分为三个步骤

第一步：使用 live build 工具构建 lite-debian 或 xfce-debian 根文件系统。使用 mk-base-debian.sh 脚本实现。

第二步：添加基于 RK 处理器进行功能增强的软件包如 GPU 驱动和硬件 firmware 等。使用 mk-buster-rootfs.sh 脚本实现。

第三步：将构建完成的根文件系统打包成 img 格式，方便烧录和下一步处理。通过 mk-image.sh 脚本实现，在第二步完成后自动调用。

13.5 搭建构建环境

在 debian 目录下执行以下命令

```
sudo apt-get install binfmt-support qemu-user-static
sudo dpkg -i ubuntu-build-service/packages/*
sudo apt-get install -f
```

上面的命令执行过程中可能有警告或报错，这是正常现象，我们直接忽略报错即可。

```
jiawen@dev120:~/RK356X_LINUX_SDK/debian$     sudo apt-get install binfmt-support qemu-user-static
ges/*
正在读取软件包列表... 完成f
正在分析软件包的依赖关系树
正在读取状态信息... 完成
binfmt-support 已经是最新版 (2.1.8-2)。
qemu-user-static 已经是最新版 (1:2.11+dfsg-1ubuntu7.40)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 39 个软件包未被升级。
jiawen@dev120:~/RK356X_LINUX_SDK/debian$     sudo dpkg -i ubuntu-build-service/packages/*
dpkg: 警告: 即将把 debootstrap 从 1.0.123 降级到 1.0.87
(正在读取数据库 ... 系统当前共安装有 193074 个文件和目录。)
正准备解包 .../debootstrap_1.0.87_all.deb  ...
正在将 debootstrap (1.0.87) 解包到 (1.0.123) 上 ...
dpkg: 警告: 即将把 linaro-image-tools 从 2016.05-1 降级到 2012.12-0ubuntu1~linaro1
正准备解包 .../linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb  ...
正在将 linaro-image-tools (2012.12-0ubuntu1~linaro1) 解包到 (2016.05-1) 上 ...
dpkg: 警告: 即将把 live-build 从 1:20210902 降级到 3.0.5-1linaro1
正准备解包 .../live-build_3.0.5-1linaro1_all.deb  ...
正在将 live-build (3.0.5-1linaro1) 解包到 (1:20210902) 上 ...
dpkg: 警告: 即将把 python-linaro-image-tools 从 2016.05-1 降级到 2012.12-0ubuntu1~linaro1
正准备解包 .../python-linaro-image-tools_2012.12-0ubuntu1~linaro1_all.deb  ...
正在将 python-linaro-image-tools (2012.12-0ubuntu1~linaro1) 解包到 (2016.05-1) 上 ...
正在设置 debootstrap (1.0.87) ...
正在设置 live-build (3.0.5-1linaro1) ...
dpkg: 依赖关系问题使得 python-linaro-image-tools 的配置工作不能继续:
 python-linaro-image-tools 依赖于 python-support (>= 0.90.0)；然而:
  未安装软件包 python-support。

dpkg: 处理软件包 python-linaro-image-tools (--install)时出错:
 依赖关系问题 - 仍未被配置
dpkg: 依赖关系问题使得 linaro-image-tools 的配置工作不能继续:
 linaro-image-tools 依赖于 python-linaro-image-tools (>= 2012.12-0ubuntu1~linaro1)；然而:
  软件包 python-linaro-image-tools 尚未配置。

dpkg: 处理软件包 linaro-image-tools (--install)时出错:
 依赖关系问题 - 仍未被配置
正在处理用于 man-db (2.8.3-2ubuntu0.1) 的触发器 ...
在处理时有错误发生:
 python-linaro-image-tools
 linaro-image-tools
```

```
jiawen@dev120:~/RK356X_LINUX_SDK/debian$     sudo apt-get install -f
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
正在修复依赖关系... 完成
将会同时安装下列软件：
  python-linaro-image-tools
下列软件包将被升级：
  python-linaro-image-tools
升级了 1 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 39 个软件包未被升级。
有 2 个软件包没有被完全安装或卸载。
需要下载 115 kB 的归档。
解压缩后会消耗 61.4 kB 的额外空间。
您希望继续执行吗？ [Y/n]
获取:1 http://mirrors.ustc.edu.cn/ubuntu bionic/universe amd64 python-linaro-image-tools all 2016.05
-1.1 [115 kB]
已下载 115 kB，耗时 0秒 (565 kB/s)
(正在读取数据库 ... 系统当前共安装有 193066 个文件和目录。)
正准备解包 .../python-linaro-image-tools_2016.05-1.1_all.deb  ...
正在将 python-linaro-image-tools (2016.05-1.1) 解包到 (2012.12-0ubuntu1~linaro1) 上 ...
正在设置 python-linaro-image-tools (2016.05-1.1) ...
正在设置 linaro-image-tools (2012.12-0ubuntu1~linaro1) ...
jiawen@dev120:~/RK356X_LINUX_SDK/debian$
```

13.6 构建 Debian 根文件系统镜像

在 ubuntu-build-service 目录下，根据 lite 或 xfce 版本，armhf 或 arm64 架构的不同，已经保存了 live build 的一些预设文件，如软件包列表、用户名、密码、用户组、时区等配置。

我们使用 mk-base-debian.sh 脚本来调用 live build 构建相应的 Debian 根文件系统。

理论上生成的根文件系统已经能在我们的板卡上运行了，不过还没有添加针对板卡的配置，如网络，显示等，只能运行核心的服务。

下面我们来看一下具体的构建过程：

13.6.1 构建 debian-base 基础根文件系统

我们在 debian 目录下运行下面的命令

```
./mk-base-debian.sh
```

选择要构建的 Debian 版本, 这里我们选择 xfce 版本，输入 2 并按下 Enter 按键，根据提示输入用户密码

```
-------------------------------------------------------
please enter TARGET version number:
请输入要构建的根文件系统版本:
[0] Exit Menu
[1] lite
[2] xfce
[3] xfce-full
-------------------------------------------------------
2
```

整个构建时间较长，等待命令结束以后我们看一下 debian 目录下的文件变化：

```
ls -hgG

-rw-rw-rw-  1 370M 3 月  10 14:04 linaro-buster-xfce-alip-20230418.tar.gz
-rwxrwxr-x  1 1.1K 3 月  10 11:04 mk-base-debian.sh
-rwxrwxr-x  1 8.1K 3 月  10 11:25 mk-buster-rootfs.sh
-rwxrwxr-x  1  477 3 月  10 10:49 mk-image.sh
-rwxrwxr-x  1  441 12 月 26 15:18 mk-rootfs.sh
drwxrwxr-x  7   67 11 月 30 11:19 overlay
drwxrwxr-x  5   49 11 月 30 11:19 overlay-debug
drwxrwxr-x  4   28 11 月 30 11:19 overlay-firmware
drwxrwxr-x  3   19 11 月 30 11:19 packages
drwxrwxr-x  5   47 11 月 30 11:19 packages-patches
-rwxrwxr-x  1 3.0K 11 月 30 11:19 post-build.sh
```

(下页继续)

(续上页)

```
-rw-rw-r--  1 2.6K 3 月  10 11:17 readme.md
drwxrwxr-x  7  164 11 月 30 11:19 ubuntu-build-service
```

新增的文件 linaro-buster-xfce-alip-20230418.tar.gz 就是刚刚通过 live build 构建的基础 debian 根文件系统的压缩包

13.6.2 构建完整的 debian 根文件系统

通过上一步骤构建的根文件系统已经可以在板卡上运行了，为了进一步优化在 LubanCat 上运行的效果，我们还要添加 Rockchip overlay 层，里面主要是一些配置文件和固件，用于添加或覆盖根文件系统中原有的配置文件。

```
./mk-buster-rootfs.sh
```

选择要构建的 Debian 版本, 这里我们选择 xfce 版本，输入 2 并按下 Enter 按键，根据提示输入用户密码

```
-------------------------------------------------------------
please enter TARGET version number:
请输入要构建的根文件系统版本:
[0] Exit Menu
[1] lite
[2] xfce
[3] xfce-full
-------------------------------------------------------------
2
```

```
drwxr-xr-x 23  270 3 月  10 14:57 binary
-rw-rw-rw-  1 370M 3 月  10 14:04 linaro-buster-xfce-alip-20230418.tar.gz
-rw-rw-r--  1 1.3G 3 月  10 14:11 linaro-xfce-rootfs.img
-rwxrwxr-x  1 1.1K 3 月  10 11:04 mk-base-debian.sh
-rwxrwxr-x  1 8.1K 3 月  10 11:25 mk-buster-rootfs.sh
```

(下页继续)

(续上页)

```
-rwxrwxr-x  1   477 3 月  10 10:49 mk-image.sh
-rwxrwxr-x  1   441 12 月 26 15:18 mk-rootfs.sh
drwxrwxr-x  7    67 11 月 30 11:19 overlay
drwxrwxr-x  5    49 11 月 30 11:19 overlay-debug
drwxrwxr-x  4    28 11 月 30 11:19 overlay-firmware
drwxrwxr-x  3    19 11 月 30 11:19 packages
drwxrwxr-x  5    47 11 月 30 11:19 packages-patches
-rwxrwxr-x  1 3.0K 11 月 30 11:19 post-build.sh
-rw-rw-r--  1 2.6K 3 月  10 11:17 readme.md
drwxrwxr-x  7   164 11 月 30 11:19 ubuntu-build-service
```

构建完成后，在 debian 目录下增加了 binary 目录。里面存放的是解压后的根文件系统，我们将要覆盖或添加的文件复制进去，然后通过 chroot 命令进行修改。

13.6.3 打包 debian-lite 根文件系统镜像

在脚本./mk-buster-rootfs.sh 的最后，自动调用了 IMAGE_VERSION=$TARGET ./mk-image.sh 脚本来打包镜像, TARGET 就是我们在运行脚本时选择的 Debian 版本。

脚本运行完成后，我们就得到了名为 linaro-xfce-rootfs.img 的根文件系统镜像文件。

13.7 定制 Debian 根文件系统

由于镜像体积的限制，我们提供的定制 Debian 镜像预装了一部分常用软件，但是在用户开发时可能还需要预装更多的软件，以及对根文件系统做进一步的定制。以下部分将对根文件系统的修改做具体说明。

13.7.1 添加预装软件包

对于预装软件的添加，我们建议放在 mk-buster-rootfs.sh 脚本中，这样在我们做修改以后，只要重复添加 Rockchip overlay 层和打包 img 镜像的过程即可，可以节约大量的开发时间。

比如我们想要预装 git 和 vim 到根文件系统中，则可以在 mk-buster-rootfs.sh 添加以下内容

```
export APT_INSTALL="apt-get install -fy --allow-downgrades"
# 添加的位置在 export APT_INSTALL 下一行

# 添加的内容是
echo -e "\033[47;36m ---------- LubanCat -------- \033[0m"
\${APT_INSTALL} git vim
```

13.7.2 添加外设 firmware

如果我们使用无线网卡这样的外设，就需要向根文件系统中添加网卡的 firmware，这时直接将对应的 firmware 存放在 overlay-firmware/目录下，按根文件系统中的路径保存。

13.7.3 添加服务项及配置文件

我们希望对有些服务项的配置进行自定义，就可以在 overlay/目录下添加对应的配置文件。制作根文件系统的过程中，在添加 Rockchip overlay 层的时候，就会添加或替换根文件系统中原有的配置文件，以实现对配置文件自定义的效果。

这里我们以 Debian 控制台登录时的 banner 配置为例，他的配置文件在根文件系统的/etc/update-motd.d 目录下，这里对应的是 overlay/etc/update-motd.d/目录。

我们在 overlay/etc/update-motd.d/中新建一个名为 00-header 的文件，在文件中添加以下内容：

```
#!/bin/sh
#

```

(下页继续)

(续上页)

```
[ -r /etc/lsb-release ] && . /etc/lsb-release

printf "\n"
echo "\e[1;34m
        OO                              OO
        OOOO                          OOOO
        OOOOO                       OOOOOO
        OOOOOOO                   .OOOOOOO
        OOOOOOOOO               OOOOOOOOO
        OOOOOOOOOOO           OOOOOOOOOOO
        OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
        OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
        OOOOO\    OOOOOOOOOOOO    /OOOOO
        OOOOOOOOO    OOOOOO    OOOOOOOO
        OOOOO/    OOOOOOOOOOOOO   \OOOOO
        OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
        OOOOOOOOOOOOOO\ - /OOOOOOOOOOOOOO
        OOOOOOOOOOOOOOO\/OOOOOOOOOOOOOOO
        OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
 _          _                  ____      _
| |   _   _| |__   __ _ _ __  / ___|__ _| |_
| |  | | | | '_ \ / _' | '_ \| |   / _' | __|
| |__| |_| | |_) | (_| | | | | |__| (_| | |_
|_____\__._|_.__/ \__'_|_| |_|\____\__'_|\__|
\e[0m"

if [ -z "$DISTRIB_DESCRIPTION" ] && [ -x /usr/bin/lsb_release ]; then
    # Fall back to using the very slow lsb_release utility
    DISTRIB_DESCRIPTION=$(lsb_release -s -d)
fi
printf "\n"
printf "Welcome to %s (%s %s %s)\n" "$DISTRIB_DESCRIPTION" "$(uname -o)" "$(uname -r)" "$(uname -m)"
```

这个脚本的作用是生成动态的 banner。

在添加配置文件以后，我们重新构建镜像，再烧录到板卡启动，就可以达到我们想要的 banner 效果。

> **注意：** 如果添加的是 shell 脚本，需要在创建文件后修改文件权限为 775，否则在根文件系统中可能无法执行。

13.7.4 重新打包根文件系统镜像

在对根文件系统的构建脚本做出修改之后，我们要重新打包 debian-xfce/lite-rootfs.img 镜像。

- 如果我们没有修改 live build 的配置文件，则不需要重复构建基础镜像部分。如果修改了配置文件，则需重新构建，要手动删除基础镜像压缩包，然后使用 mk-base-debian.sh 脚本构建。

```
# 删除基础镜像压缩包
rm linaro-*-alip-*.tar.gz

# 构建基础镜像
mk-base-debian.sh
```

- 如果修改了 mk-buster-rootfs.sh、overlay、overlay-debug、overlay-firmware、packages 的内容，则需要执行

```
# 构建完整的根文件系统镜像
./mk-buster-rootfs.sh
```

13.8 使用 LubanCat-SDK 一键构建

在完成 **搭建构建环境**以后，我们也可以直接使用一键构建命令，就可以构建我们提供的定制根文件系统镜像了，还可以借助 SDK 的镜像打包功能，将 U-boot、内核等部分也一并打包成一个完整的系统镜像。

13.8.1 SDK 配置文件说明 (Gen)

LubanCat 板卡的 SDK 配置文件在 device/rockchip/.chips 目录内按照处理器型号存放，以 LubanCat_CPU 型号 _ 系统类型 _ 系统版本 _defconfig 命名

由于 LubanCat_Linux_Generic_SDK 的配置系统，可以处理各模块和参数的依赖关系，此处存放的仅为最小配置文件，完整的 SDK 配置文件在 SDK 构建时会自动生成，并保存在 output/.config 中

13.8.2 SDK 配置文件说明 (Chip)

LubanCat 板卡的 SDK 配置文件存放在 device/rockchip/rk356x/目录内，以 BoardConfig-LubanCat-CPU 型号-系统类型-系统版本.mk 来命名

我们来看 Debian 根文件系统的配置文件，这里以 BoardConfig-LubanCat-RK3568-debian-xfce.mk 为例，主要说明与 Debian 根文件相关的设置。

```
1  # SOC
2  export RK_SOC=rk356x
3
4  # build.sh save 打包时名称
5  export RK_PKG_NAME=lubancat-${RK_UBOOT_DEFCONFIG}
6
7  # 定义默认 rootfs 为 debian
8  export RK_ROOTFS_SYSTEM=debian
9
10 # 设置 debian 版本 (10: buster)
11 export RK_DEBIAN_VERSION=10
```

(下页继续)

(续上页)

```
12
13 # 定义默认 rootfs 是否为桌面版  xfce : 桌面版        lite : 控制台版 xfce-full : 桌
   面版 + 推荐软件包
14 export RK_ROOTFS_TARGET=xfce
15
16 # 定义默认 rootfs 是否添加 DEBUG 工具  debug : 添加  none : 不添加
17 export RK_ROOTFS_DEBUG=debug
```

- RK_SOC: 定义 SOC 类型, 在./mk-buster-rootfs.sh 中使用
- RK_PKG_NAME：定义镜像发布打包时的名称
- RK_ROOTFS_SYSTEM：定义根文件系统类型
- RK_DEBIAN_VERSION：定义 Debian 发行版，目前只支持 buster
- RK_ROOTFS_TARGET：定义根文件系统版本
- RK_ROOTFS_DEBUG：是否添加 rockchip overlay_debug

13.8.3 build.sh 中的自动构建脚本

Debian 根文件系统的一键构建功能，主要由 build.sh 脚本中的以下函数实现

注意： 以下内容中对 function build_debian() 的描述仅适用于 LubanCat_Chip_SDK。在 LubanCat_Gen_SDK 中编译流程基本相同，具体内容可以查看 device/rockchip/common/scripts/mk-rootfs.sh

```
1 function build_debian(){
2     ARCH=${RK_DEBIAN_ARCH:-${RK_ARCH}}
3     case $ARCH in
4         arm|armhf) ARCH=armhf ;;
5         *) ARCH=arm64 ;;
```

(下页继续)

(续上页)

```
    esac

    echo "=========Start building debian for $ARCH========="
    echo "RK_DEBIAN_VERSION: $RK_DEBIAN_VERSION"
    echo "RK_ROOTFS_TARGET: $RK_ROOTFS_TARGET"
    echo "RK_ROOTFS_DEBUG: $RK_ROOTFS_DEBUG"
    echo " "

    cd debian

    if [[ "$RK_DEBIAN_VERSION" == "stretch" || "$RK_DEBIAN_VERSION" == "9"␣
↪]]; then
        RELEASE='stretch'
    elif [[ "$RK_DEBIAN_VERSION" == "buster" || "$RK_DEBIAN_VERSION" == "10
↪" ]]; then
        RELEASE='buster'
    elif [[ "$RK_DEBIAN_VERSION" == "bullseye" || "$RK_DEBIAN_VERSION" ==
↪"11" ]]; then
        RELEASE='bullseye'
    else
        echo -e "\033[36m please input the os type,stretch or buster...... \
↪033[0m"
    fi

    if [ ! -e linaro-$RK_ROOTFS_TARGET-rootfs.img ]; then
        echo "[ No linaro-$RK_ROOTFS_TARGET-rootfs.img, Run Make Debian␣
↪Scripts ]"
        if [ ! -e linaro-$RELEASE-$RK_ROOTFS_TARGET-alip-*.tar.gz ]; then
            echo "[ build linaro-$RELEASE-$RK_ROOTFS_TARGET-alip-*.tar.gz ]"
            RELEASE=$RELEASE TARGET=$RK_ROOTFS_TARGET ARCH=$ARCH ./mk-base-
↪debian.sh
        fi
```

(下页继续)

(续上页)

```
32
33          RELEASE=$RELEASE TARGET=$RK_ROOTFS_TARGET VERSION=$RK_ROOTFS_DEBUG␣
↪SOC=$RK_SOC ARCH=$ARCH ./mk-rootfs.sh
34      else
35          echo "[     Already Exists IMG,  Skip Make Debian Scripts     ]"
36          echo "[ Delate linaro-$RK_ROOTFS_TARGET-rootfs.img To Rebuild␣
↪Debian IMG ]"
37      fi
38
39      finish_build
40  }
```

其工作流程如下

- 使用 echo 命令打印相关配置信息。
- 判断 linaro-$RK_ROOTFS_TARGET-rootfs.img 是否存在 (**$RK_ROOTFS_TARGET** 为配置文件中定义的是否为桌面版)，存在则跳过构建过程，不存在则运行构建命令。根文件系统构建时间很长，不希望频繁构建根文件系统或使用已经构建好的根文件系统镜像。
- 判断构建基础镜像是否存在，不存在则构建基础镜像，若存在则跳过基础镜像构建过程。除了首次构建外，我们一般不会去修改基础镜像。
- 以基础镜像为基础，添加 rockchip overlay。
- 打包完整镜像。

13.8.4 编译前的准备工作

在编译开始前，首先要确保 SDK 的配置文件与要编译的 rootfs 一致，如果当前配置文件与要编译的 rootfs 不一致，需要先切换配置文件。

```
1  # 选择 SDK 配置文件 LubanCat_Chip_SDK 直接指定
2  ./build.sh BoardConfig-xxx-debian-版本.mk
```

(下页继续)

(续上页)

```
3
4 # 选择 SDK 配置文件 LubanCat_Gen_SDK 直接指定
5 ./build.sh LubanCat_xxx_debian_ 版本_defconfig
6
7 # 选择 SDK 配置文件-按序号选择
8 ./build.sh lunch
```

在设置正确的配置文件后，我们就可以进行下一步的构建工作了

13.8.5 单独构建 rootfs 并打包

我们使用以下命令构建 Debian 根文件系统，

```
1 # 构建 Debian
2 ./build.sh debian
```

编译生成的 rootfs 镜像为 linaro-$RK_ROOTFS_TARGET-rootfs.img，同时被软链接到 rockdev/rootfs.ext4。

注意： 只有不存在 linaro-$RK_ROOTFS_TARGET-rootfs.img 时，才会重新构建 Debian 根文件系统。如果修改了 Debian 根文件系统的配置文件或构建脚本，要先将 buntu/linaro-$RK_ROOTFS_TARGET-rootfs.img 手动删除后再重新构建。

构建完成后就可以将独立的分区表、boot.img、uboot.img 等分区镜像打包成一个完整的镜像了。

在执行以下操作前，请确保已经事先完成了 U-Boot 编译、Kernel 编译、以及刚刚完成的单独构建 rootfs 的过程。

确保无误后，执行以下命令

```
1 # 固件打包
2 ./mkfirmware.sh
```

(下页继续)

(续上页)

```

# 生成 update.img
./build.sh updateimg
```

打包完成后，生成的镜像为 rockdev/update.img，我们可以使用烧录工具将 update.img 烧录到板卡 eMMC 中或 SD 卡中。

13.8.6 一键构建完整镜像

在设置正确的配置文件后, 执行以下命令就可以一键完成 U-Boot、Kernel 和 rootfs 的编译，并生成 update.img 镜像

```
# 一键编译
./build.sh
```

打包完成后，生成的镜像为 rockdev/update.img，我们可以使用烧录工具将 update.img 烧录到板卡 eMMC 中或 SD 卡中。

第 14 章　Ubuntu 根文件系统构建

借助 LubanCat-SDK 我们可以方便的一键构建 Ubuntu 镜像，但是 Ubuntu 根文件系统的构建过程是相对独立的，不依赖 SDK 的其他部分。

我们可以使用 Ubuntu 官方的 Ubuntu-base 根文件系统来构建适配我们板卡的根文件系统。

我们的 Ubuntu 构建仓库由很多构建脚本组成，可以最大化的减少人工操作，使构建的根文件系统具有一致性，同时我们的构建脚本支持 Ubuntu18.04/20.04/22.04 三个版本的构建。

目前 Ubuntu20.04 为我们主要支持的版本，本文档也是以 Ubuntu20.04 为例进行讲解的，不过三个版本的构建方法基本相同，详细构建命令请查看对应分支的 readme 文件。

注解： 如果需要自行构建根文件系统，为了解决不同版本的 Ubuntu 镜像构建环境依赖问题，请参考 **Docker 构建根文件系统**章节

14.1 Ubuntu 系统支持情况

14.1.1 LubanCat_Linux_Generic_SDK

支持 Ubuntu 20.04 “Focal” 和 Ubuntu 22.04 “Jammy”

根据内核版本，使用 Kernel-5.10 的板卡默认使用 Ubuntu20.04, 使用 Kernel-6.1 的板卡默认支持 Ubuntu22.04

- lite 无桌面版本
- gnome 桌面版本 (默认桌面)
- xfce 桌面版本 (默认桌面)

14.1.2 LubanCat_Linux_rk356x_SDK

仅支持 ubuntu 20.04 “Focal”

- lite 无桌面版本
- xfce4 桌面版本
- xfce4-full 桌面版本

14.1.3 LubanCat_Linux_rk3588_SDK

仅支持 ubuntu 20.04 “Focal”

- lite 无桌面版本
- gnome 桌面版本 (默认桌面)
- xfce 桌面版本 (无技术支持)

14.1.4 Ubuntu 源码仓库说明

https://github.com/LubanCat/ubuntu 存档仓库，包含多个版本和分支

- ubuntu18.04：ubuntu18.04 版本，停止支持
- ubuntu20.04：ubuntu20.04 版本，适用于 LubanCat_Linux_Generic_SDK 适用 5.10 内核的板卡，停止更新，在独立仓库中更新
- ubuntu22.04：ubuntu22.04 版本，适用于 LubanCat_Linux_Generic_SDK 适用 6.1 内核的板卡，停止更新，在独立仓库中更新
- stable-4.19-rk356x：适用于 LubanCat_Linux_rk356x_SDK，继续更新中
- stable-5.10-rk3588：适用于 LubanCat_Linux_rk3588_SDK，继续更新中

由于 ubuntu 仓库体积过大，后续拆分为不同版本的独立仓库，并清理历史提交，减小仓库体积，以下是拆分后的仓库

Ubuntu20.04 https://github.com/LubanCat/ubuntu20.04

- ubuntu20.04：ubuntu20.04 版本，适用于 LubanCat_Linux_Generic_SDK 适用 5.10 内核的板卡

Ubuntu22.04 https://github.com/LubanCat/ubuntu22.04

- ubuntu22.04：ubuntu22.04 版本，适用于 LubanCat_Linux_Generic_SDK 适用 6.1 内核的板卡

14.2 什么是 Ubuntu Base

Ubuntu 针对不同的 CPU 架构提供相应的 Ubuntu base 根文件系统，目前提供的架构有 amd64、arm64、armhf、i386、s390x、ppc64.

Ubuntu Base 是用于为特定需求创建自定义映像的最小 rootfs，是 Ubuntu 可以运行的最小环境。

Ubuntu Base 的下载地址是 http://cdimage.ubuntu.com/ubuntu-base/releases

	ubuntu-base-20.04.1-base-amd64.tar.gz.zsync	2020-08-06 15:19	93K
	ubuntu-base-20.04.1-base-arm64.tar.gz	2020-07-31 16:51	25M
	ubuntu-base-20.04.1-base-arm64.tar.gz.zsync	2020-08-06 15:19	88K
	ubuntu-base-20.04.1-base-armhf.tar.gz	2020-07-31 17:09	22M
	ubuntu-base-20.04.1-base-armhf.tar.gz.zsync	2020-08-06 15:19	78K
	ubuntu-base-20.04.1-base-ppc64el.tar.gz	2020-07-31 17:09	32M
	ubuntu-base-20.04.1-base-ppc64el.tar.gz.zsync	2020-08-06 15:19	111K
	ubuntu-base-20.04.1-base-s390x.tar.gz	2020-07-31 16:34	26M
	ubuntu-base-20.04.1-base-s390x.tar.gz.zsync	2020-08-06 15:19	90K
	ubuntu-base-20.04.2-base-amd64.tar.gz	2021-02-01 11:15	26M
	ubuntu-base-20.04.2-base-amd64.tar.gz.zsync	2021-02-04 17:39	93K
	ubuntu-base-20.04.2-base-arm64.tar.gz	2021-02-01 11:30	25M
	ubuntu-base-20.04.2-base-arm64.tar.gz.zsync	2021-02-04 17:39	88K
	ubuntu-base-20.04.2-base-armhf.tar.gz	2021-02-01 11:32	22M
	ubuntu-base-20.04.2-base-armhf.tar.gz.zsync	2021-02-04 17:39	78K
	ubuntu-base-20.04.2-base-ppc64el.tar.gz	2021-02-01 11:16	32M
	ubuntu-base-20.04.2-base-		

14.3 Ubuntu 根文件系统构建仓库

在我们发布的 LubanCat-SDK 中就已经包含了完整的 Ubuntu 根文件系统构建项目，保存在 SDK 的 ubuntu 目录下。

注意： LubanCat_Linux_Generic_SDK 同时包含 ubuntu20.04 和 ubuntu22.04，通过构建脚本自动切换，以下内容也适用

进入 ubuntu 目录下，有以下文件

```
ls -hgG

-rwxrwxr-x  1  765 9 月  29  2022 ch-mount.sh
-rwxrwxr-x  1 7.6K 4 月  18 10:18 mk-base-ubuntu.sh
-rwxrwxr-x  1  841 4 月  12 14:16 mk-image.sh
-rwxrwxr-x  1  11K 4 月  18 10:18 mk-ubuntu-rootfs.sh
drwxrwxr-x  5   39 3 月  31 16:41 overlay
drwxrwxr-x  5   49 9 月  29  2022 overlay-debug
drwxrwxr-x  3   17 9 月  29  2022 overlay-firmware
drwxrwxr-x  3   19 3 月  31 16:41 packages
-rwxrwxr-x  1 1.8K 9 月  29  2022 post-build.sh
-rw-rw-r--  1 1.1K 4 月  18 11:10 readme.md
-rw-rw-r--  1  962 9 月  29  2022 sources.list
drwxrwxr-x  3   58 9 月  29  2022 ubuntu-build-service
```

- mk-base-ubuntu.sh: 在 Ubuntu Base 上安装软件包，以构建基础的 ubuntu 根文件系统。
- mk-ubuntu-rootfs.sh：在基础的 lite 版本根文件系统上添加 Rockchip overlay 层。
- mk-image.sh：将根文件系统打包成 img 镜像文件
- sources.list：软件源地址
- overlay：Rockchip overlay 层，主要是 rootfs 中的配置文件

- overlay-debug：Rockchip overlay 层，主要是 debug 脚本和工具
- overlay-firmware：Rockchip overlay 层，主要是 wifi/bt/npu 的固件
- packages：硬件加速包
- ubuntu-build-service：用于搭建构建环境的依赖文件

目前构建脚本支持五种版本的镜像构建

- lite：无桌面, 终端版
- xfce：使用 xfce 套件的桌面版
- xfce-full：使用 xfce 套件 + 更多推荐软件包的桌面版
- gnome：使用 gnome 套件的桌面版
- gnome-full：使用 gnome 套件 + 更多推荐软件包的桌面版

注意： 由于 gnome 桌面将会耗费更多资源，不建议在 RK356x 处理器的板卡上使用 gnome 镜像

14.4 Ubuntu 根文件系统构建流程

Ubuntu 根文件系统的构建主要分为四个步骤

第一步：从 Ubuntu 官网下载对应版本的最小 rootfs，即下载 Ubuntu Base

第二步：在最小 rootfs 的基础上安装常用的软件包和工具，并根据是否为桌面版来决定是否添加桌面显示套件。这个过程完成后得到基础根文件系统

第三步：在第二步的基础上，我们进一步添加基于 RK 处理器进行功能增强的软件包如 GPU 驱动和硬件 firmware 等。

第四步：将构建完成的根文件系统打包成 img 格式，方便烧录和下一步处理。

一、二步使用 mk-base-ubuntu.sh 脚本来实现；

第三步使用 mk-ubuntu-rootfs.sh 来实现；

第四步通过 mk-image.sh 来实现，在第三步脚本的末尾自动调用。

14.5 搭建构建环境

在 ubuntu 目录下执行以下命令

```
sudo apt-get install binfmt-support qemu-user-static
sudo dpkg -i ubuntu-build-service/packages/*
sudo apt-get install -f
```

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ sudo apt-get install binfmt-support qemu-user-static
[sudo] jiawen 的密码：
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
binfmt-support 已经是最新版 (2.1.8-2)。
qemu-user-static 已经是最新版 (1:2.11+dfsg-1ubuntu7.40)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 37 个软件包未被升级。
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ sudo dpkg -i ubuntu-build-service/packages/*
(正在读取数据库 ... 系统当前共安装有 193074 个文件和目录。)
正准备解包 .../debootstrap_1.0.123_all.deb  ...
正在将 debootstrap (1.0.123) 解包到 (1.0.123) 上 ...
dpkg: 警告: 即将把 linaro-image-tools 从 2016.05-1.1 降级到 2016.05-1
正准备解包 .../linaro-image-tools_2016.05-1_amd64.deb  ...
正在将 linaro-image-tools (2016.05-1) 解包到 (2016.05-1.1) 上 ...
正准备解包 .../live-build_20210902_all.deb  ...
正在将 live-build (1:20210902) 解包到 (1:20210902) 上 ...
dpkg: 警告: 即将把 python-linaro-image-tools 从 2016.05-1.1 降级到 2016.05-1
正准备解包 .../python-linaro-image-tools_2016.05-1_all.deb  ...
正在将 python-linaro-image-tools (2016.05-1) 解包到 (2016.05-1.1) 上 ...
正准备解包 .../qemu_5.2+dfsg-11_amd64.deb  ...
正在将 qemu (1:5.2+dfsg-11) 解包到 (1:5.2+dfsg-11) 上 ...
正在设置 debootstrap (1.0.123) ...
正在设置 live-build (1:20210902) ...
正在设置 python-linaro-image-tools (2016.05-1) ...
正在设置 qemu (1:5.2+dfsg-11) ...
正在设置 linaro-image-tools (2016.05-1) ...
正在处理用于 man-db (2.8.3-2ubuntu0.1) 的触发器 ...
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ sudo apt-get install -f
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 39 个软件包未被升级。
```

上面的命令执行过程中可能有警告或报错，这是正常现象，我们直接忽略报错即可。

14.6 构建 base 镜像

我们以 ubuntu-base 镜像为基础，构建我们自己的 base 镜像。

这里我们自己的 base 镜像是指以 ubuntu-base 镜像为基础，安装了我们指定软件包，并进行一些基础设置，如用户名、密码、用户组、时区等配置的根文件系统。

理论上这个根文件系统已经能在我们的板卡上运行了，不过还没有添加针对板卡的配置，如网络，显示等，只能运行核心的服务。

下面我们来看一下具体的构建过程：

14.6.1 构建基础根文件系统

我们在 ubuntu 目录下运行下面的命令

```
./mk-base-ubuntu.sh
```

选择要构建的 Ubuntu 版本, 这里我们选择 xfce 版本，输入 2 并按下 Enter 按键，根据提示输入用户密码

```
-------------------------------------------------------
please enter TARGET version number:
请输入要构建的根文件系统版本:
[0] Exit Menu
[1] gnome
[2] xfce
[3] lite
[4] gnome-full
[5] xfce-full
-------------------------------------------------------
2
```

等待命令结束以后我们看一下 ubuntu 目录下的文件变化：

```
drwxr-xr-x 19 4.0K 4 月  19 16:13 binary
-rwxrwxr-x  1  765 9 月  29  2022 ch-mount.sh
-rwxrwxr-x  1 7.6K 4 月  18 10:18 mk-base-ubuntu.sh
-rwxrwxr-x  1  841 4 月  12 14:16 mk-image.sh
-rwxrwxr-x  1  11K 4 月  18 10:18 mk-ubuntu-rootfs.sh
drwxrwxr-x  5   39 3 月  31 16:41 overlay
drwxrwxr-x  5   49 9 月  29  2022 overlay-debug
drwxrwxr-x  3   17 9 月  29  2022 overlay-firmware
drwxrwxr-x  3   19 3 月  31 16:41 packages
-rwxrwxr-x  1 1.8K 9 月  29  2022 post-build.sh
-rw-rw-r--  1 1.1K 4 月  18 11:10 readme.md
drwxrwxr-x  2    6 4 月  19 16:13 rootfs
-rw-rw-r--  1  962 9 月  29  2022 sources.list
-rw-rw-r--  1  26M 8 月  30  2022 ubuntu-base-20.04.5-base-arm64.tar.gz
-rw-r--r--  1 756M 4 月  13 16:42 ubuntu-base-xfce-arm64-20230413.tar.gz
drwxrwxr-x  3   58 9 月  29  2022 ubuntu-build-service
```

我们看到，新增了三个文件

- binary: 存放解压后的根文件系统, 构建过程在这个文件夹中进行
- ubuntu-base-20.04.5-base-arm64.tar.gz：在 cdimage.ubuntu.com 下载的 ubuntu-base 根文件系统压缩包
- ubuntu-base-xfce-arm64-20230413.tar.gz：我们通过上述脚本构建好的基础版根文件系统后的压缩包

上面的命令是使用./mk-base-ubuntu.sh 脚本来构建根文件系统。他具体的工作流程如下：

- 下载指定版本的 ubuntu-base 根文件系统压缩包并解压
- 向解压后的根文件系统添加软件源、DNS 服务
- 根据 ARM 架构添加模拟器
- **使用 chroot 来修改根文件系统**

  - 升级并安装系统软件包
  - 创建用户和密码，并设置权限
  - 设定主机名、时区、服务项设置等
  - 清理软件安装缓存
- 将设置好的根文件系统打包成压缩包便于保存管理

经过以上步骤，一个基础版本的根文件系统就制作好了

14.7 构建完整的 Ubuntu 根文件系统镜像

完整版的 Ubuntu 根文件系统镜像主要是添加了 Rockchip overlay 层，里面主要是一些配置文件和固件，用于添加或覆盖根文件系统中原有的配置文件，以达到我们想要的定制效果。

14.7.1 根文件系统构建

我们在 ubuntu 目录下, 根据想要构建的根文件系统版本

```
# 构建完整的根文件系统
./mk-ubuntu-rootfs.sh
```

先选择要构建的 CPU 型号，我们选择 rk3566/rk3568，输入 1 然后按回车键，接着选择要构建的 Ubuntu 版本, 这里我们选择 xfce 版本，输入 2 并按下 Enter 按键，根据提示输入用户密码

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu$ ./mk-ubuntu-rootfs.sh
-----------------------------------------------------------
please enter soc number:
请输入要构建 CPU 的序号:
[0] Exit Menu
[1] rk3566/rk3568
[2] rk3588/rk3588s
-----------------------------------------------------------
```

(下页继续)

(续上页)

```
1
set SOC=rk356x......
-------------------------------------------------------
please enter TARGET version number:
请输入要构建的根文件系统版本:
[0] Exit Menu
[1] gnome
[2] xfce
[3] lite
[4] gnome-full
[5] xfce-full
-------------------------------------------------------
2
```

上述命令中的脚本在最后调用了./mk-image.sh 脚本，会自动将 binary 文件夹中的文件打包成 img 镜像：

等打包过程结束以后我们看一下 ubuntu 目录下的文件。

```
drwxr-xr-x 19 4.0K 4 月  19 16:13 binary
-rwxrwxr-x  1  765 9 月  29  2022 ch-mount.sh
-rwxrwxr-x  1 7.6K 4 月  18 10:18 mk-base-ubuntu.sh
-rwxrwxr-x  1  841 4 月  12 14:16 mk-image.sh
-rwxrwxr-x  1  11K 4 月  18 10:18 mk-ubuntu-rootfs.sh
drwxrwxr-x  5   39 3 月  31 16:41 overlay
drwxrwxr-x  5   49 9 月  29  2022 overlay-debug
drwxrwxr-x  3   17 9 月  29  2022 overlay-firmware
drwxrwxr-x  3   19 3 月  31 16:41 packages
-rwxrwxr-x  1 1.8K 9 月  29  2022 post-build.sh
-rw-rw-r--  1 1.1K 4 月  18 11:10 readme.md
drwxrwxr-x  2    6 4 月  19 16:13 rootfs
-rw-rw-r--  1  962 9 月  29  2022 sources.list
-rw-rw-r--  1  26M 8 月  30  2022 ubuntu-base-20.04.5-base-arm64.tar.gz
```

(下页继续)

(续上页)

```
15 -rw-r--r--  1 756M 4 月  13 16:42 ubuntu-base-xfce-arm64-20230413.tar.gz
16 drwxrwxr-x  3   58 9 月  29  2022 ubuntu-build-service
17 -rw-rw-r--  1 3.5G 4 月  19 15:55 ubuntu-xfce-rootfs.img
```

我们看到，新增了三个文件

- ubuntu-xfce-rootfs.img：打包好的完整根文件系统镜像
- rootfs：mk-image.sh 脚本打包的临时目录

mk-ubuntu-rootfs.sh 脚本的构建流程如下：

- 清空 binary 目录，并将对应的 base 根文件系统压缩包解压
- 根据参数将 packages/overlay/overlay-firmware/overlay-debug 目录中的文件复制到解压后的根文件系统中
- 根据 ARM 架构添加模拟器
- **使用 chroot 来修改根文件系统**
  - 升级并安装系统软件包
  - 安装本地 packages 目录内的硬件加速包
  - 根据上面的 SOC 参数安装对应的 GPU 驱动
  - 清理软件安装缓存
- 调用 mk-image.sh 脚本打包 img 镜像

14.8 定制 Ubuntu 根文件系统

由于镜像体积的限制，我们提供的定制 Ubuntu 镜像预装了一部分常用软件，但是在用户开发时可能还需要预装更多的软件，以及对根文件系统做进一步的定制。以下部分将对根文件系统的修改做具体说明。

14.8.1 添加预装软件包

对于预装软件的添加，我们建议根据不同的版本放在 mk-ubuntu-rootfs.sh 脚本中，这样在我们做修改以后，只要重复添加 Rockchip overlay 层和打包 img 镜像的过程即可，可以节约大量的开发时间。

比如我们想要预装 git 和 vim 到根文件系统中，则可以在 mk-lite/desktop-rootfs.sh 添加以下内容

```
export APT_INSTALL="apt-get install -fy --allow-downgrades"
# 添加的位置在 export APT_INSTALL 下一行

# 添加的内容是
echo -e "\033[47;36m ---------- LubanCat -------- \033[0m"
\${APT_INSTALL} git vim
```

14.8.2 添加外设 firmware

如果我们使用无线网卡这样的外设，就需要向根文件系统中添加网卡的 firmware，这时直接将对应的 firmware 存放在 overlay-firmware/目录下，按根文件系统中的路径保存。

14.8.3 添加服务项及配置文件

我们希望对有些服务项的配置进行自定义，就可以在 overlay/目录下添加对应的配置文件。制作根文件系统的过程中，在添加 Rockchip overlay 层的时候，就会添加或替换根文件系统中原有的配置文件，以实现对配置文件自定义的效果。

这里我们以网络配置工具 netplan 的配置为例，netplan 的配置文件在根文件系统的/etc/netplan/目录下，这里对应的是 overlay/etc/netplan/目录。

我们在 overlay/etc/netplan/中新建一个名为 01-network-manager-all.yaml 的文件，在文件中添加以下内容：

```
network:
    renderer: NetworkManager
    ethernets:
        eth0:
            dhcp4: true
        eth1:
            dhcp4: true
```

上面文件的内容是使用 netplan 配置网络管理器为 NetworkManager，设置 eth0 和 eth1 都开启 dhcp。

在添加配置文件以后，我们重新构建镜像，再烧录到板卡启动，就可以达到我们想要的网络配置效果。

注意： 如果添加的是 shell 脚本，需要在创建文件后修改文件权限为 775，否则在根文件系统中可能无法执行。

14.8.4 重新打包根文件系统镜像

在对根文件系统的构建脚本做出修改之后，我们要重新打包 ubuntu-xfce/lite-rootfs.img 镜像。

- 如果我们没有修改 mk-base-ubuntu.sh 脚本中的内容，则不需要重复构建 base 镜像部分。如果修改了则需要重新构建基础根文件系统

```
./ mk-base-ubuntu.sh
```

- 如果修改了 mk-ubuntu-rootfs.sh、overlay、overlay-debug、overlay-firmware、packages 的内容，则需要执行以下命令

```
./mk-ubuntu-rootfs.sh
```

14.9 使用 LubanCat-SDK 一键构建

在完成 **拉取 Ubuntu 构建仓库**和 **搭建构建环境**这两个步骤以后，我们也可以直接使用一键构建命令，就可以构建我们提供的定制根文件系统镜像了，还可以借助 SDK 的镜像打包功能，将 U-boot、内核等部分也一并打包成一个完整的系统镜像。

14.9.1 SDK 配置文件说明 (Gen)

LubanCat 板卡的 SDK 配置文件在 device/rockchip/.chips 目录内按照处理器型号存放，以 LubanCat_CPU 型号 _ 系统类型 _ 系统版本 _defconfig 命名

由于 LubanCat_Linux_Generic_SDK 的配置系统，可以处理各模块和参数的依赖关系，此处存放的仅为最小配置文件，完整的 SDK 配置文件在 SDK 构建时会自动生成，并保存在 output/.config 中

14.9.2 SDK 配置文件说明 (Chip)

LubanCat 板卡的 SDK 配置文件存放在 device/rockchip/rk356x/目录内，以 BoardConfig-LubanCat-CPU 型号-系统类型-系统版本.mk 来命名

我们来看 Ubuntu 根文件系统的配置文件，BoardConfig-LubanCat-RK3568-ubuntu-xfce.mk 为例，主要说明与 Ubuntu 根文件相关的设置。

```
1  # SOC
2  export RK_SOC=rk356x
3
4  # build.sh save 打包时名称
5  export RK_PKG_NAME=lubancat-${RK_UBOOT_DEFCONFIG}
6
7  # 定义默认 rootfs 为 ubuntu
8  export RK_ROOTFS_SYSTEM=ubuntu
9
10 # 默认 Ubuntu 版本
11 export RK_UBUNTU_VERSION=20.04
```

(下页继续)

(续上页)

```
12
13 # 定义默认 rootfs 是否为桌面版  xfce : 桌面版        lite : 控制台版 xfce-full : 桌
   面版 + 推荐软件包
14 export RK_ROOTFS_TARGET=xfce
15
16 # 定义默认 rootfs 是否添加 DEBUG 工具  debug: 添加   none: 不添加
17 export RK_ROOTFS_DEBUG=debug
```

- RK_SOC: 定义 SOC 类型
- RK_PKG_NAME：定义镜像发布打包时的名称
- RK_ROOTFS_SYSTEM：定义根文件系统类型
- RK_UBUNTU_VERSION：定义 Ubuntu 发行版，一般无需修改，发布打包时使用
- RK_ROOTFS_TARGET：定义根文件系统版本
- RK_ROOTFS_DEBUG：是否添加 rockchip overlay_debug

14.9.3 build.sh 中的自动构建脚本

Ubuntu 根文件系统的一键构建功能，主要由 build.sh 脚本中的以下函数实现

注意： 以下内容中对 function build_ubuntu() 的描述仅适用于 LubanCat_Chip_SDK。在 LubanCat_Gen_SDK 中编译流程基本相同，具体内容可以查看 device/rockchip/common/scripts/mk-rootfs.sh

```
1 function build_ubuntu(){
2     ARCH=${RK_UBUNTU_ARCH:-${RK_ARCH}}
3     case $ARCH in
4         arm|armhf) ARCH=armhf ;;
5         *) ARCH=arm64 ;;
```

(下页继续)

(续上页)

```
    esac

    echo "=========Start building ubuntu for $ARCH========="
    echo "==RK_ROOTFS_DEBUG:$RK_ROOTFS_DEBUG RK_ROOTFS_TARGET:$RK_ROOTFS_TARGET=="
    cd ubuntu


    if [ ! -e ubuntu-$RK_ROOTFS_TARGET-rootfs.img ]; then
        echo "[ No ubuntu-$RK_ROOTFS_TARGET-rootfs.img, Run Make Ubuntu Scripts ]"
        if [ ! -e ubuntu-base-$RK_ROOTFS_TARGET-$ARCH-*.tar.gz ]; then
            ARCH=arm64 TARGET=$RK_ROOTFS_TARGET ./mk-base-ubuntu.sh
        fi

        VERSION=$RK_ROOTFS_DEBUG TARGET=$RK_ROOTFS_TARGET ARCH=$ARCH SOC=$RK_SOC ./mk-ubuntu-rootfs.sh
    else
        echo "[     Already Exists IMG,  Skip Make Ubuntu Scripts     ]"
        echo "[ Delate Ubuntu-$RK_ROOTFS_TARGET-rootfs.img To Rebuild Ubuntu IMG ]"
    fi

    finish_build
}
```

其工作流程如下

- 使用 echo 命令打印相关配置信息。
- 判断 ubuntu-$RK_ROOTFS_TARGET-rootfs.img 是否存在 (**$RK_ROOTFS_TARGET** 为配置文件中定义的是否为桌面版)，存在则跳过构建过程，不存在则运行构建命令。根文件系统构建时间很长，不希望频繁构建根文件系统或使用已经构建好的根文件系统镜像。
- 判断构建 base 镜像是否存在，不存在则构建 base 镜像，若存在则跳过 base 镜像构建过程。

除了首次构建外，我们一般不会去修改 base 镜像。

- 以 base 镜像为基础，添加 rockchip overlay。
- 打包镜像。

14.9.4 编译前的准备工作

在编译开始前，首先要确保 SDK 的配置文件与要编译的 rootfs 一致，如果当前配置文件与要编译的 rootfs 不一致，需要先切换配置文件。

```
# 选择 SDK 配置文件 LubanCat_Chip_SDK 直接指定
./build.sh BoardConfig-xxx-ubuntu-版本.mk

# 选择 SDK 配置文件 LubanCat_Gen_SDK 直接指定
./build.sh LubanCat_xxx_ubuntu_ 版本_defconfig

# 选择 SDK 配置文件-按序号选择
./build.sh lunch
```

在设置正确的配置文件后，我们就可以进行下一步的构建工作了

14.9.5 单独构建 rootfs 并打包

我们使用以下命令构建 Ubuntu 根文件系统，

```
# 构建 Ubuntu
./build.sh ubuntu
```

编译生成的 rootfs 镜像为 ubuntu/ubuntu-$RK_ROOTFS_TARGET-rootfs.img，同时被软链接到 rockdev/rootfs.ext4。

注意： 只有不存在 ubuntu/ubuntu-$RK_ROOTFS_TARGET-rootfs.img 时，才会重新构建 Ubuntu 根文件系统。如果修改了 Ubuntu 根文件系统的配置文件或构建脚本，要先将 buntu/ubuntu-$RK_ROOTFS_TARGET-rootfs.img 手动删除后再重新构建。

构建完成后就可以将独立的分区表、boot.img、uboot.img 等分区镜像打包成一个完整的镜像了。

在执行以下操作前，请确保已经事先完成了 U-Boot 编译、Kernel 编译以及刚刚完成的单独构建 rootfs 的过程。

确保无误后，执行以下命令

```
# 固件打包
./mkfirmware.sh

# 生成 update.img
./build.sh updateimg
```

打包完成后，生成的镜像为 rockdev/update.img，我们可以使用烧录工具将 update.img 烧录到板卡 eMMC 中或 SD 卡中。

14.9.6 一键构建完整镜像

在设置正确的配置文件后，执行以下命令就可以一键完成 U-Boot、Kernel、rootfs 的编译，并生成 update.img 镜像

```
# 一键编译
./build.sh
```

打包完成后，生成的镜像为 rockdev/update.img，我们可以使用烧录工具将 update.img 烧录到板卡 eMMC 中或 SD 卡中。

第 15 章　使用 Docker 构建根文件系统

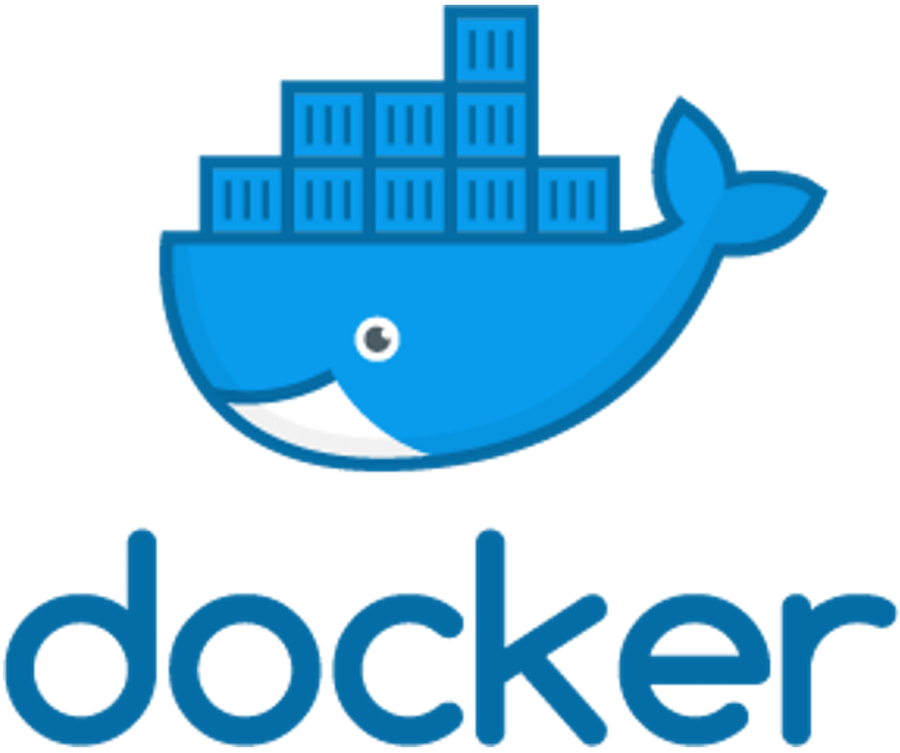
我们的 SDK 使用 Ubuntu20.04 作为标准开发环境，但是由于不同版本的 Ubuntu 镜像构建环境不同，导致我们在构建不同版本的根文件系统时要频繁安装新的构建环境，不仅浪费了大量时间，还容易导致构建失败。

为了解决以上问题，我们借助 Docker 来隔离我们的根文件系统构建环境。

15.1 什么是 Docker

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 或 Windows 操作系统的机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

简单理解，Docker 就是一个小型的虚拟化系统，可以起到与主机环境隔离的作用。下图是 docker 的 logo，是一个小鲸鱼上托着很多集装箱，很生动形象的描述了 docker 的作用。



docker 的官网是 https://www.docker.com/

15.2 Docker 的安装

在不同的操作系统上安装 Docker 有不同的方法，我们主要使用的是 Ubuntu 操作系统。其他系统的安装方法可以参考官方文档 https://docs.docker.com/engine/install/

我们在 Ubuntu 虚拟机上安装 Docker Engine 即可，具体安装过程如下

15.2.1 卸载旧版本

在尝试安装新版本之前卸载旧版本

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

15.2.2 使用存储库安装

在新主机上首次安装 Docker Engine 之前，需要设置 Docker 存储库。之后就可以从存储库安装和更新 Docker。

15.2.2.1 设置存储库

- 更新 apt 包索引并安装相关软件包以允许 apt 通过 HTTPS 使用 docker 存储库：

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
```

- 添加 Docker 的官方 GPG 密钥：

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
↪dearmor -o /etc/apt/keyrings/docker.gpg
```

- 设置存储库：

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/
↪docker.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >␣
↪/dev/null
```

15.2.2.2 安装 Docker engine

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-
↪plugin
```

15.2.2.3 验证安装完成

使用以下命令打印出版本号即为安装成功

```
sudo docker -v

# 打印出版本号
Docker version 20.10.22, build 3a2c30b
```

15.3 创建本地 docker 镜像

由于 Docker 的镜像服务器在国外，访问速度很慢，我们使用本地 Dockerfile 来创建 Docker 镜像。

本地 Dockerfile 存放在 Ubuntu 根文件系统构建的脚本仓库的 ubuntu22.04 分支下，我们克隆相应的分支。

```
git clone --branch=ubuntu22.04 --depth=1 https://github.com/LubanCat/ubuntu.
↪git
```

然后进入克隆的 ubuntu 目录下，执行 docker build 命令构建 Docker 镜像，build_rootfs:2204 是我们设置的镜像名称。

```
cd ubuntu/Docker

sudo docker build -t build_rootfs:2204 .
```

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$ sudo docker build -t build_roo
tfs:2204 .
Sending build context to Docker daemon   2.56kB
Step 1/6 : FROM ubuntu:22.04
 ---> 27941809078c
Step 2/6 : MAINTAINER hejiawen
 ---> Running in 2e0d07e90994
Removing intermediate container 2e0d07e90994
 ---> 69a4fe34ae51
Step 3/6 : RUN sed -i -r 's#http://(archive|security).ubuntu.com#http://mirror
s.tuna.tsinghua.edu.cn#g' /etc/apt/sources.list   && ln -sf /usr/share/zoneinf
o/Asia/Shanghai /etc/localtime   && echo 'Asia/Shanghai' >/etc/timezone
 ---> Running in 2627f4ec1467
Removing intermediate container 2627f4ec1467
 ---> 1cd4785a7aa8
Step 4/6 : RUN apt update   && apt install --no-install-recommends -y locales
apt-utils binfmt-support qemu-user-static      make sudo cpio bzip2 curl wget
language-selector-common   && apt-get autoremove   && apt-get clean   && rm -r
f /var/lib/apt/lists/*
 ---> Running in a81aa91cea2e
Setting up libpython3-stdlib:amd64 (3.10.6-1~22.04) ...
Setting up python3.10 (3.10.6-1~22.04.2) ...
Setting up python3 (3.10.6-1~22.04) ...
running python rtupdate hooks for python3.10...
running python post-rtupdate hooks for python3.10...
Setting up python3-dbus (1.2.18-3build1) ...
Setting up python3-apt (2.3.0ubuntu2.1) ...
Setting up language-selector-common (0.219) ...
Processing triggers for libc-bin (2.35-0ubuntu3) ...
Processing triggers for dbus (1.12.20-2ubuntu4.1) ...
Reading package lists...
Building dependency tree...
Reading state information...
0 upgraded, 0 newly installed, 0 to remove and 24 not upgraded.
Removing intermediate container a81aa91cea2e
 ---> 0f6b83b7a84c
Step 5/6 : RUN localedef -c -f UTF-8 -i zh_CN zh_CN.utf8
 ---> Running in 876c9f4a2b56
Removing intermediate container 876c9f4a2b56
 ---> 50f9fe138cf8
Step 6/6 : ENV LANG zh_CN.utf8
 ---> Running in 847abf762ffc
Removing intermediate container 847abf762ffc
 ---> 4fa5fe7845df
Successfully built 4fa5fe7845df
Successfully tagged build_rootfs:2204
```

等待 docker 镜像构建完成以后，我们用下面命令查看构建好的 docker 镜像。

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$ sudo docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
build_rootfs                2204      4fa5fe7845df   2 minutes ago    278MB
debos-radxa                 1         74091305febd   7 months ago     2.19GB
ubuntu                      22.04     27941809078c   7 months ago     77.8MB
ubuntu                      18.04     ad080923604a   7 months ago     63.1MB
debian                      testing   46d3de8d9f7e   7 months ago     119MB
debian                      buster    354ff99d6bff   7 months ago     114MB
ghcr.io/kendryte/k510_env   latest    e959ea9429f9   9 months ago     809MB
hello-world                 latest    feb5d9fea6a5   15 months ago    13.3kB
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$
```

我们可以看到镜像名是 build_rootfs，他的 TAG 是 2204，和我们设置的符合。他的镜像 ID 是 4fa5fe7845df，大小是 278M。

15.4 创建 docker 容器

执行 docker run 命令运行 docker 镜像，并将 SDK 挂载到运行 docker 镜像的容器中，进入工作目录。

```
1 sudo docker run --name build_rootfs -it --privileged -v /home/jiawen/RK356X_
  ↪LINUX_SDK:/works build_rootfs:2204 /bin/bash
2
3 cd works/
```

- docker run： 运行容器。
- –name build_rootfs：设置容器的名称。
- -it：以交互模式运行容器。
- –privileged：以特权模式运行容器。
- -v /home/jiawen/RK356X_LINUX_SDK:/works 设置挂载到容器的目录，：前是宿主机的路径，：后是挂载到容器的路径，使用绝对路径。这里把整个 SDK 挂载到容器的/works 目录下。
- build_rootfs:2204 容器使用的镜像，这里指定了镜像的名称，使用镜像 ID 也可以。
- /bin/bash 交互模式中使用的 shell 解释器。

```
jiawen@dev120:~/RK356X_LINUX_SDK/ubuntu/Docker$     sudo docker run --name bui
ld_rootfs -it --privileged -v /home/jiawen/RK356X_LINUX_SDK:/works build_rootf
s:2204 /bin/bash
root@27ddbf0ffab2:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  works
root@27ddbf0ffab2:/# cd works/
root@27ddbf0ffab2:/works# ls
app
br.log
buildroot
build.sh
debian
device
```

图中可以看到，执行完运行运行容器的命令后，我们从 jiawen@dev120 变成了 root@27ddbf0ffab2，这说明我们现在已经进入了容器，以 root 用户运行，容器 ID 是 27ddbf0ffab2。

注意此时不要关闭当前的终端，关闭中断则失去了和当前容器的连接。

15.5 构建根文件系统

根文件系统的构建方法和在虚拟机中是一样的，我们可以查看当前根文件系统构建脚本仓库中的 readme 文件，这里做简单演示。

```
# 进入 ubuntu 根文件系统构建脚本仓库
cd /works/ubuntu

# 安装相关构建依赖
apt-get install binfmt-support qemu-user-static
dpkg -i ubuntu-build-service/packages/*
apt-get install -f

# 构建镜像
./mk-base-ubuntu.sh
./mk-ubuntu-rootfs.sh
```

运行完成以上命令后，Ubuntu 根文件系统就构建完成了。

15.6 docker 的相关操作命令

15.6.1 退出当前容器

在当前终端运行 **exit** 即可停止并退出当前容器

15.6.2 查看当前正在运行的容器

```
sudo docker ps
```

-a 选项，查看所有容器，包括退出的和正在运行的。

15.6.3 重新运行停止的容器并进入容器

```
sudo docker start build_rootfs
```

重新运行停止的容器，可以指定容器名称，也可以指定容器 ID

```
sudo docker attach 27ddbf0ffab2
```

进入正在运行的容器，可以指定容器名称，也可以指定容器 ID

15.6.4 删除容器

```
sudo docker rm build_rootfs
```

删除容器，可以指定容器名称，也可以指定容器 ID

-f 选项，强制删除未停止的容器

15.6.5 删除镜像

```
sudo docker rmi build_rootfs:2204
```

删除镜像，可以指定镜像名称:TAG，也可以指定镜像 ID

第 16 章　Buildroot 根文件系统的构建

注意： 当前仅支持使用 LubanCat_Gen_SDK 构建基于 RK3576 和 RK3562 主芯片的鲁班猫板卡的 buildroot 镜像

注意： 包含 buildroot 组件的 LubanCat_Gen_SDK 仅提供源码压缩包，可通过百度网盘获取。压缩包名称为 LubanCat_Linux_Generic_Full_SDK.tgz，此源码压缩包无法通过网络进行在线更新。

错误：buildroot 相关内容不提供技术支持，仅供有能力自行开发的用户使用!!!

buildroot 相关内容不提供技术支持，仅供有能力自行开发的用户使用!!!

buildroot 相关内容不提供技术支持，仅供有能力自行开发的用户使用!!!

16.1 什么是 Buildroot？

Buildroot 是 Linux 平台上一个开源的嵌入式 Linux 系统自动构建框架。整个 Buildroot 是由 Makefile 脚本和 Kconfig 配置文件构成的。你可以和编译 Linux 内核一样，通过 buildroot 配置，menuconfig 修改，编译出一个完整的可以直接烧写到机器上运行的 Linux 系统软件。

借助 LubanCat_Gen_SDK，可以更方便的进行 Buildroot 根文件系统的构建和 Buildroot 系统镜像的制作。

LubanCat_Gen_SDK 使用了 Buildroot-2024.02 作为基础版本，并在此基础上修改和添加了一些由 Rockchip 开发的适用于 RK 处理器软件包，以及由野火添加的适配 LubanCat 板卡功能的软件包，其包含了基于 Linux 系统开发用到的各种系统源码，驱动，工具，应用软件包。

Buildroot 有以下几点优势：

- 通过源码构建，有很大的灵活性；
- 方便的交叉编译环境，可以进行快速构建；
- 方便各系统组件配置及定制开发。

使用 Buildroot 最出名的项目就是 OpenWrt，由它制作出的镜像可以跑在搭载 16Mb SPI Flash 的路由器上，系统基本没包含多余的东西。得益于 Buildroot 的简单化，整个 Buildroot project 可以在一个 git 仓库维护。Buildroot 使用 kconfig 和 make，一个 defconfig 配置代表一种 BSP 支持。

16.1.1 SDK 目录说明

- app：存放供 buildroot 构建使用的应用 demo
- buildroot：存放 buildroot 源码
- external：存放供 buildroot 使用的第三方仓库

16.2 Buildroot 构建

由于 LubanCat_Gen_SDK 中就已经包含了编译所需的 Buildroot 源码和工具链，可以直接进行编译。

在构建 Buildroot 根文件系统前，先要根据对应的板卡选择 SDK 配置文件

目前 LubanCat_Gen_SDK 中的 buildroot 项目支持以下板卡：

- 鲁班猫 1HS 系列, 使用 RK3562/RK3562J(工业级) 主芯片
- 鲁班猫 3 系列, 使用 RK3576 主芯片

```
1 # 选择将要构建的板卡使用的主芯片
2 ./build.sh chip
3
```

(下页继续)

(续上页)

```
# 选择将要使用的配置文件
./build.sh lunch
```

```
jiawen@dev120:~/RK3576_LINUX_SDK$ ./build.sh chip

############### LubanCat Linux SDK ###############

Manifest: lubancat_linux_full_dev.xml

Log colors: message notice warning error fatal

Log saved at /home/jiawen/RK3576_LINUX_SDK/output/sessions/2025-03-06_14-56-00
Pick a chip:

1. rk312x
2. rk3528
3. rk3562
4. rk3566_rk3568
5. rk3576
6. rk3588
Which would you like? [1]: 5
Switching to chip: rk3576
Pick a defconfig:

1. rockchip_defconfig
2. LubanCat_rk3576_buildroot_extboot_defconfig
3. LubanCat_rk3576_buildroot_rkboot_defconfig
4. LubanCat_rk3576_debian_gnome_defconfig
5. LubanCat_rk3576_debian_lite_defconfig
6. LubanCat_rk3576_debian_xfce_defconfig
7. LubanCat_rk3576_ubuntu_gnome_defconfig
8. LubanCat_rk3576_ubuntu_lite_defconfig
9. rockchip_rk3576_evb1_v10_defconfig
10. rockchip_rk3576_industry_evb_v10_defconfig
11. rockchip_rk3576_iotest_v10_defconfig
12. rockchip_rk3576_ipc_evb1_v10_defconfig
13. rockchip_rk3576_multi_ipc_evb1_v10_defconfig
14. rockchip_rk3576_test1_v10_defconfig
15. rockchip_rk3576_test2_v10_defconfig
Which would you like? [1]: 2
Switching to defconfig: /home/jiawen/RK3576_LINUX_SDK/device/rockchip/.chip/LubanCat_rk3576_buildroot_extboot_defconfig
#
# configuration written to /home/jiawen/RK3576_LINUX_SDK/output/.config
#
Using preferred kernel version(6.1)
jiawen@dev120:~/RK3576_LINUX_SDK$
```

每个芯片可选的 buildroot 配置文件有两个，分别是：

- LubanCat_(主芯片型号)_buildroot_extboot_defconfig
- LubanCat_(主芯片型号)_buildroot_rkboot_defconfig

这两个配置文件的主要差异是使用的分区打包格式不同，导致功能有差异。有关 rkboot 和 extboot 的详细差异请查看章节 **LubanCat_Chip_SDK - extboot 与 rkboot 分区对比**此处仅列出 buildroot 镜像的差异

分区格式	extboot	rkboot
支持板卡	所有使用同一主芯片的 LubanCat 板卡	指定型号的 LubanCat 板卡
配置文件	高度适配 LubanCat 板卡	基于官方配置简单修改，仅作为示例，无法直接使用
使用方式	直接使用	需要基于目标板卡修改设备树和 buildroot 配置文件
自动切换设备树	支持	不支持
设备树插件	支持	不支持
支持 OTA	不支持	支持
功能适配	部分 RK 开发的特性不支持	完整支持
开发难度	高 (学习和体验 Buildroot)	超高 (资深开发者进行功能高度定制)

有关 rkboot 配置文件的修改，将会在本章后面的内容介绍

选择了 exboot 分区的配置文件之后，就可以开始编译构建了。

```
# 一键编译完整的 Buildroot 系统镜像
./build.sh

# 只构建 Buildroot 根文件系统
./build.sh buildroot
```

```
========================================
         Start building rootfs(buildroot)
========================================
========================================
         Start building buildroot(2024.02)
========================================

Your network is not able to access sources.buildroot.net!
Please retry later (it could be down for a while) or setup a VPN to bypass the GFW.
Will continue in 5 seconds ...
5 ...
4 ...
3 ...
2 ...
1 ...

make: Entering directory '/home/jiawen/RK3576_LINUX_SDK/buildroot'
  GEN     /home/jiawen/RK3576_LINUX_SDK/buildroot/output/rockchip_rk3576_lubancat/Makefile
Parsing defconfig: /home/jiawen/RK3576_LINUX_SDK/buildroot/configs/rockchip_rk3576_lubancat_defconfig
Using configs/rockchip/base/kernel.config as base
Merging configs/rockchip/fs/e2fs.config
Merging configs/rockchip/base/common.config
Merging configs/rockchip/base/base.config
Merging configs/rockchip/bus/can.config
Merging configs/rockchip/bus/pci.config
Merging configs/rockchip/chips/rk3576.config
Value of BR2_ROOTFS_OVERLAY is redefined by configs/rockchip/chips/rk3576.config:
Previous value: BR2_ROOTFS_OVERLAY="board/rockchip/common/base"
Modify value:   BR2_ROOTFS_OVERLAY+="board/rockchip/rk3576/fs-overlay/"
New value:      BR2_ROOTFS_OVERLAY="board/rockchip/common/base board/rockchip/rk3576/fs-overlay/"

Merging configs/rockchip/chips/rk3576_aarch64.config
Merging configs/rockchip/font/font.config
Merging configs/rockchip/font/chinese.config
Merging configs/rockchip/fs/exfat.config
Merging configs/rockchip/fs/ntfs.config
Merging configs/rockchip/fs/vfat.config
Merging configs/rockchip/gpu/gpu.config
Merging configs/rockchip/multimedia/audio.config
Merging configs/rockchip/multimedia/camera.config
Merging configs/rockchip/multimedia/gst/audio.config
Merging configs/rockchip/multimedia/gst/camera.config
Merging configs/rockchip/multimedia/gst/rtsp.config
Merging configs/rockchip/multimedia/gst/video.config
Merging configs/rockchip/multimedia/mpp.config
```

```
#
# merged configuration written to /home/jiawen/RK3576_LINUX_SDK/buildroot/output/rockchip_rk3576_lubancat/.config.in (needs m
ake)
#
#
# configuration written to /home/jiawen/RK3576_LINUX_SDK/buildroot/output/rockchip_rk3576_lubancat/.config
#
make: Leaving directory '/home/jiawen/RK3576_LINUX_SDK/buildroot'
3c3
< # Buildroot linux-6.1-stan-rkr5-4-ge44264f794-dirty Configuration
---
> # Buildroot linux-6.1-stan-rkr5-4-ge44264f794 Configuration
Buildroot config changed!
You might need to clean it before building:
rm -rf /home/jiawen/RK3576_LINUX_SDK/buildroot/output/rockchip_rk3576_lubancat

2025-03-06T15:29:07 >>> host-gcc-initial 12.4.0 Building
2025-03-06T15:29:15 >>> host-gcc-initial 12.4.0 Installing to host directory
2025-03-06T15:29:25 >>> host-lzip 1.23 Extracting
2025-03-06T15:29:25 >>> host-lzip 1.23 Patching
2025-03-06T15:29:25 >>> host-lzip 1.23 Configuring
2025-03-06T15:29:25 >>> host-lzip 1.23 Building
2025-03-06T15:29:27 >>> host-lzip 1.23 Installing to host directory
2025-03-06T15:29:29 >>> host-make 4.3 Extracting
2025-03-06T15:29:29 >>> host-make 4.3 Patching
2025-03-06T15:29:29 >>> host-make 4.3 Updating config.sub and config.guess
2025-03-06T15:29:29 >>> host-make 4.3 Patching libtool
2025-03-06T15:29:30 >>> host-make 4.3 Configuring
2025-03-06T15:29:45 >>> host-make 4.3 Building
2025-03-06T15:29:46 >>> host-make 4.3 Installing to host directory
2025-03-06T15:29:49 >>> host-autoconf-archive 2023.02.20 Extracting
2025-03-06T15:29:49 >>> host-autoconf-archive 2023.02.20 Patching
2025-03-06T15:29:49 >>> host-autoconf-archive 2023.02.20 Updating config.sub and config.guess
2025-03-06T15:29:49 >>> host-autoconf-archive 2023.02.20 Patching libtool
2025-03-06T15:29:49 >>> host-autoconf-archive 2023.02.20 Configuring
2025-03-06T15:29:51 >>> host-autoconf-archive 2023.02.20 Building
2025-03-06T15:29:51 >>> host-autoconf-archive 2023.02.20 Installing to host directory
2025-03-06T15:29:53 >>> host-expat 2.6.1 Extracting
2025-03-06T15:29:53 >>> host-expat 2.6.1 Patching
2025-03-06T15:29:53 >>> host-expat 2.6.1 Updating config.sub and config.guess
2025-03-06T15:29:53 >>> host-expat 2.6.1 Patching libtool
```

提示： 在 buildroot 构建时，会下载大量的软件包源码存放在 buildroot/dl 目录下，我们已经将 buildroot 默认的软件源修改为国内镜像源，但是由于仍然有部分源码服务器位于国外，所以编译时请保证网络畅通和较高的源码压缩包下载速度，否则会导致构建时间过长或构建失败。

buildroot 的构建目录在 buildroot/output 目录下，这里存放了所有被编译的软件包的源码和构建文件。通过配置文件的命名来区分不同方案的构建目录，这样可以支持多个方案切换而不用清空构建好的内容，提升开发效率。

buildroot/output/latest 指向最后一次构建的方案路径，以 rockchip_rk3576_lubancat 为例

- build：软件包源码解压及编译的目录，包括宿主机所需的工具和目标系统所需的软件包
- host：包含交叉编译工具链、构建工具和宿主机所使用的工具
- images：最终生成的 buildroot 相关镜像保存目录
- staging：目标系统的开发环境，包含编译软件包所需的库和头文件，软链接到了 host/aarch64-buildroot-linux-gnu/sysroot 目录
- target：根文件系统环境，软件包编译后生成的产物将会被安装到这里，最终打包为根文件系统镜像。

构建生成的 buildroot 根文件系统镜像:buildroot/output/latest/images/rootfs.ext4

一键构建完成的完整系统镜像:output/firmware/update.img

16.3 Buildroot 开发流程

16.3.1 修改 Buildroot 配置文件

buildroot 的每一个配置文件就是一个构建方案，配置文件保存在 buildroot/configs 目录下

鲁班猫板卡使用的配置文件是：

- rockchip_rk3576_lubancat_defconfig
- rockchip_rk3562_lubancat_defconfig

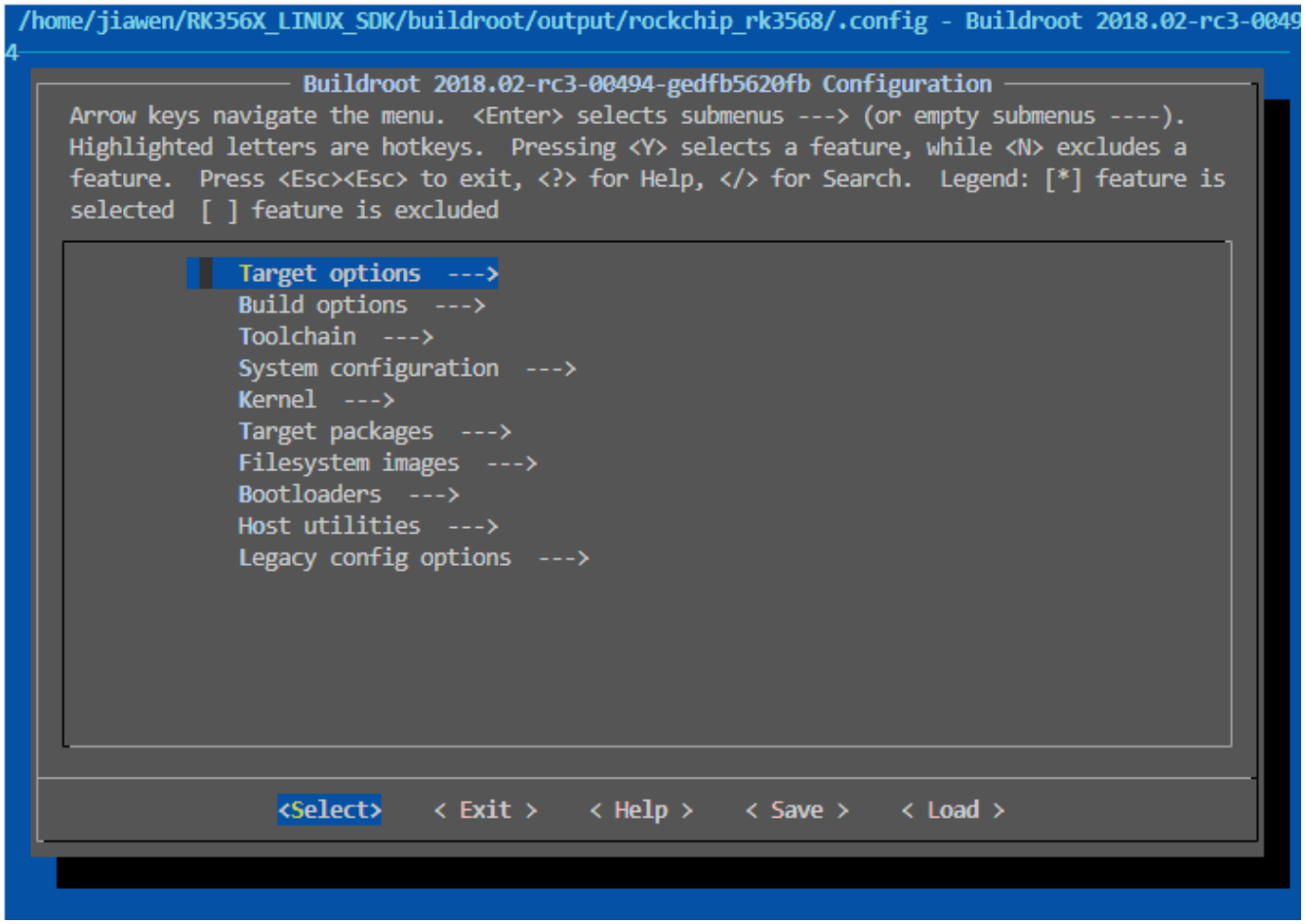
通过观察配置文件，发现配置文件使用了 #include “xxx.config” 的方式来引用其他配置文件。这是由于 Rockchip 对配置文件的格式做了修改，便于用户切换预定义的组件。

除 #include 外的其他内容就是野火针对 LubanCat 板卡做的修改。

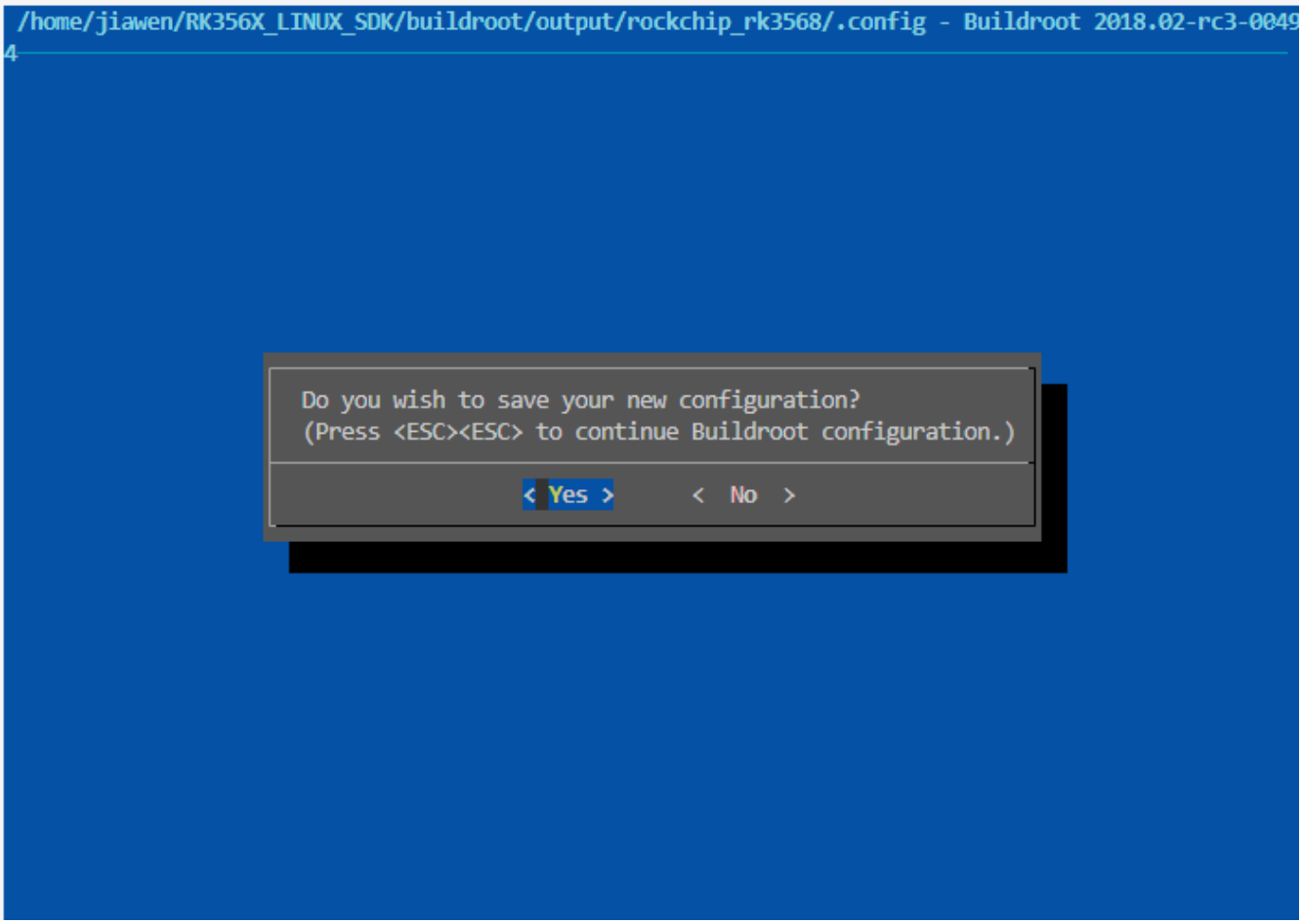
可以手动修改 configs 目录下的配置文件，也可以使用 SDK 提供的配置文件修改命令。对于 rk 这套 #include 没有深入了解的不建议手动修改，以防止配置出现冲突。

```
# SDK 配置
./build.sh bconfig
```

Buildroot 使用 kconfig 进行配置，其使用方法与 kernel 的配置基本相同，可以按层级打开或关闭配置，也可以直接搜索



当修改完成配置以后，多次按 Esc 键，退出图形配置界面，如果提示是否保存配置文件，选择 Yes



使用./build.sh bconfig 修改完成配置文件以后，会自动覆盖原有的配置文件。

16.3.2 buildroot 构建清理

由于 buildroot 的构建机制导致所有生成的文件都会被安装到 target 目录下并打包成 rootfs 镜像。当修改配置文件之后，buildroot 仅会将新选择的软件包编译并安装，如果在配置文件中移除了一些软件包，或者由于依赖关系，一些软件包已经不再需要时，buildroot 不会主动去清理这些软件包。

并且，buildroot 不会去检测软件包是否被修改，如果修改了某个已经编译过的软件包的内容，在构建时不会重新构建这个修改过的软件包。

为了让修改能被应用，可以先清理之前构建时生成的文件。

根据之前对输出目录的介绍，可知 build 和 target 目录一定要清理，images 目录下的内容会被覆盖，host 目录为宿主机使用的工具一般不用清理。

由于 Rockchip 对 buildroot 的 make 进行了修改，有些原生 buildroot 支持的 make 命令无法使用，这里可以使用手动删除的方法。

找到选择的配置文件对应的输出目录，如 rockchip_rk3576_lubancat_defconfig 对应的输出目录是 buildroot/output/rockchip_rk3576_lubancat，删除这个目录下对应的要清理的目录即可

16.3.3 Rootfs-Overlay

除了通过软件包向根文件系统安装文件以外，还可以通过 rootfs-overlay 的方式，在 rootfs 镜像打包前将文件覆盖到 target 目录中。

rootfs-overlay 方式主要是用来覆盖或写入一些配置文件或脚本。

可以通过 buildroot 的 BR2_ROOTFS_OVERLAY 选项来进行配置，例如 rockchip_rk3562_lubancat_defconfig 配置文件

```
BR2_ROOTFS_OVERLAY="board/rockchip/common/base board/rockchip/rk3562/fs-
↪overlay/ board/rockchip/common/lubancat/"
```

设置了三个路径，中间用空格隔开，在 buildroot 打包镜像前会按顺序进行覆盖，在配置时需要注意文件覆盖顺序。

16.3.4 编译单个软件包

如前文所说，buildroot 不会去检测软件包是否被修改，如果修改了某个已经编译过的软件包的内容，在构建时不会重新构建这个修改过的软件包。

由于 buildroot 的编译过程很慢，如果每次都清除整个 build 目录重新编译，这将耗费大量时间，尤其是使用软件包较多的项目。

可以只对单个软件包进行操作。命令格式为 make <package>-<command>

source	从网络获取源代码
depends	构建和安装构建此软件包所需的依赖
extract	将源代码放在包构建目录中（解压、复制源代码等）
patch	应用补丁，如果有的话
configure	运行 configure 命令，仅部分软件包支持
build	运行编译命令
install-staging	安装软件包
install	执行 build 和 install-staging
show-depends	显示构建软件包所需的依赖
clean	从 host 和 target 目录中卸载软件; 仅部分软件包支持
dirclean	删除整个包构建目录
rebuild	重新运行编译命令
reconfigure	重新运行配置命令
reinstall	重新安装软件包

在 buildroot 目录下执行以下命令，以 vim 软件包为例

```
# 获取软件包，如果本地 dl 目录中不存在会从网络获取
make vim-source

# 解压软件包到 build 目录，然后向解压到 build 目录的软件包打补丁
# 常常在软件包开发调试时使用
make vim-extract
make vim-patch

# 编译软件包
make vim-configure
make vim-build
make vim-install

# 清除软件包
make vim-dirclean
```

(下页继续)

(续上页)

```

# 重新编译软件包
make vim-reconfigure
make vim-rebuild
make vim-reinstall
```

16.3.5 添加新的软件包

在 buildroot 的开发过程中，有时需要添加一些自己的软件包，这里以野火为 LubanCat 板卡添加的 lbc-firmware 软件包为例

在合适的位置创建一个目录，为了和 Rockchip 的修改统一，在 buildroot/package/rockchip 目录下创建 lbc-firmware 目录

16.3.5.1 Config.in

将软件包加入 buildroot 配置工具的管理

```
config BR2_PACKAGE_LBC_FIRMWARE
    bool "lbc-firmware"
    help
      copy lubancat firmware to rootfs
```

添加 Config.in 文件以后，就可以通过 buildroot 的 menuconfig 的 BR2_PACKAGE_LBC_FIRMWARE 来选择开启和关闭配置项。

这里需要注意，配置选项的命令需要遵循一定的规则，软件包的开启和关闭一般以 BR2_PACKAGE_(软件包所在目录的全大写)

除了开启或关闭软件包之外，还可以添加选项来配置软件包的编译条件，这里的示例比较简单，更复杂的可以参考 buildroot 其他软件包的 Config.in

16.3.5.2 .mk

.mk 文件对软件包的下载、配置、构建、安装等细节进行了描述。mk 文件的内容比较灵活，需要根据实际软件包的编译工具使用不同的结构。

```
LBC_FIRMWARE_VERSION = 1.0
LBC_FIRMWARE_SITE_METHOD = local
LBC_FIRMWARE_SITE = $(TOPDIR)/../external/firmware
FIRMWARE_DIR := $(TARGET_DIR)/lib/firmware

define LBC_FIRMWARE_INSTALL_TARGET_CMDS
   mkdir -p $(TARGET_DIR)/lib/firmware
   cp -r $(LBC_FIRMWARE_SITE)/* $(FIRMWARE_DIR)
endef

$(eval $(generic-package))
```

.mk 文件的文件名应与软件包所在的目录一致，如软件包配置文件保存的目录是 lbc-firmware，则 mk 的名称应该是 lbc-firmware.mk。

在 mk 文件内，配置项的名称也与 mk 文件的名称保持一致，但是要使用字母全部大写的格式，并且要将分隔符 - 变为分隔符 _

- VERSION：软件包的版本
- SITE_METHOD：软件包源码保存地址的类型，local 为本地软件包，还有 git、svn 等，如果不指定默认为 tarball
- SITE：软件包源码路径，local 需要指定本地路径，git、svn、tarball 等还要指定网址，如果不指定的话默认从 buildroot 自己的软件源中下载。
- INSTALL_TARGET_CMDS：软件包安装命令

这里简单对 mk 文件的配置进行了说明，如果有更复杂的功能需要实现，可以参考其他软件包的 mk 文件或查看 buildroot 官方文档

16.4 rkboot 配置文件的修改

根据前文描述，rkboot 配置仅基于官方配置进行了简单适配，以使资深开发者在 LubanCat 板卡上进行 Rockchip 原生 buildroot 固件的开发。

> **注意：** 此配置选项不提供任何技术支持，仅提供给有开发能力的用户使用

为了进一步适配用户手中的 LubanCat 板卡，需要对一些文件及配置进行修改。

16.4.1 LubanCat_(主芯片型号)_buildroot_rkboot_defconfig

此文件为 SDK 配置文件

```
RK_ROOTFS_HOSTNAME_CUSTOM=y
RK_ROOTFS_HOSTNAME="LubanCat"
RK_ROOTFS_INSTALL_MODULES=y
# RK_WIFIBT is not set
RK_UBOOT_SPL=y
RK_KERNEL_PREFERRED="6.1"
RK_KERNEL_CFG="lubancat_linux_rk3576_defconfig"
RK_KERNEL_DTS_NAME="rk3576-lubancat-generic"
RK_USE_FIT_IMG=y
```

- RK_ROOTFS_HOSTNAME：设置 hostname 为 LubanCat
- RK_ROOTFS_INSTALL_MODULES：将内核编译的 ko 文件安装到 rootfs
- # RK_WIFIBT is not set：不使用 rk-wifibt 驱动，使用内核中的
- RK_KERNEL_PREFERRED：指定内核版本为 6.1
- RK_KERNEL_CFG：指定内核配置文件为 lubancat_linux_rk3576_defconfig，根据芯片选择
- RK_KERNEL_DTS_NAME：设备树名称

用户需要将 RK_KERNEL_DTS_NAME 修改为自己板卡的设备树，具体名称可以查看：kernel-6.1/arch/arm64/boot/dts/rockchip/Makefile

RK_KERNEL_CFG 不建议修改，如果要增删内核驱动的话用./build.sh kconfig 直接修改已经指定的配置文件即可。

16.4.2 buildroot 配置文件

rkboot 使用了 Rockchip 官方的 buildroot 配置文件，未进行修改。配置文件保存在 buildroot/configs 目录下

用户如果要修改配置文件可以使用./build.sh bconfig 去修改。

LubanCat 的修改内容可以参考 rockchip_(主芯片型号)_lubancat_defconfig

16.4.3 板卡设备树

由于 rkboot 不支持设备树插件，因此一些硬件功能需要用户自行添加并编译到镜像中。

打开前面配置文件指定的设备树，此处以 LubanCat-3 为例。打开文件 kernel-6.1/arch/arm64/boot/dts/rockchip/rk3576-lubancat-3.dts

在设备树文件靠前的位置，已经添加了一些常用的外设配置，通过 #include 的方式引用。

```
// #include "rk3576-lubancat-3-dsi-1080x1920-5-5inch-ebf410125.dtsi"
// #include "rk3576-lubancat-3-dsi-1024x600-7inch-ebf410173.dtsi"
// #include "rk3576-lubancat-3-dsi-800x1280-10-1inch-ebf410177.dtsi"

#include "rk3576-lubancat-3-csi.dtsi"
```

如上所示，#include 了几个 dtsi 文件，其中有三个屏幕配置，需要将实际使用的屏幕配置取消注释 (删除//)。

需要注意，一般同一类型的 dtsi 文件同一时间只可开启一个，上方代码中有三个使用 dsi 接口的不同屏幕，就只能选择一个。如果配置了两个不同的 dsi 接口，则有时可以同时开启两个 (还需根据其他情况具体决定)。

除此还要注意，有些 dtsi 文件原本是搭配设备树插件使用的，里面的节点并未“okay”而是处于“disabled”的状态。这时就需要参考设备树插件来开启对应的节点。

打开 kernel-6.1/arch/arm64/boot/dts/rockchip/uEnv/rk3576/uEnvLubanCat3.txt 文件，这里保存了所有 LubanCat-3 板卡所能使用的设备树插件的列表。设备树插件保存在 kernel-6.1/arch/arm64/boot/dts/rockchip/overlay 目录下

如果要在设备树中开启相应的功能，可以参考设备树插件中配置的设备树节点手动在板卡设备树中开启。

例如设备树插件 rk3576-lubancat-3-dp-disabled-ovetlay.dts 的作用是关闭 Type-C 接口的 DP 视频输出，源文件内容如下：

```
/dts-v1/;
/plugin/;

#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/pinctrl/rockchip.h>
#include <dt-bindings/display/drm_mipi_dsi.h>
#include <dt-bindings/interrupt-controller/irq.h>

/ {
    fragment@0 {
        target = <&dp>;

        __overlay__ {
                status = "disabled";
        };
    };
    fragment@1 {
        target = <&dp0>;

        __overlay__ {
                status = "disabled";
        };
```

(下页继续)

(续上页)

```
    };
    fragment@2 {
        target = <&dp0_in_vp0>;

        __overlay__ {
                status = "disabled";
        };
    };
    fragment@3 {
        target = <&dp0_in_vp1>;

        __overlay__ {
                status = "disabled";
        };
    };
    fragment@4 {
        target = <&dp0_in_vp2>;

        __overlay__ {
                status = "disabled";
        };
    };
    fragment@5 {
        target = <&vp1>;

        __overlay__ {
                status = "disabled";
        };
    };
    fragment@6 {
        target = <&route_dp0>;

```

(下页继续)

(续上页)

```
55          __overlay__ {
56                  status = "disabled";
57          };
58      };
59      fragment@7 {
60          target = <&dp0_sound>;
61
62          __overlay__ {
63                  status = "disabled";
64          };
65      };
66      fragment@8 {
67          target = <&spdif_tx3>;
68
69          __overlay__ {
70                  status = "disabled";
71          };
72      };
73  };
```

那么如何移植到板卡设备树 rk3576-lubancat-3.dts 中呢？需要在板卡设备树中搜索所有设备树插件里 **target = <&xxx>;** 对应的节点，例如在 rk3576-lubancat-3.dts 中搜索 **&dp** ，将 **&dp** 中的 **status** 改为和设备树插件一致。

```
1  &dp {
2      status = "disabled";
3  };
4
5  &dp0 {
6      status = "disabled";
7  };
8
9  &dp0_in_vp0 {
```

(下页继续)

(续上页)

```
    status = "disabled";
};

&dp0_in_vp1 {
    status = "disabled";
};

&dp0_in_vp2 {
    status = "disabled";
};

&vp1 {
    status = "disabled";
};

&route_dp0 {
    status = "disabled";
    connect = <&vp1_out_dp0>;
};

&dp0_sound {
    status = "disabled";
};

&spdif_tx3 {
    status = "disabled";
};
```

有时会出现在板卡设备树没有设备树插件中同样节点的情况，这时在板卡设备树的最后手动追加即可

例如设备树插件 rk3576-lubancat-uart8-m2-overlay.dts 的功能是开启 uart8 并使用 m2 这一组引脚

设备树插件内容如下

```
/dts-v1/;
/plugin/;

/ {
    fragment@0 {
        target = <&uart8>;

        __overlay__ {
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <&uart8m2_xfer>;
        };
    };
};
```

在板卡设备树 rk3576-lubancat-3.dts 中搜索发现并没有 **&uart8** 这个节点，此时就需要在板卡设备树的最后手动追加

将设备树插件中的对应节点复制，删除画红线的部分，最后整理一下格式

```
# 假如这是原本的最后一行

/ {
    fragment@0 {
        target = <&uart8>;

        __overlay__ {
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <&uart8m2_xfer>;
        };
    };
};
```

```
&uart8 {
    status = "okay";
```

(下页继续)

(续上页)

```
3     pinctrl-names = "default";
4     pinctrl-0 = <&uart8m2_xfer>;
5 };
```

修改完设备树以后使用./build.sh kernel 命令重新打包 boot 分区就会应用修改后的设备树

16.5 参考资料

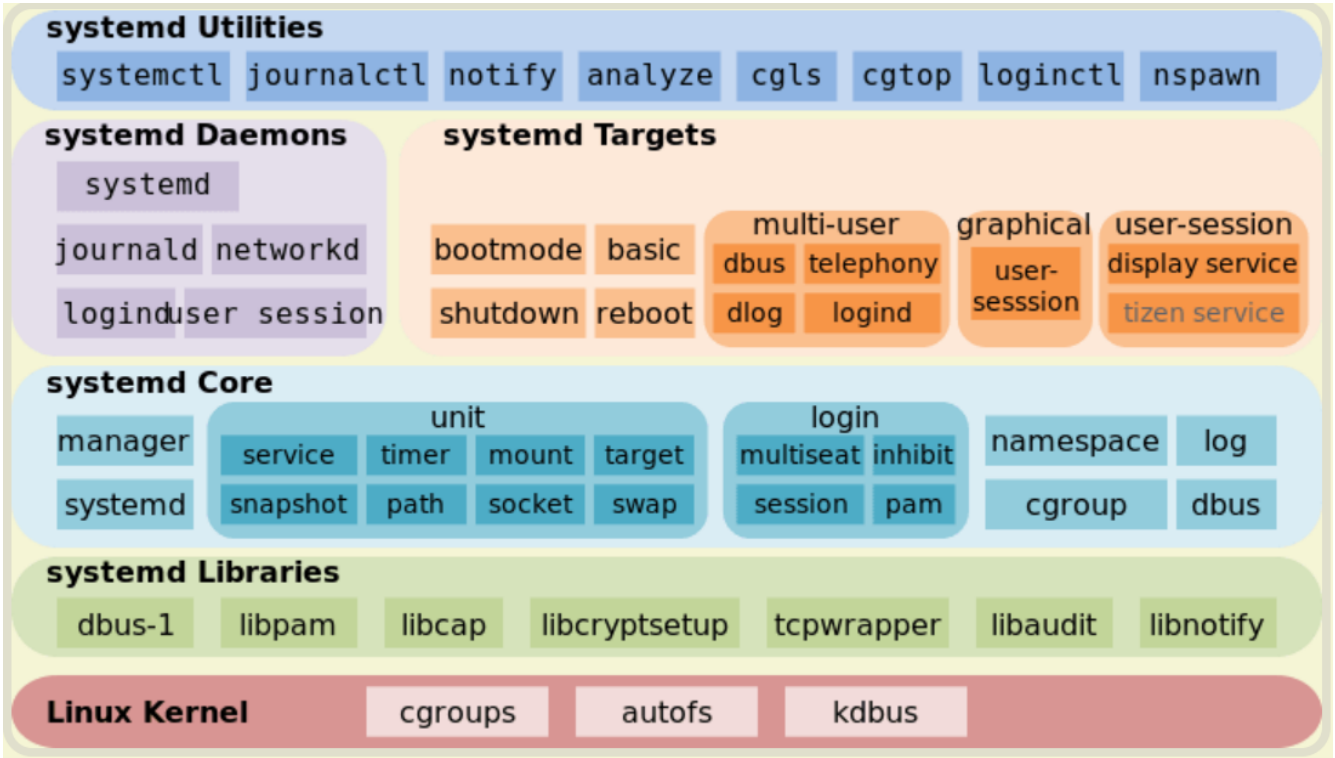
《Rockchip_Developer_Guide_Buildroot_CN.pdf》

《The Buildroot user manual》https://buildroot.org/downloads/manual/manual.html

第 17 章　探索 Systemd

17.1 系统管理

systemd 不是一个命令，而是一组命令的集合，提供了一个系统和服务管理器，运行为 PID 1 并负责启动其它程序。下图为 systemd 的架构图。



systemd 功能包括：支持并行化任务；同时采用 socket 式与 D-Bus 总线式激活服务；按需启动守护进程（daemon）；利用 Linux 的 cgroups 监视进程；支持快照和系统恢复；维护挂载点和自动挂载点；各服务间基于依赖关系进行精密控制。systemd 支持 SysV 和 LSB 初始脚本，可以替代 sysvinit。除此之外，功能还包括日志进程、控制基础系统配置，维护登陆用户列表以及系统账户、运行时目录和设置，可以运行容器和虚拟机，可以简单的管理网络配置、网络时间同步、日志转发和名称解析等。

17.2 Systemd-核心概念

17.2.1 unit

单元，单元文件是 ini 风格的纯文本文件，Systemd 可以管理所有系统资源，不同的资源统称为 Unit（单位）。其封装了 12 种对象的信息：服务 (service)、套接字 (socket)、设备 (device)、挂载点 (mount)、自动挂载点 (automount)、启动目标 (target)、交换分区或交换文件 (swap)、被监视的路径 (path)、任务计划 (timer)、资源控制组 (slice)、一组外部创建的进程 (scope)、快照 (snapshot)。

- service：一个后台服务进程，其封装了守护进程的启动、停止、重启与重载等操作。
- socket：此类配置单元封装系统和互联网中的一个套接字。当下，systemd 支持流式，数据报和连续包的 AF_INET，AF_INET6，AF_UNIX socket。每个套接字配置单元都有一个相应的服务配置单元，相应的服务在第一个“连接”进入套接字时就会启动（例如：nscd.socket 在有新连接后便启动 nscd.service），监控系统或者网络的数据信息。
- device：此类配置单元封装一个存在于 Linux 设备树中的设备。每个使用 udev 规则标记的设备都将会在 systemd 中作为一个设备配置单元出现，定义设备之间的依赖关系。
- mount：此类配置单元封装文件系统结构层次中的一个挂载点。systemd 将对这个挂载点进行监控和管理。比如，可以在启动时自动将其挂载，可以在某些条件下自动卸载。systemd 会将/etc/fstab 中的条目都转换为挂载点，并在开机时处理。
- automount：此类配置单元封装系统结构层次中的一个自挂载点。每个自挂载配置单元对应一个挂载配置单元，当该自动挂载点被访问时，systemd 执行挂载点中定义的挂载行为。
- Swap：和挂载配置单元类似，交换配置单元用来管理交换分区。用户可以用交换配置单元来定义系统中的交换分区，可以让这些交换分区在启动时被激活。
- target：此类配置单元为其他配置单元进行逻辑分组。它们本身实际上并不做什么，只是引用其他配置单元而已，这样便可以对配置单元做一个统一的控制，就可以实现大家都非常熟悉的运行级别的概念。比如，想让系统进入图形化模式，需要运行许多服务和配置命令，这些操作都由一个个的配置单元表示，将所有的这些配置单元组合为一个目标（target），就表示需要将这些配置单元全部执行一遍，以便进入目标所代表的系统运行状态（例如：multi-user.target 相当于在传统使用 sysv 的系统中运行级别 5）。

- timer：定时器配置单元用来定时触发用户定义的操作。这类配置单元取代了 atd，crond 等传统的定时服务。
- snapshot：与 target 配置单元相似，快照是一组配置单元，它保存了系统当前的运行状态。
- slice：表示一个 CGroup 的树。
- path：监控指定目录或文件的变化，并触发其它 Unit 的运行。
- scope：它用于描述一些系统服务的分组信息。

每个配置单元都有一个对应的配置文件，比如一个 avahi-daemon 服务对应一个 avahi-daemon.service 文件。这种配置文件的语法非常简单，用户不需要再编写和维护复杂的 sysv 脚本了。

```
# 列出正在运行的 Unit
$ systemctl list-units
```

```
root@lubancat:~# systemctl list-units
UNIT                               LOAD   ACTIVE     SUB        JOB   DESCRIPTION
sys-devices-platform-3c0800000.pcie-pci0002:20-0002:20:00.0-0002:21:00.0-nvme-nv
sys-devices-platform-backlight-backlight-backlight.device loaded active     plug
sys-devices-platform-fe010000.ethernet-net-eth1.device loaded active     plugged
sys-devices-platform-fe2a0000.ethernet-net-eth0.device loaded active     plugged
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0.device
sys-devices-platform-fiq_debugger.0-tty-ttyFIQ0.device loaded active     plugged
sys-devices-platform-rk809\x2dsound-sound-card0.device loaded active     plugged
sys-devices-virtual-block-ram0.device loaded active     plugged         /sys/dev
sys-devices-virtual-block-zram0.device loaded active     plugged         /sys/de
sys-devices-virtual-misc-rfkill.device loaded active     plugged         /sys/de
sys-devices-virtual-tty-ttyGS0.device loaded active     plugged         /sys/dev
sys-module-configfs.device loaded active     plugged         /sys/module/configf
lines 1-23
```

```
# 列出所有 Unit，包括没有找到配置文件的或者启动失败的
$ systemctl list-units --all
```

(下页继续)

(续上页)

```
# 列出所有没有运行的 Unit
$ systemctl list-units --all --state=inactive
```

```
root@lubancat:~# systemctl list-units --all --state=inactive
  UNIT                              LOAD      ACTIVE   SUB  JOB   DESCRIPTION
  proc-sys-fs-binfmt_misc.automount loaded    inactive dead       Arbitrary Exec
  dev-hugepages.mount               loaded    inactive dead       Huge Pages File Syste
  proc-sys-fs-binfmt_misc.mount     loaded    inactive dead       Arbitrary Executab
  anacron.service                   loaded    inactive dead start Run anacron jobs
● apparmor.service                  not-found inactive dead       apparmor.service
  apt-daily-upgrade.service         loaded    inactive dead       Daily apt upgrade and
  apt-daily.service                 loaded    inactive dead       Daily apt download ac
  async.service                     loaded    inactive dead       enable ASYNC for Debi
● auditd.service                    not-found inactive dead       auditd.service
● connman.service                   not-found inactive dead       connman.service
● console-screen.service            not-found inactive dead       console-screen.servic
  emergency.service                 loaded    inactive dead       Emergency Shell
  fstrim.service                    loaded    inactive dead       Discard unused blocks
  getty-static.service              loaded    inactive dead       getty on tty2-tty6 if
● kbd.service                       not-found inactive dead       kbd.service
  kmod-static-nodes.service         loaded    inactive dead       Create list of requir
● lighdm.service                    not-found inactive dead       lighdm.service
● plymouth-quit-wait.service        not-found inactive dead       plymouth-quit-wait.se
● plymouth-quit.service             not-found inactive dead       plymouth-quit.service
● plymouth-start.service            not-found inactive dead       plymouth-start.servic
  pppd-dns.service                  loaded    inactive dead       Restore /etc/resolv.c
  rescue.service                    loaded    inactive dead       Rescue Shell
lines 1-23
```

```
# 列出所有加载失败的 Unit
$ systemctl list-units --failed

# 列出所有正在运行的、类型为 service 的 Unit
$ systemctl list-units --type=service
```

```
root@lubancat:~# systemctl list-units --type=service
UNIT                            LOAD   ACTIVE     SUB     JOB   DESCRIPTION
acpid.service                   loaded active     running       ACPI event daemon
adbd.service                    loaded active     running       adbd for Debian
alsa-restore.service            loaded active     exited        Save/Restore Sound Ca
alsa-state.service              loaded active     running       Manage Sound Card Sta
anacron.service                 loaded inactive   dead    start Run anacron jobs
binfmt-support.service          loaded active     exited        Enable support for ad
bluetooth.service               loaded active     running       Bluetooth service
console-setup.service           loaded active     exited        Set console font and
cpufrequtils.service            loaded active     exited        LSB: set CPUFreq kern
dbus.service                    loaded active     running       D-Bus System Message
getty@tty1.service              loaded active     running       Getty on tty1
ifupdown-pre.service            loaded active     exited        Helper to synchronize
ifupdown-wait-online.service loaded active     exited        Wait for network to
keyboard-setup.service          loaded active     exited        Set the console keybo
lightdm.service                 loaded active     running       Light Display Manager
loadcpufreq.service             loaded active     exited        LSB: Load kernel modu
networking.service              loaded active     exited        Raise network interfa
NetworkManager.service          loaded active     running       Network Manager
ntp.service                     loaded active     running       Network Time Service
openvpn.service                 loaded active     exited        OpenVPN service
polkit.service                  loaded active     running       Authorization Manager
rc-local.service                loaded active     exited        /etc/rc.local Compati
lines 1-23
```

```
# 显示某个 Unit 是否正在运行
$ systemctl is-active systemd-timesyncd.service
```

```
root@lubancat:~# systemctl is-active systemd-timesyncd.service
inactive
root@lubancat:~#
```

```
# 显示某个 Unit 服务是否建立了启动链接
$ systemctl is-enabled systemd-timesyncd.service
```

```
root@lubancat:~#  systemctl is-enabled systemd-timesyncd.service
enabled
root@lubancat:~#
```

17.2.2 unit-管理

```
# 立即启动一个服务
$ sudo systemctl start sshd.service

# 立即停止一个服务
$ sudo systemctl stop sshd.service

# 重启一个服务
$ sudo systemctl restart sshd.service

# 杀死一个服务的所有子进程
$ sudo systemctl kill sshd.service

# 重新加载一个服务的配置文件
$ sudo systemctl reload sshd.service

# 重载所有修改过的配置文件
$ sudo systemctl daemon-reload

# 显示某个 Unit 的所有底层参数
$ systemctl show httpd.service
```

```
root@lubancat:~# systemctl show httpd.service
Restart=no
NotifyAccess=none
RestartUSec=100ms
TimeoutStartUSec=1min 30s
TimeoutStopUSec=1min 30s
RuntimeMaxUSec=infinity
WatchdogUSec=0
WatchdogTimestampMonotonic=0
RootDirectoryStartOnly=no
RemainAfterExit=no
GuessMainPID=yes
MainPID=0
ControlPID=0
FileDescriptorStoreMax=0
NFileDescriptorStore=0
StatusErrno=0
Result=success
UID=[not set]
GID=[not set]
NRestarts=0
ExecMainStartTimestampMonotonic=0
ExecMainExitTimestampMonotonic=0
ExecMainPID=0
lines 1-23
```

```
# 显示某个 Unit 的指定属性的值
$ systemctl show -p CPUShares avahi-daemon.service
```

17.2.3 unit-依赖

尽管 systemd 将大量的启动工作解除了依赖，使得它们可以并行启动。但还是存在一些任务，它们之间存在天生的依赖关系，不能用“套接字激活”（socket activation），D-Bus activation 和 autofs 三大方法来解除依赖。比如，挂载必须等待挂载点在文件系统中被创建；挂载也必须等待相应的物理设备就绪。为了解决这类依赖问题，systemd 的配置单元之间可以彼此定义依赖关系。比如，unit Q 依赖 unit W，可以在 unit W 的定义中用“require Q”来表示，这样 systemd 就会保证先启动 Q 再启动 W。systemd 能保证事务完整性。systemd 的事务概念和数据库中的有所不同，主要是为了保证多个依赖的配置单元之间没有环形引用。若存在循环依赖，那么 systemd 将无法启动任意一个服务。此时，systemd 将会尝试解决这个问题，因为配置单元之间的依赖关系有两种：requireds 为强依赖，wants 为弱依赖，systemd 将去掉 wants 关键字指定的依赖看看是否能打破循环。如果无法修复，systemd 会报错。systemd 能够自动检测和修复这类配置错误，极大地减轻了管理员的拔锚负担。

```
# 命令列出一个 Unit 的所有依赖
$ systemctl list-dependencies sshd.service
```

```
root@lubancat:~# systemctl list-dependencies sshd.service
sshd.service
● ├─-.mount
● ├─system.slice
● └─sysinit.target
●   ├─dev-hugepages.mount
●   ├─dev-mqueue.mount
●   ├─keyboard-setup.service
●   ├─kmod-static-nodes.service
●   ├─proc-sys-fs-binfmt_misc.automount
●   ├─resolvconf.service
●   ├─sys-fs-fuse-connections.mount
●   ├─sys-kernel-config.mount
●   ├─sys-kernel-debug.mount
●   ├─systemd-ask-password-console.path
●   ├─systemd-binfmt.service
●   ├─systemd-hwdb-update.service
●   ├─systemd-journal-flush.service
●   ├─systemd-journald.service
●   ├─systemd-machine-id-commit.service
●   ├─systemd-modules-load.service
●   ├─systemd-random-seed.service
●   ├─systemd-sysctl.service
●   ├─systemd-sysusers.service
lines 1-23
```

上面命令的输出结果之中，有些依赖是 Target 类型，而 Target 类型默认是不会展开显示的。若要展开显示 Target，则需要添加–all 参数。

```
# 命令列出一个 Unit 的所有依赖，并展开显示 Target 依赖类型
$ systemctl list-dependencies --all sshd.service
```

17.2.4 unit-配置文件

systemd 的配置文件默认会存放于文件系统中的/etc/systemd/system 或/usr/lib/systemd/system 目录下，我们输入“ls -al”命令查看一下该目录的内容。

```
root@lubancat:~# ls -al /etc/systemd/system
total 48
drwxr-xr-x 12 root root 4096 Feb 14  2019 .
drwxr-xr-x  5 root root 4096 Sep 27 16:48 ..
lrwxrwxrwx  1 root root    9 Sep 28 09:48 NetworkManager-wait-online.service -> /dev/n
ull
drwxr-xr-x  2 root root 4096 Sep 27 16:46 bluetooth.target.wants
drwxr-xr-x  2 root root 4096 Feb 14  2019 boot-complete.target.requires
lrwxrwxrwx  1 root root    9 Sep 27 16:45 bootlogs.service -> /dev/null
lrwxrwxrwx  1 root root    9 Sep 27 16:45 bootmisc.service -> /dev/null
lrwxrwxrwx  1 root root    9 Sep 27 16:45 brightness.service -> /dev/null
lrwxrwxrwx  1 root root    9 Sep 27 16:45 checkfs.service -> /dev/null
lrwxrwxrwx  1 root root    9 Sep 27 16:45 checkroot-bootclean.service -> /dev/null
lrwxrwxrwx  1 root root    9 Sep 27 16:45 checkroot.service -> /dev/null
lrwxrwxrwx  1 root root   33 Feb 14  2019 ctrl-alt-del.target -> /lib/systemd/system/r
eboot.target
lrwxrwxrwx  1 root root   42 Sep 27 16:46 dbus-fi.w1.wpa_supplicant1.service -> /lib/s
ystemd/system/wpa_supplicant.service
lrwxrwxrwx  1 root root   37 Sep 27 16:46 dbus-org.bluez.service -> /lib/systemd/syste
m/bluetooth.service
lrwxrwxrwx  1 root root   44 Feb 14  2019 dbus-org.freedesktop.network1.service -> /li
b/systemd/system/systemd-networkd.service
lrwxrwxrwx  1 root root   53 Sep 27 16:47 dbus-org.freedesktop.nm-dispatcher.service -
> /lib/systemd/system/NetworkManager-dispatcher.service
lrwxrwxrwx  1 root root   44 Feb 14  2019 dbus-org.freedesktop.resolve1.service -> /li
b/systemd/system/systemd-resolved.service
lrwxrwxrwx  1 root root   45 Sep 27 16:39 dbus-org.freedesktop.timesync1.service -> /l
ib/systemd/system/systemd-timesyncd.service
lrwxrwxrwx  1 root root   35 Sep 27 16:47 display-manager.service -> /lib/systemd/syst
em/lightdm.service
```

可以看到该目录下有很多链接符号“->”，这代表着文件的实体是“->”指向的文件，可以说是软链接的关系，和 windows 上的快捷方式类似（有点类似月老牵线 ^-^），而这些软链接所指向的目录绝大多数是/lib/systemd/system 目录，也有指向/dev/null 文件的，这代表它是一个空文件，初始化过程中 systemd 只执行/etc/systemd/system 目录里面的配置文件。当你安装完 systemd 程序之后，他会自动的在/lib/systemd/system 目录下生成一个与该程序对应的配置文件。你可以使用“systemctl enable xxx.service”的方式来建立一个服务软链接，若设置了开机启动，则“systemctl enable”相当于使能开机启动，而“systemctl disable”命令与之相反，他会断开软链接，所以开机就不会启动。

```
# 检查某个单元是否是开机自启动的（建立的启动链接）
$ systemctl is-enabled sshd.service
```

```
root@lubancat:~# systemctl is-enabled sshd.service
enabled
root@lubancat:~#
```

17.2.5 unit-系统管理

```
# 重启系统（异步操作）
$ systemctl reboot
```

```
root@lubancat:~# systemctl reboot
root@lubancat:~# [  OK  ] Closed Load/Save RF Kill Switch Status /dev/rfkill Watch.
         Stopping Session c1 of user cat.
         Stopping Session c2 of user root.
[ 7589.470863] ttyFIQ ttyFIQ0: tty_port_close_start: tty->count = 1 port count = 2
         Stopping Setup zram based device zram0...
[  OK  ] Stopped target Sound Card.
         Stopping Save/Restore Sound Card State...
[  OK  ] Stopped target Remote Encrypted Volumes.
[  OK  ] Stopped Daily Cleanup of Temporary Directories.
[  OK  ] Stopped target Host and Network Name Lookups.
         Stopping Bluetooth service...
         Stopping Light Display Manager...
         Stopping Disk Manager...
         Stopping Daemon for power management...
         Stopping OpenBSD Secure Shell server...
         Stopping LSB: set CPUFreq kernel parameters...
         Stopping triggerhappy global hotkey daemon...
         Stopping LSB: layer 2 tunelling protocol daemon...
         Stopping Network Time Service...
[  OK  ] Stopped target Login Prompts.
         Stopping Serial Getty on ttyFIQ0...
         Stopping Getty on tty1...
         Stopping System Logging Service...
[  OK  ] Stopped OpenVPN service.
         Stopping ACPI event daemon...
         Stopping Authorization Manager...
```

```
# 关闭系统，切断电源（异步操作）
$ systemctl poweroff

# 仅 CPU 停止工作，其他硬件仍处于开机状态（异步操作）
$ systemctl halt
```

（下页继续）

(续上页)

```
# 暂停系统（异步操作），执行 suspend.target
$ systemctl suspend

# 使系统进入冬眠状态（异步操作），执行 hibernate.target
$ systemctl hibernate
```

17.2.6 unit-日志管理

Systemd 统一管理所有 Unit 的启动日志。带来的好处就是，可以只用 journalctl 一个命令，查看所有日志（内核日志和应用日志）。

日志配置文件位于/etc/systemd/journald.conf，其保存目录为/var/log/journal/ 默认情况下日志最大限制为所在文件系统容量的 10%，可通过/etc/systemd/journald.conf 中的 SystemMaxUse 字段来指定日志最大限制。

注意： /var/log/journal/目录是 systemd 软件包的一部分。若被删除，systemd 不会自动创建它，直到下次升级软件包时重建该目录。如果该目录缺失，systemd 会将日志记录写入/run/systemd/journal。这意味着，系统重启后日志将丢失。

```
# 查看所有日志
$ sudo journalctl
```

```
root@lubancat:~# sudo journalctl
-- Logs begin at Tue 2022-10-11 13:43:18 CST, end at Tue 2022-10-11 13:46:08 CST
Oct 11 13:43:18 lubancat kernel: Booting Linux on physical CPU 0x0000000000 [0x4
Oct 11 13:43:18 lubancat kernel: Linux version 4.19.232 (jiawen@dev120.embedfire
Oct 11 13:43:18 lubancat kernel: Machine model: EmbedFire LubanCat2 HDMI
Oct 11 13:43:18 lubancat kernel: earlycon: uart8250 at MMIO32 0x00000000fe660000
Oct 11 13:43:18 lubancat kernel: bootconsole [uart8250] enabled
Oct 11 13:43:18 lubancat kernel: cma: Reserved 16 MiB at 0x000000007ec00000
Oct 11 13:43:18 lubancat kernel: On node 0 totalpages: 519680
Oct 11 13:43:18 lubancat kernel:   DMA32 zone: 8184 pages used for memmap
Oct 11 13:43:18 lubancat kernel:   DMA32 zone: 0 pages reserved
Oct 11 13:43:18 lubancat kernel:   DMA32 zone: 519680 pages, LIFO batch:63
Oct 11 13:43:18 lubancat kernel: psci: probing for conduit method from DT.
Oct 11 13:43:18 lubancat kernel: psci: PSCIv1.1 detected in firmware.
Oct 11 13:43:18 lubancat kernel: psci: Using standard PSCI v0.2 function IDs
Oct 11 13:43:18 lubancat kernel: psci: Trusted OS migration not required
Oct 11 13:43:18 lubancat kernel: psci: SMC Calling Convention v1.2
Oct 11 13:43:18 lubancat kernel: percpu: Embedded 24 pages/cpu s57896 r8192 d322
Oct 11 13:43:18 lubancat kernel: pcpu-alloc: s57896 r8192 d32216 u98304 alloc=24
Oct 11 13:43:18 lubancat kernel: pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
Oct 11 13:43:18 lubancat kernel: Detected VIPT I-cache on CPU0
Oct 11 13:43:18 lubancat kernel: CPU features: detected: Virtualization Host Ext
Oct 11 13:43:18 lubancat kernel: CPU features: detected: Speculative Store Bypas
Oct 11 13:43:18 lubancat kernel: Built 1 zonelists, mobility grouping on.  Total
lines 1-23
```

```
# 指定日志文件占据的最大空间
$ sudo journalctl --vacuum-size=8M
```

17.3 Systemd-实例分析

17.3.1 启动顺序及依赖

前面我们讲到了服务的配置文件，系统上电后，systemd 便会读取每个服务地配置文件，然后根据配置文件执行每个系统服务，配置文件详细地描述了一个服务是如何启动的. 我们以 ssh 服务为例，详细分析其配置文件。在/etc/systemd/system 目录下找到 sshd.service 文件，它描述了如何启动一个 ssh 服务，使用 vim.tiny 打开该配置文件，如下图所示：

```
[Unit]
Description=OpenBSD Secure Shell server
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target
Alias=sshd.service
~
"/etc/systemd/system/sshd.service" 22L, 538C                  1,1           All
```

可以看到，配置文件一共有三个区块，分别是 Unit、Service、Install，每个区块又包含了许多键值对。

其中 Unit 区块中，Description 描述了当前服务，Documentation 字段给出了文档位置，紧接着的是比较重要的 After 字段，它指定了服务的启动顺序，但是不涉及依赖关系，与之对应的是 Before 字段。以本配置文件为例，After 表示的是当前服务需要在 network.target 及 auditd.service 两个服务之后启动。而 Wants 和 Requires 字段只涉及依赖关系，他与启动顺序是无关的，默认为同时启动。如果想要设置服务之间的依赖关系，及使用 Wants 和 Requires 字段即可，Wants 为“弱依赖”，Requires 为“强依赖”。

17.3.2 启动命令

EnvironmentFile 字段指定了当前服务的环境参数文件，注意等号后面的“-”，它表示如果/etc/default/ssh 文件不存在也不会抛出错误，ExecStart 是配置文件最重要的字段，它定义了启动一个进程需要执行的命令。图中执行的命令是“/usr/sbin/sshd -D $SSHD_OPTS”，其中的变量 $SSHD_OPTS 就是来自于 EnvironmentFile 字段所指定的环境参数文件。ExecStartPre 字段表示启动服务之前需要执行的命令。

17.3.3 启动类型与行为

1. Type 字段指定了服务的启动类型，它的类型如下所示。
    - simple（默认值）：ExecStart 字段启动的进程为主进程
    - forking：ExecStart 字段将以 fork() 方式启动，此时父进程将会退出，子进程将成为主进程
    - oneshot：类似于 simple，但只执行一次，Systemd 会等它执行完，才启动其他服务
    - dbus：类似于 simple，但会等待 D-Bus 信号后启动
    - notify：类似于 simple，启动结束后会发出通知信号，然后 Systemd 再启动其他服务
    - idle：类似于 simple，但是要等到其他任务都执行完，才会启动该服务。一种使用场合是为让该服务的输出，不与其他服务的输出相混合
2. KillMode 字段定义了 systemd 如何停止 ssh 服务，本例设置为 process，表示只停止主进程，但不停止 sshd 的子进程。
3. Restart 字段定义了 sshd 退出后，systemd 的重启方式，Restart 设为 on-failure，表示任何意外的失败，就将重启 sshd。如果 sshd 正常停止（如执行 systemctl stop 命令），它就不会重启。其可设置成下面值。
    - no（默认值）：退出后不会重启
    - on-success：只有正常退出时（退出状态码为 0），才会重启
    - on-failure：非正常退出时（退出状态码非 0），包括被信号终止和超时，才会重启
    - on-abnormal：只有被信号终止和超时，才会重启
    - on-abort：只有在收到没有捕捉到的信号终止时，才会重启
    - on-watchdog：超时退出，才会重启
    - always：不管是什么退出原因，总是重启

17.3.4 安装方式

WantedBy 字段表示当前服务所在的 Target,Target 表示的是服务组，sshd 所在的服务组为 multi-user.target。当执行 “systemctl enable sshd.service” 命令的时候，sshd.service 的一个符号链接，就会放在/etc/systemd/system 目录下面的 multi-user.target.wants 子目录之中。

如果我们修改了配置文件，就需要重新加载配置文件，然后重启该服务。

```
# 重新加载配置文件
$ sudo systemctl daemon-reload

# 重启相关服务
$ sudo systemctl restart ssh
```

17.4 Systemd-创建自己的 Systemd 服务

我们经常有这样的需求，自己写好一个应用，想要它实现开机自启动的功能，那么我们可以通过创建一个 Systemd 服务服务来实现。下面我以创建一个简单的 hello.service 服务为例子，教大家如何创建自己的 Systemd 服务。

17.4.1 编写脚本

cd 进入/opt 目录下，使用 vim 编写一个 hello.sh 脚本。

```
1 #!/bin/bash
2
3 while true
4 do
5     echo Hello Lubancat >> /tmp/hello.log
6     sleep 3
7 done
```

该脚本实现的功能是每隔 3 秒就打印“Hello Lubancat”字符串到/tmp/hello.log 文件中。编写好后记得赋予 hello.sh 可执行权限。

```
sudo chmod 0755 hello.sh
```

17.4.2 创建配置文件

在/etc/systemd/system/目录下创建一个 hello.service 配置文件，内容如下。

```
[Unit]
Description = hello daemon

[Service]
ExecStart = /opt/hello.sh
Restart = always
Type = simple

[Install]
WantedBy = multi-user.target
```

其中 ExecStart 字段定义了 hello.service 服务的自启动脚本为/opt/hello.sh，当我们使能了 hello.service 开机自启功能，在开机后便会执行/opt/hello.sh。Restart = always 表示指进程或服务意外故障的时候可以自动重启的模式。Type = simple 为默认的，可以不填。WantedBy 指定服务由谁启动，WantedBy = multi-user.target 表示在系统启动到多用户模式时就会把这个服务添加到启动顺序中。如果需要在图形用户界面后再启动，需要设置为 WantedBy = graphical.target，表示需要在系统进入图形用户界面模式时启动。

也就是说，WantedBy 参数对于服务是否需要桌面环境十分重要，如果服务需要桌面环境，然而设置 WantedBy = multi-user.target 在桌面启动前就调用自己的启动脚本，将会导致服务启动失败！

```
# 在桌面前就启动服务
WantedBy = multi-user.target

```

(下页继续)

(续上页)

```
4
5  # 进入桌面后启动服务
6  WantedBy = graphical.target
```

17.4.3 使能 hello.service 开机自启功能

输入命令“sudo systemctl list-unit-files –type=service | grep hello”查看 hello.service 是否被添加到了服务列表。

```
sudo systemctl list-unit-files --type=service | grep hello
```

```
root@lubancat:/etc/systemd/system# sudo systemctl list-unit-files --type=service | gre
p hello
hello.service                              disabled
root@lubancat:/etc/systemd/system#
```

可以看到 hello.service 处于 disable 状态，如果你输入上面命令后没有任何显示，那你创建的服务就处理问题，需要仔细排查。我们输入下面命令使 hello.service 开机自启动。

```
sudo systemctl enable hello
sudo systemctl start hello
```

然后使用 reboot 命令重启系统，启动系统后输入“sudo systemctl status hello”命令即可看到 hello.service 处于运行状态。

```
sudo systemctl status hello


cat /tmp/hello.log
```

```
root@lubancat:~# sudo systemctl status hello
● hello.service - hello daemon
   Loaded: loaded (/etc/systemd/system/hello.service; enabled; vendor preset: en
   Active: active (running) since Tue 2022-10-11 13:59:32 CST; 58s ago
 Main PID: 344 (hello.sh)
   Memory: 1.3M
   CGroup: /system.slice/hello.service
           ├─ 344 /bin/bash /opt/hello.sh
           └─1046 sleep 3
root@lubancat:~# cat /tmp/hello.log
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
```

关于 systemd 的知识点还有很多，这里做简单介绍，主要是让大家了解 systemd 的基本用法及启动服务的流程，感兴趣的可以在网上查阅相关文档。

如果使用桌面环境的用户建议使用桌面系统自带的自启动方式，详情请参考：

17.5 Systemd-基本工具

17.5.1 systemctl

监视和控制 systemd 的主要命令是 systemctl。该命令可用于查看系统状态和管理系统及服务。

```
# 输出激活的单元
systemctl

# 打印信息如下

UNIT                          LOAD   ACTIVE     SUB       JOB   DESCRIPTION
sys-devices-platform-3c0800000.pcie-pci0002:20-0002:20:00.0-0002:21:00.0-
 ↪nvme-nv
sys-devices-platform-backlight-backlight-backlight.device loaded active
 ↪plug
```

(下页继续)

(续上页)

```
sys-devices-platform-fe010000.ethernet-net-eth1.device loaded active    ␣
↪plugged
sys-devices-platform-fe2a0000.ethernet-net-eth0.device loaded active    ␣
↪plugged
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0-
↪mmcblk
sys-devices-platform-fe310000.sdhci-mmc_host-mmc0-mmc0:0001-block-mmcblk0.
↪device
sys-devices-platform-fiq_debugger.0-tty-ttyFIQ0.device loaded active    ␣
↪plugged
sys-devices-platform-rk809\x2dsound-sound-card0.device loaded active    ␣
↪plugged
sys-devices-virtual-block-ram0.device loaded active     plugged         /
↪sys/dev
```

(下页继续)

(续上页)

```
sys-devices-virtual-block-zram0.device loaded active     plugged          /
↪sys/de
sys-devices-virtual-misc-rfkill.device loaded active     plugged          /
↪sys/de
sys-devices-virtual-tty-ttyGS0.device loaded active     plugged          /
↪sys/dev
sys-module-configfs.device loaded active     plugged          /sys/module/
↪configf
```

```
# 显示系统状态
systemctl status -l

# 打印信息如下
● lubancat
    State: starting
    Jobs: 11 queued
Failed: 0 units
    Since: Tue 2022-10-11 13:59:30 CST; 5min ago
CGroup: /
        ├─user.slice
        | ├─user-0.slice
        | | ├─session-c2.scope
        | | | ├─ 557 /bin/login -p --
        | | | ├─1021 -bash
        | | | ├─1160 systemctl status -l
        | | | └─1161 pager
        | | └─user@0.service
        | |   └─init.scope
        | |     ├─1006 /lib/systemd/systemd --user
        | |     └─1009 (sd-pam)
        | └─user-1000.slice
        |   ├─user@1000.service
```

(下页继续)

(续上页)

```
        |   | ├─pulseaudio.service
        |   | | └─686 /usr/bin/pulseaudio --daemonize=no --log-
↪target=journal
        |   | └─init.scope
        |   |   ├─674 /lib/systemd/systemd --user
lines 1-23
```

```
# 重启系统
$ sudo systemctl reboot

# 暂停系统
$ sudo systemctl suspend

# 关闭系统，切断电源
$ sudo systemctl poweroff

# CPU 停止工作
$ sudo systemctl halt

# 让系统进入冬眠状态
$ sudo systemctl hibernate

# 让系统进入交互式休眠状态
$ sudo systemctl hybrid-sleep

# 启动进入救援状态（单用户状态）
$ sudo systemctl rescue

# 输出运行失败的单元
$ systemctl --failed
```

17.5.2 systemd-analyze

systemd-analyze 命令用于查看启动耗时。

```
# 查看启动耗时
systemd-analyze

# 打印信息
Startup finished in 1.809s (kernel) + 49.887s (userspace) = 51.696s
graphical.target reached after 49.834s in userspace
```

```
# 查看每个服务的启动耗时
systemd-analyze blame

# 打印信息
44.661s snapd.seeded.service
 3.820s man-db.service
 3.586s blueman-mechanism.service
 3.500s fstrim.service
 2.270s networkd-dispatcher.service
 2.214s dev-mmcblk0p3.device
 2.051s adbd.service
 1.884s snapd.service
 1.504s async.service
 1.355s logrotate.service
 1.025s accounts-daemon.service
  960ms NetworkManager.service
  823ms polkit.service
  687ms systemd-resolved.service
  627ms avahi-daemon.service
  613ms ModemManager.service
  550ms systemd-journald.service
  530ms systemd-logind.service
```

(下页继续)

(续上页)

```
505ms switcheroo-control.service
481ms apport.service
451ms wpa_supplicant.service
441ms e2scrub_reap.service
427ms colord.service
425ms systemd-udev-trigger.service
408ms networking.service
391ms kerneloops.service
355ms fwupd-refresh.service
345ms ssh.service
303ms packagekit.service
284ms alsa-restore.service
265ms user@0.service
249ms user@112.service
225ms systemd-udevd.service
205ms ntp.service
184ms rsyslog.service
168ms lightdm.service
156ms pppd-dns.service
148ms rkwifibt.service
136ms systemd-user-sessions.service
127ms e2scrub_all.service
124ms systemd-fsck@dev-disk-by\x2dpartlabel-boot.service
106ms systemd-journal-flush.service
102ms dev-mqueue.mount
101ms bluetooth.service
 97ms sys-kernel-debug.mount
 93ms vsftpd.service
 90ms motd-news.service
 90ms sys-kernel-tracing.mount
 89ms systemd-sysctl.service
 88ms systemd-sysusers.service
```

(下页继续)

(续上页)

```
   86ms systemd-backlight@backlight:backlight.service
   82ms systemd-random-seed.service
   73ms modprobe@pstore_blk.service
   72ms modprobe@pstore_zone.service
   71ms modprobe@efi_pstore.service
   67ms systemd-pstore.service
   62ms modprobe@chromeos_pstore.service
   60ms systemd-modules-load.service
   58ms systemd-remount-fs.service
   49ms systemd-tmpfiles-setup.service
   47ms plymouth-quit-wait.service
   44ms systemd-update-utmp-runlevel.service
   41ms user-runtime-dir@112.service
   38ms systemd-tmpfiles-setup-dev.service
   35ms user-runtime-dir@0.service
   35ms systemd-update-utmp.service
   27ms sys-fs-fuse-connections.mount
   24ms sys-kernel-config.mount
   21ms boot.mount
   21ms plymouth-read-write.service
   18ms ifupdown-pre.service
   18ms rtkit-daemon.service
   13ms tmp.mount
    3ms snapd.socket
```

```
# 显示瀑布状的启动过程流
systemd-analyze critical-chain

# 打印信息
The time when unit became active or started is printed after the "@"␣
↪character.
The time the unit took to start is printed after the "+" character.
```

(下页继续)

(续上页)

```
graphical.target @49.834s
└─multi-user.target @49.834s
  └─snapd.seeded.service @5.169s +44.661s
    └─snapd.service @3.271s +1.884s
      └─basic.target @3.094s
        └─sockets.target @3.094s
          └─snapd.socket @3.089s +3ms
            └─sysinit.target @3.069s
              └─systemd-update-utmp.service @3.033s +35ms
                └─systemd-tmpfiles-setup.service @2.973s +49ms
                  └─local-fs.target @2.960s
                    └─boot.mount @2.938s +21ms
                      └─systemd-fsck@dev-disk-by\x2dpartlabel-boot.service␣
↪@2.809s +124ms
                        └─dev-disk-by\x2dpartlabel-boot.device @2.778s
```

```
# 显示指定服务的启动流
$ systemd-analyze critical-chain sshd.service
```

17.5.3 hostnamectl

hostnamectl 命令可用于查看当前的主机信息，当然也可以直接输入 hostname 来查看，但是当我们想要修改主机名的时候就显得没那么方便，每次都要找到 hostname 文件，并将其打开再做修改，hostnamectl 使得我们对主机名的操作更加简便。

```
# 显示当前主机的信息
hostnamectl

# 打印信息
    Static hostname: lubancat
```

(下页继续)

(续上页)

```
          Icon name: computer
         Machine ID: 994b8b2798844992a0691925b398fab5
            Boot ID: 21de30a7c6ee4a75b619fe31bd53a415
   Operating System: Debian GNU/Linux 10 (buster)
             Kernel: Linux 4.19.232
       Architecture: arm64


# 修改主机名
root@npi:~# sudo hostnamectl set-hostname cat

# 再次查看主机信息
root@npi:~# hostnamectl
    Static hostname: cat
          Icon name: computer
         Machine ID: 994b8b2798844992a0691925b398fab5
            Boot ID: 21de30a7c6ee4a75b619fe31bd53a415
   Operating System: Debian GNU/Linux 10 (buster)
             Kernel: Linux 4.19.232
       Architecture: arm64
```

可以看到主机名 Static hostname 已被修改。

17.5.4 localectl

localectl 命令可用于查询与修改系统的本地化 (locale) 与键盘布局的设置。它通过与 systemd-localed.service(8) 通信来修改例如 /etc/locale.conf 与 /etc/vconsole.conf 之类的配置文件。本地化设置控制着用户界面的语言、字符类型与字符编码、日期时间与货币符号的表达方式等许多细节。

```
# 查看本地化设置
localectl
```

(下页继续)

(续上页)

```
# 打印消息
    System Locale: LANG=C.UTF-8
        VC Keymap: n/a
       X11 Layout: us
        X11 Model: pc105


# 设置系统本地语言和键盘
root@zhan:~# sudo localectl set-locale LANG=en_GB.utf8
root@zhan:~# sudo localectl set-keymap en_GB

# 再次查看
root@lubancat:~# localectl
    System Locale: LANG=en_GB.utf8
        VC Keymap: en_GB
       X11 Layout: us
        X11 Model: pc105
```

17.5.5 timedatectl

timedatectl 命令可以查询和更改系统时钟和设置，你可以使用此命令来设置或更改当前的日期，时间和时区，或实现与远程 NTP 服务器的自动系统时钟同步。

```
# 查看当前时区设置
timedatectl

# 显示所有可用的时区
timedatectl list-timezones

# 选择中国上海的时区
timedatectl set-timezone "Asia/Shanghai"
```

(下页继续)

(续上页)

```
# 关闭网络时间同步
timedatectl set-ntp no

# 开启网络时间同步
timedatectl set-ntp yes

# 设置时间和日期
timedatectl set-time 9:40:20
timedatectl set-time 2021-4-16
```

在设置时间与日期时要关闭时间同步功能。

17.5.6 loginctl

loginctl 命令可用于检查和控制 systemd 的状态；查看已经登录的用户会话消息。

```
# 显示所有会话及属性
loginctl -a

# 显示会话配置消息
loginctl show-session

# 列出显示指定用户的信息
loginctl show-user root
```

第 18 章　Linux 制作 deb 包的方法

为了方便管理一些程序或脚本，可以将这些程序以及脚本都制作在一个 deb 包中，本章节将介绍如何制作一个 deb 包，制作 deb 的方式很多，如使用 dpkg-deb 方式、使用 checkinstall 方式、使用 dh_make 方式及修改原有的 deb 包，本章将介绍如何从零制作一个自己的 deb 包，以及修改原有的 deb 包。

18.1 什么是 deb 包？

deb 包是在 linux 系统下的一种安装包，有时我们在网上下载的 Linux 软件安装包也会以 deb 包的形式出现，由于它是基于 tar 包的，所以同样会记录着文件的权限信息（读、写、可执行）、所有者、用户组等。

我们可以使用命令：dpkg -l 来查看系统以及安装了哪些 deb 包。

```
1 cat@lubancat:~$ dpkg -l
2 Desired=Unknown/Install/Remove/Purge/Hold
3 | Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-
  ↪pend
4 |/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
5 ||/ Name                                    Version                      ␣
  ↪     Architecture Description
6 +++-=======================================-
  ↪=======================================-============
  ↪================================================================================
7 ii  acpi-support-base                       0.142-8                      ␣
  ↪     all          scripts for handling base ACPI events such as the power␣
  ↪button
8 ii  acpid                                  1:2.0.31-1                   ␣
  ↪     arm64        Advanced Configuration and Power Interface event daemon
```

(下页继续)

(续上页)

```
ii  adduser                                 3.118                ␣
↪     all          add and remove users and groups
ii  adwaita-icon-theme                     3.30.1-1             ␣
↪     all          default icon theme of GNOME
ii  alsa-utils                             1.1.8-2              ␣
↪     arm64        Utilities for configuring and using ALSA
ii  anacron                                2.3-28               ␣
↪     arm64        cron-like program that doesn't go by time
ii  apt                                    1.8.2.3              ␣
↪     arm64        commandline package manager
ii  apt-transport-https                    1.8.2.3              ␣
↪     all          transitional package for https support
ii  apt-utils                              1.8.2.3              ␣
↪     arm64        package management related utility programs
ii  aspell                                 0.60.7~20110707-6+deb10u1 ␣
↪     arm64        GNU Aspell spell-checker
ii  aspell-en                              2018.04.16-0-1       ␣
↪     all          English dictionary for GNU Aspell
ii  base-files                             10.3+deb10u13        ␣
↪     arm64        Debian base system miscellaneous files
ii  base-passwd                            3.5.46               ␣
↪     arm64        Debian base system master password and group files
ii  bash                                   5.0-4                ␣
↪     arm64        GNU Bourne Again SHell
ii  binfmt-support                         2.2.0-2              ␣
↪     arm64        Support for extra binary formats
ii  blueman                                2.0.8-1+deb10u1      ␣
↪     arm64        Graphical bluetooth manager
ii  bluez                                  5.50-1.2~deb10u2     ␣
↪     arm64        Bluetooth tools and daemons
ii  bluez-obexd                            5.50-1.2~deb10u2     ␣
↪     arm64        bluez obex daemon
```

(下页继续)

(续上页)

```
ii  bsdmainutils                                     11.1.2+b1                         arm64        collection of more utilities from FreeBSD
ii  bsdutils                                         1:2.33.1-0.1                      arm64        basic utilities from 4.4BSD-Lite
ii  bubblewrap                                       0.3.1-4                           arm64        setuid wrapper for unprivileged chroot and namespace manipulation
ii  busybox                                          1:1.30.1-4                        arm64        Tiny utilities for small and embedded systems
ii  bzip2                                            1.0.6-9.2~deb10u2                 arm64        high-quality block-sorting file compressor - utilities
ii  ca-certificates                                  20200601~deb10u2                  all          Common CA certificates
ii  camera-engine-rkaiq                              2.0x60.1                          arm64        3A libraries match Rockchip rkisp v21(rk356x).
ii  can-utils                                        2018.02.0-1                       arm64        SocketCAN userspace utilities and tools
ii  cheese                                           3.31.90-1                         arm64        tool to take pictures and videos from your webcam
ii  cheese-common                                    3.31.90-1                         all          Common files for the Cheese tool to take pictures and videos
lines 1-33
```

上面是 LubanCat 板卡 Debian 镜像所安装的 deb 包列表，Name 所对应的列为已安装的 deb 包包名，Version 表示该包的版本号，Architecture 为该包所支持的处理器架构。

18.2 deb 包的组成结构

deb 包一般分成两部分：

- 安装的内容，这部分类似 linux 的根目录，表示需要将软件安装到 linux 系统上的文件目录。
- 控制信息（放在 DEBIAN 目录下），通常 DEBIAN 目录下有如下几个文件。
  - changelog: 文件记录了 deb 包的作者、版本以及最后一次更新日期等信息；
  - control: 文件记录了包名、版本号、架构、维护者及描述等信息；
  - copyright: 文件记录了一些版权信息；
  - postinst: 软件在进行正常目录文件拷贝到系统后需要执行的脚本。
  - postrm 文件: 软件卸载后需要执行的脚本。

其中 control、postinst、postrm 为必要文件。

18.3 从零开始创建自己的 deb 包

安装工具及依赖：

```
sudo apt-get install build-essential debhelper make autoconf automake dpkg-
↪dev fakeroot pbuilder gnupg
```

首先我们创建如下目录及文件

```
hello_deb/
├── DEBIAN
│   ├── control
│   ├── postinst
│   └── postrm
└── opt
    └── hello_deb
        └── hello_deb.sh
```

在 hello_deb 目录下创建 DEBIAN 及 opt/hello_deb 目录，DEBIAN 目录下包含控制信息文件，而在 opt/hello_deb 目录下创建 hello_deb.sh 文件则表示我们需要将 hello_deb.sh 文件安装到 linux 系统的 opt/hello_deb 目录下。

然后分别给予 postinst、postrm、hello_deb.sh 文件可执行权限，postinst 和 postrm 的权限必须 >=0555 且 <=0775。

其中 control 文件所包含信息如下：

列表 1: hello_deb/DEBIAN/control

```
1 Package: hello-deb
2 Version: 1.0.0
3 Section: free
4 Priority: optional
5 Essential: no
6 Architecture: arm64
7 Maintainer: embedfire <embedfire@embedfire.com>
8 Provides: hell_deb
9 Description: deb test
```

注解： control 文件的末尾需添加一个空行，否则会报错“缺失结尾的换行符”

若以后想升级这个 deb 包，可以修改该包的版本号 Version，值得注意的是 Architecture，前面我们也有讲到，就是该 deb 包所支持的处理器架构，因为最终要将该 deb 包安装到 arm64 处理器的板卡上，所以我们应该在 Architecture 中填入 arm64 属性，大家可根据自己的需求做相应修改即可，如果不知道你的处理器架可以通过 dpkg -l 命令来查看已安装的 deb 包持支的架构，或者输入 lscpu 查看处理器信息，aarch64 就是 arm64 架构。若想支持所有架构，可以填入 all 属性，如果 Architecture 属性与当前处理器架构属性不匹配的话，deb 包将无法成功安装，且 control 的属性信息必须以字母或者数字开头，不然可能导致打包出错。

postinst 文件包含信息如下：

列表 2: hello_deb/DEBIAN/postinst

```
#!/bin/bash

if [ "$1" = "upgrade" ] || [ "$1" = "install" ];then
    echo "hello_deb installing"
fi
```

当安装了该 deb 包以后，系统会默认执行 postinst 脚本，通常我们利用该脚本来搭建一些为软件执行的环境（如创建目录、修改权限等），值得注意的是该文件需具有可执行权限。这里写的比较简单, 判断第一个参数，仅供参考。

postrm 文件包含信息如下：

列表 3: hello_deb/DEBIAN/postrm

```
#!/bin/bash

if [ "$1" = "upgrade" ] ; then
    echo "upgrade"
elif [ "$1" = "remove" ] || [ "$1" = "purge" ] ; then
    echo "remove"
fi
```

当卸载了该 deb 包以后，系统会默认执行 postrm 脚本，通常我们利用该脚本来清理环境，值得注意的是该文件具有可执行权限。这里写的比较简单, 判断第一个参数，仅供参考。

最后我们来看下真正的程序主体，为了简单起见，此处以一个简单的脚本为例。

列表 4: hello_deb/opt/hello_deb/hello_deb.sh

```
#! /bin/bash

echo Hello deb!
echo This is a test script!!!
```

脚本仅仅是打印两句信息，用户可自行设置需要执行的程序。

万事俱备，只欠东风，当备齐了制作 deb 包的基本原材料之后我们便可以开始制作属于自己的 deb 包了，进入 hello_deb 目录下，也就是 DEBIAN 及 home 文件夹所在的目录，接着输入如下命令来构建软件包。

```
sudo dpkg-deb -b ../hello_deb ../hello_deb_1.0.0_arm64.deb
```

其中 dpkg-deb 是构建 deb 包命令，-b 参数表示要构建一个 deb 包，../hello_deb 参数表示要构建 deb 包原材料的路径，../hello_deb_1.0.0_arm64.deb 参数表示将该 deb 包构建在当前目录的上级目录中，一般我们构建 deb 包的名字都会遵循这么一个原则，其命名方式为：软件名称 + 软件版本号 + 该软件所支持的处理器架构，如软件名为 hello_deb，版本号为 1.0.0，所支持的处理器架构为 arm64。

打包成功后会输出如下信息，并可在上级目录查看到 deb 安装包：

```
jiawen@dev120:~/deb/hello_deb$ sudo dpkg-deb -b ../hello_deb ../hello_deb_1.
↪0.0_arm64.deb
dpkg-deb: 正在 '../hello_deb_1.0.0_arm64.deb' 中构建软件包 'hello-deb'。
```

制作好自己的 deb 包后我们需要验证一下是否真的制作成功，可以如下命令查看已制作的 deb 包文件内容：

```
# 命令
dpkg -c  hello_deb_1.0.0_arm64.deb

# 打印信息
```

(下页继续)

(续上页)

```
drwxrwxr-x jiawen/jiawen     0 2022-10-12 09:27 ./
drwxrwxr-x jiawen/jiawen     0 2022-10-12 09:28 ./opt/
drwxrwxr-x jiawen/jiawen     0 2022-10-12 09:28 ./opt/hello_deb/
-rwxrwxrwx jiawen/jiawen    59 2022-10-12 09:41 ./opt/hello_deb/hello_deb.sh
```

也可使用如下命令查看 deb 包信息：

```
# 命令
dpkg --info  hello_deb_1.0.0_arm64.deb

# 打印信息
new Debian package, version 2.0.
size 976 bytes: control archive=496 bytes.
    190 字节,     9 行      control
    100 字节,     4 行   *  postinst             #!/bin/bash
    138 字节,     7 行   *  postrm               #!/bin/bash
Package: hello-deb
Version: 1.0.0
Section: free
Priority: optional
Essential: no
Architecture: arm64
Maintainer: embedfire <embedfire@embedfire.com>
Provides: hell_deb
Description: deb test
```

将该 deb 包拷贝到 LubanCat 板卡的文件系统下，输入“sudo dpkg -i hello_deb_1.0.0_arm64.deb”命令即可安装，其中-i 参数表示安装软件，即 install，并且在安装完软件之后可以输入“dpkg -s hello-dev”命令查看是否安装了软件。如下所示

```
cat@lubancat:~$ sudo dpkg -i hello_deb_1.0.0_arm64.deb
Selecting previously unselected package hello-deb.
(Reading database ... 74783 files and directories currently installed.)
```

(下页继续)

(续上页)

```
Preparing to unpack hello_deb_1.0.0_arm64.deb ...
Unpacking hello-deb (1.0.0) ...
Setting up hello-deb (1.0.0) ...
cat@lubancat:~$ dpkg -s hello-deb
Package: hello-deb
Status: install ok installed
Priority: optional
Section: free
Maintainer: embedfire <embedfire@embedfire.com>
Architecture: arm64
Version: 1.0.0
Provides: hell_deb
Description: deb test
```

或者输入“dpkg -l | grep hello-deb”命令查看你的软件是否在已安装软件列表里面。

```
cat@lubancat:~$ dpkg -l | grep hello-deb
ii  hello-deb          1.0.0           arm64        deb test
```

验证安装完成之后查看开发板/opt/hello_deb 目录下是否存在 hello_deb.sh 文件。

```
cat@lubancat:~$ ls /opt/
hello_deb
cat@lubancat:~$ ls /opt/hello_deb/
hello_deb.sh
```

执行看看效果

```
cat@lubancat:~$ /opt/hello_deb/hello_deb.sh
Hello deb!
This is a test script!!!
```

到此，制作 deb 包的基本流程已介绍完毕。

18.4 从根据已有 deb 包修改内容

下面介绍下如何修改已有的 deb 包，以刚刚创建的 deb 包为例，直接在 LubanCat 板卡上修改。

在 LubanCat 板卡上安装工具及依赖：

```
sudo apt-get install build-essential debhelper make autoconf automake dpkg-
↪dev fakeroot pbuilder gnupg
```

新建一个 update_deb 目录，使用 **dpkg -X** 命令将 deb 包解压到 update_deb 目录中。

```
cat@lubancat:~$ ls
Desktop    Downloads  Pictures  Templates  hello_deb_1.0.0_arm64.deb
Documents  Music      Public    Videos
cat@lubancat:~$ mkdir update_deb
cat@lubancat:~$ sudo dpkg -X  hello_deb_1.0.0_arm64.deb  update_deb/
./
./opt/
./opt/hello_deb/
./opt/hello_deb/hello_deb.sh
cat@lubancat:~$
```

进入 update_deb 目录下可看到并没 DEBIAN 相关目录，在 update_deb 目录下使用 **dpkg -e** 解压出控制文件相关信息。

```
cat@lubancat:~/update_deb$ ls
opt
cat@lubancat:~/update_deb$ sudo dpkg -e ../hello_deb_1.0.0_arm64.deb
cat@lubancat:~/update_deb$ ls -al
total 16
drwxrwxr-x  4 1001 1001 4096 Oct 12 13:37 .
drwxr-xr-x 15 cat  cat  4096 Oct 12 13:35 ..
drwxr-xr-x  2 root root 4096 Oct 12 09:28 DEBIAN
drwxrwxr-x  3 1001 1001 4096 Oct 12 09:28 opt
```

(下页继续)

(续上页)

```
cat@lubancat:~/update_deb$ tree
.
|-- DEBIAN
|   |-- control
|   |-- postinst
|   `-- postrm
`-- opt
    `-- hello_deb
        `-- hello_deb.sh

3 directories, 4 files
```

此时就可以对程序主体进行修改了，我们在 opt/hello_deb/hello_deb.sh 下添加一句打印消息，如下所示

列表 5: update_deb/opt/hello_deb/hello_deb.sh

```
#! /bin/bash

echo Hello deb!
echo This is a test script!!!
echo update deb!
```

修改 DEBIAN/control 下的版本信息，将 Version 修改为 1.0.1

列表 6: update_deb/DEBIAN/control

```
Package: hello-deb
Version: 1.0.1
Section: free
Priority: optional
Essential: no
Architecture: arm64
```

(下页继续)

(续上页)

```
Maintainer: embedfire <embedfire@embedfire.com>
Provides: hell_deb
Description: deb test
```

此时便可以重新打包应用程序并安装了。

```
cat@lubancat:~/update_deb$ sudo dpkg-deb -b ../update_deb ../hello_deb_1.0.
↪1_arm64.deb
dpkg-deb: building package 'hello-deb' in '../hello_deb_1.0.1_arm64.deb'.

cat@lubancat:~$ sudo dpkg -i hello_deb_1.0.1_arm64.deb
(Reading database ... 103608 files and directories currently installed.)
Preparing to unpack hello_deb_1.0.1_arm64.deb ...
Unpacking hello-deb (1.0.1) over (1.0.0) ...
upgrade
Setting up hello-deb (1.0.1) ...

cat@lubancat:~$ /opt/hello_deb/hello_deb.sh
Hello deb!
This is a test script!!!
update deb!
```

18.5 简单实例：通过安装 deb 包创建开机自启服务

通过安装 deb 包在 LubanCat 板卡上实现程序开机自启，在实际项目上的应用极为普遍，下面通过一个简单的实例介绍相关如何使用创建开机自动的 deb 程序。

创建一个新的 deb 包目录，文件结构如下

```
hello_deb
├── DEBIAN
│   ├── control
```

(下页继续)

(续上页)

```
4  |   ├── postinst
5  |   └── postrm
6  ├── etc
7  |   └── systemd
8  |       └── system
9  |           └── hello.service
10 └── opt
11     └── hello_deb
12         └── hello_deb.sh
```

其中 /etc/systemd/system/hello.service 为自启服务，/opt/hello_deb/hello_deb.sh 为程序主体。DEBIAN 目录为 deb 包的一些描述信息以及安装卸载执行脚本等，对于一些基本概念此处便不再赘述，直接查看各个文件中的内容。

DEBIAN/control 文件内容如下所示

列表 7: DEBIAN/control

```
1  Package: hello-deb
2  Version: 1.0.2
3  Section: free
4  Priority: optional
5  Essential: no
6  Architecture: arm64
7  Maintainer: embedfire <embedfire@embedfire.com>
8  Provides: hell_deb
9  Description: deb test
```

DEBIAN/postinst 文件内容如下所示，脚本的内容很简单只是使能了 hello 相关服务并启动它。

列表 8: DEBIAN/postinst

```
1  #!/bin/bash
2
```

(下页继续)

(续上页)

```
if [ "$1" = "upgrade" ] || [ "$1" = "install" ];then
    echo "hello_deb installing"
    systemctl enable hello
    systemctl start hello
fi
```

DEBIAN/postinst 文件内容如下所示，当卸载程序时，关闭 hello 相关服务。

列表 9: DEBIAN/postrm

```
if [ "$1" = "upgrade" ] ; then
    echo "upgrade"
elif [ "$1" = "remove" ] || [ "$1" = "purge" ] ; then
    echo "remove"
    systemctl disable hello
fi
```

/etc/systemd/system/hello.service 服务内容如下，重点在于第六行，指定需要执行的程序。

列表 10: /etc/systemd/system/hello.service

```
[Unit]
Description = hello daemon

[Service]
ExecStart = /opt/hello_deb/hello_deb.sh
Restart = always
Type = simple

[Install]
WantedBy = multi-user.target
```

/opt/hello_deb/hello_deb.sh 为程序主体，内容如下所示，以每 3 秒的速度向/tmp/hello.log 写入 Hello Embedfire。

列表 11: /opt/hello_deb/hello_deb.sh

```
#!/bin/bash

while true
do
    echo Hello Embedfire >> /tmp/hello.log
    sleep 3
done
```

将以上目录打包构建成 deb 文件并在板板卡上使用 dpkg 命令安装即可。重启后使用 “systemctl status hello“ 查看服务相关状态如下。

```
root@lubancat:~# systemctl status hello
● hello.service - hello daemon
    Loaded: loaded (/etc/systemd/system/hello.service; enabled; vendor␣
 ↪preset:>
    Active: active (running) since Wed 2022-10-12 13:54:55 CST; 12s ago
  Main PID: 372 (hello_deb.sh)
    Memory: 2.5M
    CGroup: /system.slice/hello.service
            ├─372 /bin/bash /opt/hello_deb/hello_deb.sh
            └─817 sleep 3
lines 1-8/8 (END)

root@lubancat:~# cat /tmp/hello.log
Hello Embedfire
Hello Embedfire
Hello Embedfire
Hello Embedfire
Hello Embedfire
```

也可通过查看/tmp/hello.log 文件查看程序是否正确执行。

第 19 章　添加系统自启动服务

本章总结两种自启动方式，分别通过 Systemd 服务以及桌面系统自带的自启动方式实现程序的自启动。

本章节适用于已经将镜像烧录到板卡并正常启动系统的情况。

19.1 Systemd 方式

创建 hello.service 服务的过程已经在 **探索 Systemd** 讲解过了，这里我们只对实现自启动的方式进行讲解，更多 Systemd 相关内容请查看 **探索 Systemd** 章节。

19.1.1 编写脚本

cd 进入/opt 目录下，使用 vim 编写一个 hello.sh 脚本。

```
#!/bin/bash

while true
do
    echo Hello Lubancat >> /tmp/hello.log
    sleep 3
done
```

该脚本实现的功能是每隔 3 秒就打印“Hello Lubancat”字符串到/tmp/hello.log 文件中。编写好后记得赋予 hello.sh 可执行权限。

```
sudo chmod 0755 hello.sh
```

19.1.2 创建配置文件

在/etc/systemd/system/目录下创建一个 hello.service 配置文件，内容如下。

```
[Unit]
Description = hello daemon

[Service]
ExecStart = /opt/hello.sh
Restart = always
Type = simple

[Install]
WantedBy = multi-user.target
```

其中 ExecStart 字段定义了 hello.service 服务的自启动脚本为/opt/hello.sh，当我们使能了 hello.service 开机自启功能，在开机后便会执行/opt/hello.sh。Restart = always 表示指进程或服务意外故障的时候可以自动重启的模式。Type = simple 为默认的，可以不填。WantedBy 指定服务由谁启动，WantedBy = multi-user.target 表示在系统启动到多用户模式时就会把这个服务添加到启动顺序中。如果需要在图形用户界面后再启动，需要设置为 WantedBy = graphical.target，表示需要在系统进入图形用户界面模式时启动。

也就是说，WantedBy 参数对于服务是否需要桌面环境十分重要，如果服务需要桌面环境，然而设置 WantedBy = multi-user.target 在桌面启动前就调用自己的启动脚本，将会导致服务启动失败！

```
# 在桌面前就启动服务
WantedBy = multi-user.target


# 进入桌面后启动服务
WantedBy = graphical.target
```

19.1.3 使能 hello.service 开机自启功能

输入命令“sudo systemctl list-unit-files –type=service | grep hello”查看 hello.service 是否被添加到了服务列表。

```
sudo systemctl list-unit-files --type=service | grep hello
```

```
root@lubancat:/etc/systemd/system# sudo systemctl list-unit-files --type=service | gre
p hello
hello.service                              disabled
root@lubancat:/etc/systemd/system#
```

可以看到 hello.service 处于 disable 状态，如果你输入上面命令后没有任何显示，那你创建的服务就处理问题，需要仔细排查。我们输入下面命令使 hello.service 开机自启动。

```
sudo systemctl enable hello
sudo systemctl start hello
```

然后使用 reboot 命令重启系统，启动系统后输入“sudo systemctl status hello”命令即可看到 hello.service 处于运行状态。

```
sudo systemctl status hello


cat /tmp/hello.log
```

```
root@lubancat:~# sudo systemctl status hello
● hello.service - hello daemon
   Loaded: loaded (/etc/systemd/system/hello.service; enabled; vendor preset: en
   Active: active (running) since Tue 2022-10-11 13:59:32 CST; 58s ago
 Main PID: 344 (hello.sh)
   Memory: 1.3M
   CGroup: /system.slice/hello.service
           ├─ 344 /bin/bash /opt/hello.sh
           └─1046 sleep 3
root@lubancat:~# cat /tmp/hello.log
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
```

19.2 桌面系统方式

如果使用带桌面环境的镜像，例如 xfce、xfce-full、gnome 版本的镜像，可以通过桌面系统自带的自启动服务实现，也十分建议桌面用户使用此种方式。

19.2.1 编写自启动配置脚本

在桌面登录用户的家目录下创建启动文件：

```
# 以 cat 用户为例
mkdir /home/cat/.config/autostart

# 创建配置文件
vim /home/cat/.config/autostart/xfce-terminal.desktop
```

在 xfce-terminal.desktop 文件中添加以下内容：

```
[Desktop Entry]
Type=Application
Exec=/opt/test.sh
Hidden=false
NoDisplay=false
X-GNOME-Autostart-enabled=true
Name=My App
Comment=Start My App on login
```

该配置文件的内容说明如下：

- Type=Application：指定类型为应用程序。
- Exec=/opt/test.sh：指定要执行的命令或脚本的路径。
- Hidden=false：不隐藏该自启动项。
- NoDisplay=false：在桌面环境的启动器中显示该自启动项。

- X-GNOME-Autostart-enabled=true：启用桌面环境下的自启动功能。
- Name=My App：给自启动项指定一个名称。
- Comment=Start My App on login：自启动项的注释，描述其在登录时启动的作用。

通过这个配置文件，当用户登录桌面环境时，系统会自动执行/opt/test.sh 脚本并启动相应的应用程序。

19.2.2 编写自启动脚本

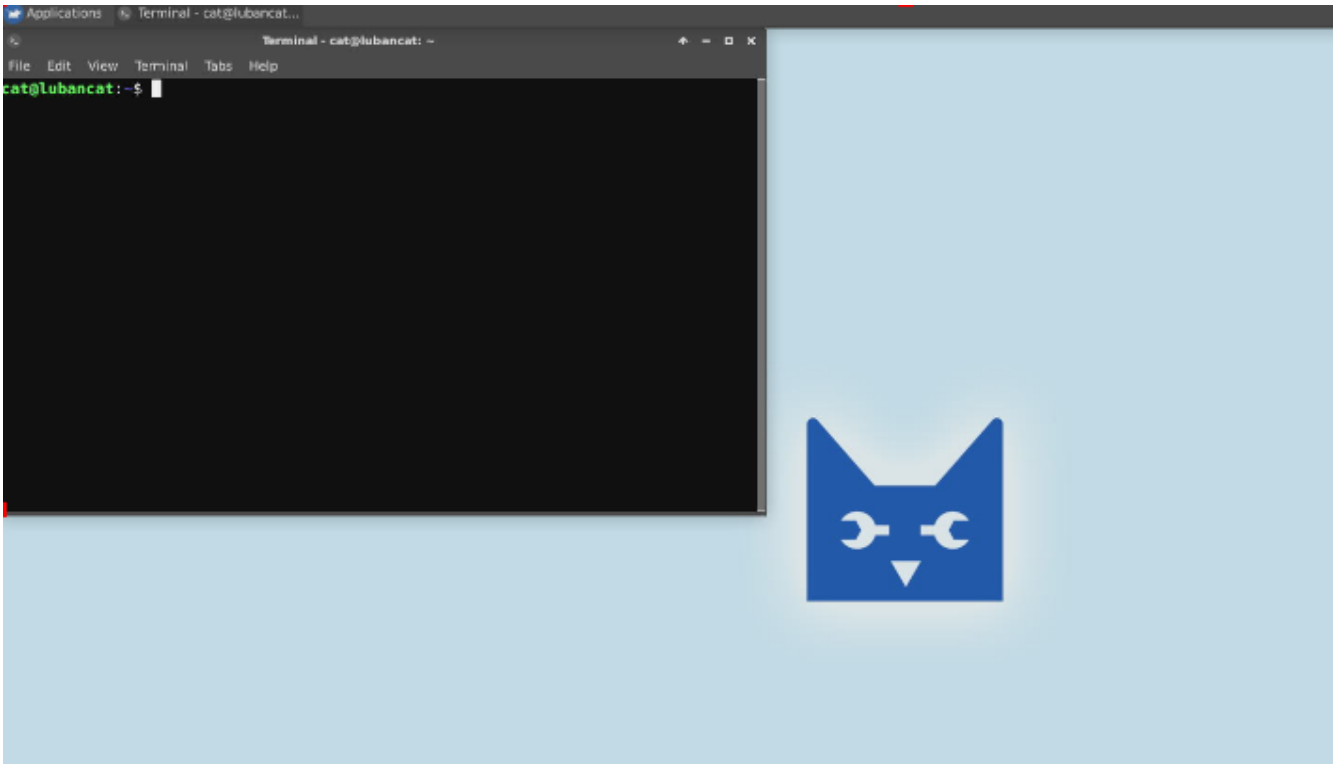
根据上面的自启动配置，创建自启动脚本，cd 进入/opt 目录下，编写一个 test.sh 脚本。

```
#!/bin/bash

# 打开桌面终端窗口
xfce4-terminal
```

以上脚本实现的功能是在桌面中打开一个终端窗口，编写完成后需要赋予权限。

```
sudo chmod 777 /opt/test.sh
```

完成以上操作后，重启开发板即可，进入桌面后会自动打开一个终端，如下图：


Applications
Terminal - cat@lubancat...
Terminal - cat@lubancat: ~
File Edit View Terminal Tabs Help
cat@lubancat:~$

第 20 章　添加系统服务到根文件系统构建脚本

除了直接在启动的板卡上创建 Systemd 服务，我们也可以在根文件系统的构建脚本中添加。

此处以 Debian 镜像构建脚本为例。

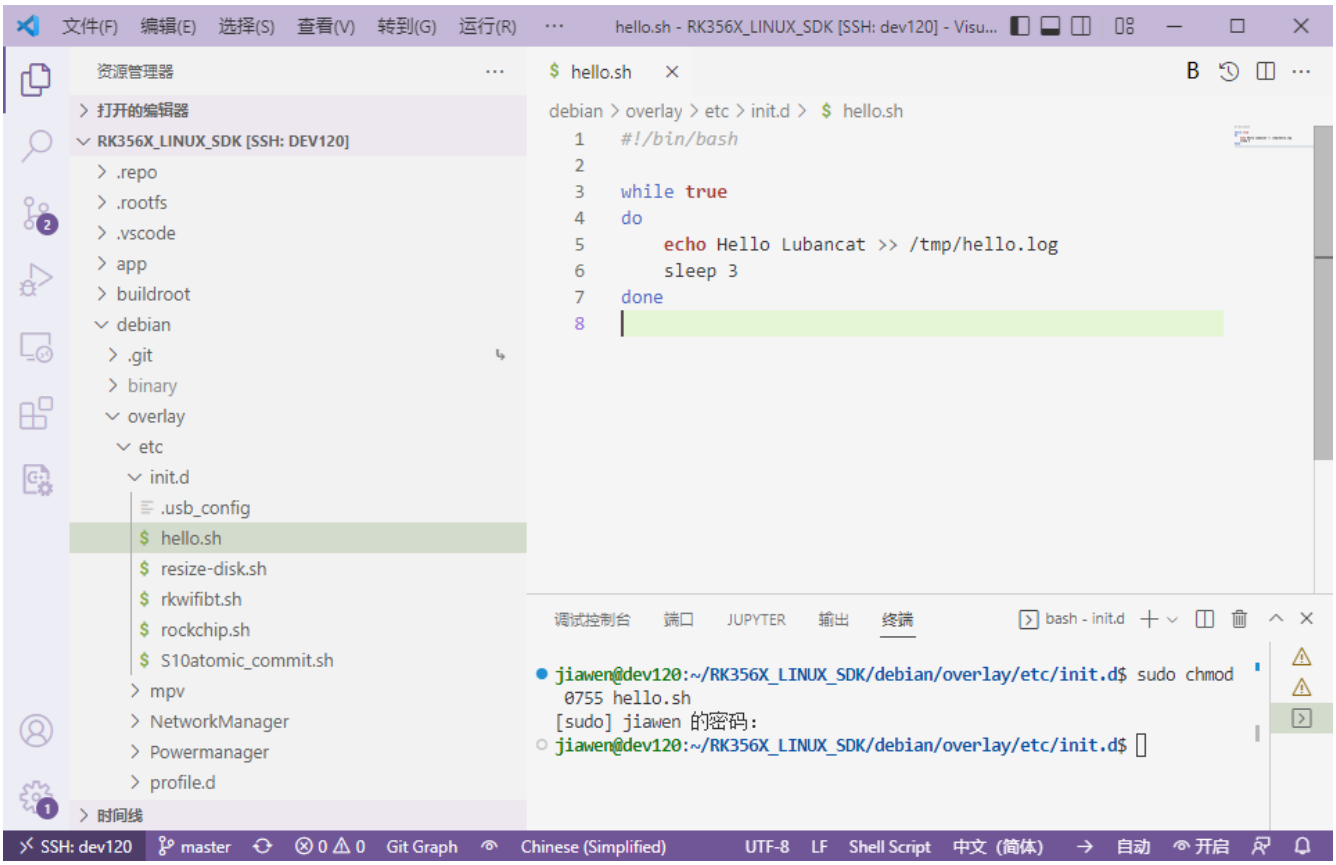
20.1 编写脚本

打开 LubanCat-SDK，并在 debian/overlay/etc/init.d 目录下创建 hello.sh 脚本

```
#!/bin/bash

while true
do
    echo Hello Lubancat >> /tmp/hello.log
    sleep 3
done
```

该脚本实现的功能是每隔 3 秒就打印“Hello Lubancat”字符串到/tmp/hello.log 文件中。编写好后记得赋予 hello.sh 可执行权限。

```
sudo chmod 0755 hello.sh
```



20.2 创建配置文件

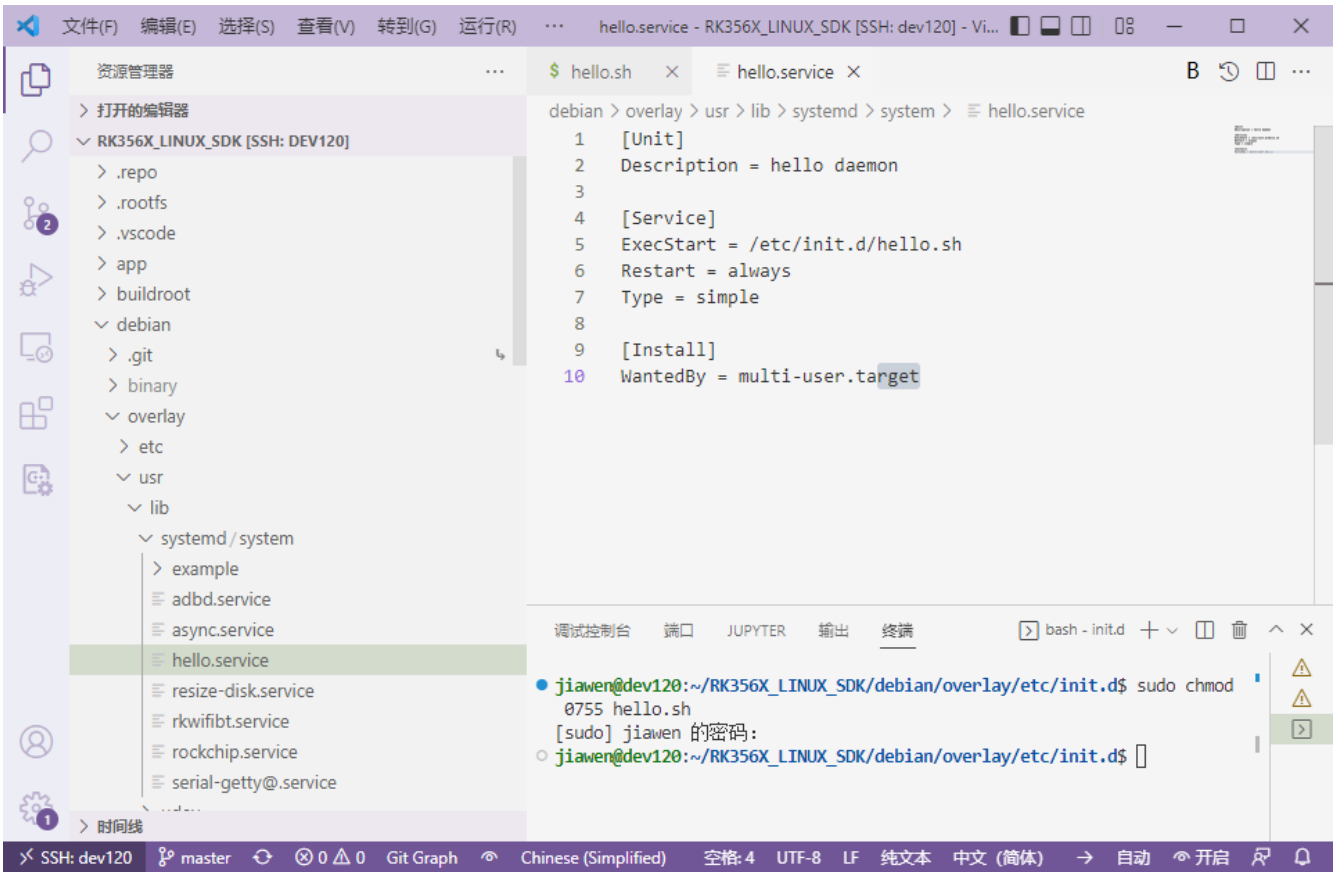
在 debian/overlay/usr/lib/systemd/system/目录下创建一个 hello.service 配置文件，内容如下。

```
[Unit]
Description = hello daemon

[Service]
ExecStart = /etc/init.d/hello.sh
Restart = always
Type = simple

[Install]
```

(下页继续)

(续上页)

```
10 WantedBy = multi-user.target
```



其中 ExecStart 字段定义了 hello.service 服务的自启动脚本为/etc/init.d/hello.sh，当我们使能了 hello.service 开机自启功能，在开机后便会执行/etc/init.d/hello.sh。Restart = always 表示指进程或服务意外故障的时候可以自动重启的模式。Type = simple 为默认的，可以不填。

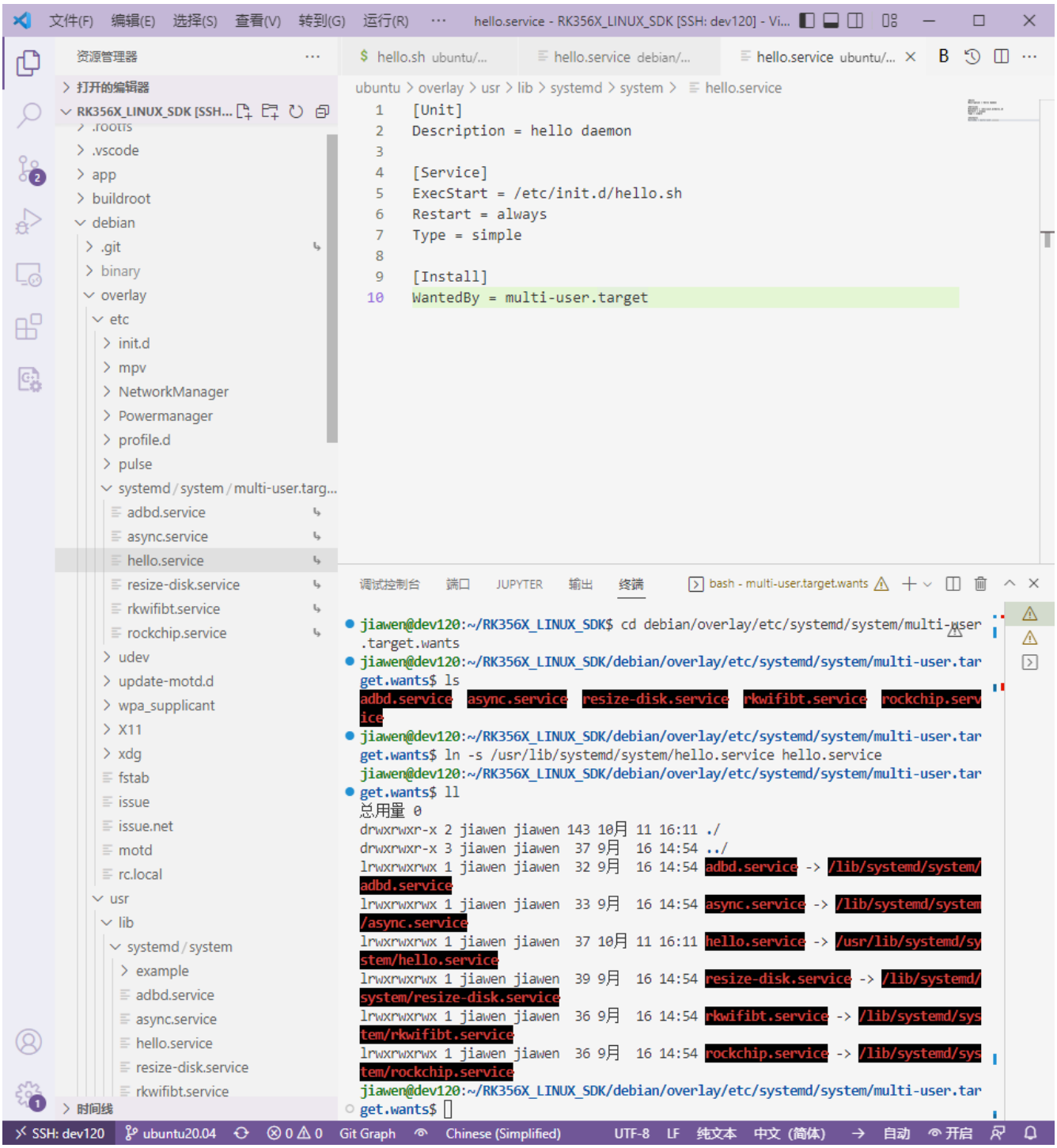
20.3 使能 hello.service 开机自启功能

我们还需要在/etc/systemd/system/的 multi-user.target.wants 目录下创建指向/usr/lib/systemd/system/hello.service 的软连接，来使 hello.service 加入 multi-user.target，在这个组里的所有服务，都将开机启动

我们来到 LubanCat-SDK 的 debian/overlay/etc/systemd/system/multi-user.target.wants 目录下，使用 ln

命令创建软连接。

```
# 进入 ulti-user.target.wants 目录
cd debian/overlay/etc/systemd/system/multi-user.target.wants

# 创建软连接
ln -s /usr/lib/systemd/system/hello.service hello.service
```



修改完成后我们重新构建根文件系统，等待。

系统镜像构建完成后重新烧录板卡，启动系统后输入“sudo systemctl status hello”命令即可看到

hello.service 处于运行状态。

```
sudo systemctl status hello


cat /tmp/hello.log
```

```
root@lubancat:~# sudo systemctl status hello
● hello.service - hello daemon
     Loaded: loaded (/lib/systemd/system/hello.service; enabled; vendor preset:
enabled)
     Active: active (running) since Wed 2022-08-31 23:27:36 CST; 1 mo
nths 10 days ago
   Main PID: 375 (hello.sh)
     Memory: 2.5M
     CGroup: /system.slice/hello.service
             ├─ 375 /bin/bash /etc/init.d/hello.sh
             └─1878 sleep 3
root@lubancat:~# cat /tmp/hello.log
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
Hello Lubancat
```

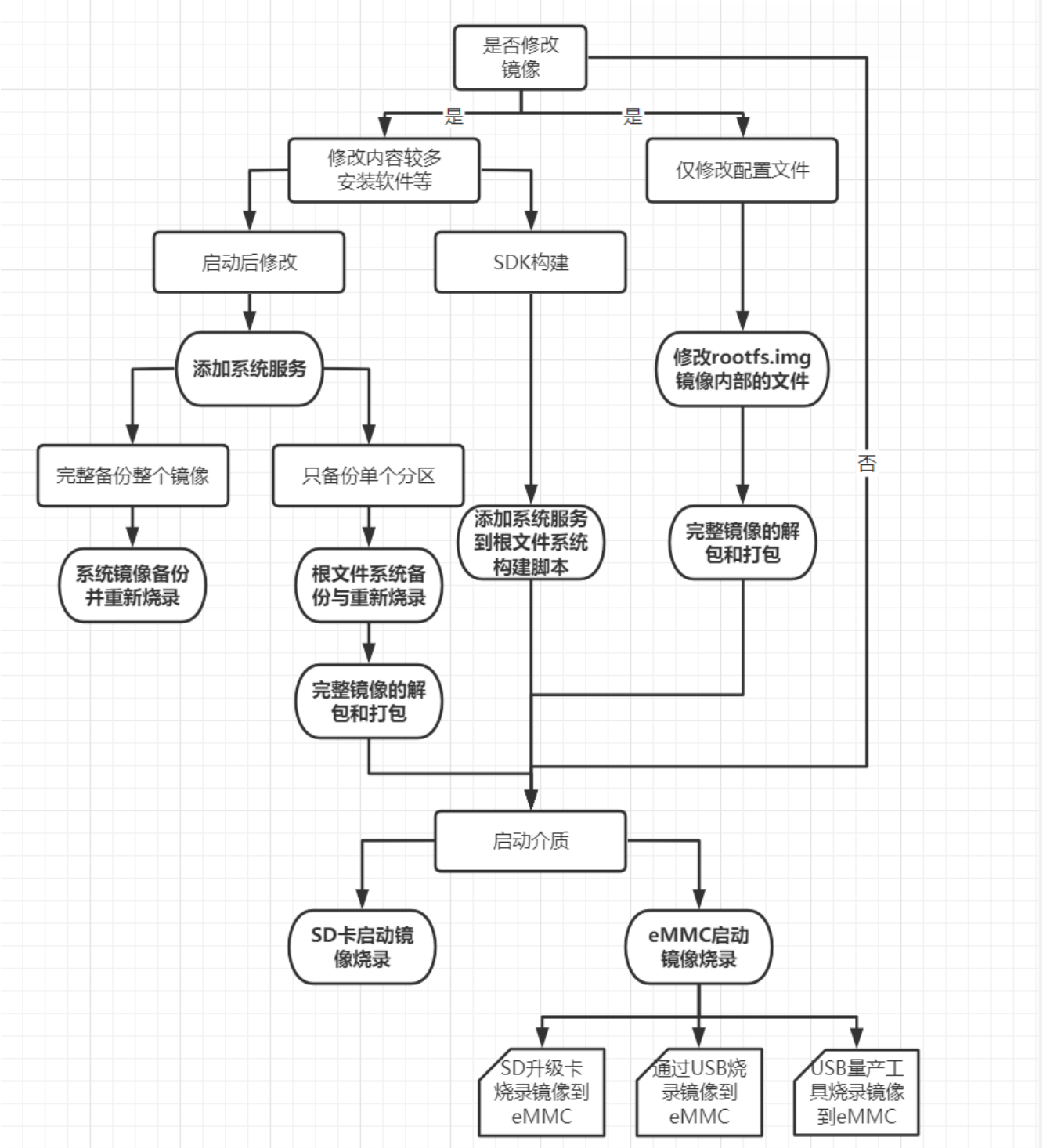
第 21 章　备份与量产说明

对于量产与备份，本章节 **很重要很重要很重要**。

野火 LubanCat-RK 系列板卡目前仅支持 SD 卡和 eMMC 两种启动方式，所以在量产过程中，主要就是根据产品来选择合适的系统镜像下载方式。

对于系统镜像存储介质，我们更推荐使用 eMMC，eMMC 具有更快的读写速度和更高的稳定系，而 SD 卡会有兼容性较差的问题，使用部分品牌部分系列的 SD 卡会导致系统无法启动或运行异常。另外，安卓系统镜像无法运行在 SD 卡上。

除了镜像下载需要特别说明之外，部分用户还有修改根文件系统，自定义系统服务等需求，导致本部分内容会过于杂乱。我们列举出了一些常用技术路线，用户可以根据自己的情况按图索骥。


是否修改镜像
是
是
否
修改内容较多安装软件等
仅修改配置文件
启动后修改
SDK构建
添加系统服务
修改rootfs.img镜像内部的文件
完整备份整个镜像
只备份单个分区
添加系统服务到根文件系统构建脚本
完整镜像的解包和打包
系统镜像备份并重新烧录
根文件系统备份与重新烧录
完整镜像的解包和打包
启动介质
SD卡启动镜像烧录
eMMC启动镜像烧录
SD升级卡烧录镜像到eMMC
通过USB烧录镜像到eMMC
USB量产工具烧录镜像到eMMC

21.1 镜像下载

野火 LubanCat-RK 系列板卡目前仅支持 SD 卡和 eMMC 两种启动方式，下面对于两种启动介质的烧录我们分别做说明。

21.1.1 SD 卡

将野火提供的 LubanCat 镜像烧写到 SD 卡目前只有一种烧录方式，就是借助瑞芯微官方烧录工具 SDDiskTool，仅提供 Windows 版本。

SD 卡烧录不支持分区烧录，只能烧录完整的 SDK 生成的 update.img 镜像

实际使用过程中，在 Win10 系统上用 SDDiskTool 重复烧录镜像到 SD 卡时，由于已经烧录过的 SD 卡存在较多的分区，会导致电脑蓝屏，暂无解决办法。如果必须使用此工具，则应在烧录前保存好其他软件内容。

SD 卡烧录的相关内容请查看以下章节：

SD 卡启动镜像烧录

21.1.2 eMMC

将野火提供的 LubanCat 镜像烧写到 eMMC 可以通过两种方式，分别是使用 SD 升级卡烧录镜像到 eMMC 和使用 USB 烧录工具烧录镜像。

SD 升级卡的制作也需要使用 SDDiskTool 工具，不支持分区烧录，只能烧录完整的 SDK 生成的 update.img 镜像。其原理是向 SD 卡内烧录完整的启动镜像，然后再保留一份 update.img 到 SD 卡内，系统从 SD 卡内启动，然后将 update.img 镜像烧录到 eMMC 中。

使用 USB 烧录工具烧录镜像到 eMMC 也分为两种方式，一个是使用瑞芯微开发工具 RKDevTool(Windows 版) 或 Linux_Upgrade_Tool(Linux 版)，另一个是使用瑞芯微 USB 量产烧录工具 FactoryTool(仅 Windows)

RKDevTool 和 Linux_Upgrade_Tool 的功能类似，都支持分区烧录和完整 update.img 镜像烧录，这两个工具主要用于开发过程中的镜像烧录，不仅拥有烧录功能，还有其他的高级功能，但是一次

只能连接一块板卡进行操作。

FactoryTool 量产工具主要用于工厂生产使用，只支持 update.img 镜像，可以同时进行多块板卡的烧录，操作简单。

综合分析以上烧录方式，FactoryTool 量产工具适合工厂大批量烧录，RKDevTool 和 Linux_Upgrade_Tool 适合开发时使用或小批量烧录使用。SD 卡烧录方式不推荐。

eMMC 烧录的相关内容及软件使用说明请查看以下章节：

eMMC 启动镜像烧录

21.2 完整镜像拆包合并

通过对上面烧录过程的了解，我们会发现，在生产过程中我们主要使用完整镜像 update.img，而我们发布的也是 xxx_update.img 的完整镜像。对于要进行二次开发的用户来说，主要的修改内容就是根文件系统这一部分，我们在备份时也是备份根文件系统。

所以一个个单独的分区镜像如何变成完整镜像 update.img，这就涉及到了完整镜像的打包和拆包，具体内容查看以下章节：

完整镜像的解包和打包

21.3 备份与还原

部分用户想要快速添加自己的功能进行二次开发，而对镜像的构建和编译步骤又不是很熟悉。此时就可以在野火 LubanCat 镜像的基础上直接进行修改，使用修改后的镜像二次分发，就涉及到了对板卡镜像的备份和再次烧录。

根据修改的内容和操作的难易程度，可以选择以下几种方法：

21.3.1 完整备份 SD 卡或 eMMC 全部内容并烧录

可以选择直接将整个 SD 卡和 eMMC 进行备份，此方法的备份过程是最简单的，但是备份得到的 RAW 格式镜像无法通过瑞芯微量产工具进行烧录。如果是使用 SD 卡启动或者验证阶段和小批量生产阶段的 eMMC 烧录可以选择这种方式。

系统镜像备份并重新烧录

21.3.2 备份根文件系统分区

如果我们的修改内容较少，也可以选择只备份根文件系统分区或其他单独分区，得到单的的分区镜像。此时我们可以选择使用瑞芯微开发工具镜像分区烧录，也可以将分区镜像打包成一个完整镜像，后续的烧录过程就与野火提供的 LubanCat 镜像烧写过程一致了。

不过这种方式的局限性是，如果备份多个分区，每个分区都要单独备份成一个分区镜像，加上镜像的拆包和打包，整个过程就会非常冗杂。

根文件系统备份与重新烧录

21.3.3 在 PC 修改 rootfs.img 镜像

如果我们只修改根文件系统的少量内容，则可以直接将 rootfs.img 挂载到 Linux PC 上进行修改。这种方法只适合熟悉根文件系统的开发者使用，如果修改不当可能造成系统无法启动或工作异常。

如果开发者对 chroot 工具的使用也很熟悉，甚至可以直接借助主机网络安装软件包，修改系统服务等。

修改 rootfs.img 镜像内部的文件

21.4 添加系统服务

在用户使用 LubanCat 板卡的时候，需要添加或修改一些的系统服务进去，根据添加的时机，可以分为以下两种方式。

21.4.1 添加或修改系统服务到板卡

镜像已经烧录到板卡，需要在启动的板卡上添加或修改系统服务，根据自己的需求即用即改。

添加系统自启动服务

21.4.2 添加或修改系统服务到构建脚本

系统镜像还未生成，要在构建根文件系统的时候将相应的系统服务添加并设置相应的参数，这样做可以保证我们的修改是确定的，可复现的。

添加系统服务到根文件系统构建脚本

第 22 章　SD 卡启动镜像烧录

在 SD 卡的烧录过程中，我们需要使用瑞芯微 SD 卡烧录工具 SDDiskTool 和板卡对应的 RK 格式完整镜像。

由于瑞芯微 SD 卡烧录工具 SDDiskTool 目前只有 Windows 版本，所以 SD 卡的烧录过程只能在 Windows 下进行。文档中使用 Windows 10 21H1 版本，其他版本兼容性未测试。

错误： 由于烧录过后的 SD 卡有较多的分区，重新使用 SDDiskTool 烧录时会有概率导致电脑蓝屏，烧录前请先保存好未保存的应用软件。

RK 格式完整镜像是指：

- 使用 SDK 直接构建的 rockdev/update.img 镜像。
- 野火发布的以 Update.img 结尾的系统镜像。

SDDiskTool 烧录软件可在网盘资源中获取。

打开烧录 SDDiskTool 的可执行文件 SD_Firmware_Tool.exe 并插入 SD 卡。

警告： 插入多个可移动存储设备时，请确定选择的 SD 卡无误后再烧录，否则会导致其他存储设备内容被覆盖。

选择正确的要烧录的 SD 卡，然后将功能模式选择为 **SD 启动**，再选择要烧录的 RK 格式完整镜像，最后点击开始创建烧录镜像到 SD 卡。

提示： 发布的镜像以 zip 格式压缩，请使用解压后 img 格式的镜像进行烧录。

耐心等待 SD 卡烧录完整，当镜像较大时，烧录的时间会相应变长。

第 23 章　eMMC 启动镜像烧录

eMMC 烧录可以分为两种方式，分别是使用 SD 升级卡烧录镜像到 eMMC 和使用 USB 烧录工具烧录镜像。

SD 升级卡的制作也需要使用 SDDiskTool 工具，不支持分区烧录，只能烧录完整的 SDK 生成的 update.img 镜像。其原理是向 SD 卡内烧录完整的启动镜像，然后再保留一份 update.img 到 SD 卡内，系统从 SD 卡内启动，然后将 update.img 镜像烧录到 eMMC 中。

使用 USB 烧录工具烧录镜像到 eMMC 也分为两种方式，一个是使用瑞芯微开发工具 RKDevTool(Windows 版) 或 Linux_Upgrade_Tool(Linux 版)，另一个是使用瑞芯微 USB 量产烧录工具 FactoryTool(仅 Windows)

RKDevTool 和 Linux_Upgrade_Tool 的功能类似，都支持分区烧录和完整 update.img 镜像烧录，这两个工具主要用于开发过程中的镜像烧录，不仅拥有烧录功能，还有其他的高级功能，但是一次只能连接一块板卡进行操作。

FactoryTool 量产工具主要用于工厂生产使用，只支持 update.img 镜像，可以同时进行多块板卡的烧录，操作简单。

综合分析以上烧录方式，FactoryTool 量产工具适合工厂大批量烧录，RKDevTool 和 Linux_Upgrade_Tool 适合开发时使用或小批量烧录使用。SD 卡烧录方式不推荐。

提示： 强烈建议预留 USB-OTG 接口用于 eMMC 固件下载及量产烧录

23.1 SD 升级卡烧录镜像到 eMMC

注意： SD 升级卡烧录镜像到 eMMC 目前仅支持 Buildroot 镜像

在量产时，我们可以借助 SD 卡进行 eMMC 的烧录。其原理是先在 SD 卡中烧录完整的系统启动镜像，然后再存放一份 update.img 镜像到 SD 卡中，在板卡上电后，优先从 SD 卡中启动操作系统，待操作系统启动完成后，将 SD 卡中的 update.img 镜像烧录到 eMMC 中。

首先我们要创建用于烧录的 SD 卡，我们称为 SD 升级卡，也是用 SDDiskTool 工具和板卡对应的 RK 格式完整镜像进行创建。

注意： 制作 SD 升级卡要使用 SDDiskTool_v1.7 及以上版本

由于瑞芯微 SD 卡烧录工具 SDDiskTool 目前只有 Windows 版本，所以 SD 升级卡的制作过程只能在 Windows 下进行。文档中使用 Windows 10 21H1 版本，其他版本兼容性未测试。

错误： 由于烧录过后的 SD 卡有较多的分区，重新使用 SDDiskTool 烧录时会有概率导致电脑蓝屏，烧录前请先保存好未保存的应用软件。

RK 格式完整镜像是指：

- 使用 SDK 直接构建的 rockdev/update.img 镜像。
- 野火发布的以 Update.img 结尾的系统镜像。

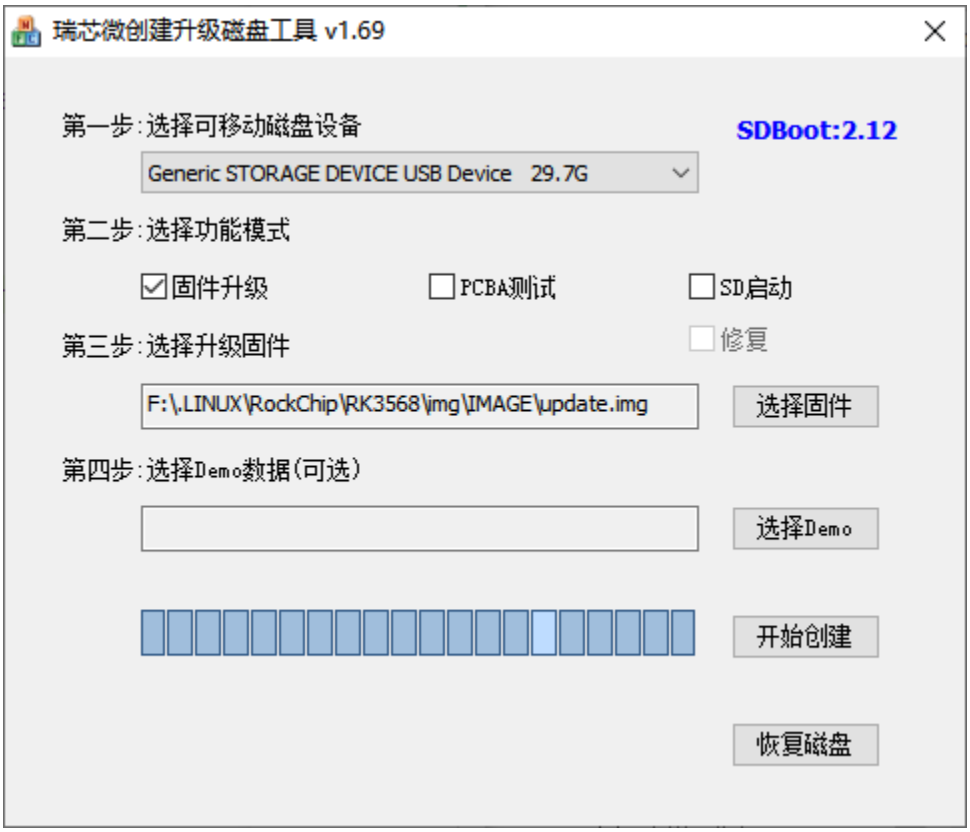
SDDiskTool 烧录软件可在网盘资源中获取。

打开烧录 SDDiskTool 的可执行文件 SD_Firmware_Tool.exe 并插入 SD 卡。

警告： 插入多个可移动存储设备时，请确定选择的 SD 卡无误后再烧录，否则会导致其他存储设备内容被覆盖。

选择正确的要烧录的 SD 卡，然后将功能模式选择为 **固件升级**，再选择要烧录的 RK 格式完整镜像，最后点击开始创建烧录镜像到 SD 卡。

提示： 发布的镜像以 zip 格式压缩，请先解压压缩包，然后使用 img 格式镜像进行烧录。



耐心等待 SD 卡烧录完成，当镜像较大时，烧录的时间会相应变长。

烧录完成后, 会新建一个 NTFS 格式的分区，我们打开该分区可以看到被重命名为 sdupdate.img 的 update.img 文件。

烧录之前，我们可以连接 Debug 串口或接入屏幕，在烧录时会有提示信息。所有准备工作做好后，在板卡上插入 SD 卡，并重新上电。

系统启动后屏幕上会出现一个安卓机器人的界面，同时会有 parameter waiting…uboot waiting.. 的文字提示，这是正在烧录对应的分区。由于 rootfs 分区较大，会长时间停留在 rootfs waiting 阶段。

在串口终端输出以下内容时，eMMC 烧录开始：

```
Starting input-event-daemon: input-event-daemon: Start parsing /etc/input-event-daemon.conf...
input-event-daemon: Adding device: /dev/input/event0...
input-event-daemon: Adding device: /dev/input/event1...
input-event-daemon: Adding device: /dev/input/event2...
input-event-daemon: Adding device: /dev/input/event3...
input-event-daemon: Adding device: /dev/input/event4...
input-event-daemon: Adding device: /dev/input/event5...
input-event-daemon: Adding device: /dev/input/event6...
input-event-daemon: Adding device: /dev/input/event7...
input-event-daemon: Start listening on 8 devices...
done
[root@RK356X:/]# [    5.049542]  mmcblk0: p1 p2 p3 p4 p5 p6 p7 p8
```

在分区烧录完成后，会进行各分区的校验，待校验完成后会提示 **update.img Installation success** 然后在屏幕和串口打印出 **Please remove SD CARD!!!, wait for reboot.**

此时拔出 SD 卡，板卡会自动重启，并使用 eMMC 中烧录的镜像启动操作系统。

23.2 瑞芯微开发工具通过 USB 烧录镜像到 eMMC

通过板卡预留的 USB-OTG 接口，我们可以很方便的将固件烧录到 eMMC 中。而使用瑞芯微开发工具，我们不仅能进行完整镜像的烧录，还可以分区烧录镜像，对于开发者来说，可以大大加快开发节奏。

注解： 由于部分板卡下载接口不支持自动切换 OTG 模式或与供电接口共用，在板卡或外壳上的丝印为 Download，其本质还是 USB-OTG 接口。为便于理解，下文中统称为下载接口。

注解： 使用 USB 下载方式，除工厂生产首次烧录外，都需通过按键进入 maskrom 模式，部分板卡按键丝印为 MASKROM，部分为 MR，以下统称为 MR 按键。

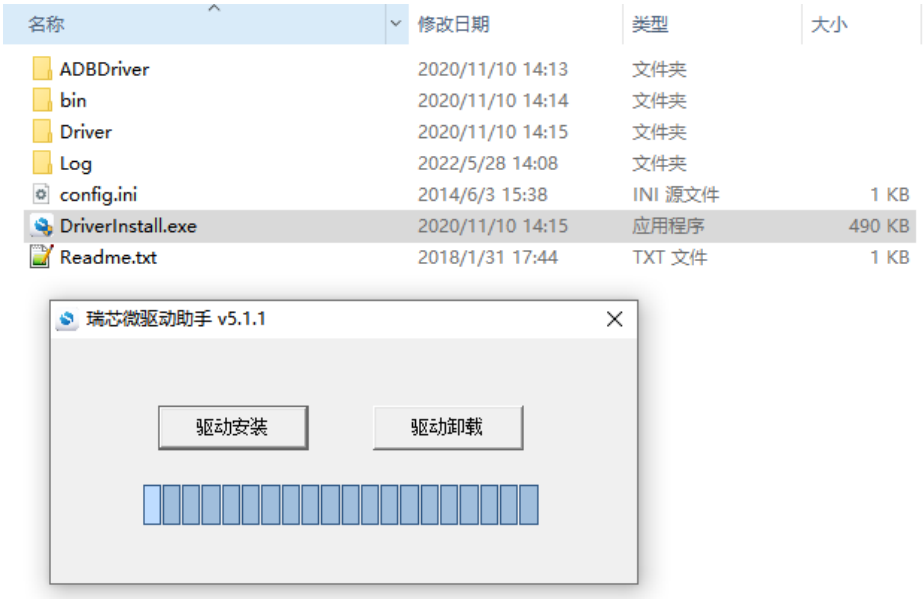
23.2.1 RKDevTool(Windows)

由于在开发过程中，编译都是在公司的编译服务器上进行的，我们本地使用 WindowsPC 通过 ssh、ftp 等方式连接到服务器。

这种本地使用 Windows 环境，远程使用 Linux 编译服务器的场景就很适合使用瑞芯微开发工具 RKDevTool，推荐使用 v2.91 及以上版本。

23.2.1.1 准备工作

- 安装 Windows 平台瑞芯微 USB 设备驱动程序。下载并解压软件压缩包，双击 DriverInstall.exe 进入驱动安装界面。



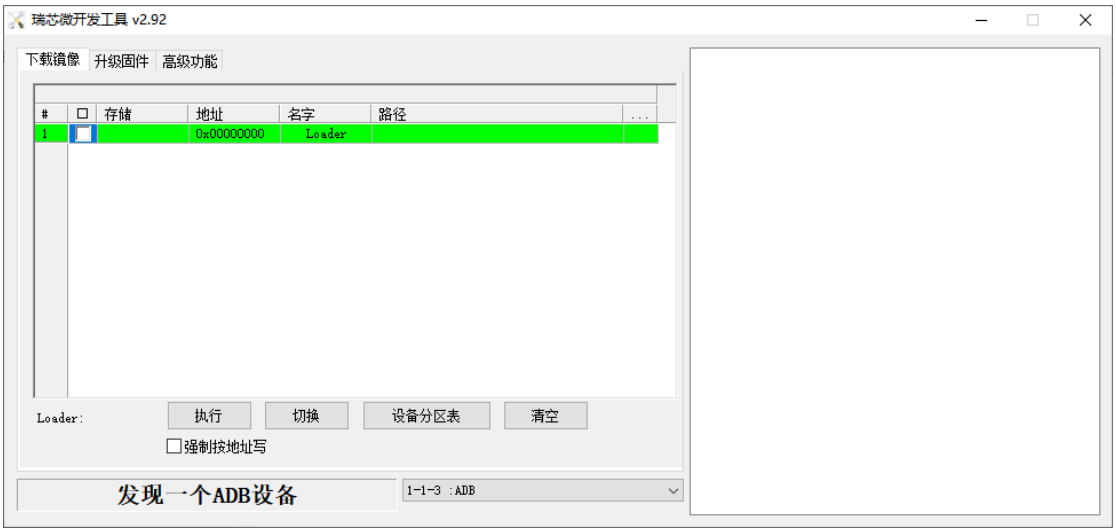
  点击 **驱动安装**即可开始安装驱动。如果不确定以前是否安装过旧版驱动，先点击驱动卸载移除旧版本驱动程序，再点击驱动安装。

- 质量较好的 USB 线。**USB-A to USB-C** 或 **双公头 USB-A** 线缆，依据下载口母座形式和实际情况决定。

- 安装 Windows 平台瑞芯微开发工具

  解压压缩包后无需安装即可使用，双击 RKDevTool.exe 进入软件界面。

  软件主要有三大部分，分别是下载镜像、升级固件和高级功能



其各个部分功能的使用将在具体使用时讲解，完整使用方法请查看工具压缩包内的：《开发工具使用文档》

- 进入烧录模式

  1. 将准备好的 USB 线缆，一端连接电脑，一端预留备用

  2. 将所有有可能给板卡供电的接线都断开，如电源线，其他 USB 线等

  3. 按住 RB 按键不松开，先将准备好的 USB 线插入板卡的下载接口，再插入电源。部分下载接口与电源接口共用的板卡无需再插入额外的电源，但需保证电脑 USB 接口能提供至少 5V 0.5A 的供电。

  4. 等待软件提示发现一个 MASKROM 设备，继续等待 5 秒以上再松开按键

  5. 如果不成功，重复 2-4 步骤。

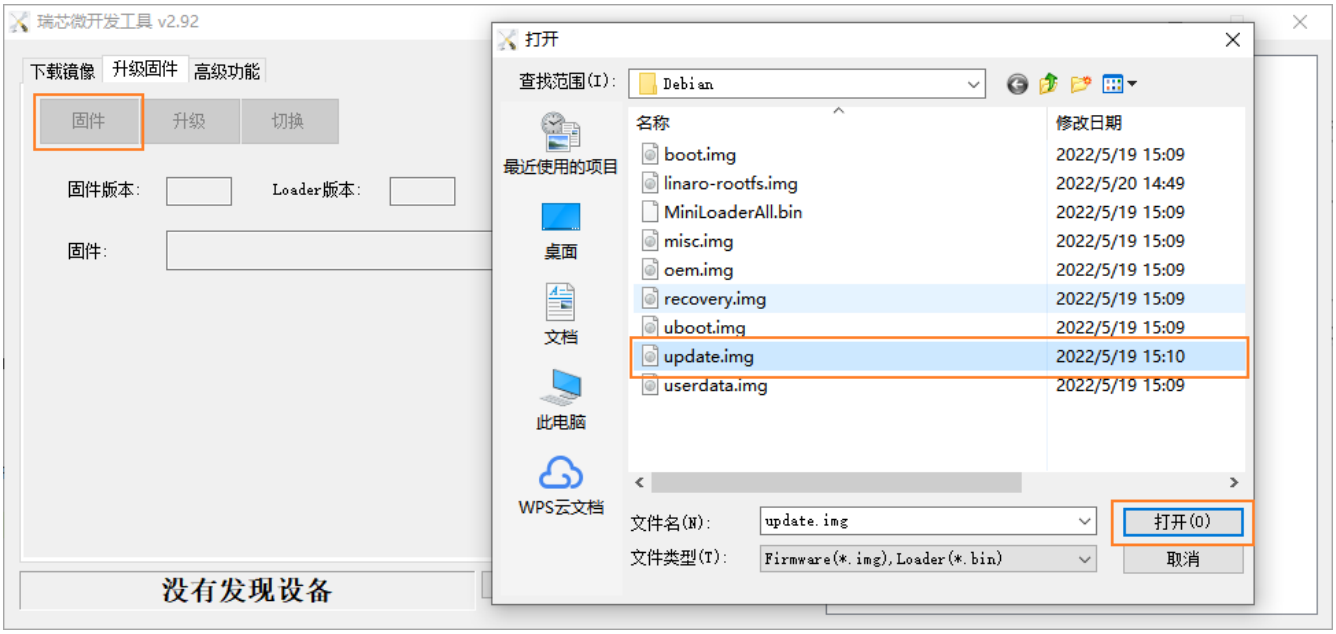
23.2.1.2 完整镜像烧录

完整镜像包含分区镜像的全部文件，在对外分发或量产时更容易管理镜像文件。其烧录过程简单，可以一键烧录。

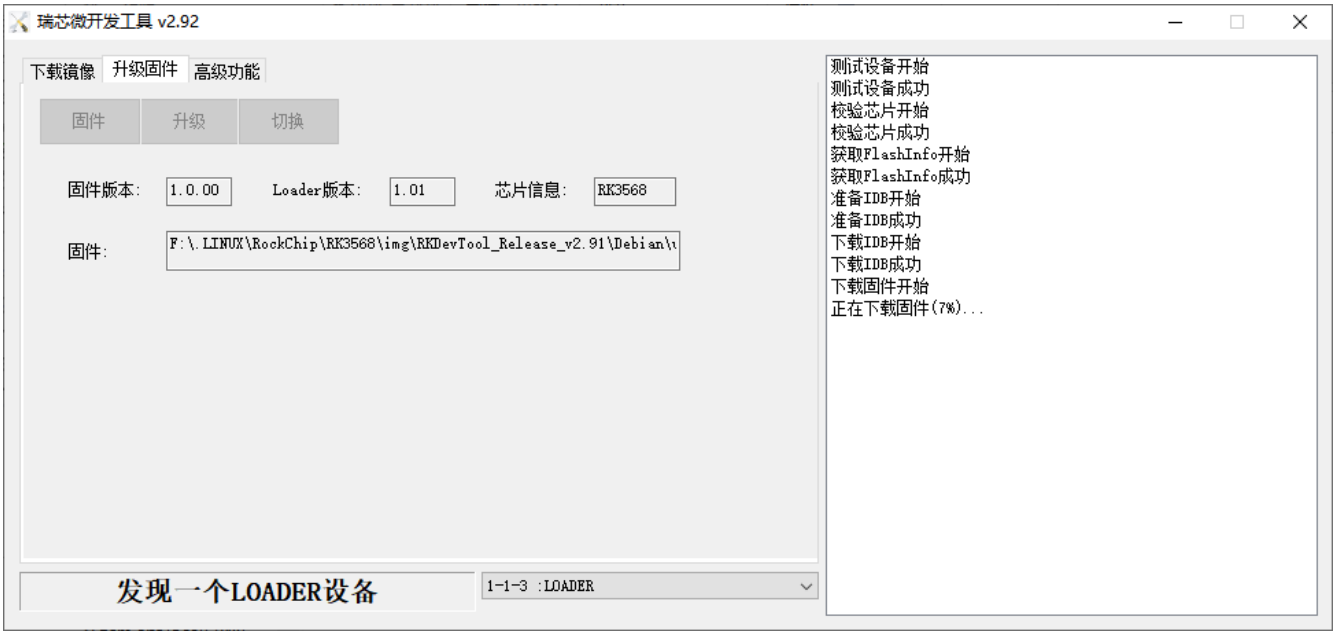
我们打开 **RKDevTool** 烧录工具，并设置板卡进入烧录模式。

将下载好的野火以 zip 压缩包格式发布的系统镜像解压为 img 格式结尾的镜像文件，也可以直接使用 SDK 编译好的 update.img 镜像。

点击 **固件**选择要烧录的镜像，这里以 update.img 为例，然后打开



等待固件加载完成，然后点击 **升级**，开始烧录固件



23.2.1.3 分区镜像烧录

在开发阶段，分区烧录镜像可以大大缩减我们在烧录过程中消耗的时间。在完整烧录过整个镜像后，如果我们对某一部分做了修改，我们就可以单独烧录这一部分，而不必重新烧录整个完整的镜像。

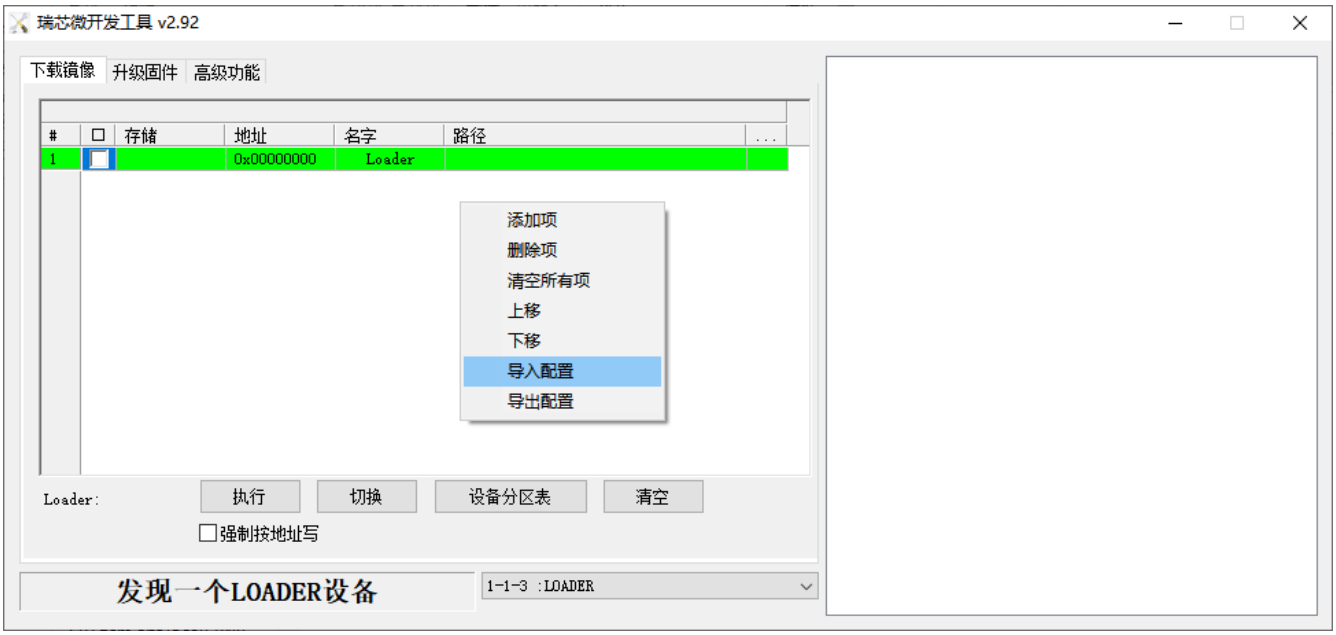
RK 固件的组合规则是基于 GPT 分区表的，这些分立的文件在 SDK 编译后，存放在 rockdev/ 路径下，这些文件如下图所示。

图中文件仅作为参考，不同版本的镜像可能不同，具体规则参考分区表。

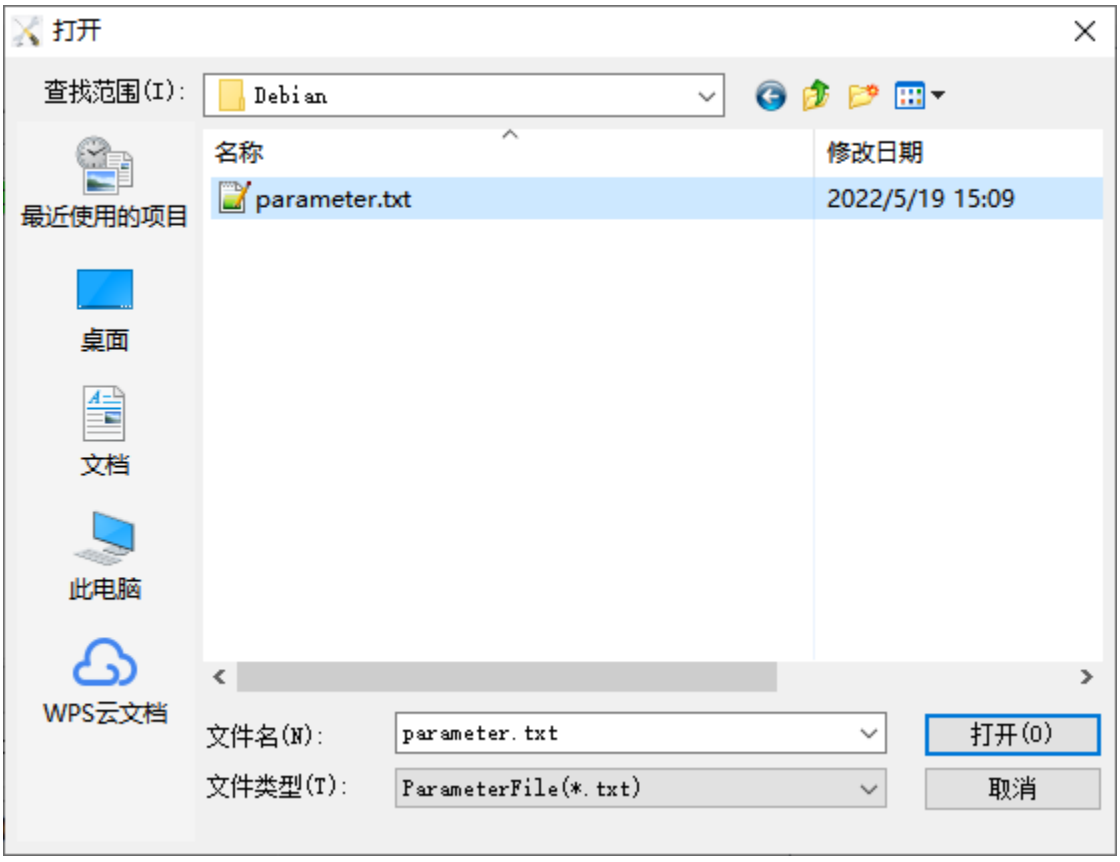
名称	修改日期	类型	大小
boot.img	2022/5/19 15:09	光盘映像文件	23,214 KB
linaro-rootfs.img	2022/5/20 14:49	光盘映像文件	3,429,024...
MiniLoaderAll.bin	2022/5/19 15:09	BIN 文件	455 KB
misc.img	2022/5/19 15:09	光盘映像文件	48 KB
oem.img	2022/5/19 15:09	光盘映像文件	17,408 KB
parameter.txt	2022/5/19 15:09	TXT 文件	1 KB
recovery.img	2022/5/19 15:09	光盘映像文件	29,951 KB
uboot.img	2022/5/19 15:09	光盘映像文件	4,096 KB
update.img	2022/5/19 15:10	光盘映像文件	3,570,083...
userdata.img	2022/5/19 15:09	光盘映像文件	5,120 KB

我们打开 **RKDevTool** 烧录工具，并设置板卡进入烧录模式。

鼠标右键点击图中空白位置，选择导入配置。

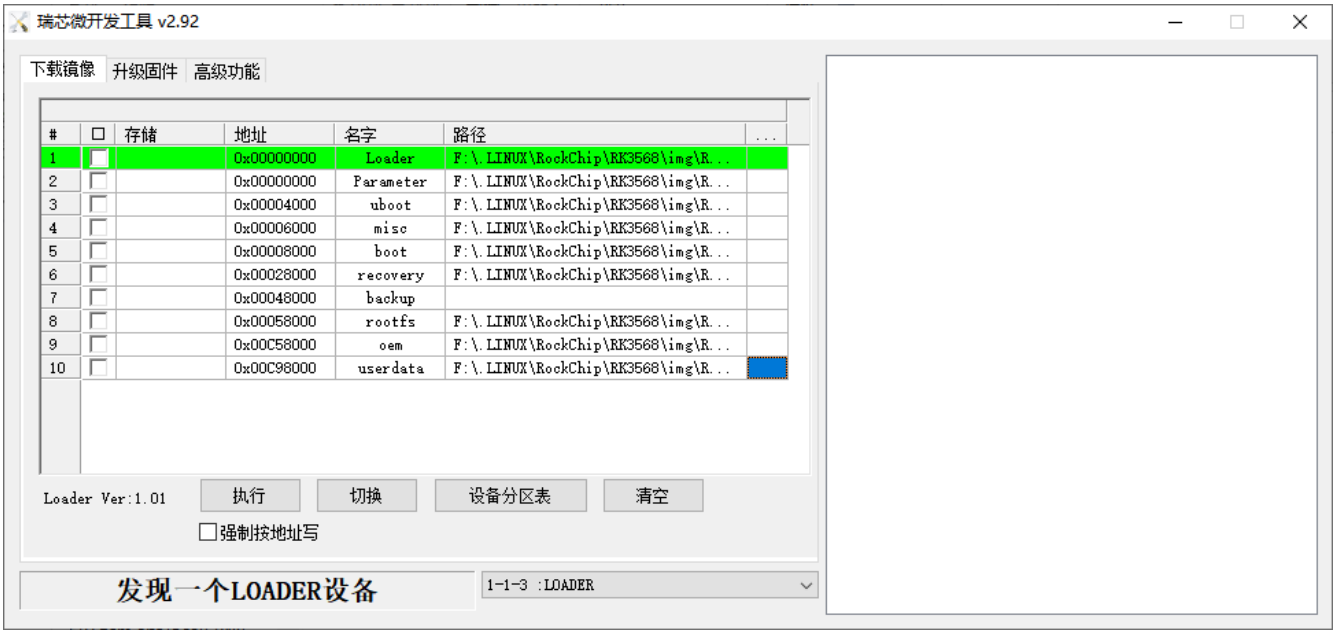


文件类型选择 *.txt，然后进入分区文件所在的目录，打开分区表文件，我们就成功的导入配置了。



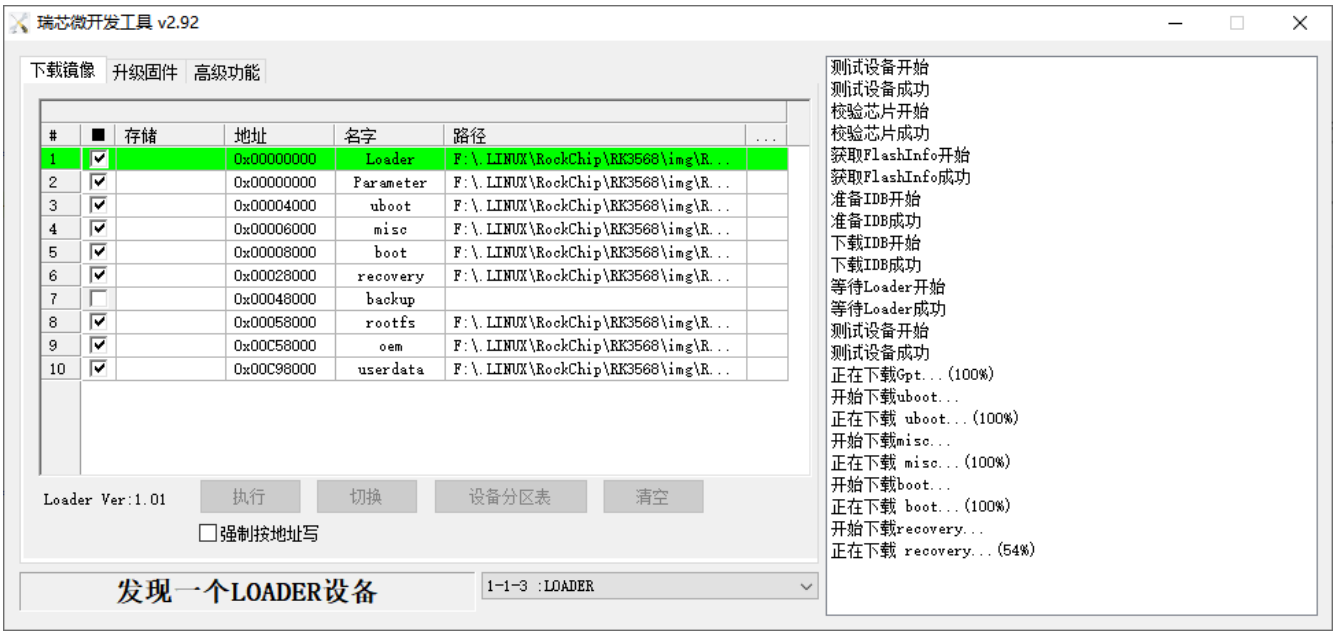
根据分区表中的配置文件，我们点击最后一栏 …选择对应的文件，对应关系如下表

注解： 部分系统分区会少于下表中所列出的项目，具体以分区表 parameter.txt 中的内容为准，不同操作系统的分区表不能混用。

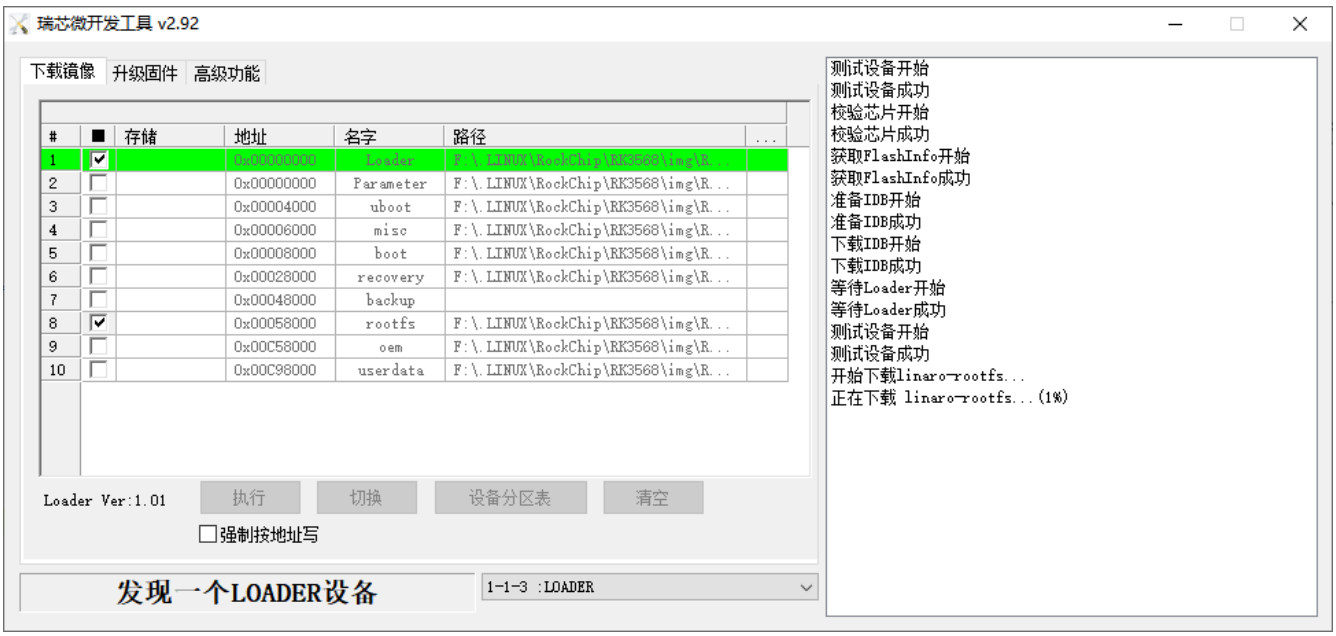


名字	对应文件名	备注
bootloader	MiniLoaderAll.bin	
parameter	parameter.txt	
uboot	uboot.img	
misc	misc.img	部分系统无此分区
boot	boot.img	
recovery	recovery.img	部分系统无此分区
backup	不添加，保留	
rootfs	rootfs.img	单独提供的文件系统名字会有变化，但都是 rootfs-xxx.img
oem	oem.img	部分系统无此分区
userdata	userdata.img	部分系统无此分区

烧录全部分区时，全选所有分区，然后把保留分区去掉，点击执行开始烧录，右侧会有进度提示



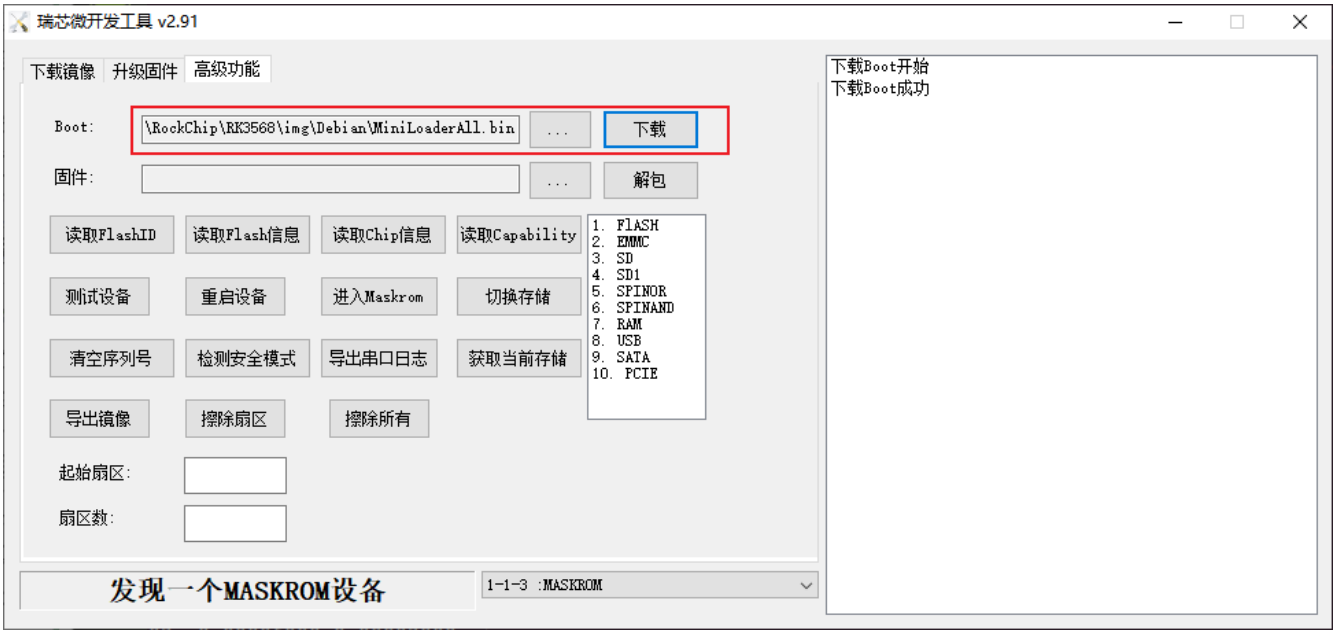
烧录单个或部分分区时，勾选要烧录的分区和 loader 分区，点击执行开始烧录，如图单独烧录 rootfs 分区



23.2.1.4 高级功能介绍

在高级功能界面主要是两部分功能，一个是通过加载 Boot 来实现与板卡的交互，另一个是对完整固件进行解包。解包部分的内容我们单独在 **完整镜像的拆包和打包**章节进行讲解。

要想使用开发工具提供的高级功能，第一步就是下载 Boot 镜像。其实下载 Boot 镜像的描述是不准确的，准确来说这一步应该是把 bootloader 加载到内存中，以实现与板卡的交互操作。

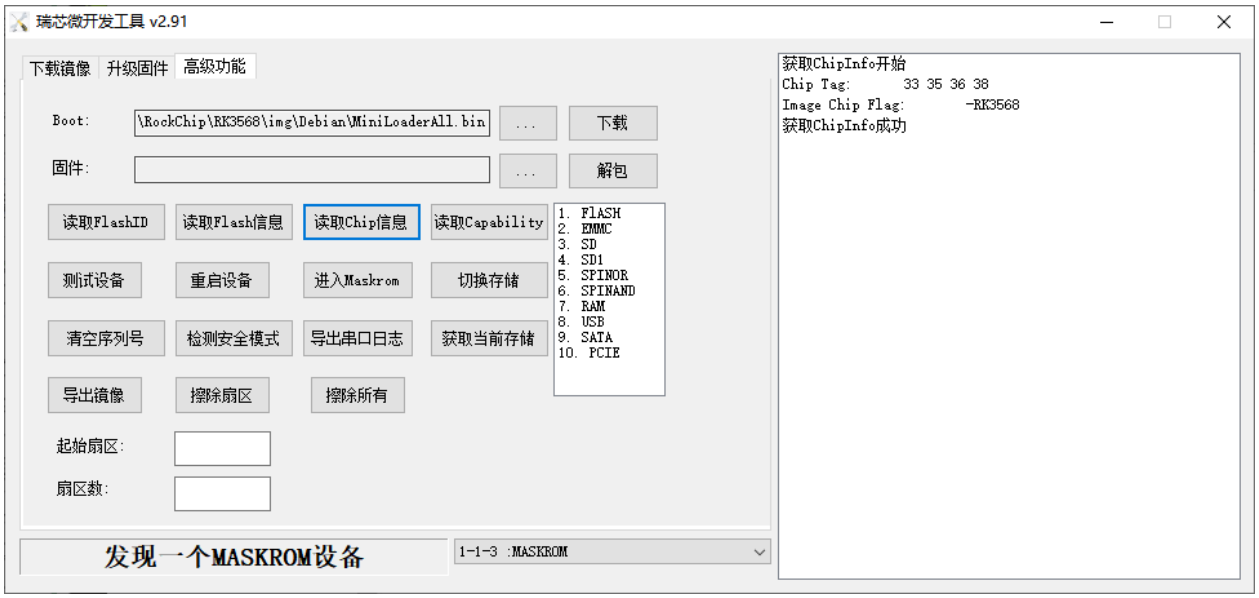


选择芯片对应的 MiniLoaderAll.bin 固件，然后下载到板卡中。

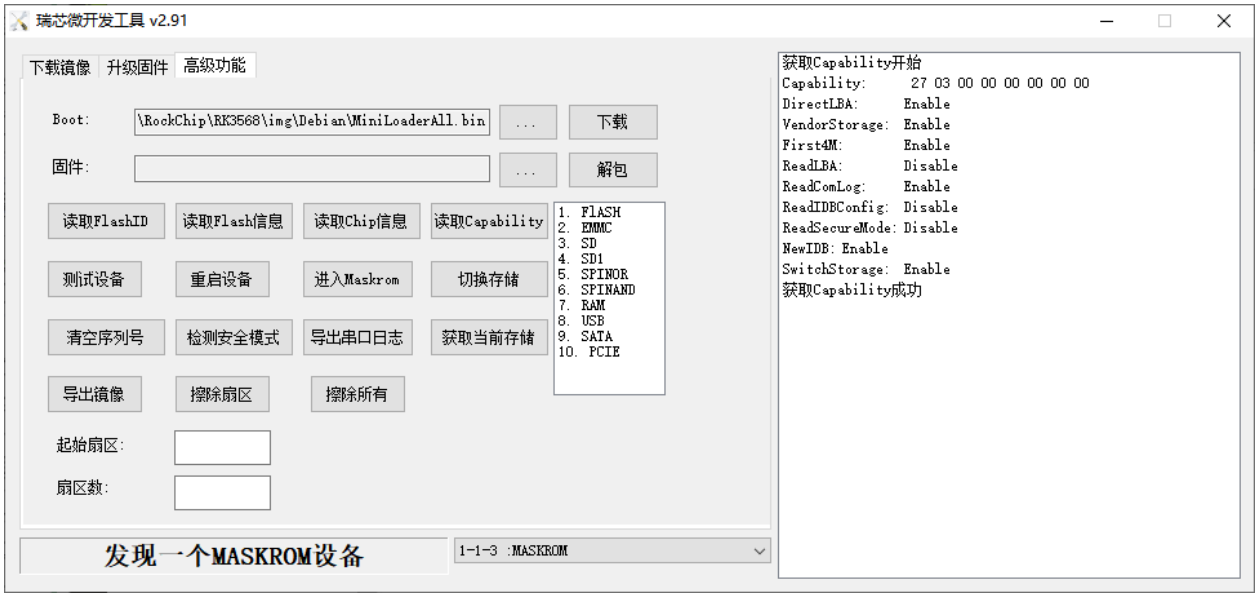
- 读取 FlashID

- 读取 Flash 信息，读取到的厂商信息是错误的，容量等信息都是正确的

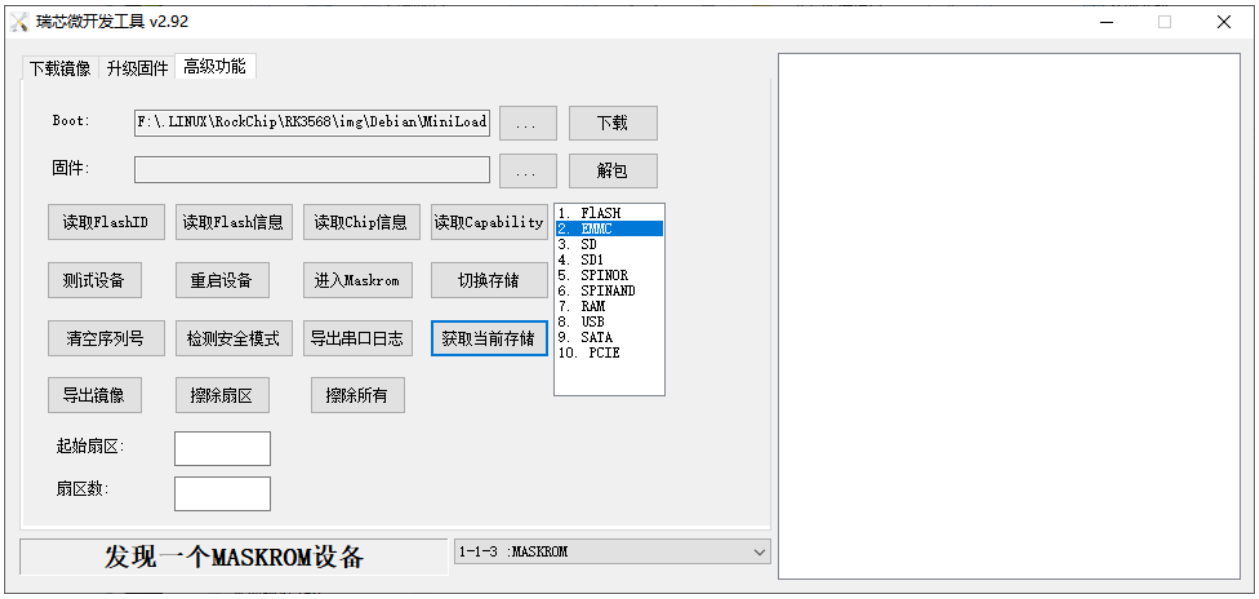
- 读取 chip 信息



- 读取 Capability



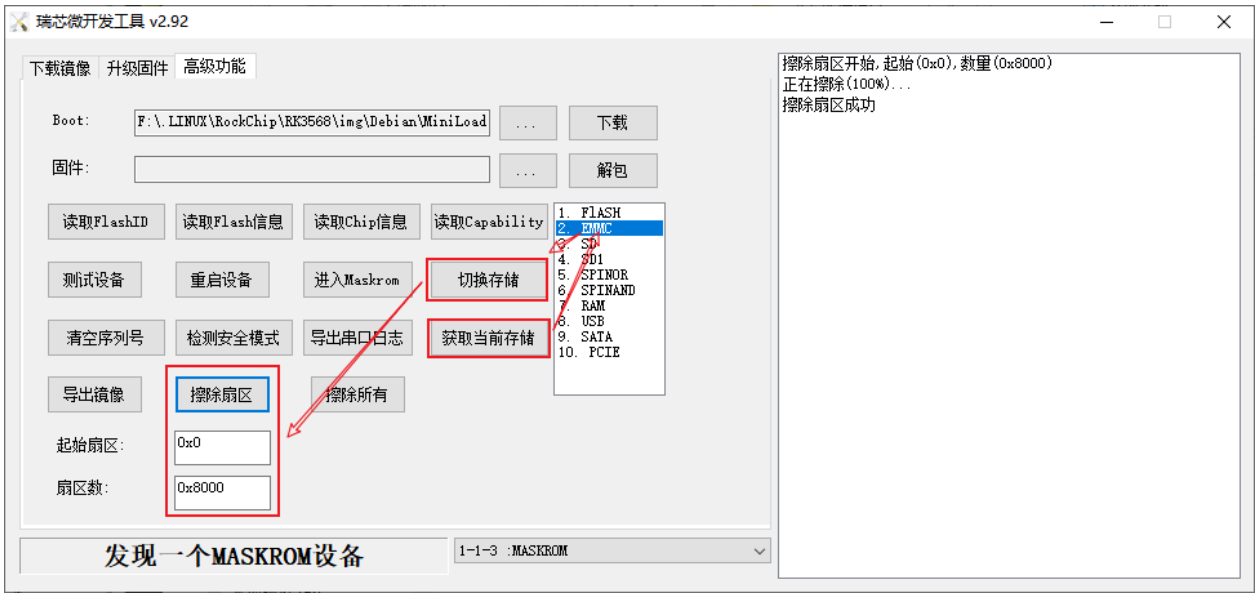
- 获取当前存储



- 擦除扇区，根据”起始扇区”和”扇区数”进行扇区擦除，只支持 eMMC。

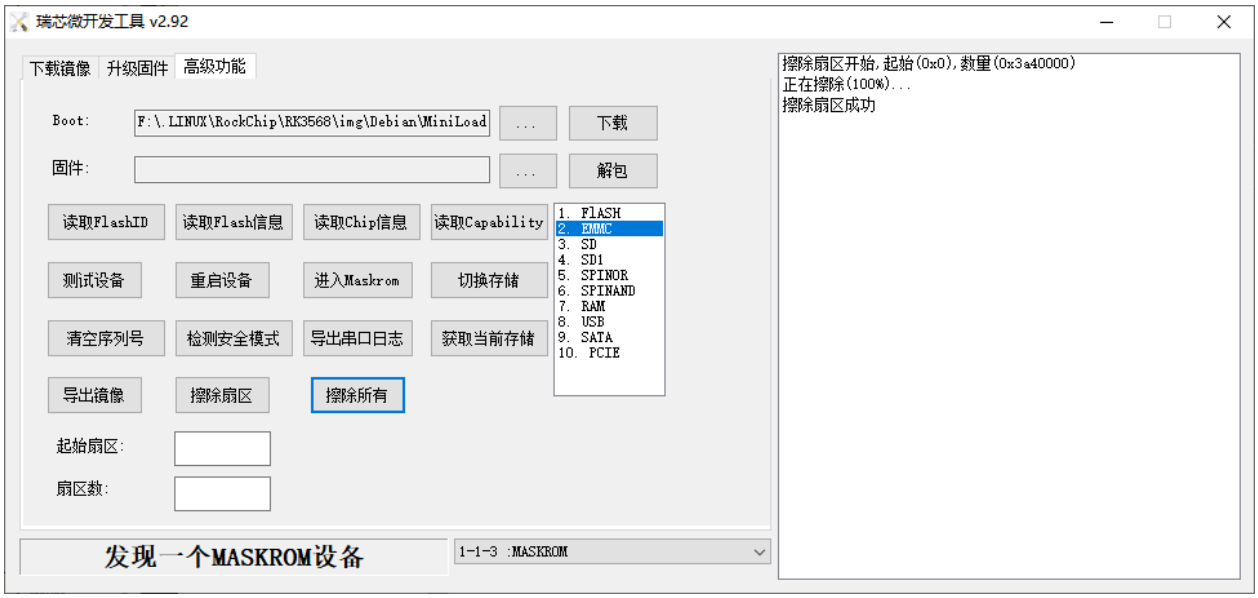
  先获取当前存储设备，选择 EMMC，然后点击切换存储，此时操作的就是 eMMC 了。

  设置起始扇区和扇区数，然后擦除扇区



- 擦除所有，即擦除对应存储设备的所有扇区，支持 eMMC/nor/nand

  先获取当前存储设备，选择 EMMC 或其他存储设备，然后点击切换存储，此时操作的就是对应的存储了。



23.2.2 Linux_Upgrade_Tool(Linux)

对于在本地 Linux 操作系统 PC 上或 Linux 虚拟机上进行开发的用户而言，Linux_Upgrade_Tool 工具用起来不需要切换操作系统环境，使用起来十分便利。

23.2.2.1 安装 Linux 开发工具

先将获取的 Linux_Upgrade_Tool 压缩包使用以下命令解压，然后将其移动到系统二进制软件包的存放路径，就可以直接使用此工具了。

```
# 解药压缩包，这里使用的是 v2.1 的版本
unzip Linux_Upgrade_Tool_v2.1.zip

# 赋予可执行权限
chmod 775 Linux_Upgrade_Tool_v2.1/upgrade_tool

# 复制到系统二级制软件包目录下
sudo cp Linux_Upgrade_Tool_v2.1/upgrade_tool /usr/local/bin/

```

(下页继续)

(续上页)

```
# 验证工具是否正常使用，返回版本号则正常
upgrade_tool -V

# 查看下载工具的使用信息
upgrade_tool -h
```

23.2.2.2 查看连接到 PC 的板卡

我们按住下载按键，然后将 USB 线接入板卡下载口，待板卡进入下载模式。

```
# 输入以下命令，查看是否有板卡连接
upgrade_tool LD

# 打印出的信息
Program Log will save in the /home/hh/upgrade_tool/log/
List of rockusb connected(1)
DevNo=1     Vid=0x2207,Pid=0x350a,LocationID=32     Mode=Maskrom    ␣
↪SerialNo=rockchip
```

根据打印出的信息我们可以得知，目前已经连接了一个设备，他的设备序号是 1，ID 是 32, 目前处于 Maskrom 模式

板卡正确连接以后，我们就可以进行固件的烧录了。

这里以 SDK 生成的镜像为例，在 SDK 的 rockdev 目录下进行以下操作

23.2.2.3 烧录完整镜像

我们在 rockdev 目录，直接烧录完整镜像到板卡，烧录前要确认板卡处于 Maskrom 模式。

```
# 烧录完整镜像
upgrade_tool uf update.img

```

(下页继续)

(续上页)

```
# 打印以下信息
Program Log will save in the /home/hh/upgrade_tool/log/
Loading firmware...
Support Type:RK3568 FW Ver:1.0.00   FW Time:2022-09-19 11:45:09
Loader ver:1.01     Loader Time:2022-09-19 11:41:11
Upgrade firmware ok.
```

烧录完成后板卡会自动重启

23.2.2.4 烧录分区镜像

我们在 rockdev 目录，将分区镜像到板卡，烧录前要确认板卡处于 Maskrom 模式。

```
# 烧录 loader 不重启设备, 不论下载哪个分区都要执行
upgrade_tool ul MiniLoaderAll.bin -noreset

# 下载分区表
    upgrade_tool di -p parameter.txt

# 下载单独分区, 一条命令下载一个分区
    upgrade_tool di -u uboot.img
    upgrade_tool di -b boot.img
    upgrade_tool di -r recovery.img
    upgrade_tool di -m misc.img
    upgrade_tool di -oem oem.img
    upgrade_tool di -userdata userdata.img
    upgrade_tool di -rootfs rootfs.img

# 连续下载多个分区
upgrade_tool di -u uboot.img  -b boot.img

# 重启
    upgrade_tool rd
```

23.2.2.5 其他软件操作

23.2.2.5.1 加载 loader 到内存

设备在 Maskrom 时，要先加载 Boot 才能进行通讯

```
upgrade_tool db MiniLoaderAll.bin
```

23.2.2.5.2 读取设备信息

```
# 读取存储信息
upgrade_tool rfi

# 读取芯片 id
upgrade_tool rci

# 读取分区表
upgrade_tool pl
```

23.2.2.5.3 按地址读写文件

```
# 按地址写文件到 LBA 0x12000
upgrade_tool wl 0x12000 oem.img

# 按地址读数据保存到文件，例子：从 0x12000 开始读取 0x2000 扇区数据到 out.img
upgrade_tool rl 0x12000 0x2000 out.img
```

23.2.2.5.4 设备擦除

```
# 擦除存储的所有数据，请进入 maskrom 执行，不需要提前下载 Boot
upgrade_tool ef rkxxloader.bin

# 按地址进行扇区擦除，只支持 emmc 存储，例子：从第 0 扇区开始擦除 0x2000 个扇区
upgrade_tool el 0 0x2000
```

23.3 USB 量产工具烧录镜像到 eMMC

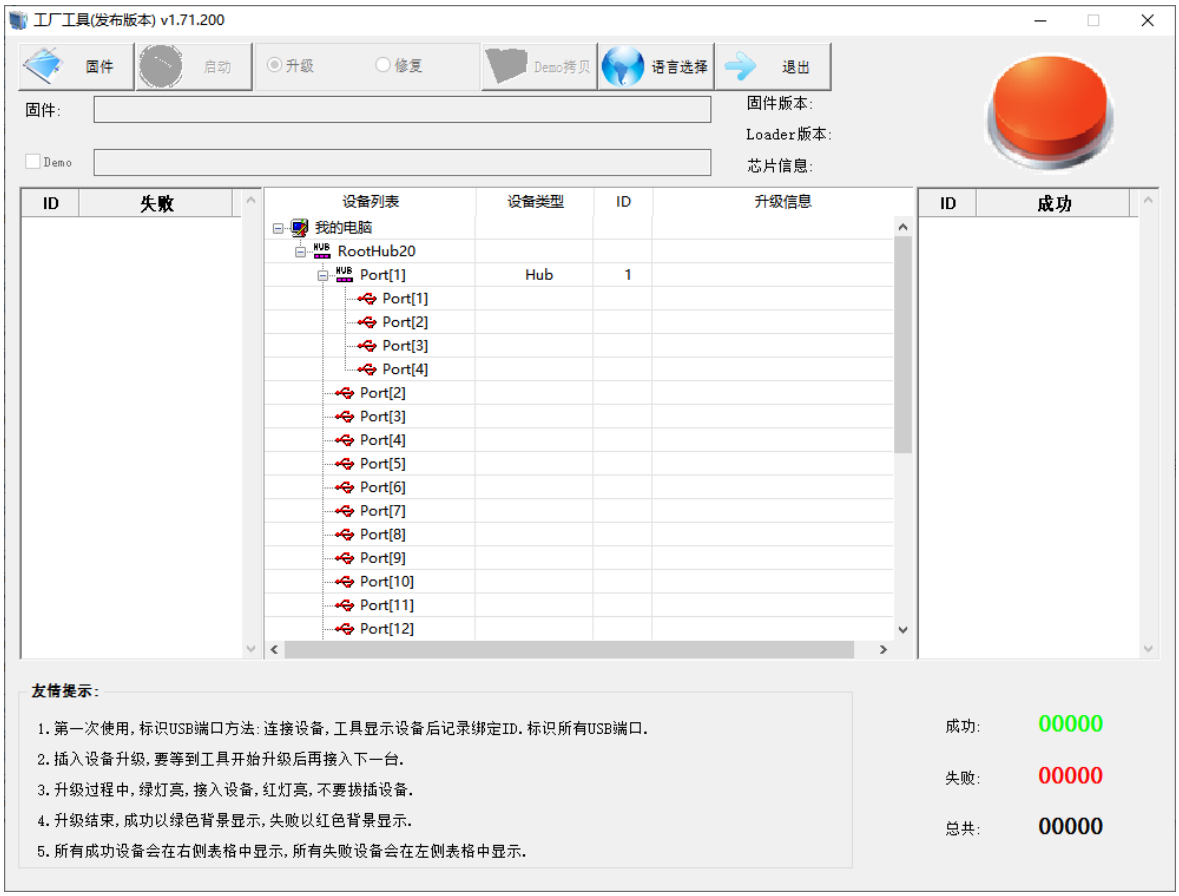
USB 量产工具 FactoryTool 仅支持 Windows 平台，使用 USB 量产工具烧录的准备工作与 RKDevTool 的准备工作一致，都需要先安装 DriverInstall 驱动，然后准备对应的接口的 USB 线缆。

FactoryTool 可以同时对多块板卡进行下载，所以我们可以准备多条下载线缆，对于部分 USB 接口较少的电脑，也可以使用 USB-HUB 来扩展 USB 口的数量。

下面我们来讲解 FactoryTool 的使用。

联系技术支持并获取 FactoryTool 软件包的压缩包，然后解压到本地，无需安装即可使用。

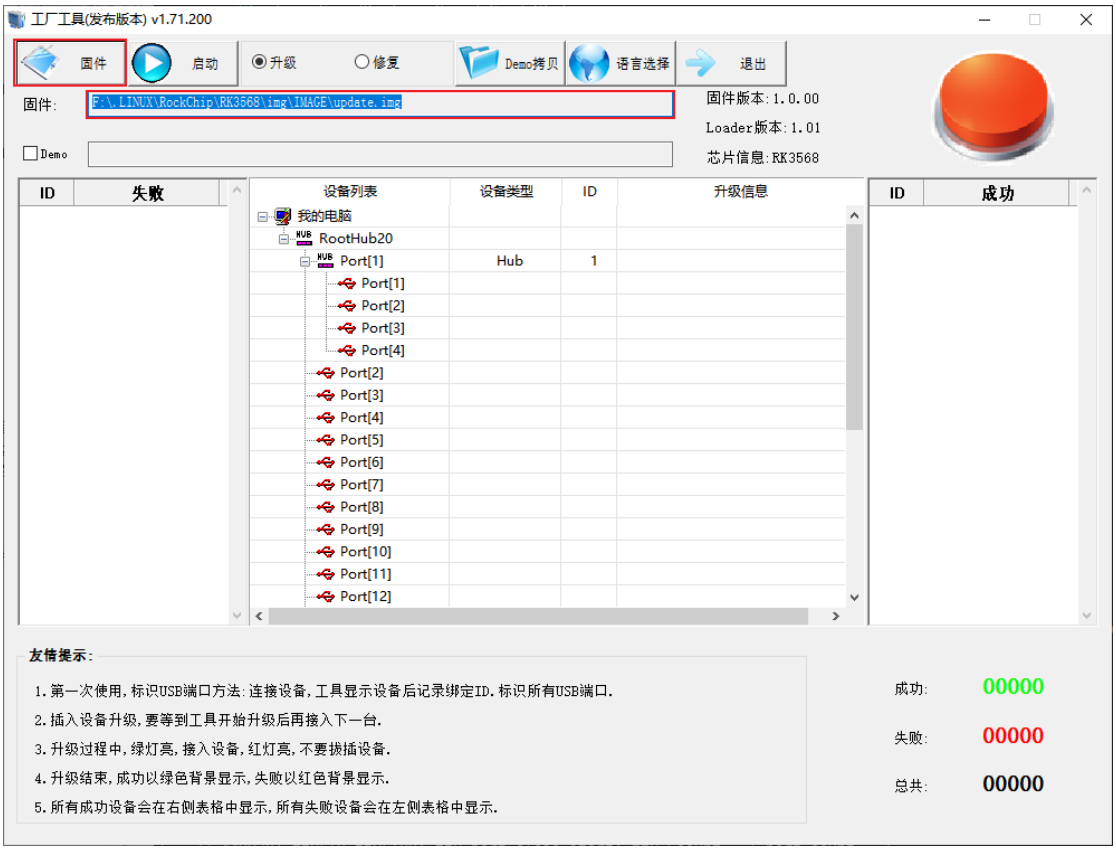
双击 FactoryTool.exe 进入软件界面。



整个软件界面整洁高效，分为三大区域。上部是配置区，用于选择烧录的固件和启动烧录功能；中间是功能区，中间显示 USB 接口的信息及烧写进度，左右两侧分别显示烧录成功和烧录失败的设备；下部是提示区，简单明了的标明了软件的使用方法，还有一小部分区域用于记录烧写次数及失败成功的次数。

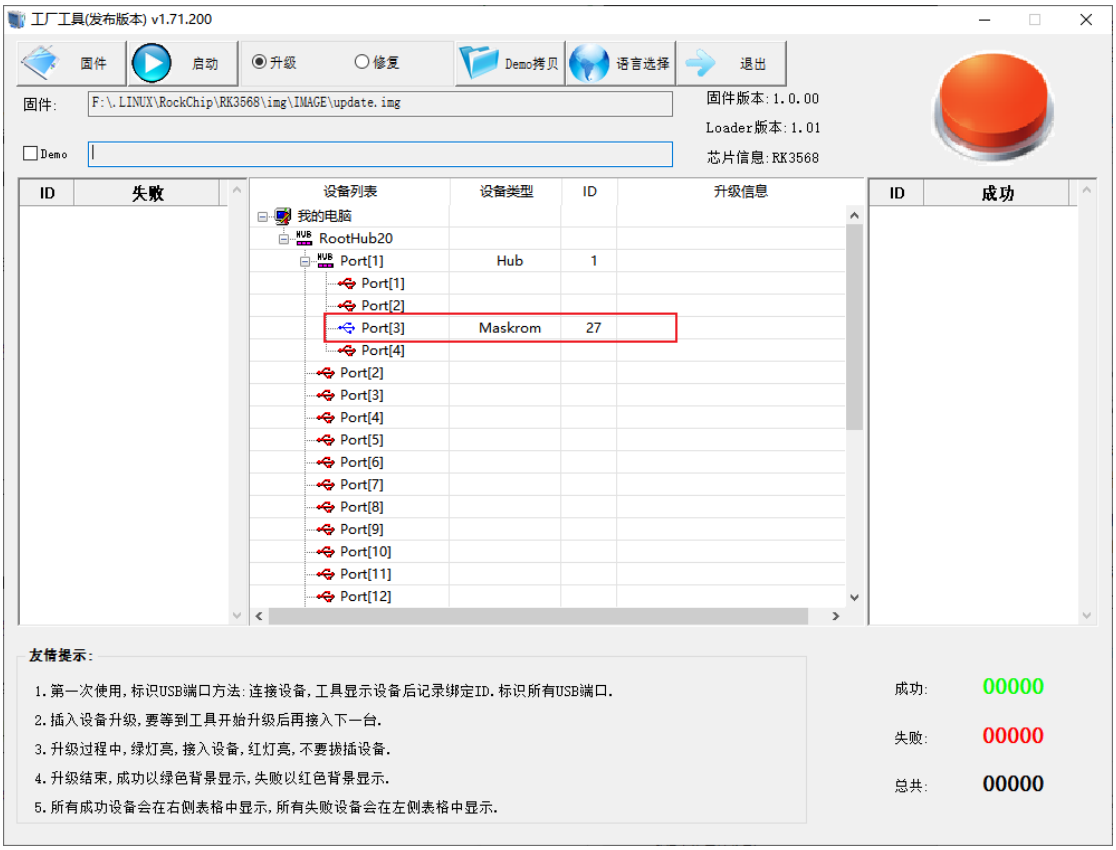
介绍完了界面，我们开始实际的操作。

- 点击固件按钮，选择要烧录的镜像，必须是 RK 格式打包的完整镜像，这里以 update.img 为例。



- 标记设备列表里的端口与实际的下载线的对应关系，便于分辨哪块板卡下载完成，哪块下载失败。

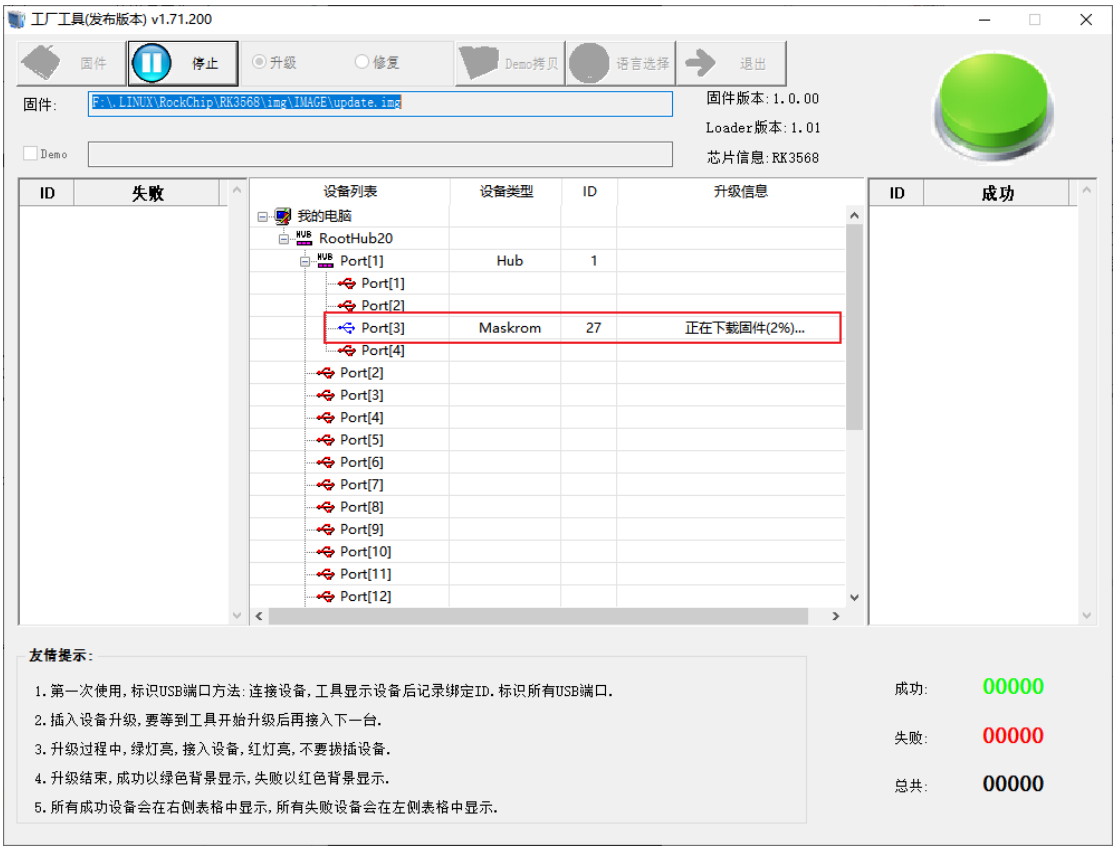
  具体的方法是，连接板卡到第一个 USB 端口，使板卡进入 MASKROM 模式，软件列表中会显示相应的 ID，我们将 ID 号标记到对应的接口或下载线上，然后更换端口，重复上述过程。



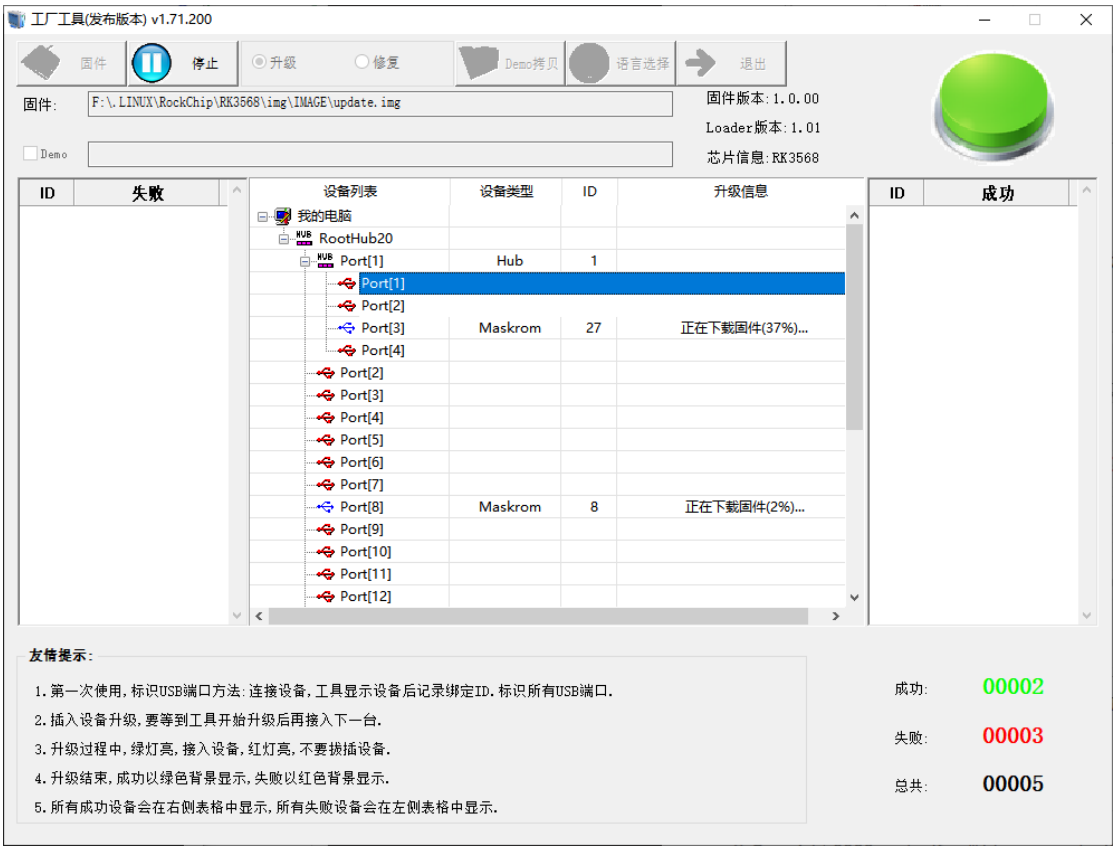
从上图可以看出，我们将板卡接在了 USB-HUB1 的 Port3 上，ID 是 27。

- 点击 **启动**按钮，开始进入烧录状态。

  烧录工具进入烧录状态后，如果连接了进入 Maskrom 模式的板卡，将会自动下载。



- 我们接入多块板卡，将会同时进行多块板卡的下载。



注解：在不插入 SD 启动卡且 eMMC 中无启动镜像是，不用按下载键就可以进入 Maskrom 模式

第 24 章　完整镜像的解包和打包

为了编译发布和镜像管理，我们提供了完整打包好的系统镜像，完整镜像按照一定的规则将系统所需的各个分区打包在了一起。

由于在后面的备份及烧录过程中涉及了对单独分区的操作，而部分用户又不想使用 SDK 来自行构建分区。为了便于用户对系统镜像进行二次修改和分发，这里我们讲解不借助 SDK 对系统镜像镜像单独解包和打包。

注意：通用镜像必须在 Linux 下，使用 Linux_Pack_Firmware 工具进行打包、解包，否则打包出来的镜像无法正常使用。**专用镜像**在 windows 和 linux 下都可以正常打包、解包。

24.1 RKDevTool 解包和打包 (Windows)

24.1.1 解包

我们需要借助 RKDevTool 来将完整的 update.img 镜像解包成分区镜像的形式。

RKDevTool 的安装请参考 eMMC 烧录章节。

首先将从网盘获取的 Update.zip 镜像解压成.img 格式，也可以直接使用 SDK 生成的 update.img 镜像。

打开 RKDevTool，进入高级功能选项页，找到解包功能，选择要解包的镜像，点击解包按钮，等待解包完成。

解包后的文件保存在 RKDevTool 的 Output 目录中。

Output 目录中有两个文件一个文件夹

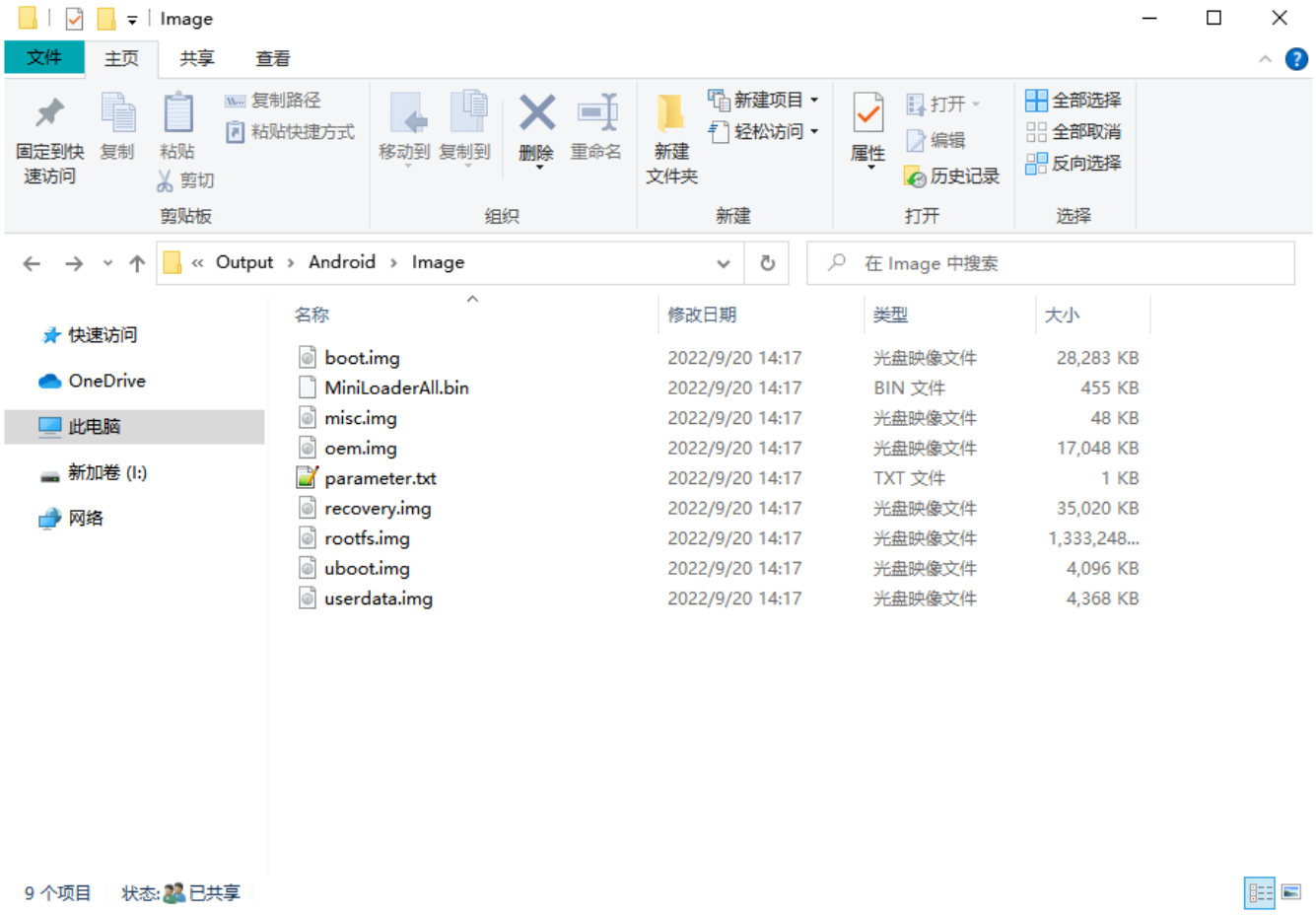
- boot.bin：打包时的 loader, 也就是 MiniLoaderAll.bin 文件
- firmware.img：打包时 afptool 生成的固件
- Android 目录下为 firmware.img 的展开内容

进入 Android 目录，有一个文件一个文件夹

- package-file：分区与分区镜像名的对应关系
- Image：firmware.img 展开后的内容，也就是解包后的分区镜像。

进入 Image 目录，根据不同操作系统，有以下三类文件

- 分区表：parameter.txt
- loader 文件：MiniLoaderAll.bin
- 分区镜像：boot.img、misc.img 等以 img 结尾的分区镜像文件

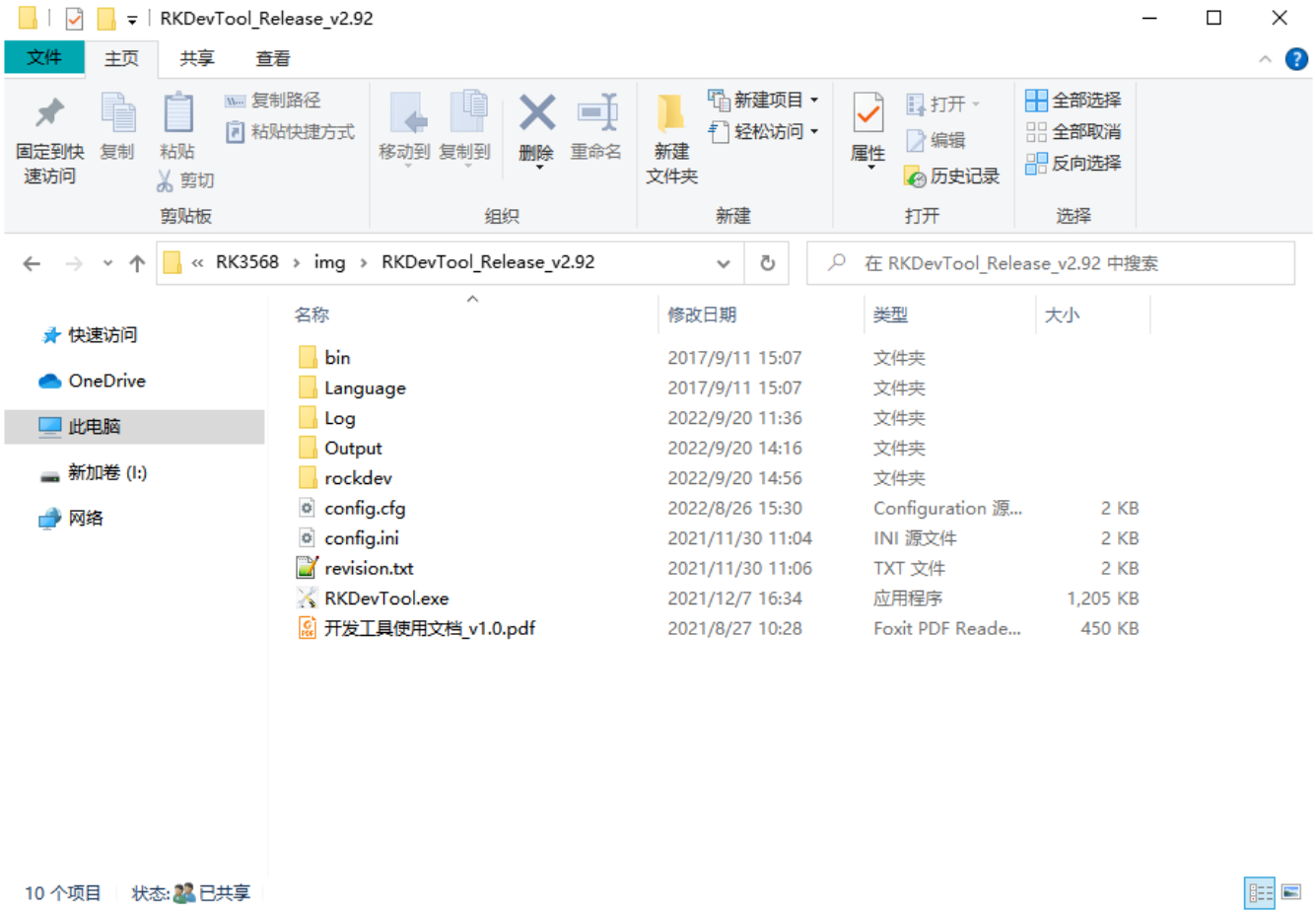


24.1.2 打包

一般是在修改或备份了根文件系统之后，我们需要将修改后的分区镜像重新打包成一个完整的固件。

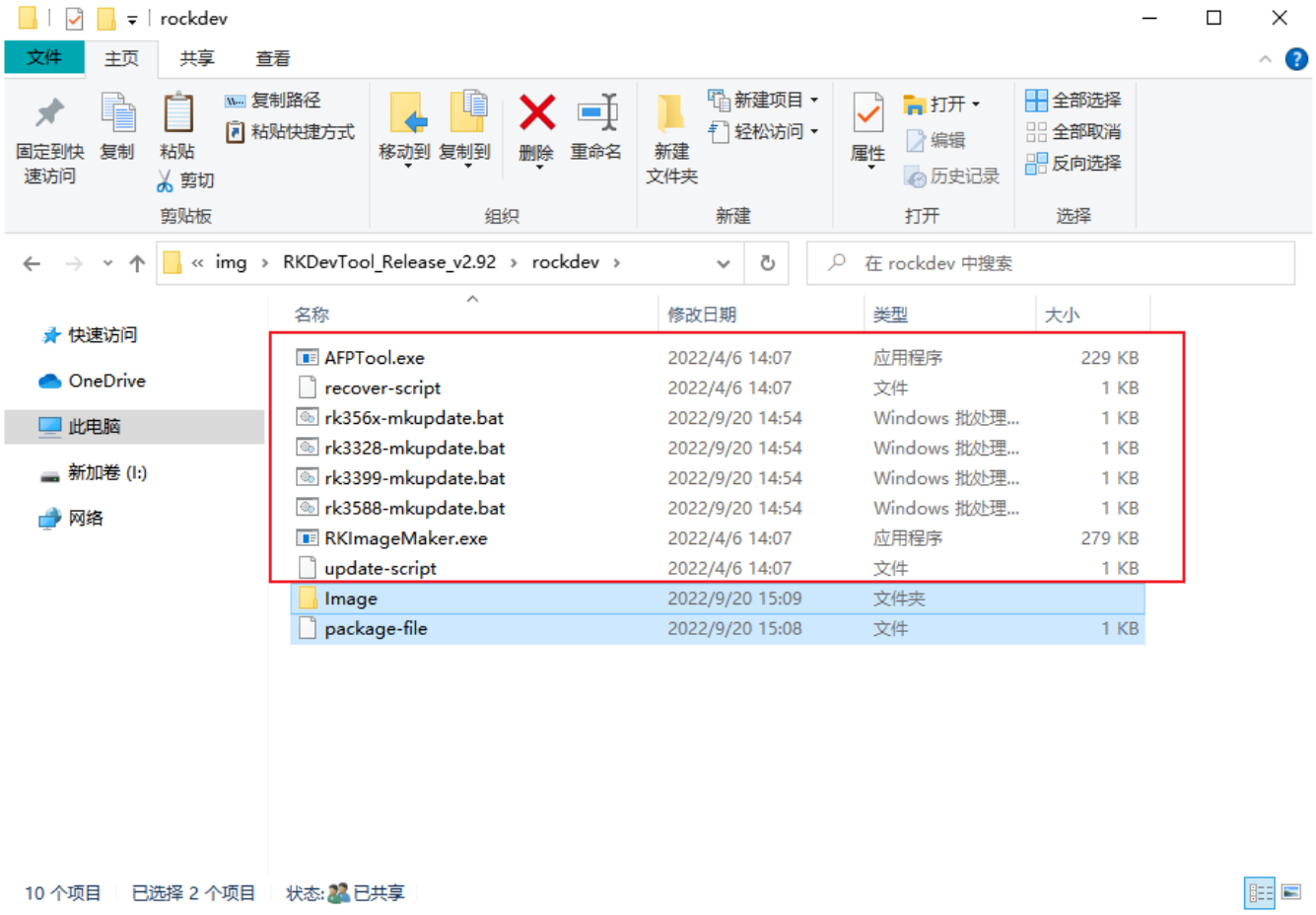
Windows 环境下打包工具存放在 SDK/tools/windows/RKDevTool/rockdev/目录下

由于以上打包工具都是针对 SDK 设计的，我们对打包工具做了部分修改，使其可以不依托于 SDK 使用。我们将修改后的打包工具存放在了 RKDevTool_Release_v2.92 软件压缩包的 rockdev 目录下，可以搭配 RKDevTool 的解包功能使用。



在上面的解包过程中，我们得到了 Output 文件夹的内容。

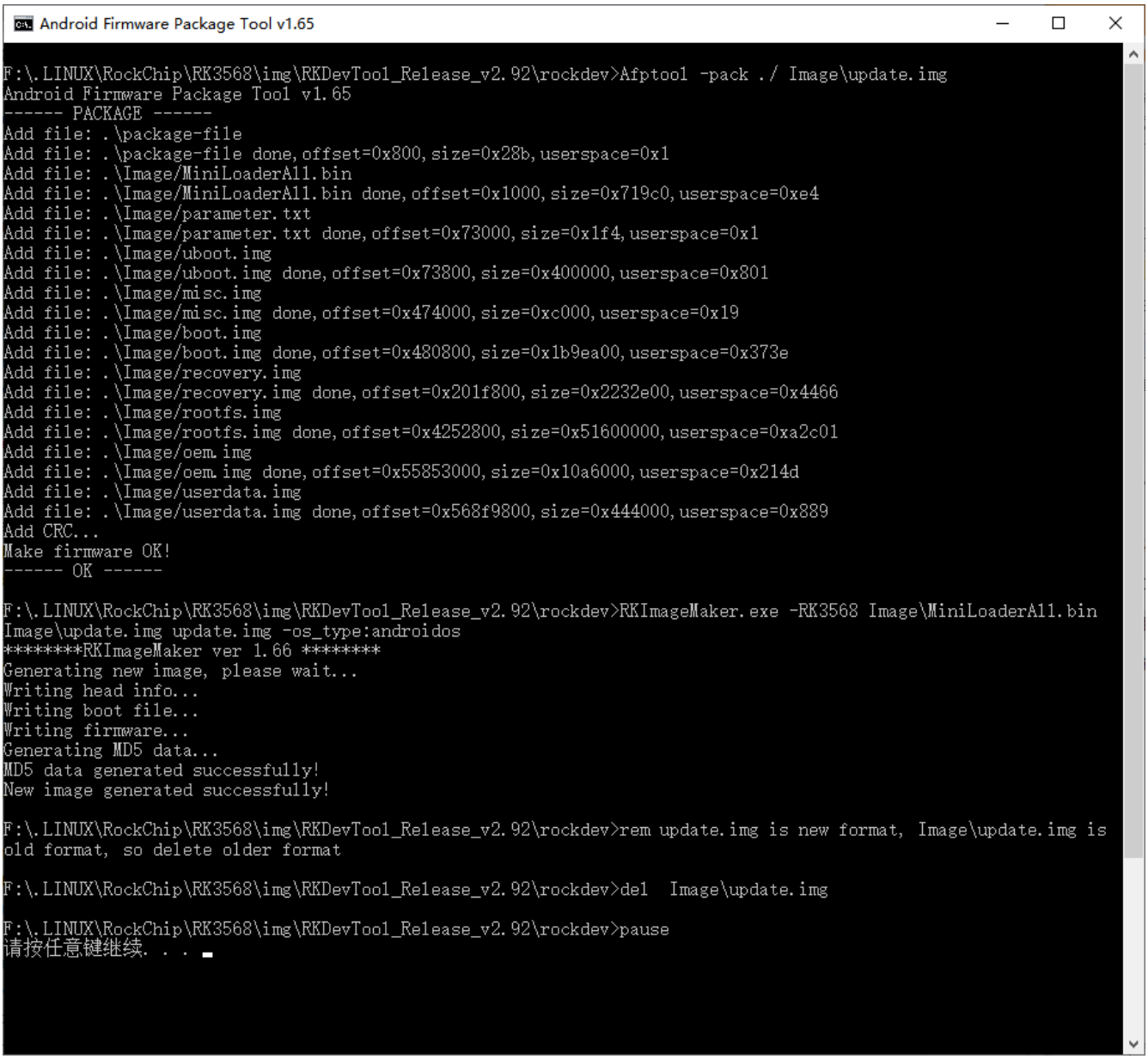
首先我们将 Output\Android 目录下的内容复制到 rockdev 目录下



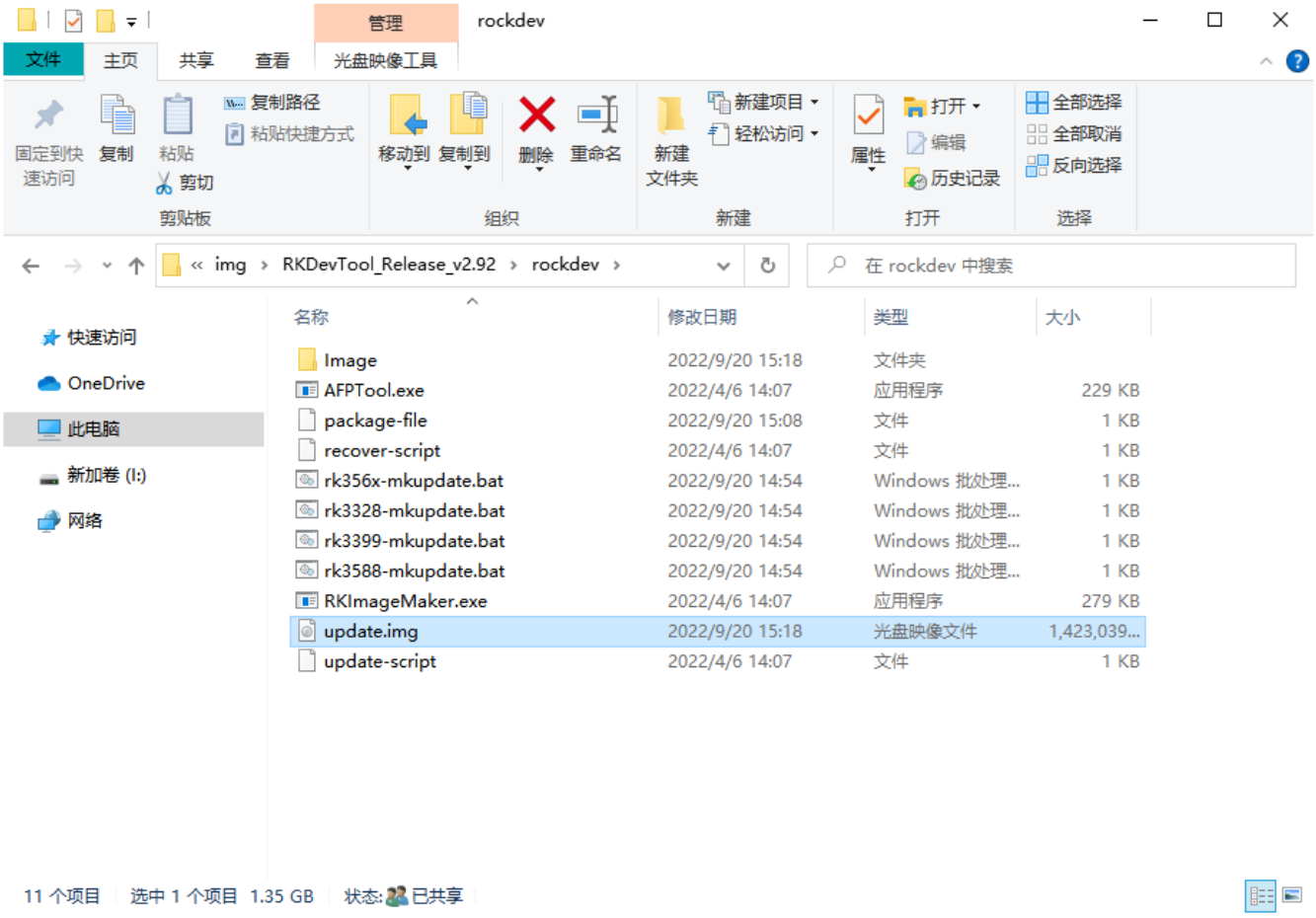
红框中的内容是我们的打包工具及辅助打包的可执行文件，Image 目录下存放了我们的分区镜像，package-file 是分区与分区镜像文件的对应关系。

如果我们对某一分区镜像文件做出了改变，在打包时要注意镜像文件名正确，与 package-file 文件中的命名及路径一致。

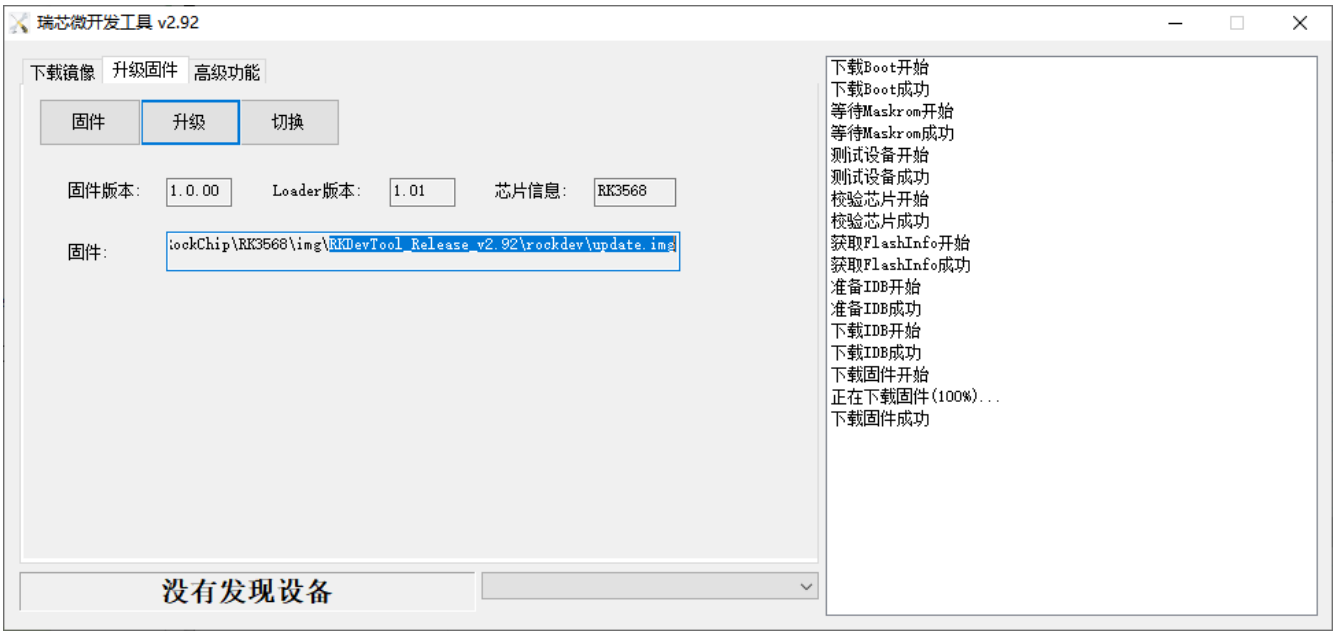
双击板卡主芯片对应的 bat 文件，开始进行打包。LubanCat-1、2、Zero 都使用了 rk356x 主芯片，这里我们双击 rk356x-mkupdate.bat



打包完成后，当前文件夹内出现了 update.img 文件，这就是打包后的完整镜像文件



我们使用烧录工具，将打包好的镜像烧录到板卡中，可以正常启动。


瑞芯微开发工具 v2.92
下载镜像
升级固件
高级功能
固件
升级
切换
固件版本:
1.0.00
Loader版本:
1.01
芯片信息:
RK3568
固件:
iockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev\update.img
没有发现设备
下载Boot开始
下载Boot成功
等待Maskrom开始
等待Maskrom成功
测试设备开始
测试设备成功
校验芯片开始
校验芯片成功
获取FlashInfo开始
获取FlashInfo成功
准备IDB开始
准备IDB成功
下载IDB开始
下载IDB成功
下载固件开始
正在下载固件(100%)...
下载固件成功

```
U-Boot 2017.09-gf403aeeed3-220621 #jiawen (Sep 16 2022 - 15:40:12 +0800)

ADC Channel:2 reading_value: 796, CH_ID: 0x05
ADC Channel:3 reading_value: 186, CH_ID: 0x00
Board Version: Board_ID 0x0500
Board: LubanCat2N
PreSerial: 2, raw, 0xfe660000
DRAM:  4 GiB
Sysmem: init
Relocation Offset: ed33a000
Relocation fdt: eb9f9080 - eb9fecd8
CR: M/C/I
Using default environment

dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
DM: v1
boot mode: recovery (misc)
FIT: no signed, no conf required
** Unrecognized filesystem type **
DTB: rk-kernel.dtb
HASH(c): OK
I2c0 speed: 100000Hz
vsel-gpios- not found! Error: -2
vdd_cpu 1025000 uV
PMIC:  RK8090 (on=0x40, off=0x00)
vdd_logic init 900000 uV
vdd_gpu init 900000 uV
vdd_npu init 900000 uV
io-domain: OK
Could not find baseparameter partition
Board Version: Board_ID 0x0500
Board: LubanCat2N
Rockchip UBOOT DRM driver version: v1.0.1
VOP have 1 active VP
vp0 have layer nr:6[0 2 4 1 3 5 ], primary plane: 4
vp1 have layer nr:0[], primary plane: 0
vp2 have layer nr:0[], primary plane: 0
xfer: num: 2, addr: 0x50
xfer: num: 2, addr: 0x50
Monitor has basic audio support
can't find to match
Could not find baseparameter partition
mode:1920x1080
hdmi@fe0a0000:  detailed mode clock 148500 kHz, flags[5]
    H: 1920 2008 2052 2200
    V: 1080 1084 1089 1125
bus_format: 2025
VOP update mode to: 1920x1080p0, type: HDMI0 for VP0
rockchip_vop2_init: Failed to get hdmi0_phy_pll ret=-22
rockchip_vop2_init: Failed to get hdmi1_phy_pll ret=-22
VOP VP0 enable Smart0[772x525->772x525@574x277] fmt[1] addr[0xee029000]
CEA mode used vic=17
final pixclk = 148000000 tmdsclk = 148000000
PHY powered down in 0 iterations
```

24.2 Linux_Pack_Firmware 解包和打包 (Linux)

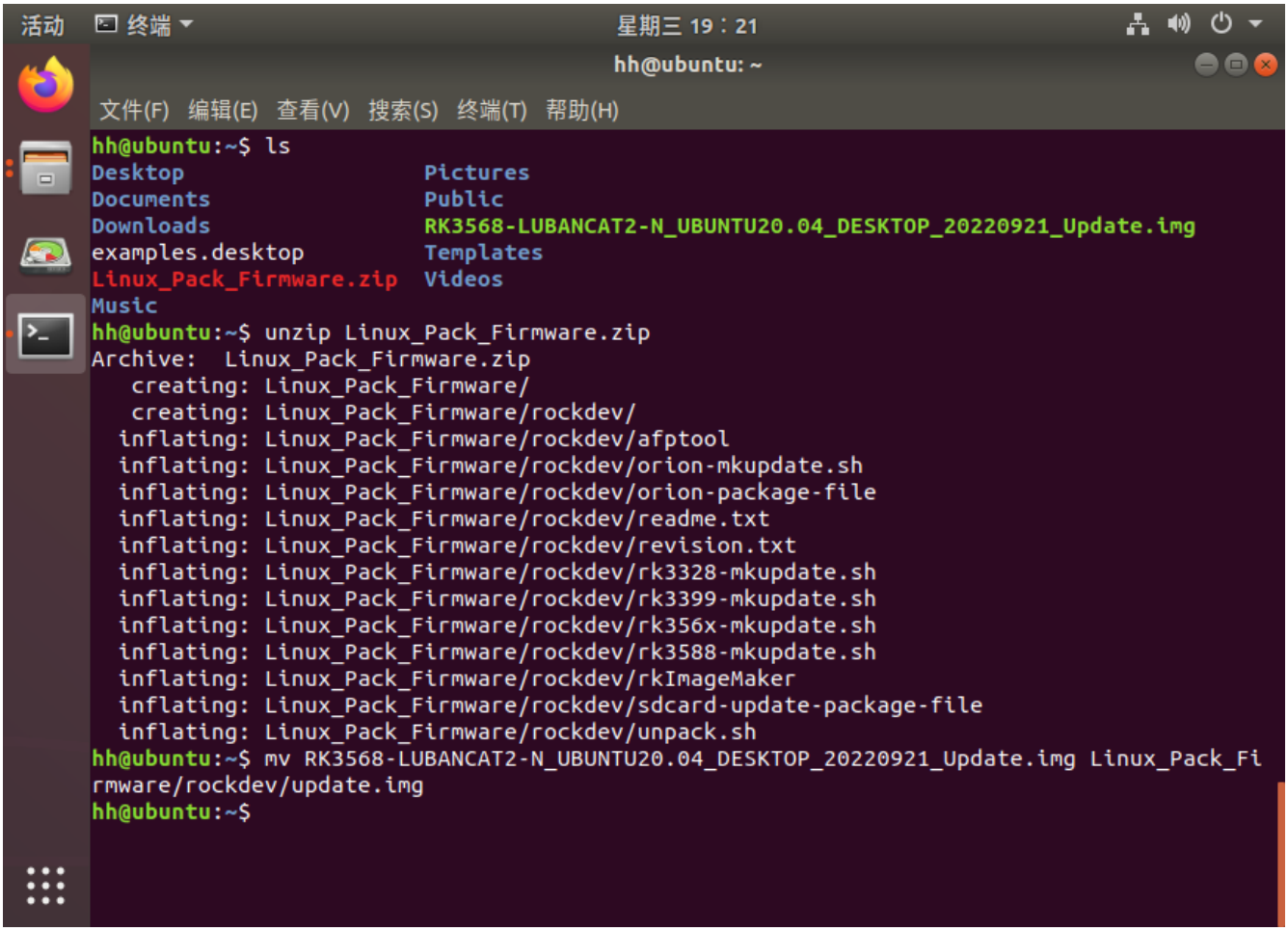
24.2.1 解包

我们也可以使用 Linux_Pack_Firmware 来将完整的 update.img 镜像解包成分区镜像的形式。

Linux_Pack_Firmware 请使用我们在网盘提供的版本, 将其解压到 Linux PC 或虚拟机中。

将从网盘获取的 Update.zip 镜像解压成.img 格式并重命名为 update.img，也可以直接使用 SDK 生成的 update.img 镜像。

将 update.img 复制到 Linux_Pack_Firmware/rockdev 目录下

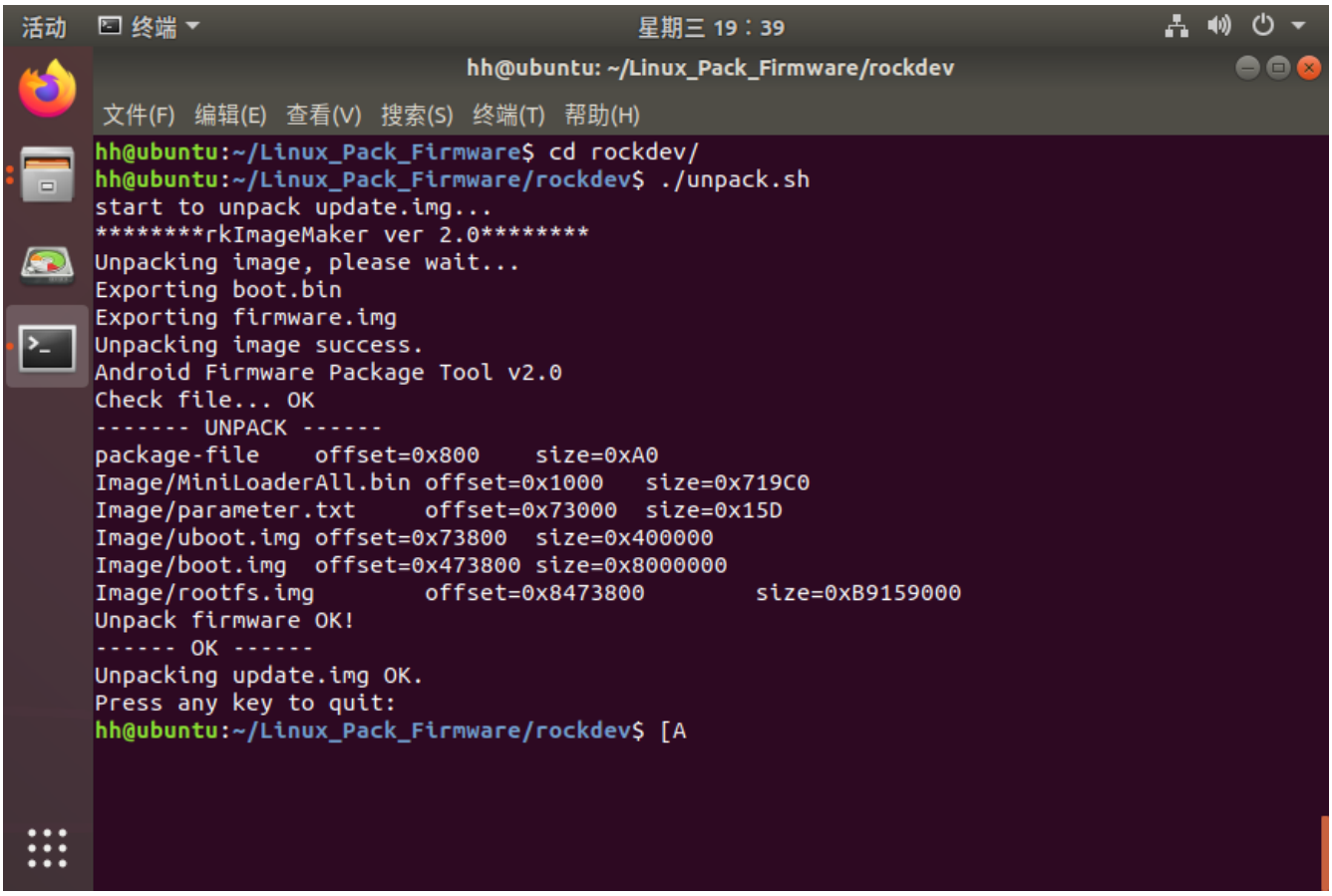


我们来看一下 Linux_Pack_Firmware 中的文件

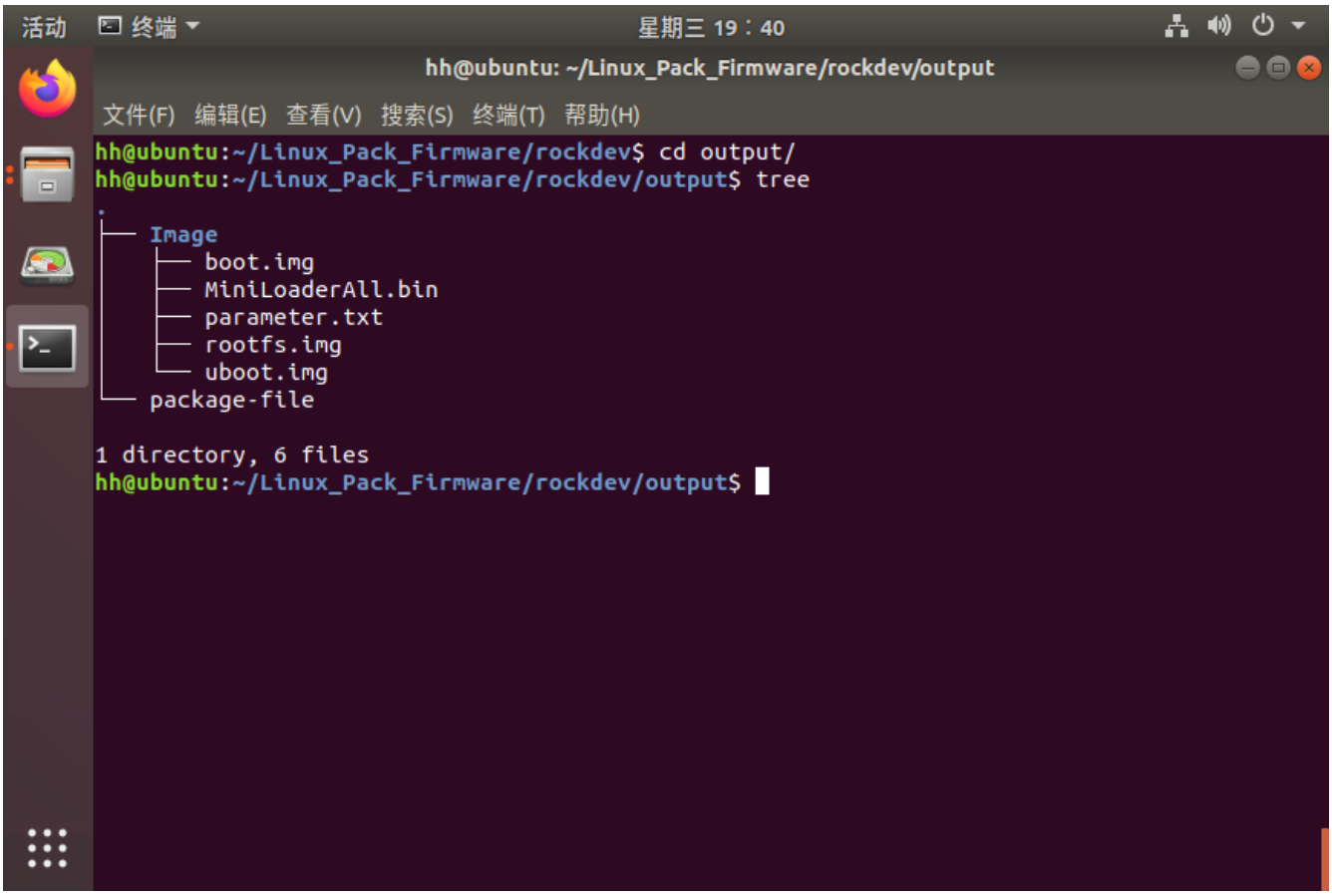
- afptool、rkImageMaker：二进制可执行文件
- rk3xxx-mkupdate.sh：镜像打包脚本
- unpack.sh：镜像解包脚本
- 其他：示例、说明文件等

运行 unpack.sh 脚本进行解包

```
unpack.sh
```



解包后的文件保存在 output 目录中，文件内容与 RKDevTool 解包后的内容相同。



output 目录中有一个文件一个文件夹

- package-file：分区与分区镜像名的对应关系
- Image：firmware.img 展开后的内容，也就是解包后的分区镜像。

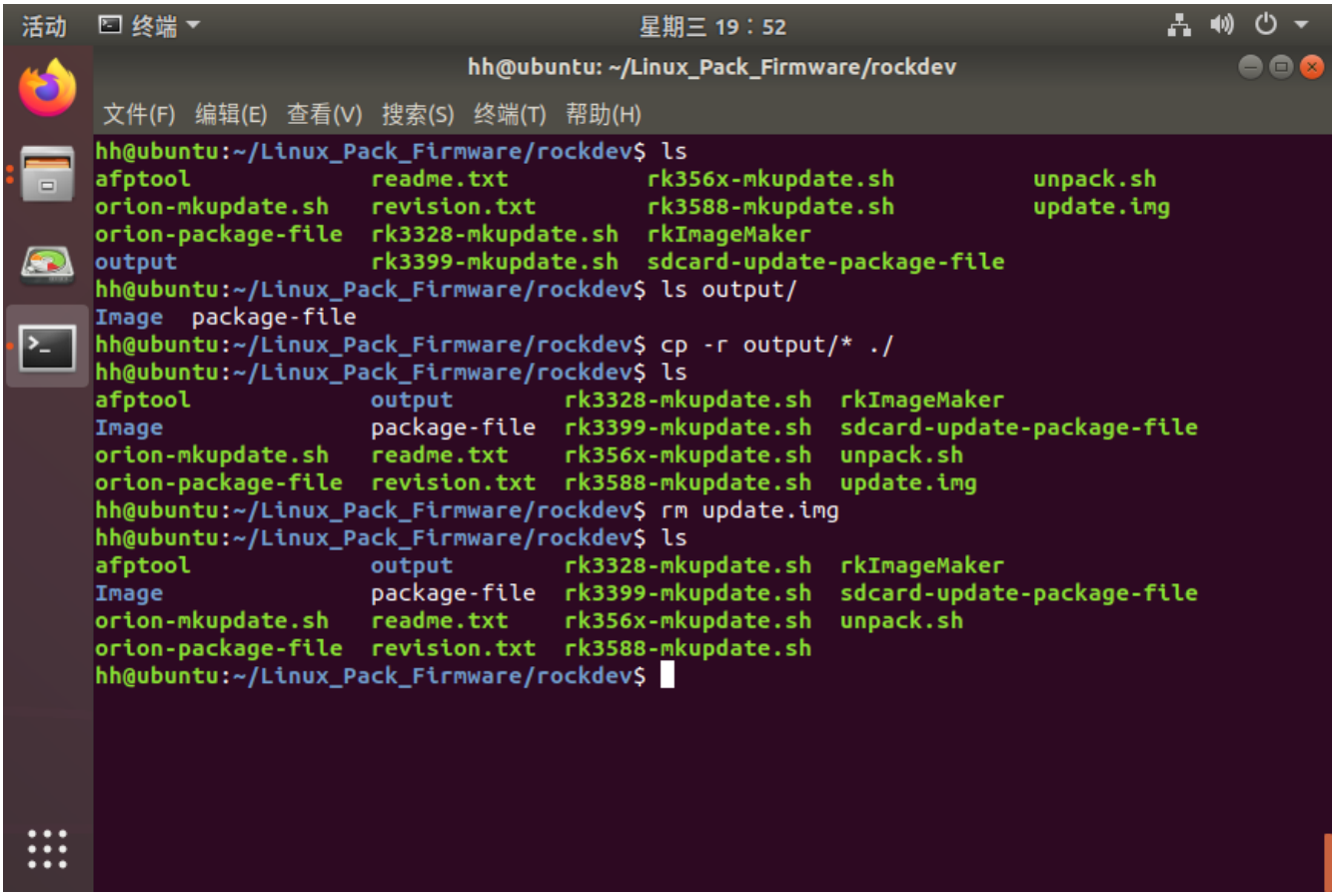
进入 Image 目录，根据不同操作系统镜像，有以下三类文件

- 分区表：parameter.txt
- loader 文件：MiniLoaderAll.bin
- 分区镜像：boot.img、uboot.img 等以 img 结尾的分区镜像文件

24.2.2 打包

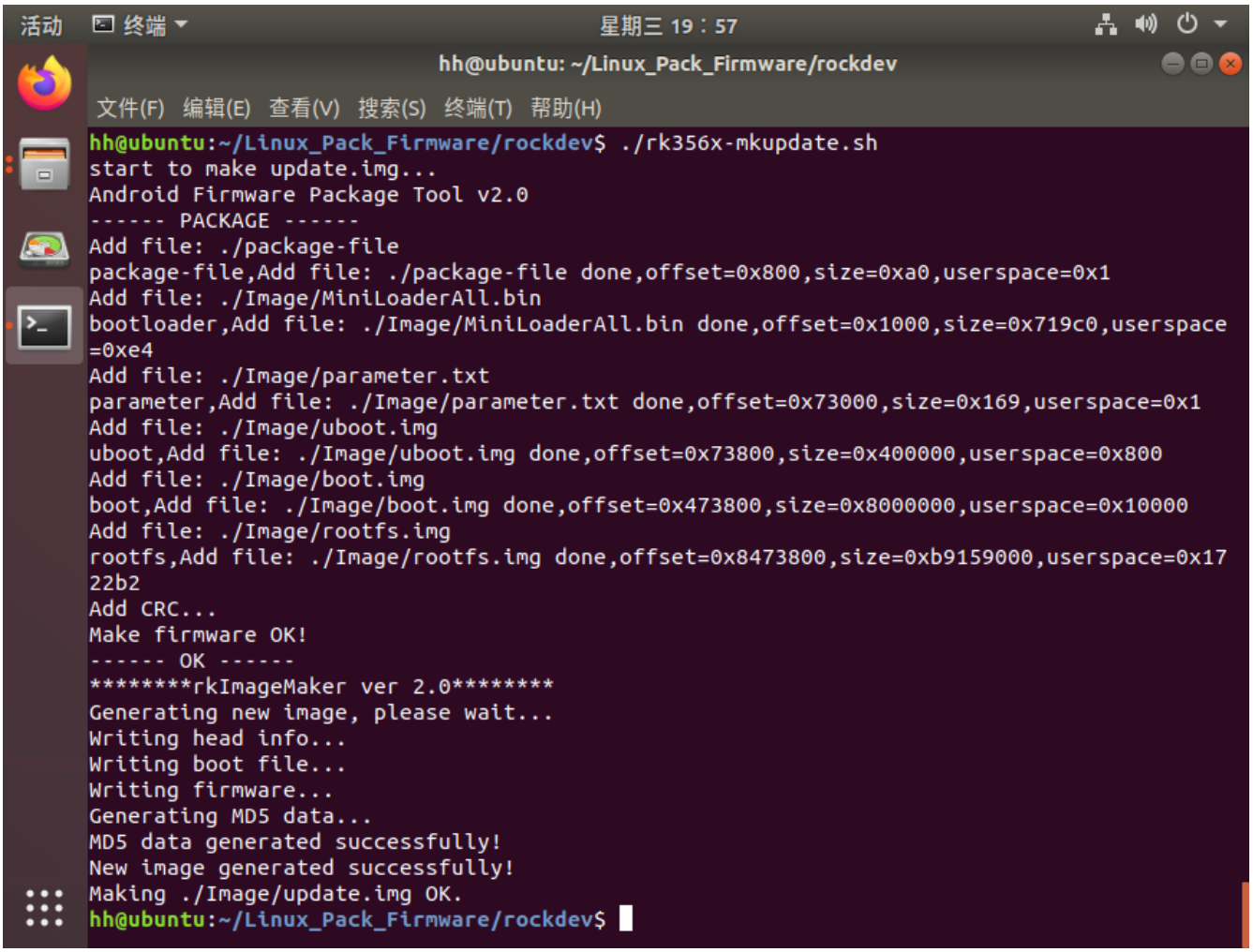
在上面的解包过程中，我们得到了 output 文件夹的内容。

首先我们将 rockdev/output 目录下的内容复制到 rockdev 目录下，删除之前用于解包的 update.img 镜像

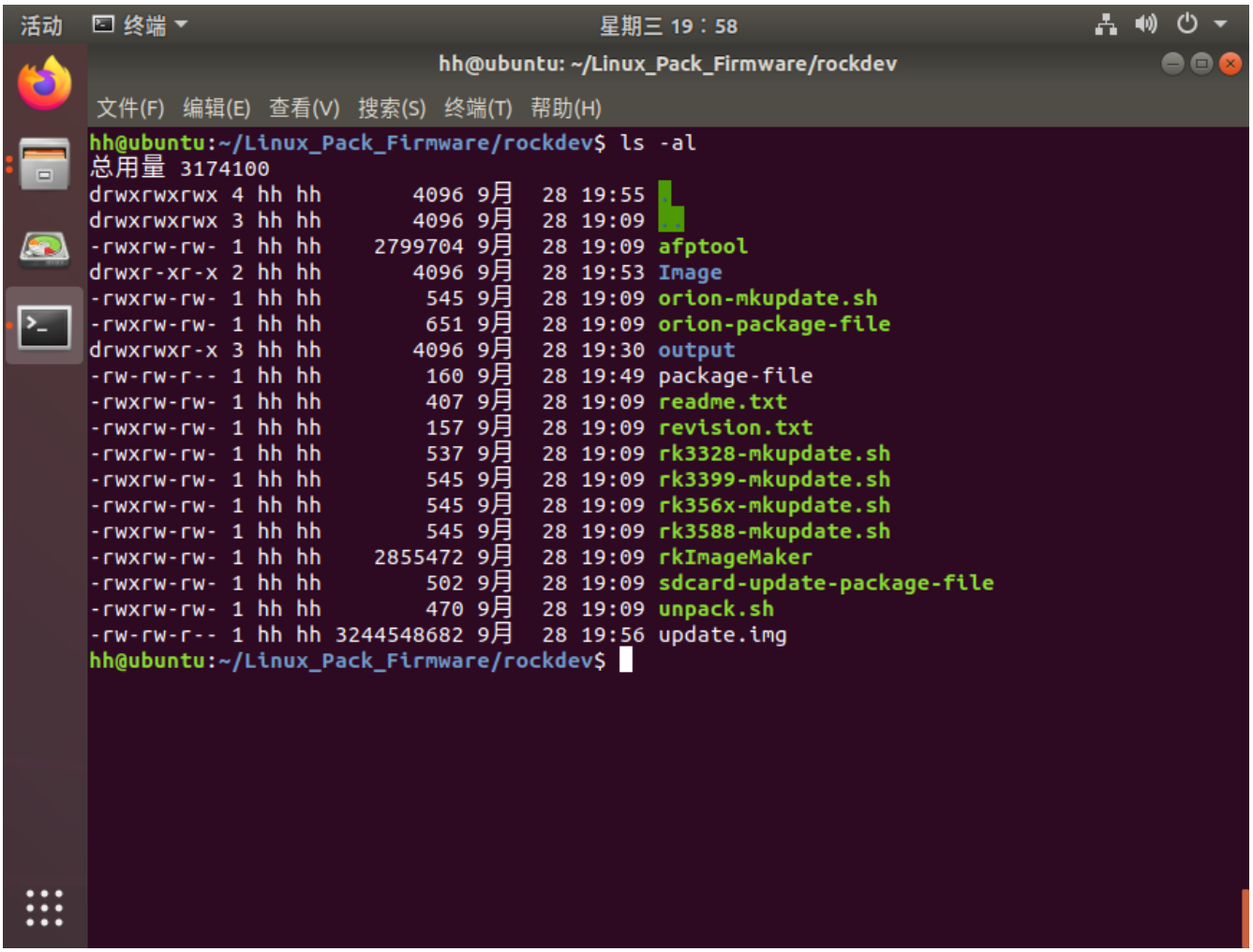


如果我们对某一分区镜像文件做出了改变，在打包时要注意镜像文件名正确，与 package-file 文件中的命名及路径一致。

LubanCat-1、2、Zero 都使用了 rk356x 主芯片，运行对应的打包脚本。



打包完成后，当前文件夹内出现了 update.img 文件，这就是打包后的完整镜像文件



我们使用烧录工具，将打包好的镜像烧录到板卡中，可以正常启动。

```
         Starting Network Manager Script Dispatcher Service...
[  OK  ] Started Network Manager Script Dispatcher Service.
         Starting Time & Date Service...
[  OK  ] Started Time & Date Service.
[  OK  ] Started Bluetooth management mechanism.

Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0

[username:password] root:root cat:temppwd

Modify information : /etc/issue

lubancat login: [   11.359201] rk-pcie 3c0000000.pcie: PCIe Link Fail
[   11.359296] rk-pcie 3c0000000.pcie: failed to initialize host

lubancat login: root
Password:


Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

 * Documentation:  http://doc.embedfire.com
 * Management:     http://www.embedfire.com

System information as of Wed Aug 31 23:27:53 CST 2022

System load:   2.37 0.56 0.19   Up time:       0 min
Memory usage:  7 % of 3900MB    IP:           192.168.103.119
CPU temp:      57°C             GPU temp:     57°C
Usage of /:    9% of 29G


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@lubancat:~#
```

第 25 章　系统镜像备份并重新烧录

在开发到生产过程中，对系统镜像的备份及再烧录过程是必不可少的，接下来介绍几种不同的备份方法和将备份镜像烧录的方法。

使用以下备份方法得到的备份镜像为 RAW 格式镜像，无法使用 USB 量产工具进行烧录，但可以使用其他支持 RAW 格式烧录的软件，如 Win32DiskImager、Etcher 等，还可以自己使用 dd 命令来进行烧录。

注解： RAW 格式原意是指未经加工的格式，野火相关文档中 RAW 格式是指镜像中已经包含了完整分区信息及分区文件，整个镜像是一个整体不可拆分。与 RK 格式 (打包时将分区标志打包进完整镜像，烧录时根据分区表将对应的分区烧录到相应的地址）做区分。

注意： 如果是希望备份 sd 卡系统烧录到 emmc 的用户，请提前修改/etc/fstab 文件，否则无法挂载 boot 分区，修改如下，如果不是则不需要修改。

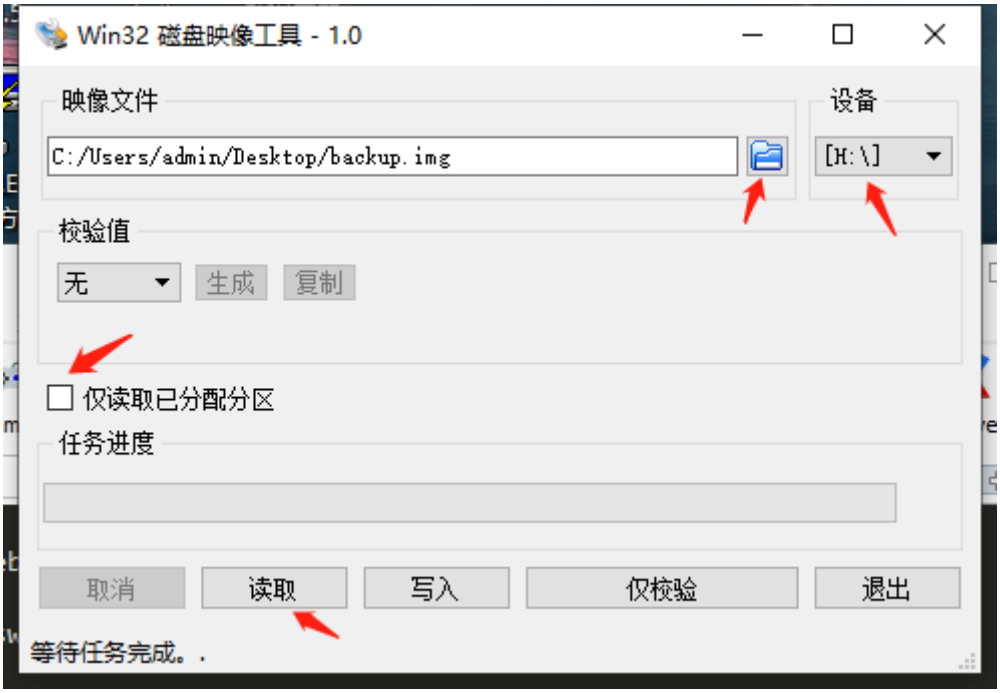
```
1 #mmcblk1p2 为 sd 卡的 boot 分区，mmcblk0p2 为 emmc 的 boot 分区，启动系统会自动挂载
  该分区到/boot 目录
2 #/dev/mmcblk1p2  /boot  auto  defaults  0 2
3
4 /dev/mmcblk0p2  /boot  auto  defaults  0 2
```

25.1 使用 Win32DiskImager 全卡备份 SD 卡系统镜像并重新烧录

使用 Win32DiskImager 进行全卡备份是最简单的一种备份 SD 卡的方法，但是由于对整个 SD 卡进行备份，会导致备份文件和 SD 卡的容量一致，而且在还原的时候必须使用比镜像更大容量的 SD 卡。

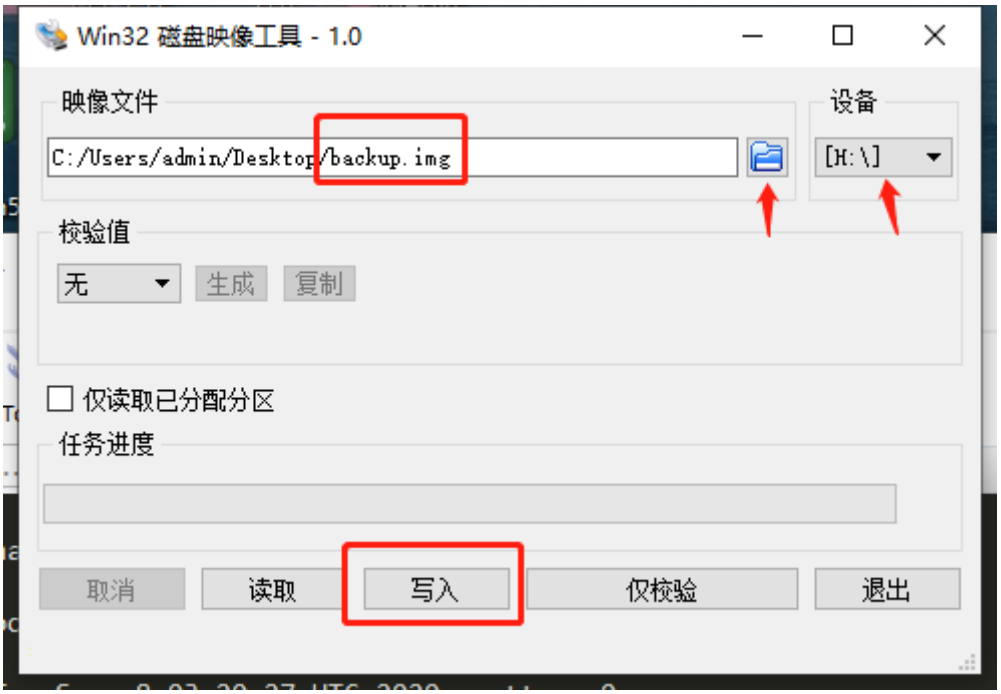
Win32 磁盘映像工具下载链接: https://win32diskimager.org/

在 windows 下新建一个空文件，后缀为 img，例如 backup.img，将需要备份镜像的 SD 卡插到 windows 上并打开 Win32 磁盘映像工具，然后点击文件夹图标，找到刚刚创建的 backup.img 文件并确认，并确认 SD 卡盘符，取消“仅读取已分配区”的勾选，最后点击读取按钮，如果弹出是否覆盖 backup.img 的对话框，点击是即可，等待镜像备份完毕。



此时镜像已经备份完毕，备份好的镜像就是 RAW 格式的 backup.img。

除了使用 Win32 磁盘映像工具烧写 backup.img 镜像到 SD 卡，也可以使用 Etcher 等支持烧录 RAW 格式镜像的软件来烧录。打开 Win32 磁盘映像工具，选择 backup.img 镜像，选择新的空白 SD 卡盘符，最后点击写入按钮即可，注意被烧录的 SD 卡大小必须大于等于镜像的大小。



25.2 使用 dd 命令压缩备份 SD 卡系统镜像并重新烧录

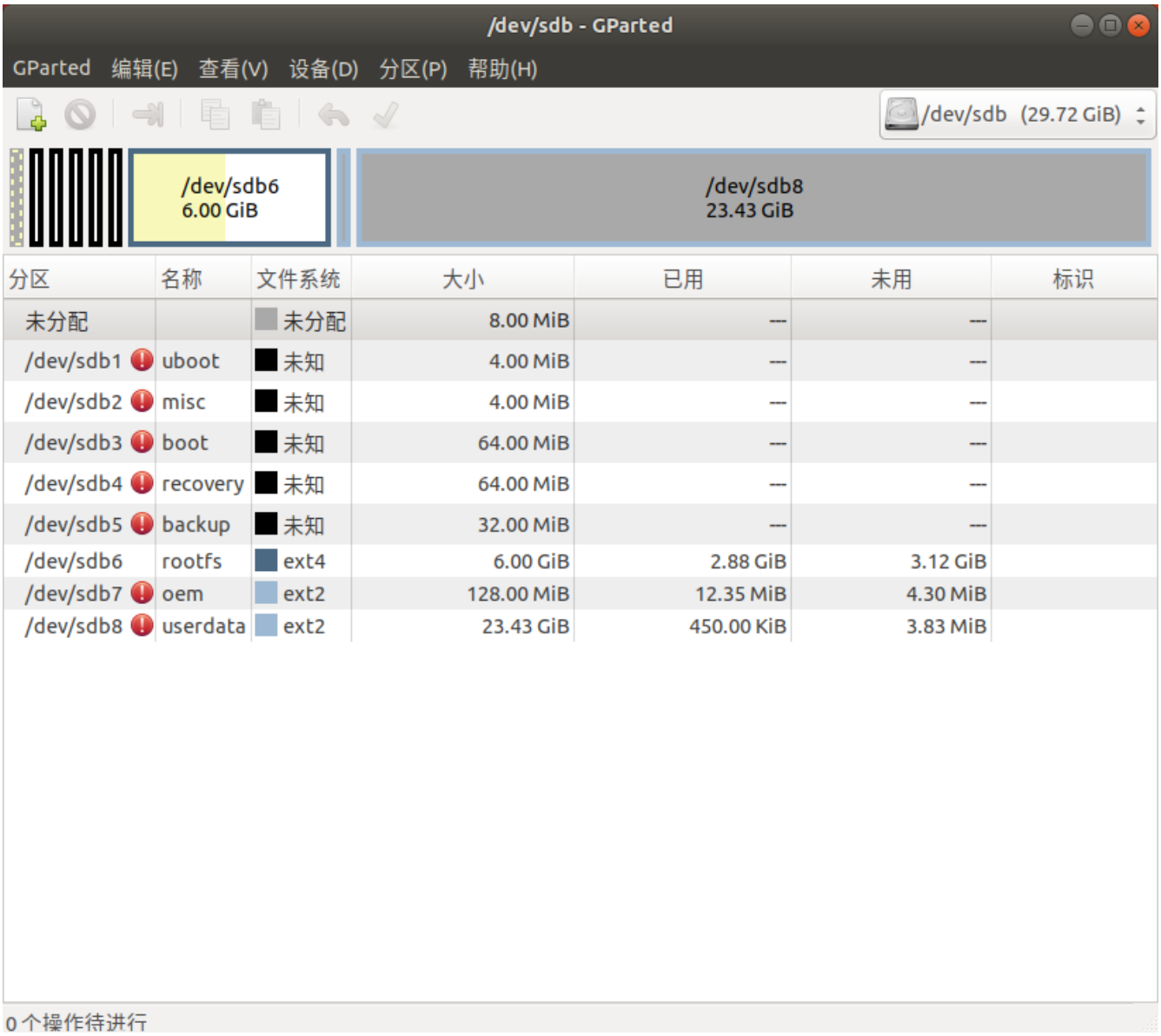
使用 dd 命令进行 SD 卡压缩备份虽然需要借助运行 Linux 系统的主机或虚拟机，但是备份出的镜像体积较小，便于再次发布和烧录。

将已经搭建好环境的 SD 卡从板卡取出并插到读卡器上，然后将读卡器接入 PC 机（Ubuntu 中），

打开 Gparted 磁盘管理工具，将磁盘切换到 SD 卡，就能看到一下信息 (烧录 debian 系统的 SD 卡为例)

没有 Gparted 的可以使用以下命令安装。

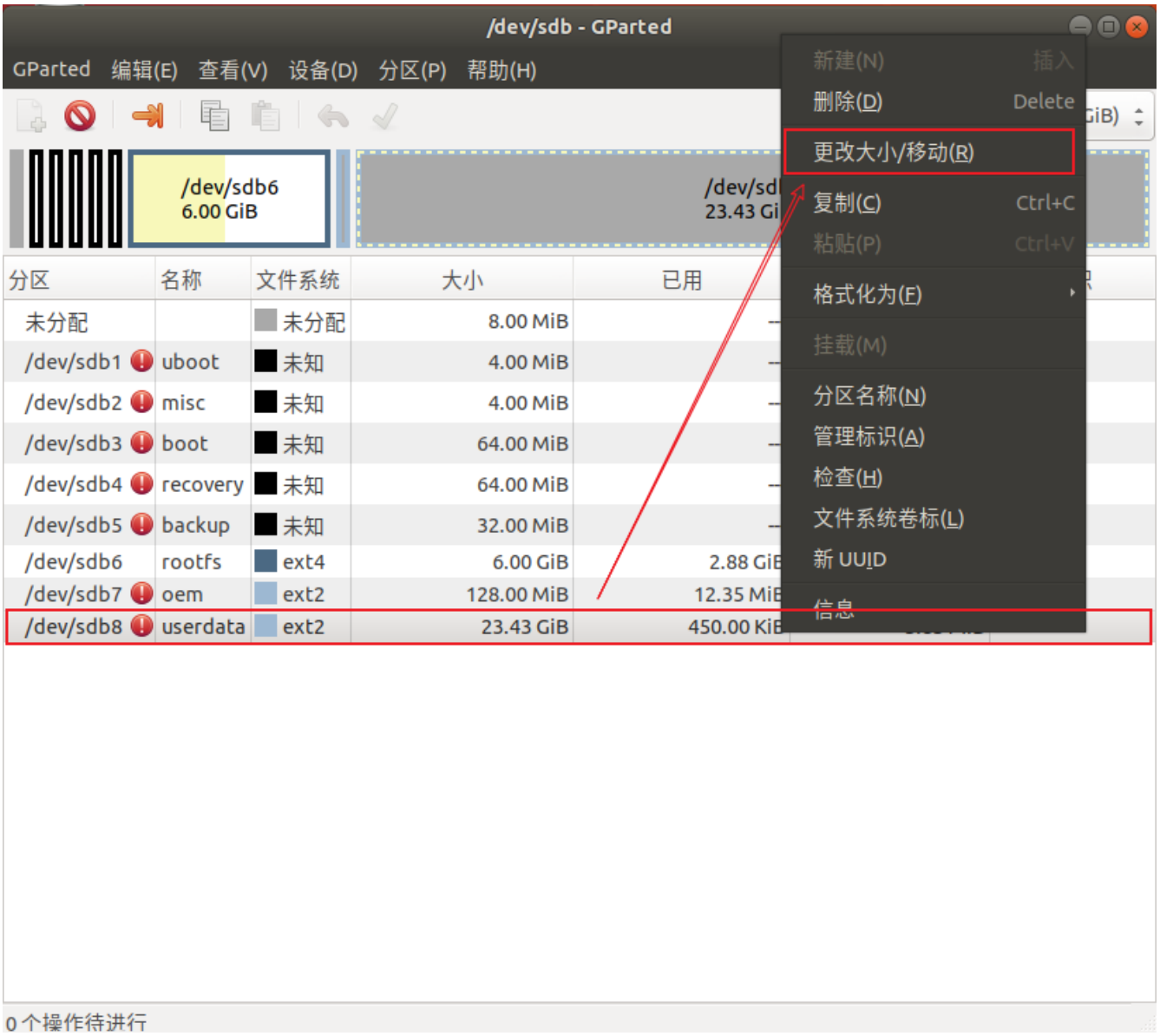
```
sudo apt install gparted
```



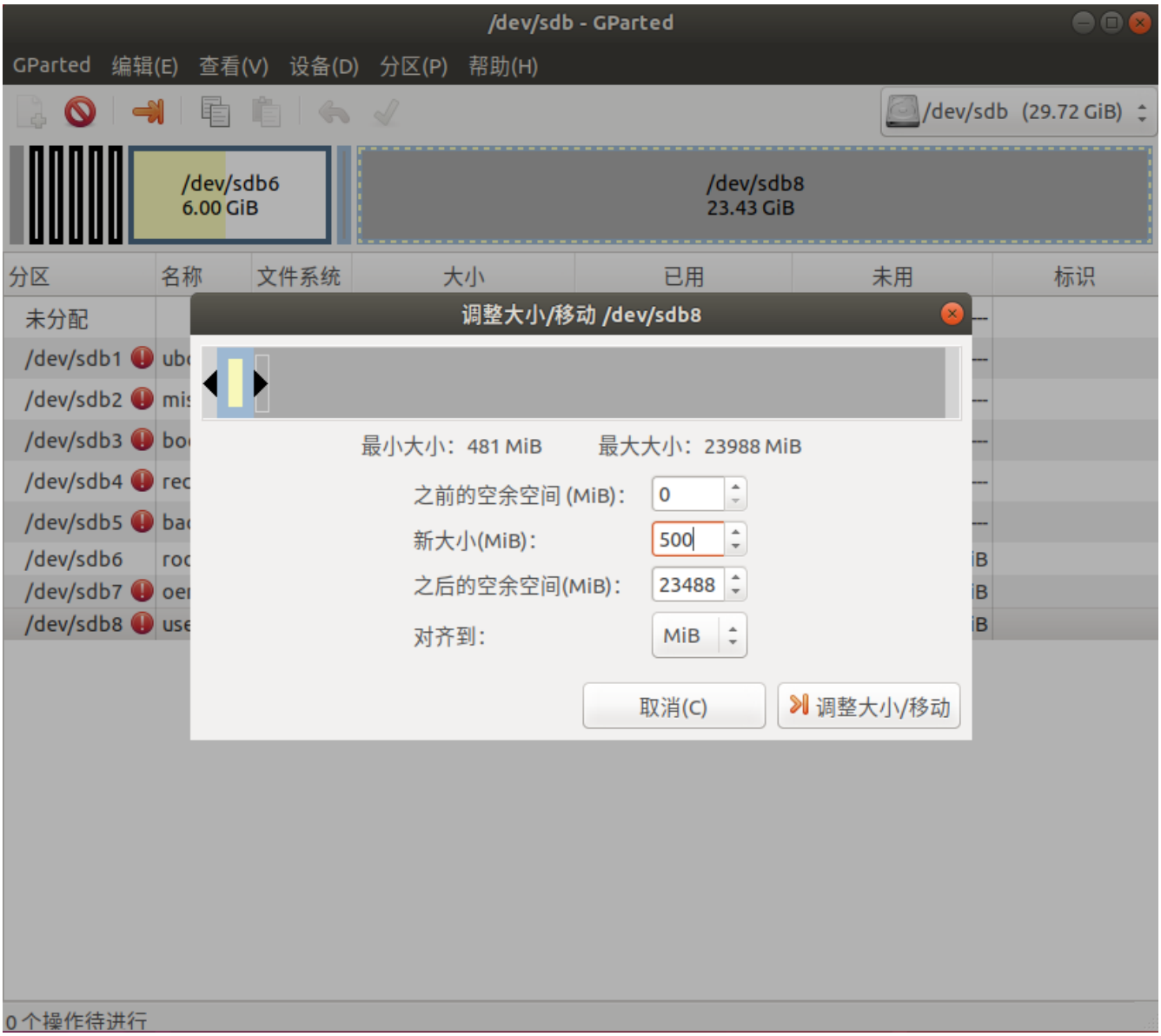
从图中可以看到 SD 卡中共 8 个已分配的分区，要备份的内容是最后一个分区及之前的内容。

最后一个分区大小为 23.43GiB，但是已用空间只有 450KiB，我们将最后一个分区的大小进行压缩，就可以减小备份镜像的大小。

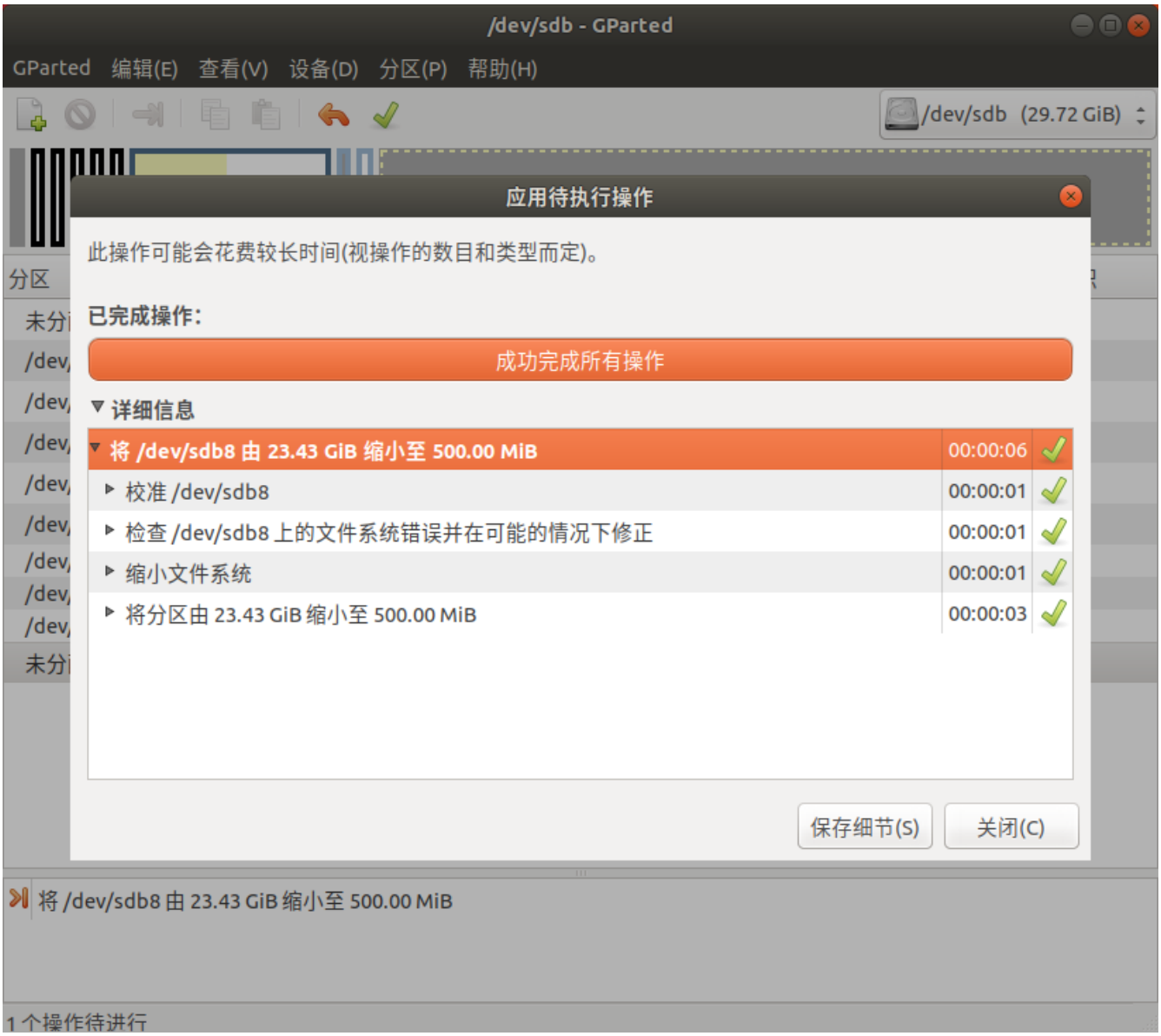
选中要调整大小的分区，点击鼠标右键，选中 **更改大小/移动**



从图中看到，此分区的最小大小为 481M，我们将它压缩到 500M，然后点击 **调整大小/移动**



配置完成后点击绿色对钩，开始执行的更改



如果希望备份的镜像可以在烧录启动时自动扩展最后一个分区，可以在终端执行以下命令。

挂载 rootfs 分区，根据上图可知，也就是/dev/sdb6，然后删除其中的一个文件

```
# 挂载 rootfs 分区
sudo mount /dev/sdb6 /mnt

# 删除此文件，使用备份镜像启动会自动扩展分区
```

(下页继续)

(续上页)

```
sudo rm /mnt/var/lib/misc/firstrun

# 删除完成后，卸载已挂载的分区
sudo umount /mnt
```

现在再计算一下需要备份的大小 (8+4+4+64+64+32+6.00x1024+128+500)MiB≈6948MiB，由于显示的大小通过四舍五入的方式保留 2 位小数，所以我们可以加一点余量，备份 7000MiB 的大小。

注解： 在 Linux 下 1MiB=1024KiB=1048576Byte，1MB=1000KB=1000000Byte。

使用 mkdir 命令创建一个新的目录，用于存放从带镜像的 SD 卡中拷贝的镜像。然后使用 dd 命令将带镜像的 SD 中的镜像拷贝到新创建的目录中。

```
# 创建新目录
mkdir backup

# 将带镜像的 SD 卡中的镜像拷贝到创建的目录中
sudo dd if=/dev/sdb of=./backup/backup.img count=7000 bs=1024k conv=sync

# 拷贝需要时间，请耐心等待，打印消息如下
记录了 7000+0 的读入
记录了 7000+0 的写出
7340032000 bytes (7.3 GB, 6.8 GiB) copied, 492.74 s, 14.9 MB/s
```

提示： 若备份的镜像烧录后仍无法正常运行，请将 bs=1024k 改为 bs=1M 并去掉 conv 参数，即 `sudo dd if=/dev/sdb of=./backup/backup.img count=7000 bs=1M`

等待 dd 命令运行完成后，就得到了 RAW 格式的 backup.img 镜像

dd 命令参数的含义：

- if= 文件名：输入文件名，缺省为标准输入。即指定源文件。< if=/dev/sdb >

- of= 文件名：输出文件名，缺省为标准输出。即指定目的文件。< of=./backup/backup.img, 这里的.img 是镜像的格式，转成.img 格式的文件后方便后续使用 etcher 烧录镜像 >
- bs = bytes：同时设置读入/输出的块大小为 bytes 个字节，此处填的是 1024k，表示 1M 大小。
- count = blocks：仅拷贝 blocks 个块，块大小等于 ibs 指定的字节数，此处设置的是 7000，表示 7000 个 bs，也就是 7000M。
- conv= sync：将每个输入块填充到 ibs 个字节，不足部分用空（NUL）字符补齐。

RAW 格式的 backup.img 镜像，可以使用使用 Win32DiskImager 或 Etcher 等软件来烧录到 SD 卡，也可以使用 dd 命令去写入到空的 SD 卡中。

将空白 SD 卡插入 PC，确认插入 SD 卡的设备号，这里演示的设备号是/dev/sdc，烧录时以 SD 卡实际对应的设备号灵活修改。

```
# 将 backup.img 写入 SD 卡
sudo dd if=./backup/backup.img of=/dev/sdc bs=1024k conv=sync

# 写入完成后，打印消息如下
记录了 7000+0 的读入
记录了 7000+0 的写出
7340032000 bytes (7.3 GB, 6.8 GiB) copied, 427.027 s, 17.2 MB/s
```

不指定 count，会将整个 backup.img 文件写入目标位置。

等待写入完成后，即可使用烧录好的 SD 卡来启动板卡了。

25.3 使用 fire-config 压缩备份 eMMC 中的系统镜像

25.3.1 说明

fire-config v1.2.4 版本添加支持了 SD 卡备份 eMMC 和重新烧录 eMMC 的功能，将操作步骤通过脚本一键完成，方便开发和生产。fire-config 执行的步骤和后一小节——“使用 dd 命令压缩备份 eMMC 中的系统镜像”的内容一样，只不过通过脚本自动执行。假如 fire-config 出现了备份镜像不完整、根文件损坏等 bug，可自行跳到 《使用 dd 命令压缩备份 eMMC 中的系统镜像》 章节进

行操作。野火也在努力修复 fire-config 备份相关的 bug，假如您出现了上述提到的问题，请把操作步骤和报错贴到 这个帖子 上反馈，我们收到后会尽快回复并修复。

- 对于备份：

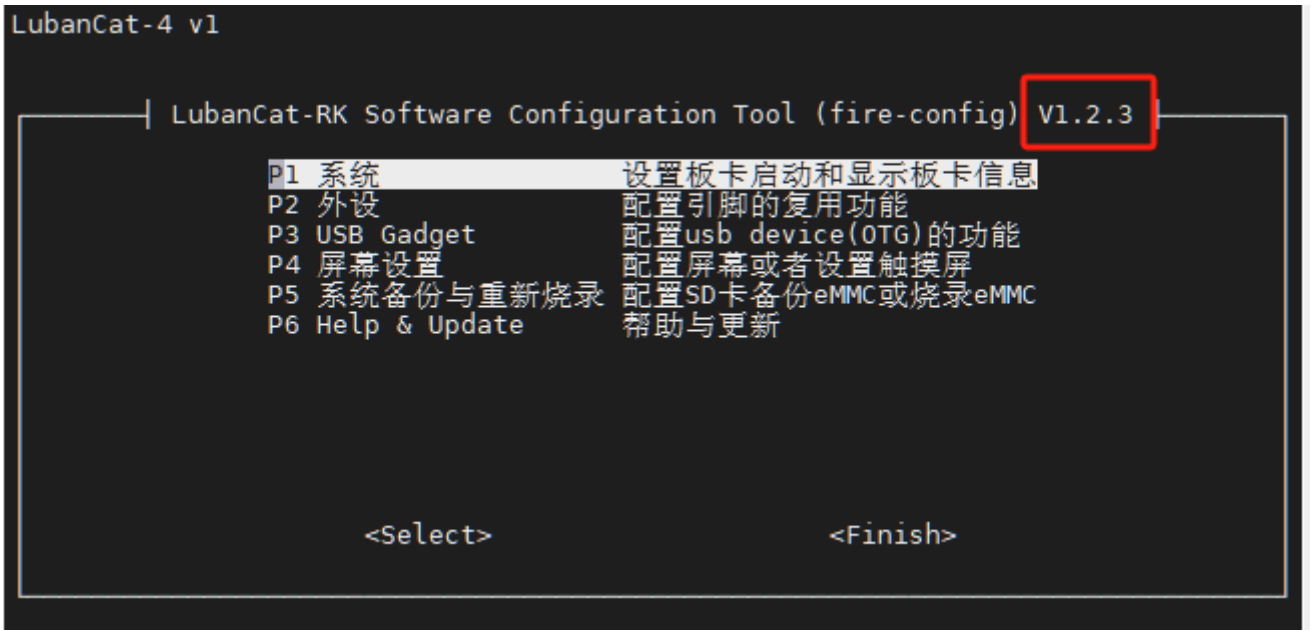
1. 自动压缩 eMMC 没有使用的空间，减小备份镜像的大小。
2. 添加随机网口 Mac 和扩充脚本。
3. 使用 dd 进行备份，与 SDK 构建的格式不一样。

- 对于烧录：

1. 将备份镜像重新烧录到 eMMC，镜像需位于/home/cat/或者/mnt/，名字为 backup.img。
2. 只能烧录 dd 打包的镜像，不能烧录 SDK 构建的通用镜像。

- 确认版本：

执行 fire-config 命令打开界面，可以从界面中找到当前版本，如下图，需要 v1.2.4 及以上的版本。



如果低于 v1.2.4 版本可以通过网络或者 deb 包进行更新：

提示： 建议将 fire-config 通过网络更新到最新版本再进行后续操作。

网络更新：

```
# 更新软件数据库
sudo apt update

# 下载最新版 fire-config
sudo apt install fire-config
```

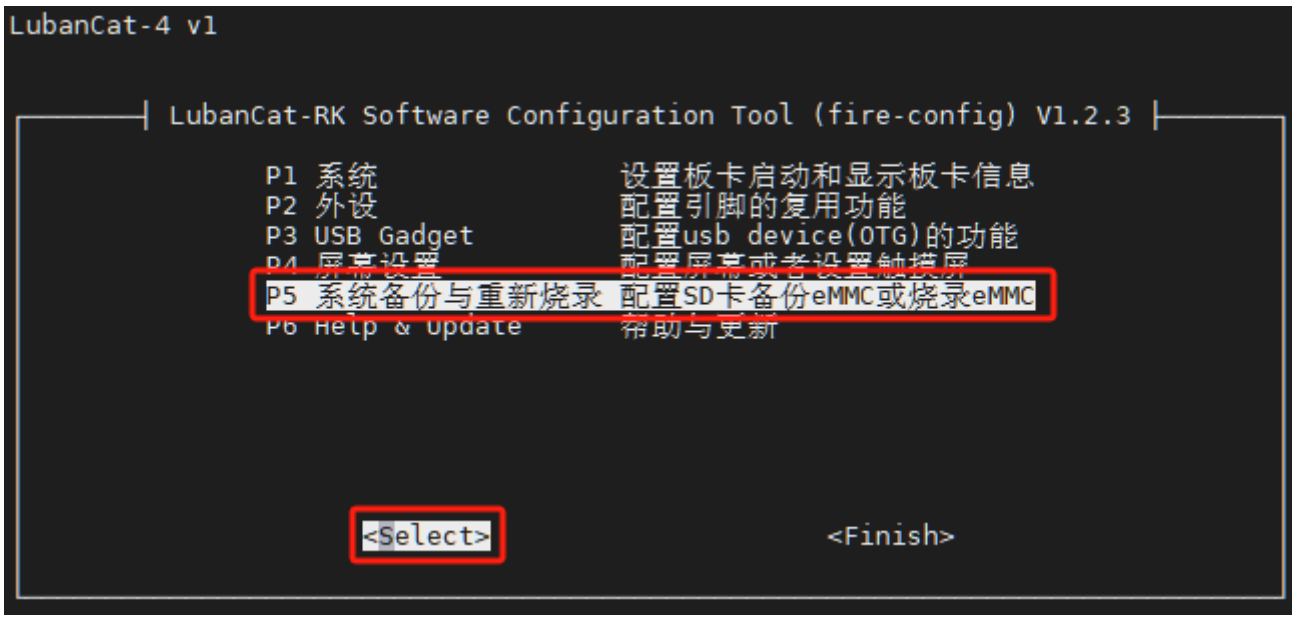
离线 deb 更新：

- fire-config-v1.2.6：`fire-config_1.2.6_arm64.deb`

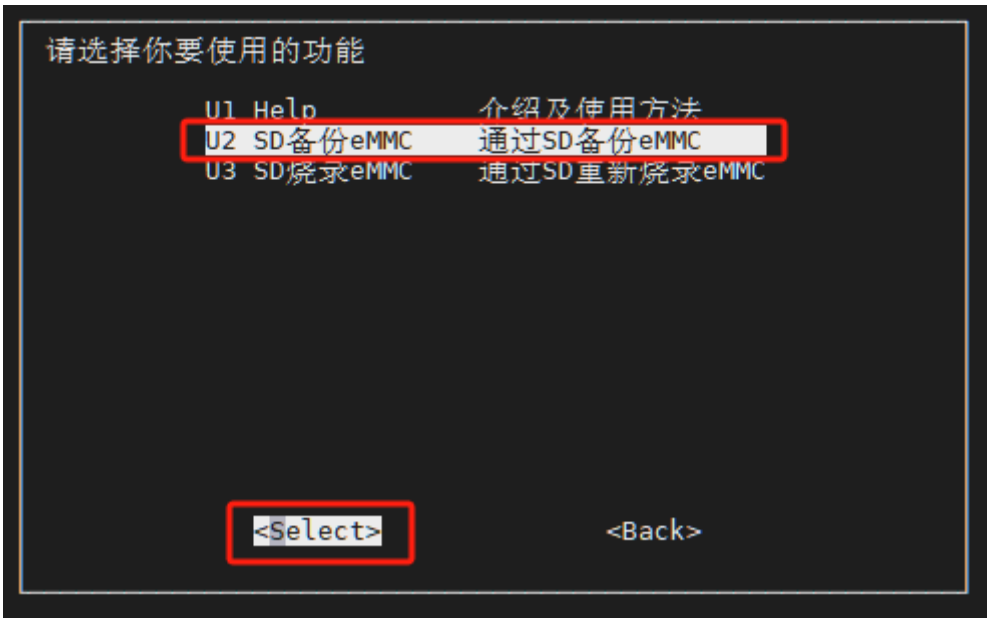
下载离线 deb 包，然后传到板卡，使用命令 sudo dpkg -i fire_config*.deb 进行安装。

25.3.2 SD 备份 eMMC

1. 命令行执行 fire-config 打开界面，选择“P5 系统备份与重新烧录选项”，如下图：



2. 进入系统备份与重新烧录界面后，选择“U2 SD 备份 eMMC”，如下图：



3. 选择后，会自动判断是否是 SD 卡启动系统，然后是 SD 卡启动，则自动执行备份命令。

信息输出如下：

```
添加随机 mac 和扩容分区内容到 eMMC boot_init.sh 初始化脚本完成
可以进行分区调整，备份的文件系统的已用大小：6486MB，调整分区大小为：6686MB
检查文件系统是否挂载...
文件系统未挂载，可进行压缩空间和备份...
e2fsck 1.46.2 (28-Feb-2021)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 3A: Optimizing directories
Pass 4: Checking reference counts
Pass 5: Checking group summary information

/dev/mmcblk0p3: ***** FILE SYSTEM WAS MODIFIED *****

      132371 inodes used (1.77%, out of 7477792)
          94 non-contiguous files (0.1%)
         164 non-contiguous directories (0.1%)
```

(下页继续)

(续上页)

```
            # of inodes with ind/dind/tind blocks: 0/0/0
            Extent depth histogram: 121444/59
   1568317 blocks used (5.14%, out of 30502904)
         0 bad blocks
         1 large file

      109957 regular files
      9930 directories
         55 character device files
         25 block device files
         0 fifos
         7 links
      12395 symbolic links (10780 fast symbolic links)
         0 sockets
------------
      132369 files
开始调整文件系统分区大小为 6686MB
resize2fs 1.46.2 (28-Feb-2021)
resize2fs: Invalid new size: 6686MB

End?  [125GB]? 6686MB
Warning: Shrinking a partition can cause data loss, are you sure you want to continue?
Yes/No? yes
Information: You may need to update /etc/fstab.

e2fsck 1.46.2 (28-Feb-2021)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
```

(下页继续)

(续上页)

```
/dev/mmcblk0p3: 132371/7477792 files (0.2% non-contiguous), 1568317/
↪30502904 blocks
文件系统分区调整完成。

正在执行 sync 同步....
开启打包镜像，请耐心等待...
7142899712 字节 (7.1 GB, 6.7 GiB) 已复制，122 s，58.5 MB/s
记录了 6886+0 的读入
记录了 6886+0 的写出
7220494336 字节 (7.2 GB, 6.7 GiB) 已复制，123.516 s，58.5 MB/s
正在执行 sync 同步....
```

- 挂载 eMMC 根文件系统到/media/emmc/，向/media/emmc/etc/init.d/boot_init.sh 初始化脚本添加扩容和随机网口 Mac 代码。
- 判断 eMMC 根文件系统是否需要压缩空间，压缩空间可以减小备份镜像的体积，重新烧录 eMMC 后会自动扩容。
- e2fsck 工具自动检测和修复文件系统，resize2fs 工具压缩文件系统，parted 工具调整根文件系统分区大小。
- 通过 dd 命令打包镜像，期间会判断当前设备空间是否充足，如果充足则打包镜像到/home/cat/backup.img，如果不充足，则需要插入 **NTFS 格式**的 U 盘，自动挂载/mnt，并将镜像打包到/mnt/backup.img。

4. 打包完成后会提示，备份已完成并提示备份镜像生成的路径，如下图：

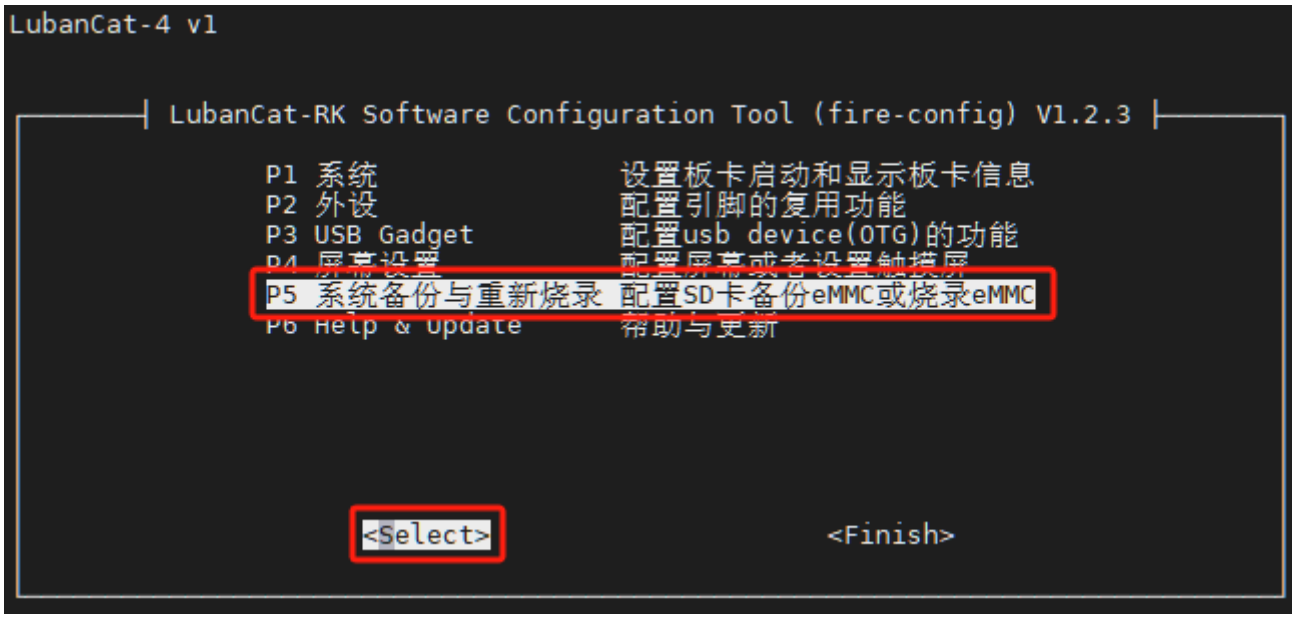
生成的 backup.img 镜像为 dd 格式，可以通过 fire-config 工具重新烧录到 eMMC，或者参照本章后续小节——“使用 RKDevTool 烧录 RAW 格式镜像到 eMMC”的内容，使用 RKDevTool 工具通过 USB 烧录到 eMMC。

25.3.3 SD 烧录 eMMC

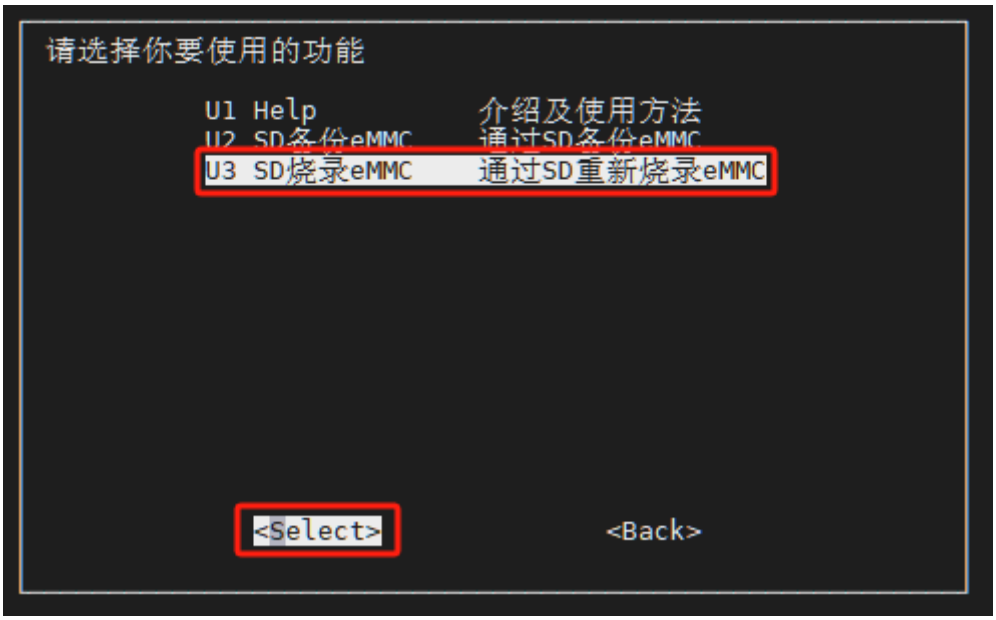
需要注意：

- 要烧录的镜像必须是 dd 打包的，不能是 SDK 构建出来的通用镜像。
- 镜像需要放在/home/cat/或/mnt，名字为 backup.img。
- 如果/home/cat/和/mnt 同时存在 backup.img 则需要删除其中一个。

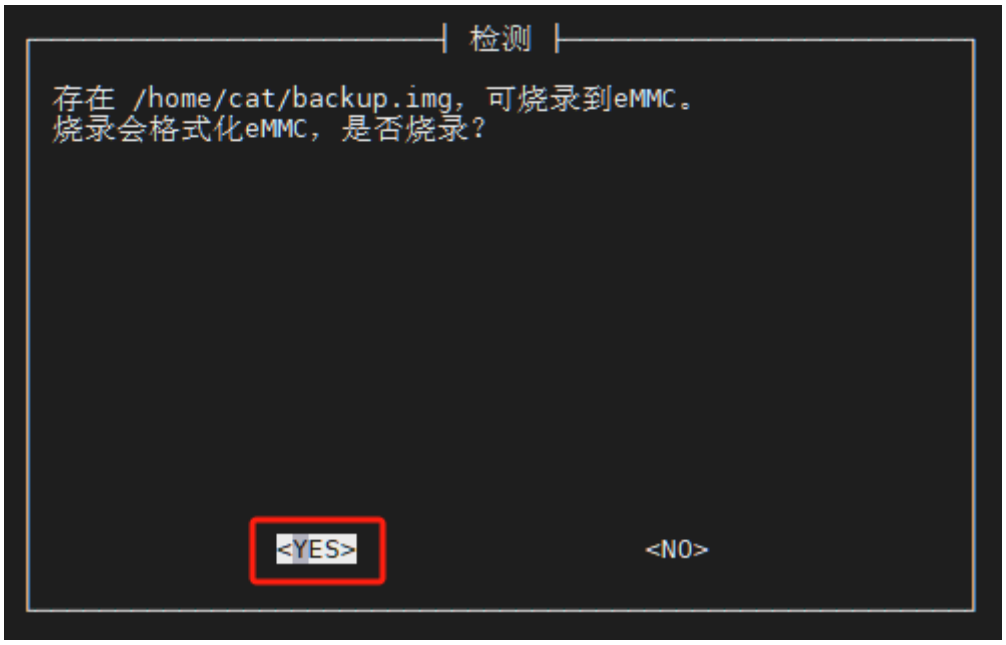
1. 命令行执行 fire-config 打开界面，选择“P5 系统备份与重新烧录选项”，如下图:



2. 进入系统备份与重新烧录界面后，选择“U3 SD 烧录 eMMC”，如下图:



3. 自动检测 backup.img，如果/home/cat/或/mnt 存在，则可以进行烧录，如下图:

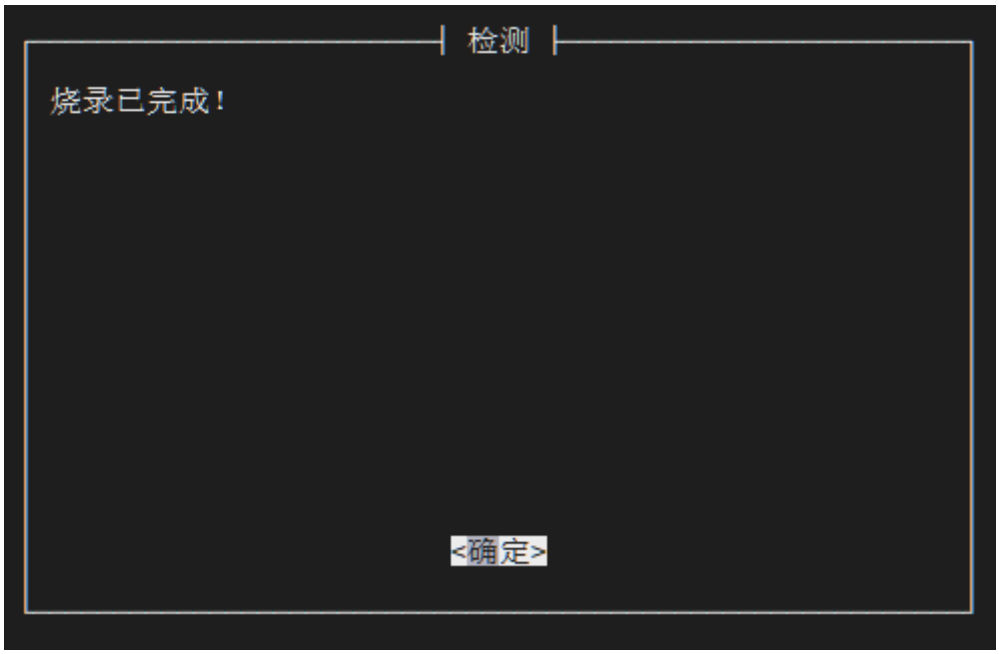


信息输出如下：

```
开始烧录 eMMC....
7052722176 字节 (7.1 GB, 6.6 GiB) 已复制, 17 s, 415 MB/s[ 3458.479757] mmcblk0: p1 p2 p3

记录了 6886+0 的读入
记录了 6886+0 的写出
7220494336 字节 (7.2 GB, 6.7 GiB) 已复制, 28.8714 s, 250 MB/s
正在执行 sync 同步....
```

- 通过 dd 命令将 backup.img 烧录到 eMMC。

4. 烧录完成后会提示烧录已完成，如下图，然后断电拔掉 SD 卡，从 eMMC 启动系统即可。



25.4 使用 dd 命令压缩备份 eMMC 中的系统镜像

使用 dd 命令对 eMMC 中的系统镜像进行备份和使用 dd 命令对 SD 卡备份的过程和原理都类似，都是在 Linux 环境下，去读取并复制要备份存储设备中的内容。

备份 SD 卡和备份 eMMC 的不同点在于，SD 卡是可移动存储设备，可以借助 PC 的 Linux 环境，而 eMMC 在板卡上不可拆卸。如果使用 SD 卡启动启动系统，就能为 eMMC 备份提供 Linux 环境了。

注解： SD 卡备份过程中以 rkboot 分区 Debian 镜像为例演示，eMMC 备份过程中将以 extboot 分区 Ubuntu 镜像进行演示，两系统镜像仅分区格式和数量有部分差异，对备份方法及过程无影响，可对照使用。

重要： 以下备份 eMMC，是以 SD 卡启动系统，通过 gparted 工具压缩 eMMC 系统空间，然后添加系统扩容和随机网卡 mac 地址，最后使用 dd 命令进行备份，以确保备份出来的镜像大小比较

小，再烧录时间短，烧录后系统分区能扩容，网口 Mac 地址不重复。

25.4.1 压缩 eMMC 系统空间

我们需要准备一张 SD 卡，并烧录用于备份的板卡镜像，此处的镜像是板卡对应的通用镜像，Debian 或 Ubuntu 通用镜像都可以，使用最新版本即可，板卡通用镜像可以从网盘下载。

还需要一个存储设备，用于存放备份后的镜像，最好是 U 盘，读写速度较快，可以加快备份速度。没有 U 盘的话，也可以使用启动板卡的 SD 卡，单独创建一个用于存放备份镜像的分区。

先用 **eMMC 启动**，创建一个标志文件，用以验证修改过的内容是否被正确备份了。

```
Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0

[username:password] root:root cat:temppwd

Modify information : /etc/issue

lubancat login: root
Password:

Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

 * Documentation:  http://doc.embedfire.com
 * Management:     http://www.embedfire.com

System information as of Wed Sep 28 11:34:02 CST 2022

System load:   2.96 1.28 0.48   Up time:       1 min
Memory usage:  8 % of 3900MB    IP:           192.168.103.123
CPU temp:      57°C             GPU temp:     57°C
Usage of /:    9% of 29G

Last login: Wed Aug 31 23:28:26 CST 2022 on ttyFIQ0
root@lubancat:~# date
Wed Sep 28 11:34:07 CST 2022
root@lubancat:~# date > backup.txt
root@lubancat:~# cat backup.txt
Wed Sep 28 11:34:16 CST 2022
root@lubancat:~#
```

使用 poweroff 命令关机，**插入 SD 卡，系统是优先 SD 卡启动的，直接上电启动即可**。

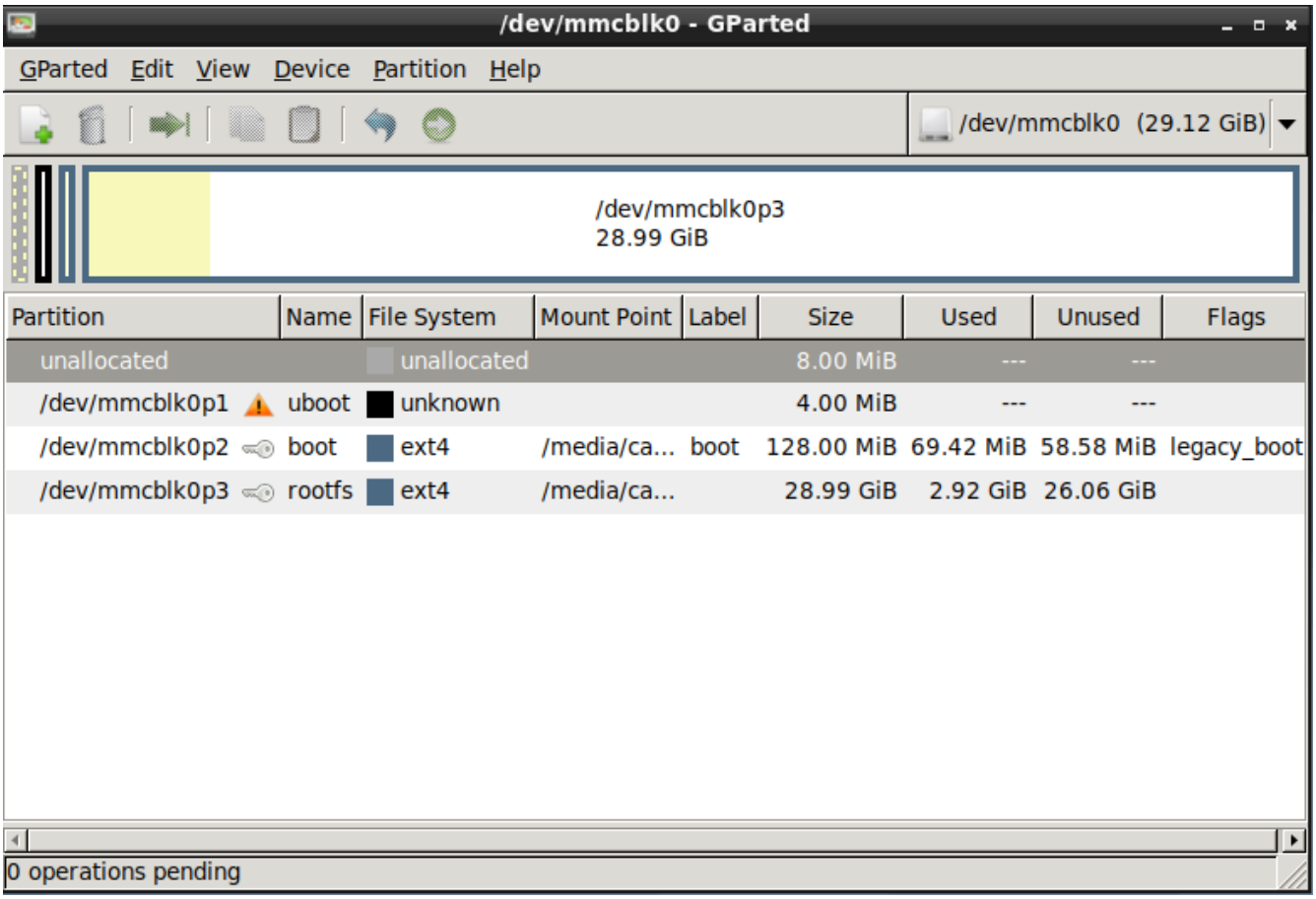
后面的步骤就和使用 Linux PC 备份 SD 卡的流程一致了，只是有部分细节需要注意。

首先要获取 eMMC 的设备号，查看/dev 目录下 mmc 设备，带有 mmcblkx[boot]x 的就是 eMMC，这里就是/dev/mmcblk0。

```
[root@RK356X:/]# ls -l /dev/mmc*
brw-rw----    1 root     disk      179,   0 Aug  4 09:00 /dev/mmcblk0
brw-rw----    1 root     disk      179,  32 Aug  4 09:00 /dev/mmcblk0boot0
brw-rw----    1 root     disk      179,  64 Aug  4 09:00 /dev/mmcblk0boot1
brw-rw----    1 root     disk      179,   1 Aug  4 09:00 /dev/mmcblk0p1
brw-rw----    1 root     disk      179,   2 Aug  4 09:00 /dev/mmcblk0p2
brw-rw----    1 root     disk      179,   3 Aug  4 09:00 /dev/mmcblk0p3
crw-------    1 root     root      238,   0 Aug  4 09:00 /dev/mmcblk0rpmb
brw-rw----    1 root     disk      179,  96 Aug  4 09:00 /dev/mmcblk1
brw-rw----    1 root     disk      179,  97 Aug  4 09:00 /dev/mmcblk1p1
brw-rw----    1 root     disk      179,  98 Aug  4 09:00 /dev/mmcblk1p2
brw-rw----    1 root     disk      179,  99 Aug  4 09:00 /dev/mmcblk1p3
brw-rw----    1 root     disk      179, 100 Aug  4 09:00 /dev/mmcblk1p4
brw-rw----    1 root     disk      179, 101 Aug  4 09:00 /dev/mmcblk1p5
brw-rw----    1 root     disk      179, 102 Aug  4 09:00 /dev/mmcblk1p6
brw-rw----    1 root     disk      179, 103 Aug  4 09:00 /dev/mmcblk1p7
brw-rw----    1 root     disk      179, 104 Aug  4 09:00 /dev/mmcblk1p8
[root@RK356X:/]#
```

桌面打开或者终端执行以下命令打开 Gparted 磁盘管理工具，将磁盘切换到 eMMC

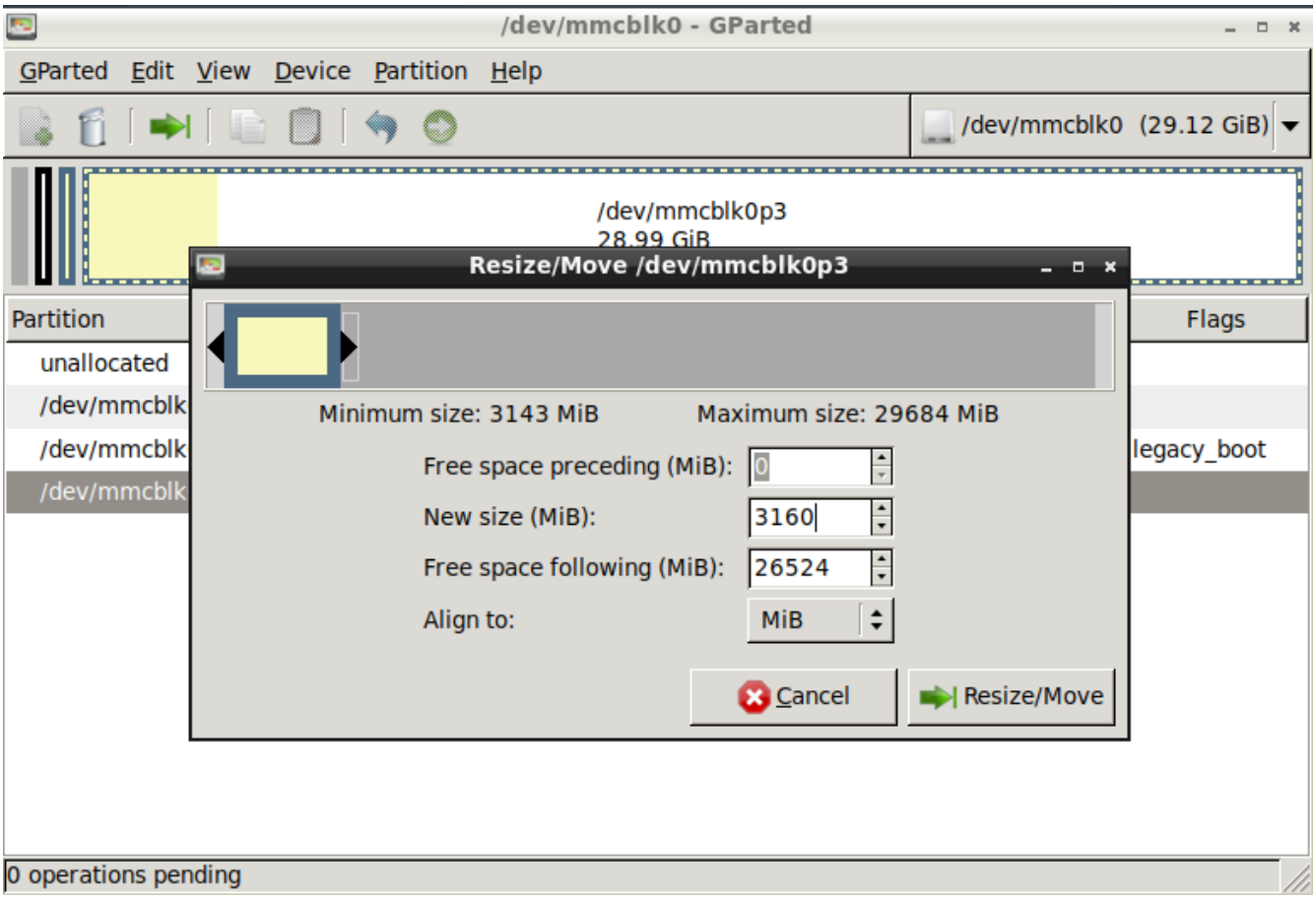
```
# 如果没有工具，可以以下命令下载工具
sudo apt install gparted

# 使用终端命令打开 Gparted 磁盘管理工具
sudo gparted
```



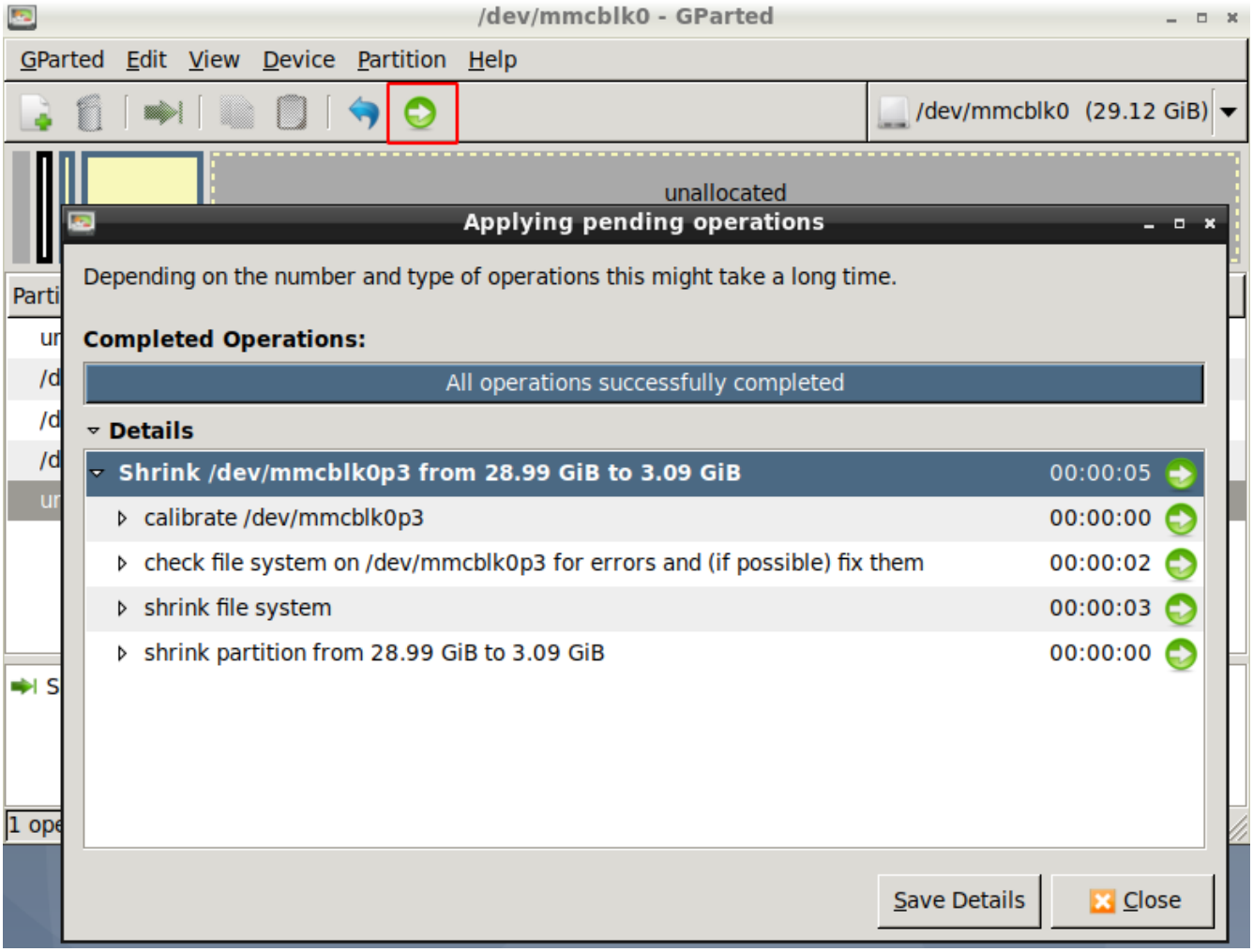
从图中可以看到 SD 卡中共 3 个已分配的分区，将要备份的内容是最后一个分区及之前的空间。

我们发现最后一个分区，也就是 rootfs 分区的总空间为 28.99GiB，但是已用空间只有 2.92GiB，我们将 rootfs 分区的大小进行压缩，就可以减小备份镜像的大小。

调整最后一个分区，也就是 rootfs 分区的大小。从图中可以看到，最小的空间是 3143MiB，我们调整到 3160MiB。



调整完成后点击绿色按钮应用修改



25.4.2 扩容和网口随机 mac

由于使用 dd 完整备份 emmc 里面的系统，网口 mac 是随机生成并固定在 uboot 环境变量的，且不能覆盖，但如果不修改 mac 会造成新烧录的一块或多块板的网口 mac 地址相同，造成网络无法通信，但是我们可以使用脚本重新随机或自定义网口 mac。

另外，希望备份的镜像可以在烧录启动时自动扩展最后一个分区，可以按以下步骤进行操作。

注解： 以下命令须使用管理员权限操作，如未使用 root 用户操作，需要在命令前加 sudo

通用镜像和专用镜像的方法不同，分别对两种镜像进行讲解。

- **通用镜像**执行以下操作：

插入烧录好系统的 SD 卡，以 SD 卡启动系统，进入系统后执行以下操作：

```
# 创建目录挂载 U 盘
mkdir -p /media/emmc

# 挂载根文件系统分区
mount /dev/mmcblk0p3 /media/emmc/

# 打开系统初始化脚本
vim /media/emmc/etc/init.d/boot_init.sh
```

在系统初始化脚本 boot_init.sh 末尾添加以下内容，boot_init.sh 脚本是初始化脚本，默认配置为了开机自启动。

```
#-------扩容------
# 判断/boot/boot_dilatation_init 文件是否存在, 不存在则进行扩容
if [ ! -e "/boot/boot_dilatation_init" ] ;
   then

   # 修改/dev/mmcblk0p3 根文件系统分区空间配置
   printf 'yes\n-1\nyes' | parted /dev/mmcblk0 resizepart 3 ---pretend-
↪input-tty

   # 根据配置重新分配空间
   resize2fs /dev/mmcblk0p3

   # 创建判断文件, 第二次启动存在该文件不再执行此扩容
   touch /boot/boot_dilatation_init
fi

#------随机网口 mac------
```

(下页继续)

(续上页)

```
#-------------------------------------------
# 遍历 eth0 到 eth3
for i in {0..3}; do
    iface="eth$i"

    # 检查接口是否存在
    if ifconfig "$iface" &> /dev/null; then

        ip link set dev "$iface" down

        # 判断/boot/boot_mac_ethX, 不存在则进行随机 mac
        if [ ! -e "/boot/boot_mac_$iface" ] ;
        then
                # 生成随机 MAC 地址
                new_mac=$(printf '%01x2:%02x:%02x:%02x:%02x:%02x' $((RANDOM%16)) $((RANDOM%256)) $((RANDOM%256)) $((RANDOM%256)) $((RANDOM%256)) $((RANDOM%256)))

                # 创建判断文件, 第二次启动存在该文件则不再执行随机 mac
                touch /boot/boot_mac_$iface

                # 保存随机生成的 mac 到判断文件中
                echo "$new_mac" > /boot/boot_mac_$iface
        fi

        # 获取生成的 mac 并修改
        mac_address=$(cat /boot/boot_mac_$iface)
        ifconfig $iface hw ether $mac_address

        ip link set dev "$iface" up
    fi
done
```

提示： 以上扩容和随机网口 Mac 都会在/boot/下生成一个判断文件，如果存在则表示已经扩容和随机 Mac，下次启动不会重复执行，如果删除对应的文件则会重新执行。

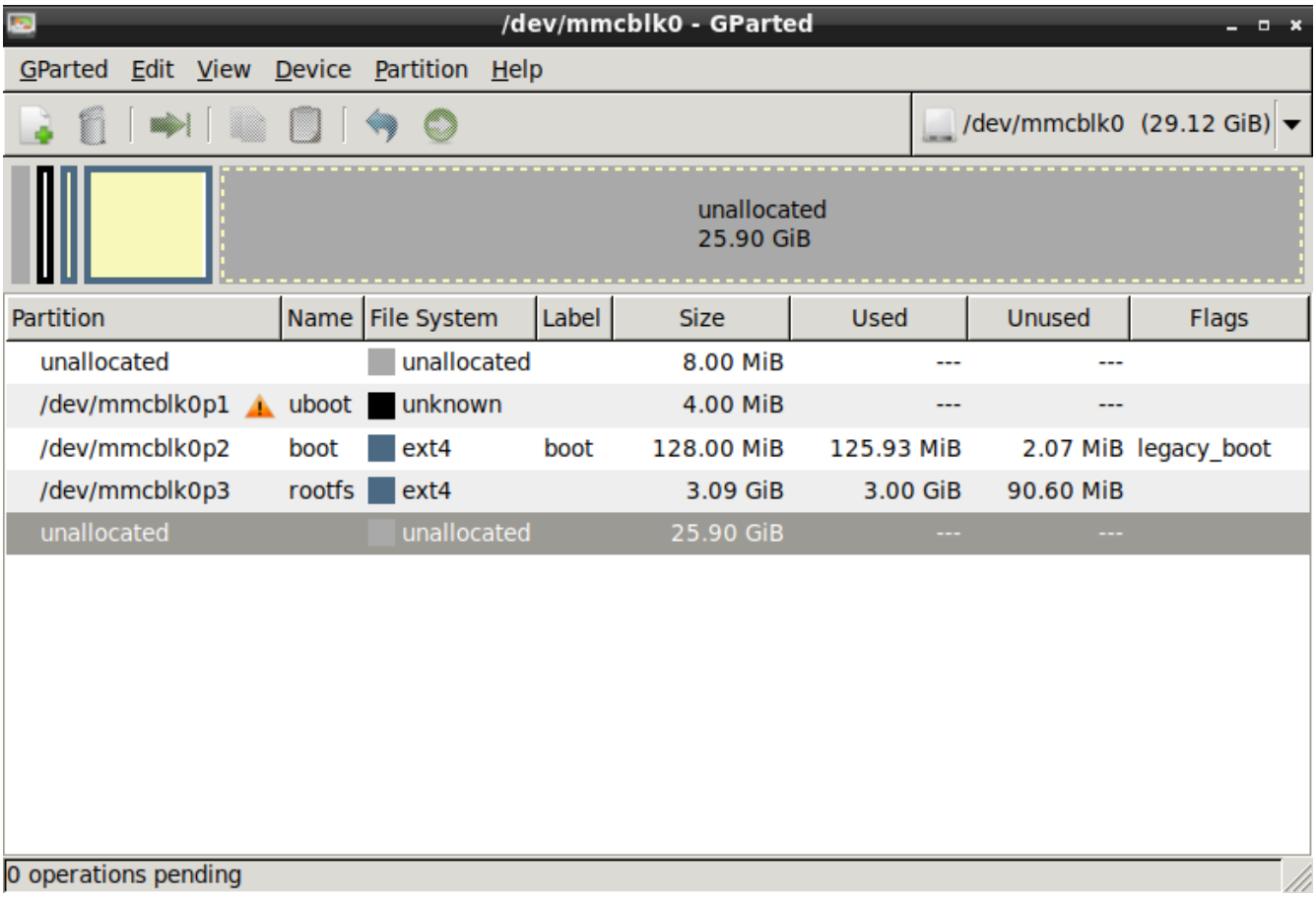
修改完成后卸载挂载的分区

```
# 修改完成后，卸载已挂载的分区
umount /media/emmc/
```

提示： 如果需要二次备份，请删除以上的判断文件，否则二次备份后烧录到其他板，文件存在则不会进行扩容和随机 mac。

- **专用镜像**执行以下操作：

```
# 创建目录挂载 U 盘
mkdir -p /media/emmc

# 挂载根文件系统分区，根据实际修改
mount /dev/mmcblk0p3 /media/emmc/

# 删除此文件，使用备份镜像启动会自动扩展分区
rm /media/emmc/var/lib/misc/firstrun

# 随机 mac 参考通用镜像，创建自启动脚本并进行随机。

# 删除完成后，卸载已挂载的分区
umount /media/emmc/
```



手动计算一下需要备份的大小 (8+4+128+3.09x1024)MiB≈3305MiB，由于显示的大小通过四舍五入的方式保留 2 位小数，所以我们可以加一点余量，备份 3350MiB 的大小。

使用 `mkdir` 命令创建一个目录用于挂载 U 盘，然后使用 `mount` 命令挂载 U 盘，最后使用 `dd` 命令将镜像备份到挂载的 U 盘中。

注意： fat32 格式的 U 盘单文件最大只能 4G，因此如果备份系统大于 4G，U 盘请不要使用 fat32 格式，可使用 exFAT 或者 NTFS 格式。

```
1 # 创建目录挂载 U 盘
2 mkdir -p /media/usb
3
```

(下页继续)

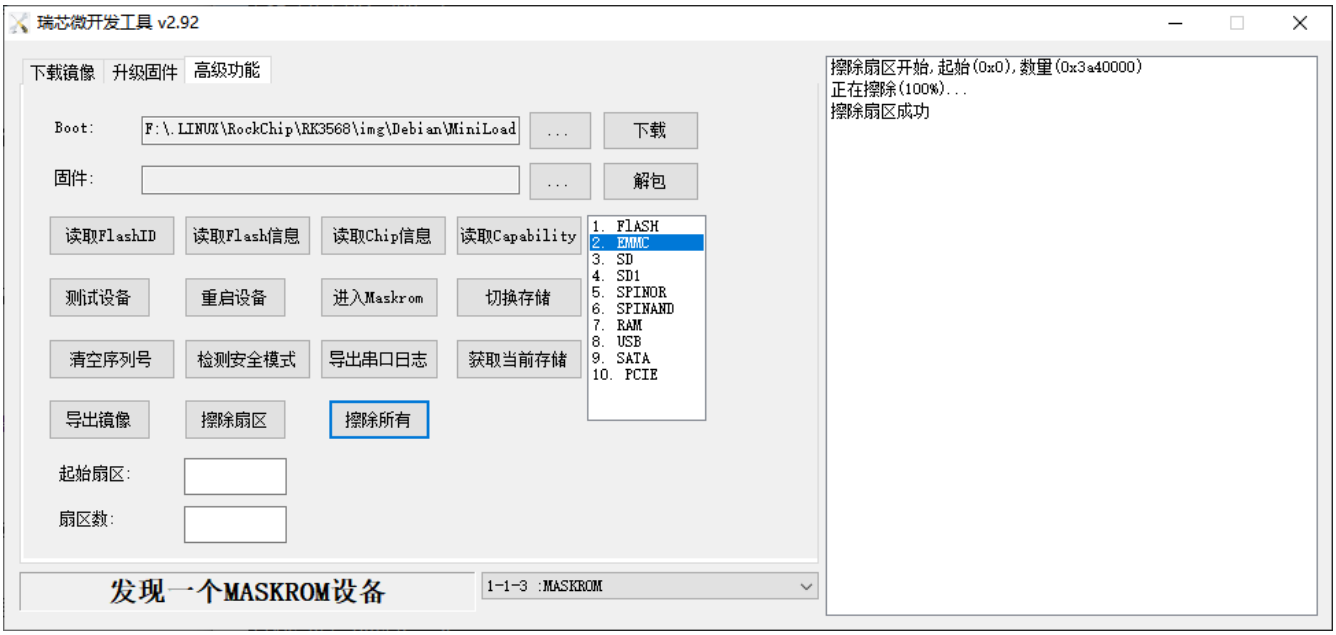
(续上页)

```
# 挂载 U 盘
mount /dev/sda1 /media/usb/

# 将镜像备份到挂载 U 盘中
dd if=/dev/mmcblk0 of=/media/usb/backup.img count=3350 bs=1024k conv=sync␣
↪status=progress

# 拷贝需要时间，请耐心等待，打印消息如下
3350+0 records in
3350+0 records out
3512729600 bytes (3.5 GB, 3.3 GiB) copied, 198.883 s, 17.7 MB/s

# 备份完成后，卸载 U 盘
umount /media/usb/
```

等待 dd 命令运行完成后，就得到了 RAW 格式的 backup.img 镜像，为了确保 U 盘中的镜像写入完整，不要直接拔出 U 盘，使用 umount 命令卸载完成后再拔出 U 盘

25.5 使用 RKDevTool 烧录 RAW 格式镜像到 eMMC

为了验证系统镜像确实被写入了 eMMC，我们先删除原来的镜像。使用 RKDevTool 高级功能中的 **擦除所有**来清空 eMMC。

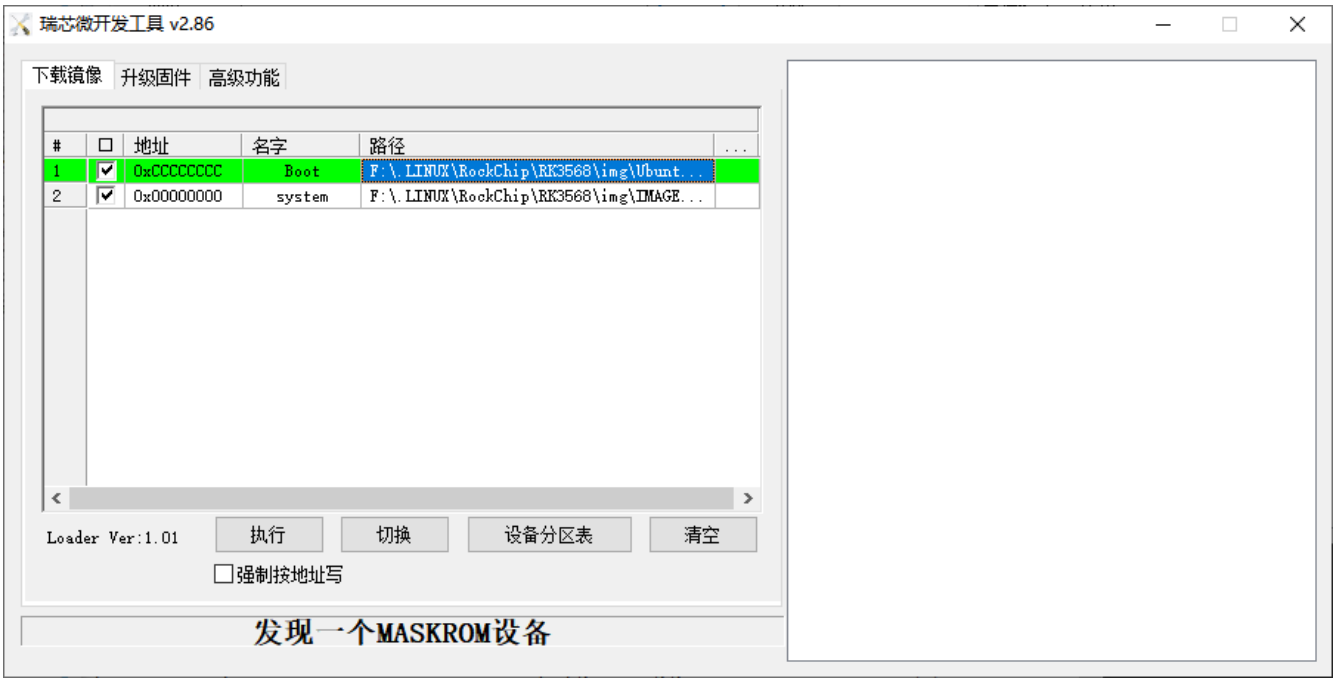


注解： RKDevTool 高级功能中的擦除所有功能虽然无法清除 Loader 文件，但是可以擦除其他分区。

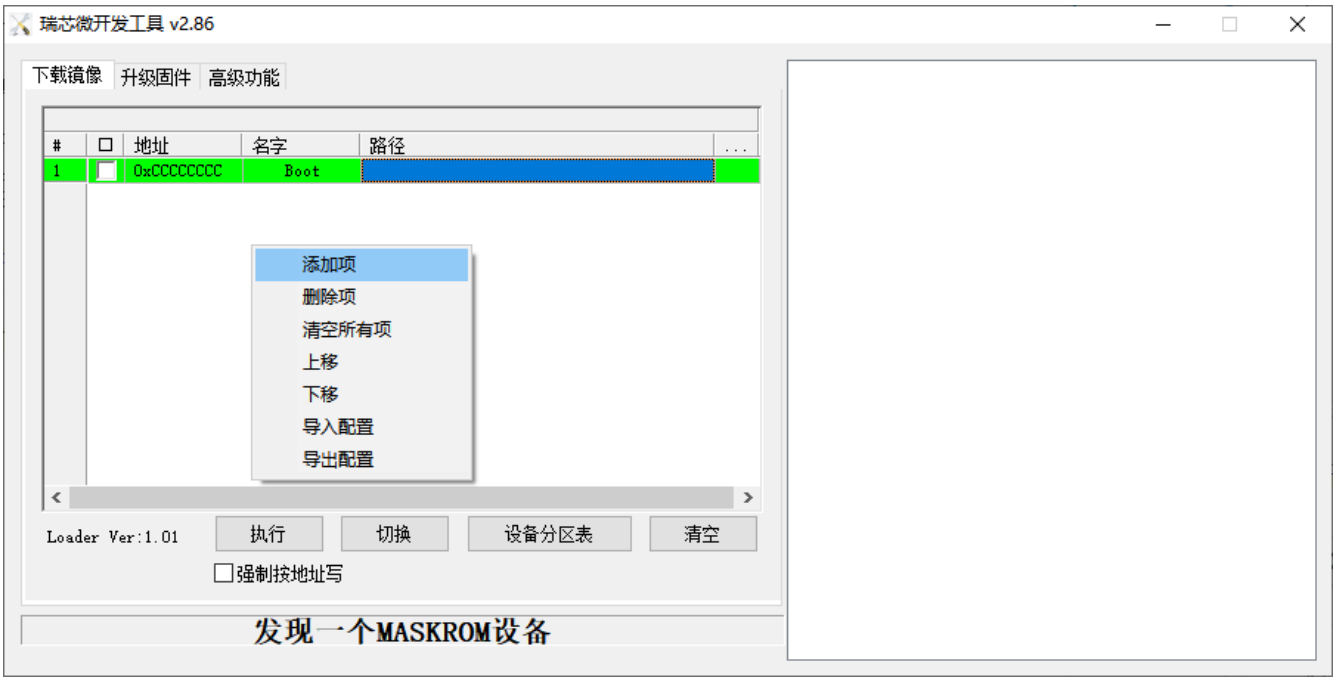
接下来使用 RKDevTool 向 eMMC 中烧录 RAW 格式的镜像。

要想烧录 RAW 格式镜像，需要使用 RKDevTool_Release_v2.86 的版本，使用最新版的烧录工具会失败。

打开 RKDevTool_Release_v2.86，默认配置文件就是用来烧录 RAW 格式的，如果与下图红框的内容不一致，可以在 **地址** 下方的空白区域点击鼠标右键导入配置，选择和 RKDevTool_Release_v2.86 同一文件夹的 RAW.cfg。



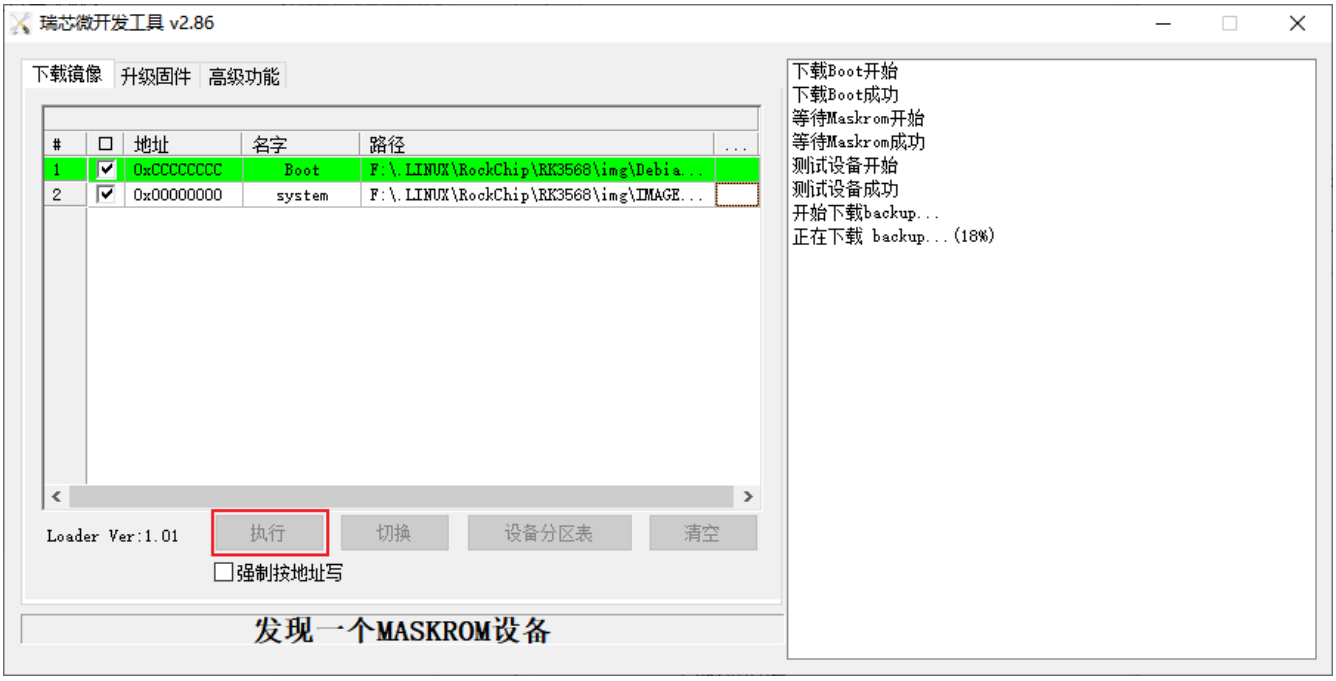
也可以按照图片中的地址和名字，手动点击 **地址**和 **名字**进行修改，还可以右键点击空白处 **添加项**



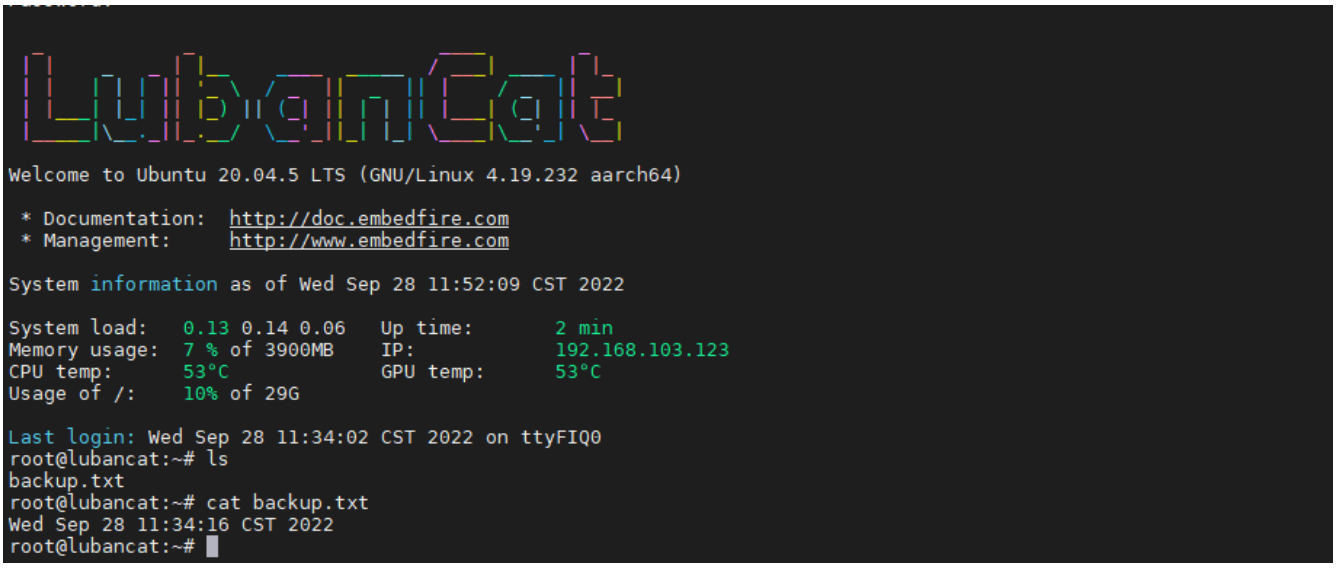
上述工作准备完成后，点击路径后的 …添加对应的文件，Boot 是指 Loader 文件，也就是芯片型号对应的 MiniLoaderAll.bin，system 对应的就是 RAW 格式的镜像，可以是备份的 RAW 格式的镜

像，也可以是 OpenWrt 的 RAW 格式镜像。

这里以备份镜像为例，选择完镜像后点击执行，开始下载。



下载完成后启动板卡，看一下备份之前 eMMC 之前创建的文件。



25.6 使用 dd 命令烧录 RAW 格式镜像到 eMMC

除了使用 RKDevTool 工具烧录 eMMC 之外，还可以通过 dd 命令将 RAW 格式的 backup.img 镜像写入 eMMC 中。

使用 dd 命令对 eMMC 烧录 RAW 格式镜像和使用 dd 命令备份 eMMC 的过程类似，备份是将 eMMC 中的内容以 RAW 格式写入 backup.img 镜像，而烧录过程就是将 RAW 格式的 backup.img 镜像写入 eMMC 中。

为了验证系统镜像确实被写入了 eMMC，我们先删除原来的镜像。使用 dd 命令将 eMMC 的前 16M 写空内容，破坏原有的分区表就可以了。

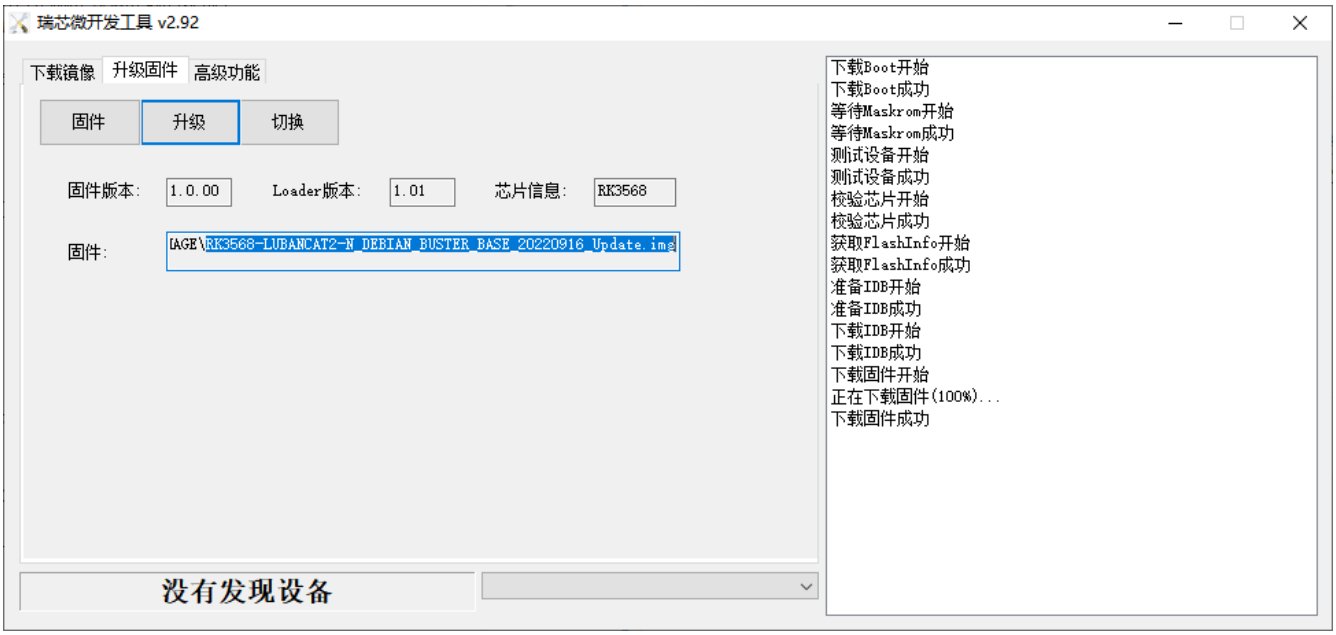
```
1 # 将 eMMC 前 16M 写空内容
2 dd if=/dev/zero of=/dev/mmcblk0 bs=1M count=16
```

移除 SD 卡，重新上电，直接进入了 Maskrom 模式.

我们需要准备一张 SD 卡，并烧录用于备份的板卡通用备份镜像 RK356X-LUBANCAT-BACKUP_xxx_Update.img，xxx 为更新日期

还需要一个存储设备，用于存放要烧录的 RAW 格式镜像，就使用刚刚用于备份的 U 盘。

为了验证通过 dd 命令烧录的镜像就是我们备份的镜像，先烧录 Debian 镜像覆盖 eMMC 中的内容。



插入 SD 卡，从 SD 卡启动操作系统。由于系统启动时会自动挂载分区，我们先要手都卸载/media/cat/目录下的所有分区，然后重新挂载。

```
# 卸载/media/cat/目录下挂载的所有分区
umount /media/cat/*

# 挂载 U 盘到/mnt 目录
mount /dev/sda1 /mnt

# 将 backup.img 镜像 dd 烧录到 emmc 中
dd if=/mnt/backup.img of=/dev/mmcblk0 bs=1024k conv=sync

# 拷贝需要时间，请耐心等待，打印消息如下
3350+0 records in
3350+0 records out
3512729600 bytes (3.5 GB, 3.3 GiB) copied, 133.043 s, 26.4 MB/s

# 卸载 U 盘
umount /mnt/

```

(下页继续)

(续上页)

```
# 将内存中的缓存写入存储设备
sync

# 关机
poweroff
```

移除 SD 卡，重新上电启动

```
Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0

[username:password] root:root cat:temppwd

Modify information : /etc/issue

lubancat login: [   11.360602] rk-pcie 3c0000000.pcie: PCIe Link Fail
[   11.360644] rk-pcie 3c0000000.pcie: failed to initialize host

lubancat login: root
Password:


Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

 * Documentation:  http://doc.embedfire.com
 * Management:     http://www.embedfire.com

System information as of Wed Aug 31 23:27:58 CST 2022

System load:   2.44 0.64 0.22   Up time:       0 min
Memory usage:  8 % of 3900MB    IP:           192.168.103.123
CPU temp:      53°C             GPU temp:     52°C
Usage of /:    9% of 29G

Last login: Wed Sep 28 11:34:02 CST 2022 on ttyFIQ0
root@lubancat:~# ls
backup.txt
root@lubancat:~# cat backup.txt
Wed Sep 28 11:34:16 CST 2022
root@lubancat:~#
```

第 26 章　根文件系统备份与重新烧录

对系统镜像镜像完整备份的好处是处理方便，步骤简单。但是完整备份的系统镜像体积较大且无法使用量产工具进行烧录。

如果我们仅对根文件系统做出了修改，则可以单独备份根文件系统分区。同理，如果我们对 boot 分区或其他分区做出了修改，也可以单独备份相应的分区。但是在开发过程中，我们一般不去修改别的分区，所以这里重点讲解 rootfs 分区的备份，其他分区的备份及烧录方式也类似。

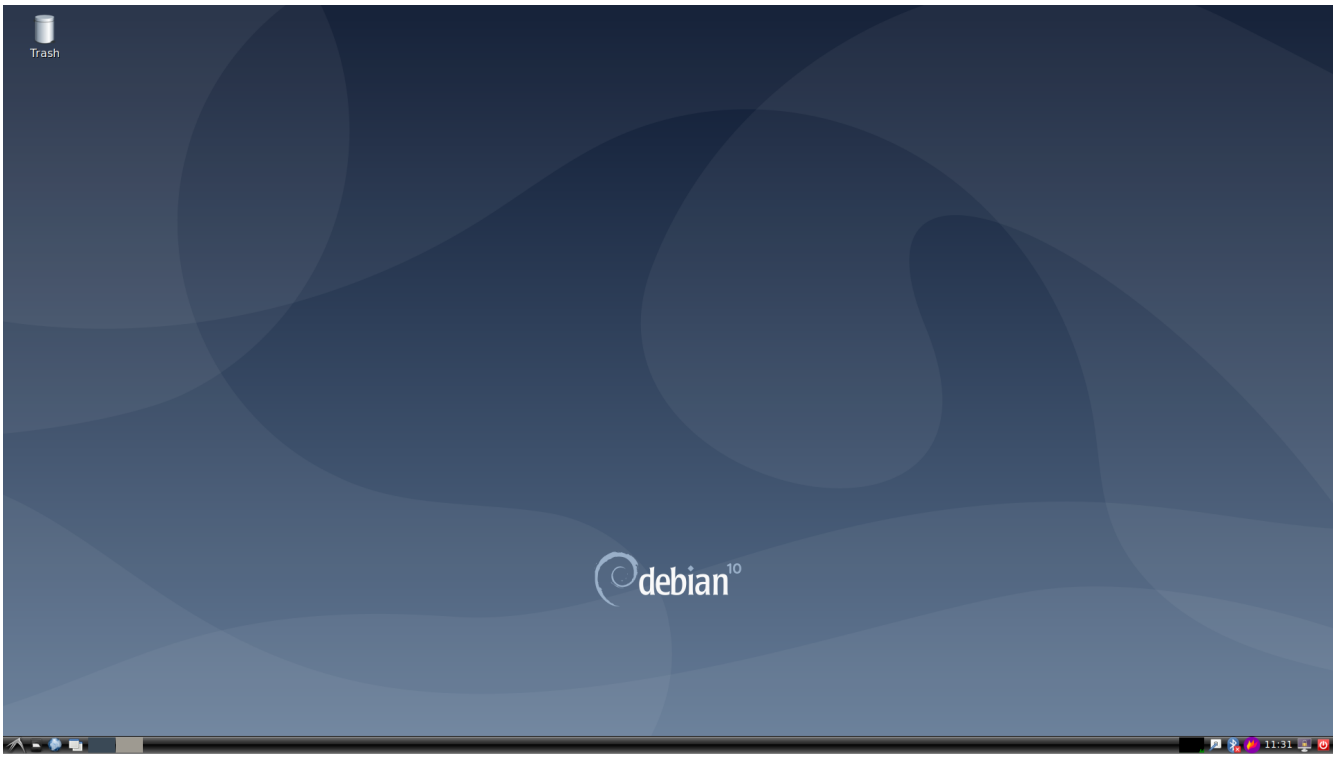
26.1 备份前的准备工作

为了顺利的进行根文件系统的备份，我们先要做一些准备工作。

- 将烧录到 eMMC 或 SD 卡的 RK 格式镜像源文件解包，得到各个分区的镜像。
- 用于后期镜像处理的 Linux PC 或虚拟机
- 如果备份的是 eMMC，则需要一张已经烧录了板卡通用备份镜像的 SD 卡。

26.2 根文件系统备份

为了验证 rootfs 分区是否被正确备份，我们将桌面背景图片换为下图所示的图片。



接着操作以下步骤，再次启动时会自动扩展分区。

- **专用镜像**执行以下操作：

```
# 删除此文件，再次启动时会自动扩展分区
rm /var/lib/misc/firstrun

# 删除完成后，关机
poweroff

通用镜像和专用镜像的扩容方法不同，分别对两种镜像进行讲解。
```

- **通用镜像**执行以下操作：

打开系统初始化脚本/etc/init.d/boot_init.sh，该脚本是初始化脚本，默认配置为了开机自启动。

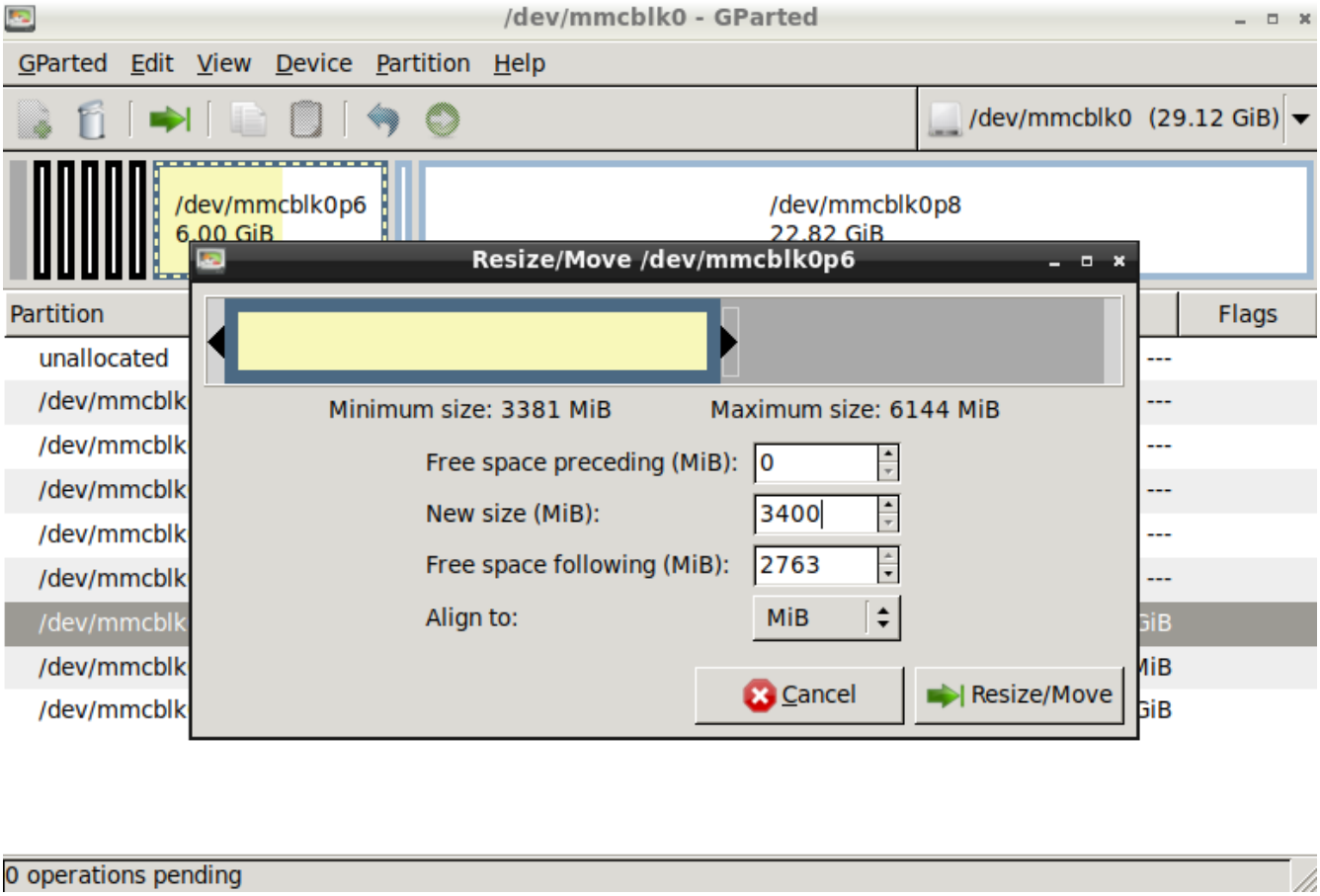
```
vim /etc/init.d/boot_init.sh
```

假定通用镜像的根文件系统分区为/dev/mmcblk0p3，可执行命令 df -h 进行确认，在系统初始化脚本末尾添加以下内容：

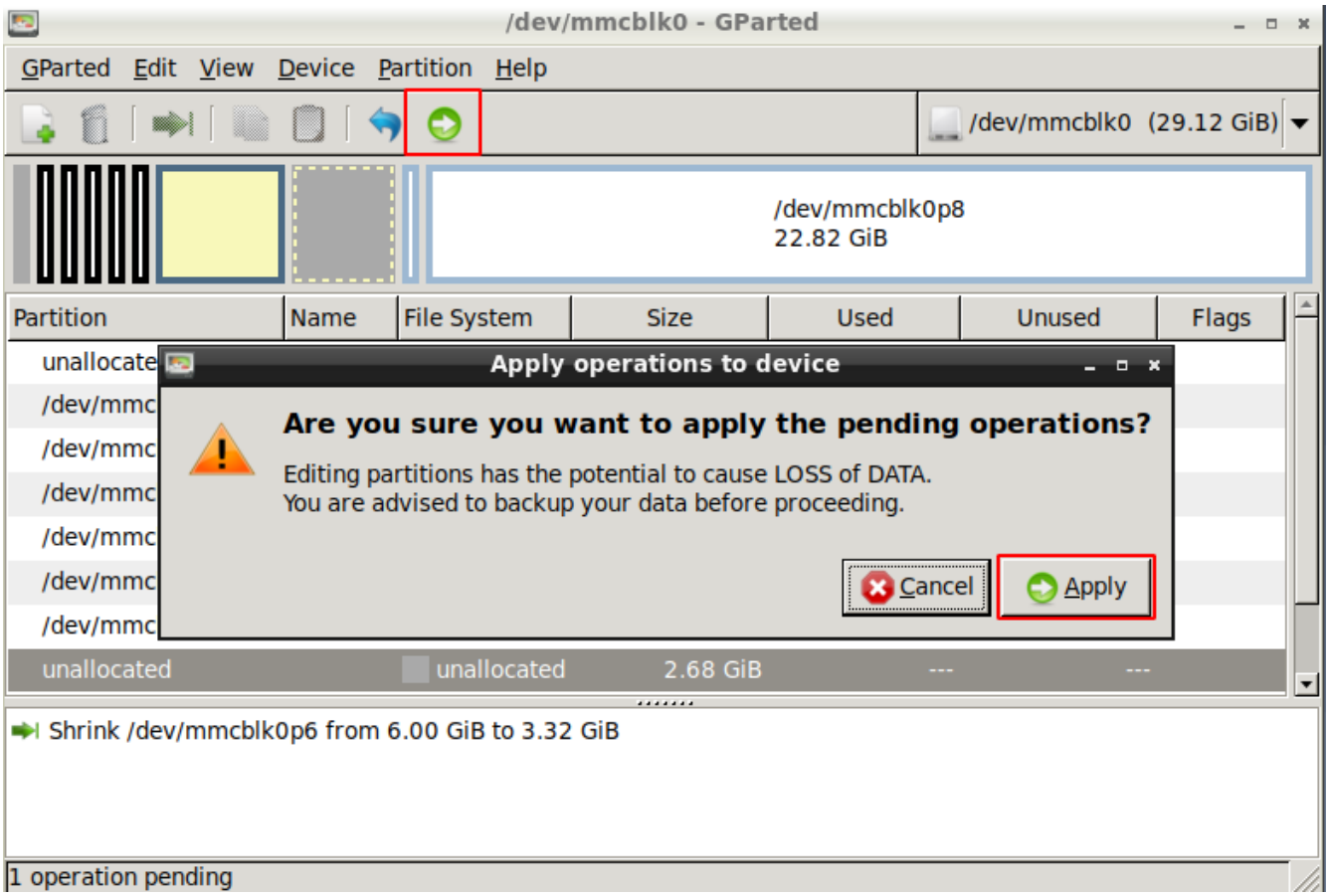
```
if [ ! -e "/boot/boot_dilatation_init" ] ;
   then

   # 修改/dev/mmcblk0p3 根文件系统分区空间配置
   printf 'yes\n-1\nyes' | parted /dev/mmcblk0 resizepart 3 ---pretend-input-tty

   # 根据配置重新分配空间
   resize2fs /dev/mmcblk0p3

   # 创建判断文件，第二次启动不再执行此扩容
   touch /boot/boot_dilatation_init
fi
```

插入烧录了板卡通用备份镜像的 SD 卡，从 SD 卡启动系统。如果是备份 SD 卡上的 rootfs 分区的话，这一步就是将要备份的 SD 卡插入 Linux PC。

打开 gparted 磁盘管理工具，卸载并压缩要备份的 rootfs 分区。



应用我们做出的修改，缩小 rootfs 的体积



使用 mkdir 命令创建一个新的目录，用于存放从带镜像的 SD 卡中拷贝的镜像。然后使用 dd 命令将带镜像的 SD 中的镜像拷贝到新创建的目录中，这里我们选择挂载的 U 盘。

刚才在压缩分区时我们得知 eMMC 的 rootfs 分区就是/dev/mmcblk0p6 分区

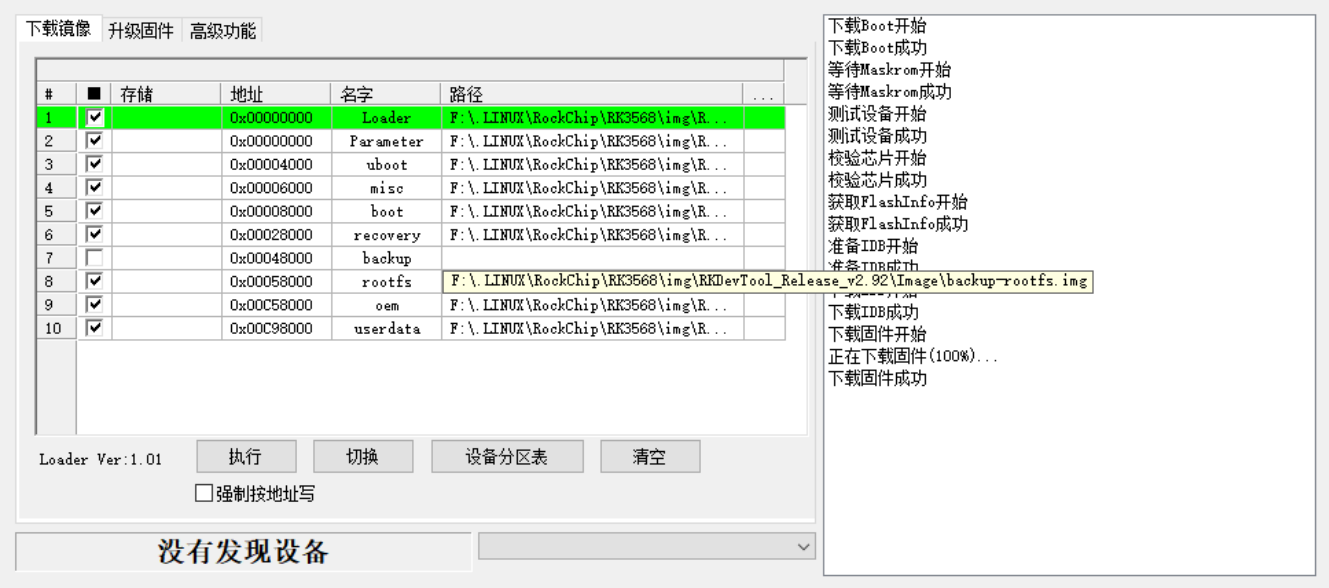
```
# 挂载 U 盘
mount /dev/sda1 /mnt/udisk/

# 将带镜像的 SD 卡中的镜像拷贝到创建的目录中
dd if=/dev/mmcblk0p6 of=/mnt/udisk/backup-rootfs.img bs=1024k conv=sync

# 拷贝需要时间，请耐心等待，打印消息如下
3400+0 records in
3400+0 records out
3565158400 bytes (3.6 GB, 3.3 GiB) copied, 185.89 s, 19.2 MB/s
```

备份完成后我们 poweroff 命令关机，移除 U 盘，这样我们就得到了单独的 rootfs.img 分区镜像。

我们可以选择直接用开发工具进行分区烧录，也可以将备份的 rootfs.img 与其他分区合并成一个完整的镜像。

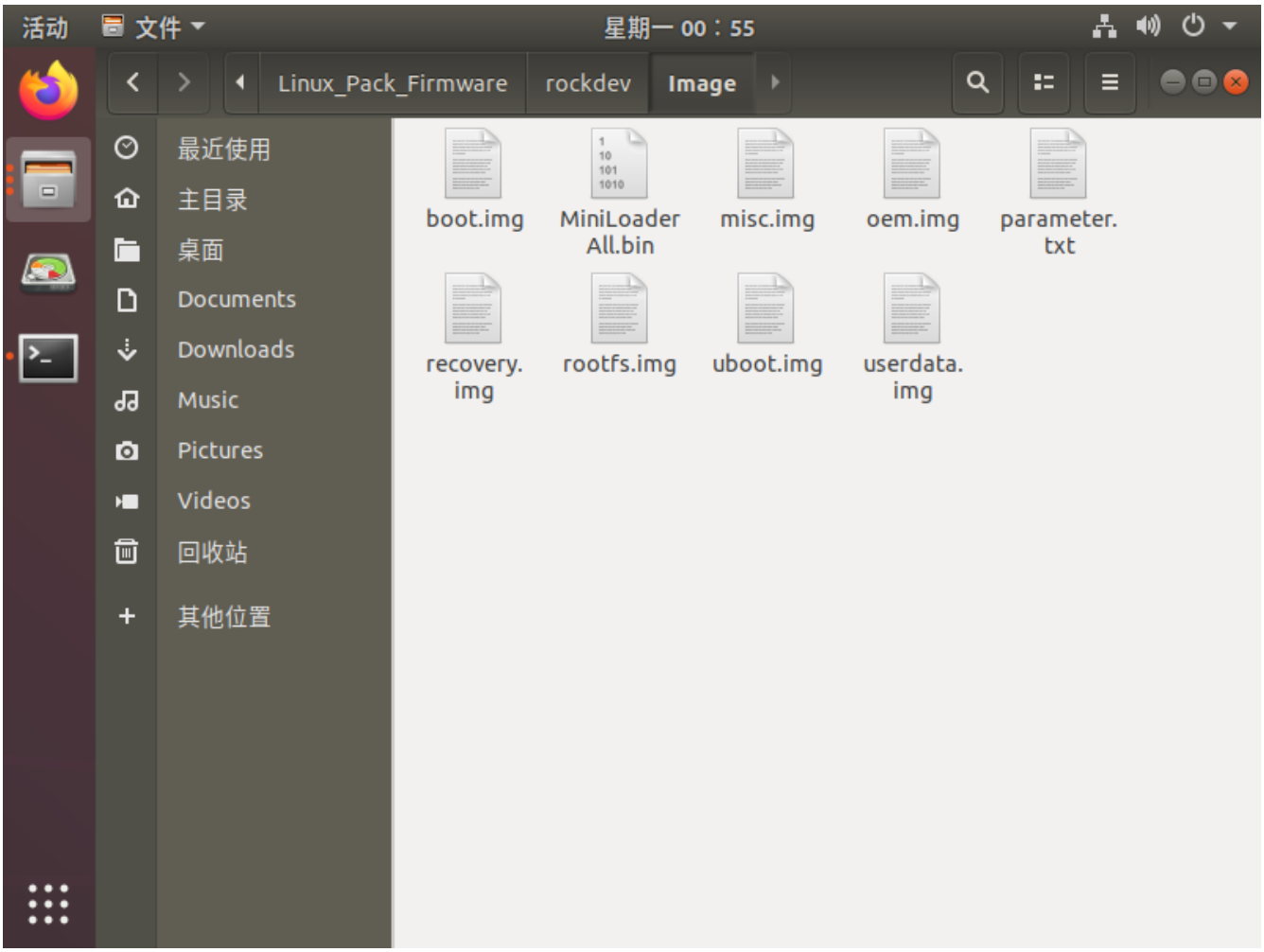
26.3 分区烧录

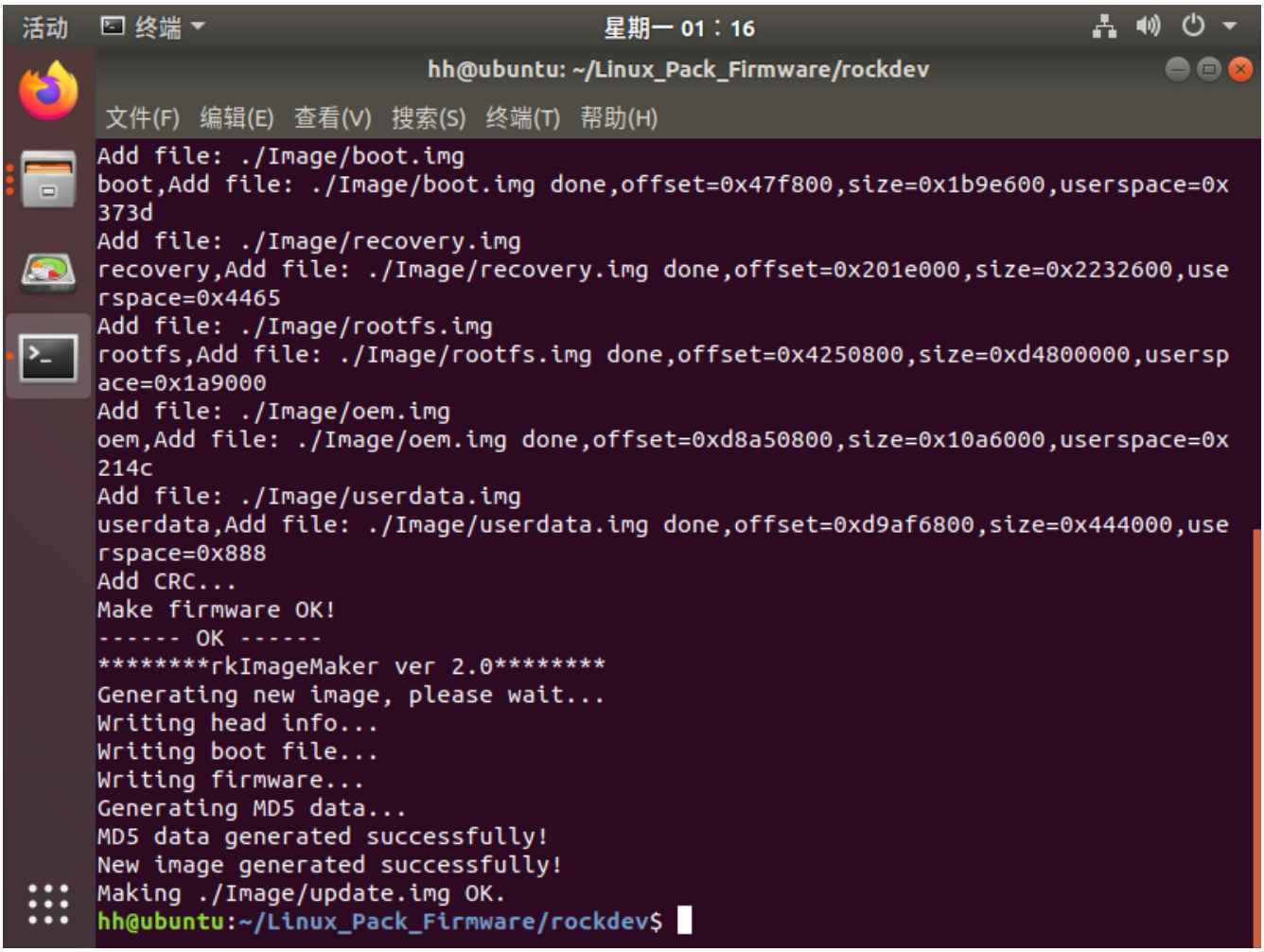
使用开发工具导入对应的分区表，然后依次选择 RK 格式完整镜像解包后的各个分区镜像，注意 rootfs 分区要选择我们刚刚备份得到的 backup-rootfs.img 镜像文件，然后进行烧录。



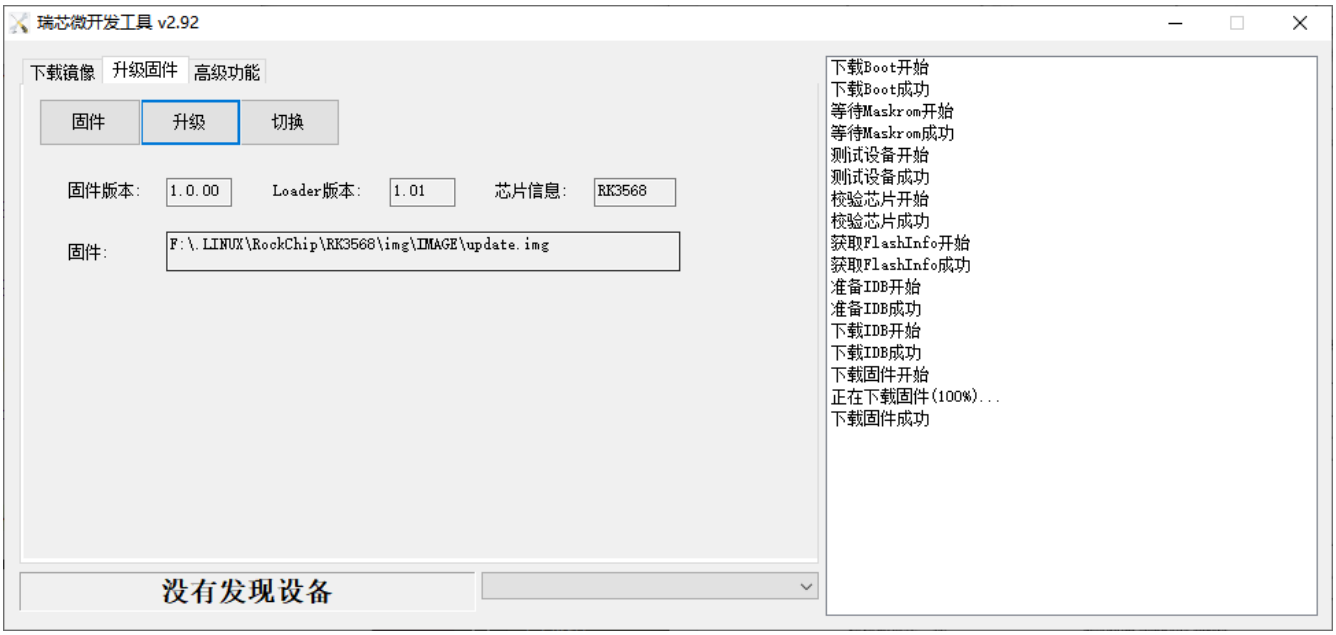
26.4 打包成完整镜像后烧录

打开我们的打包工具，将之前解包的分区镜像和刚刚备份的 backup-rootfs.img 复制到 Image 目录下，并将 backup-rootfs.img 重命名为 rootfs.img 替换原来的文件，然后运行 rk356x-mkupdate.sh 打包完整镜像。


活动
文件
星期一 00：55
Linux_Pack_Firmware
rockdev
Image
最近使用
主目录
桌面
Documents
Downloads
Music
Pictures
Videos
回收站
其他位置
boot.img
MiniLoaderAll.bin
misc.img
oem.img
parameter.txt
recovery.img
rootfs.img
uboot.img
userdata.img



将打包后的完整镜像 update.img 烧录到板卡中。



26.5 验证修改结果

重新烧录镜像后，默认桌面背景仍然是我们原来修改的图片。

第 27 章　修改 rootfs.img 镜像内部的文件

有时候我们需要修改 rootfs 里的内容，又不想从头构建根文件系统，就可以使用野火发布的镜像，直接修改 rootfs 分区中的文件。

这种方法只适用于修改量较小，且对根文件系统文件结构较为熟悉的开发人员使用。

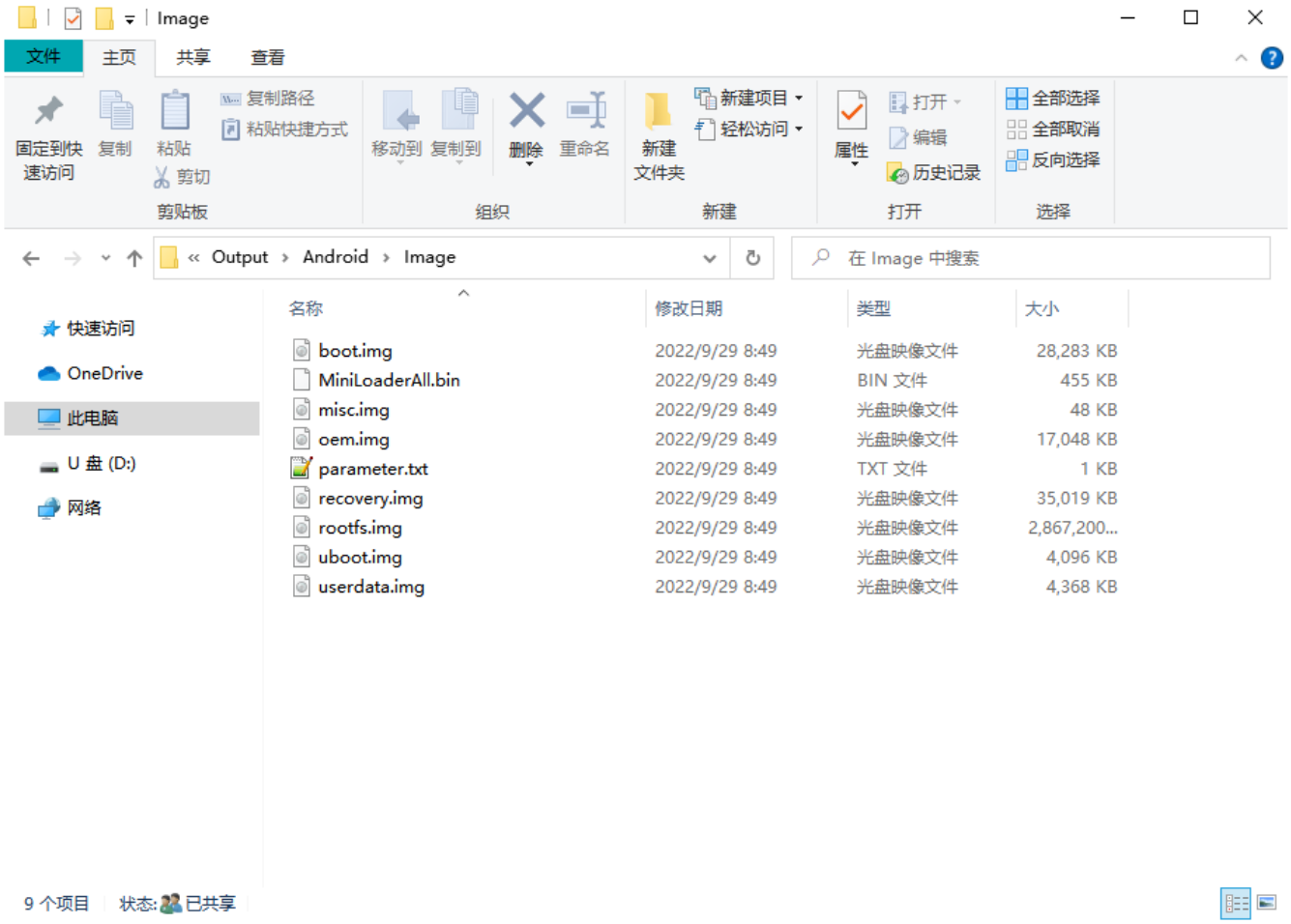
这里以 RK3568-LUBANCAT2-N_DEBIAN_BUSTER_DESKTOP_20220928_Update 镜像为例进行说明，具体步骤如下

注意：通用镜像必须在 Linux 下，使用 Linux_Pack_Firmware 工具进行打包、解包，否则打包出来的镜像无法正常使用。**专用镜像**在 windows 和 linux 下都可以正常打包、解包。

27.1 RKDevTool 解包、修改、打包 (Windows)

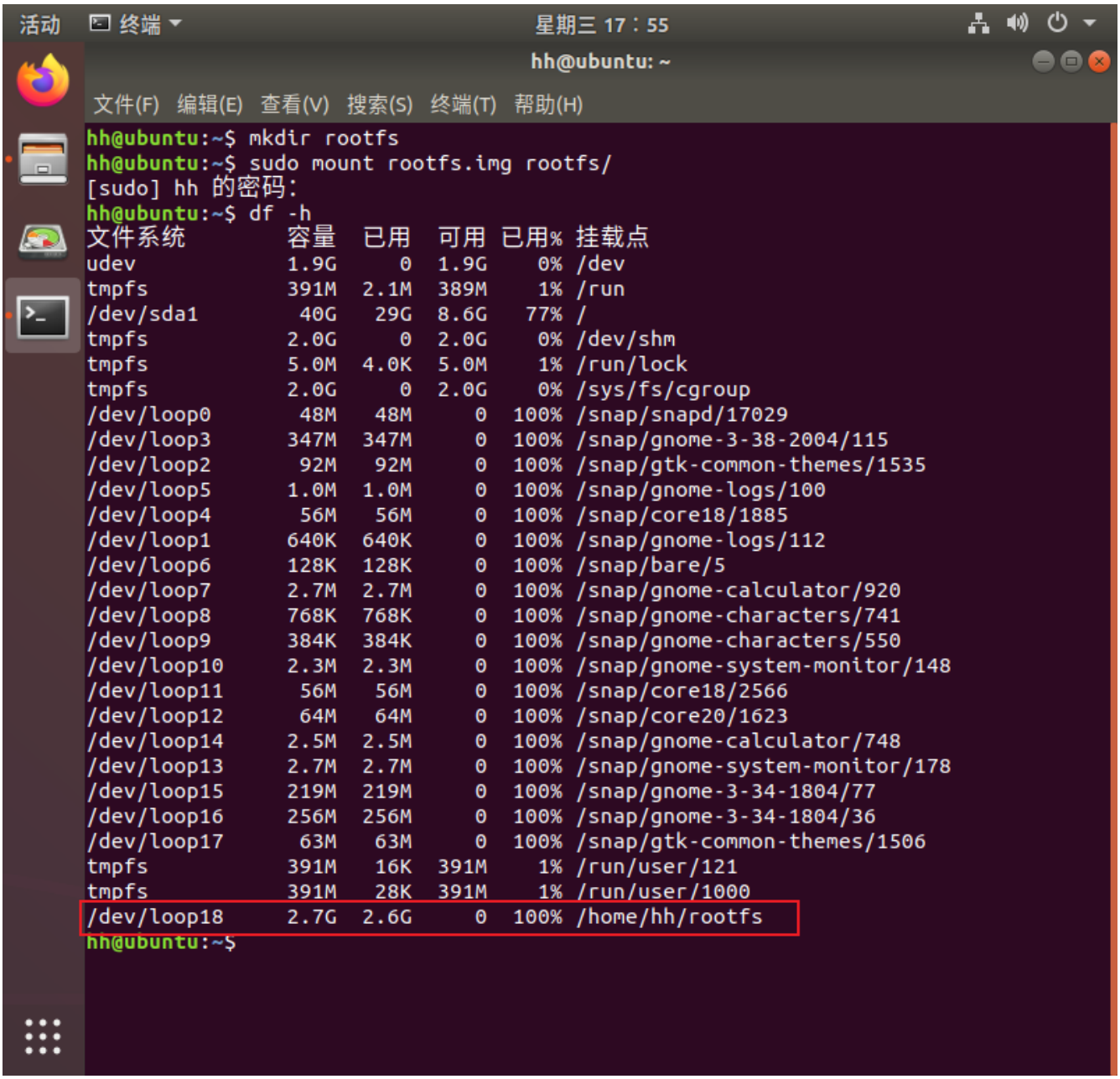
1. 使用 RKDevTool 将 update.img 完整镜像进行解包，得到单独的根文件系统镜像 rootfs.img

2. 解包得到的 rootfs.img 镜像保存在 RKDevTool_Release_v2.92\Output\Android\Image

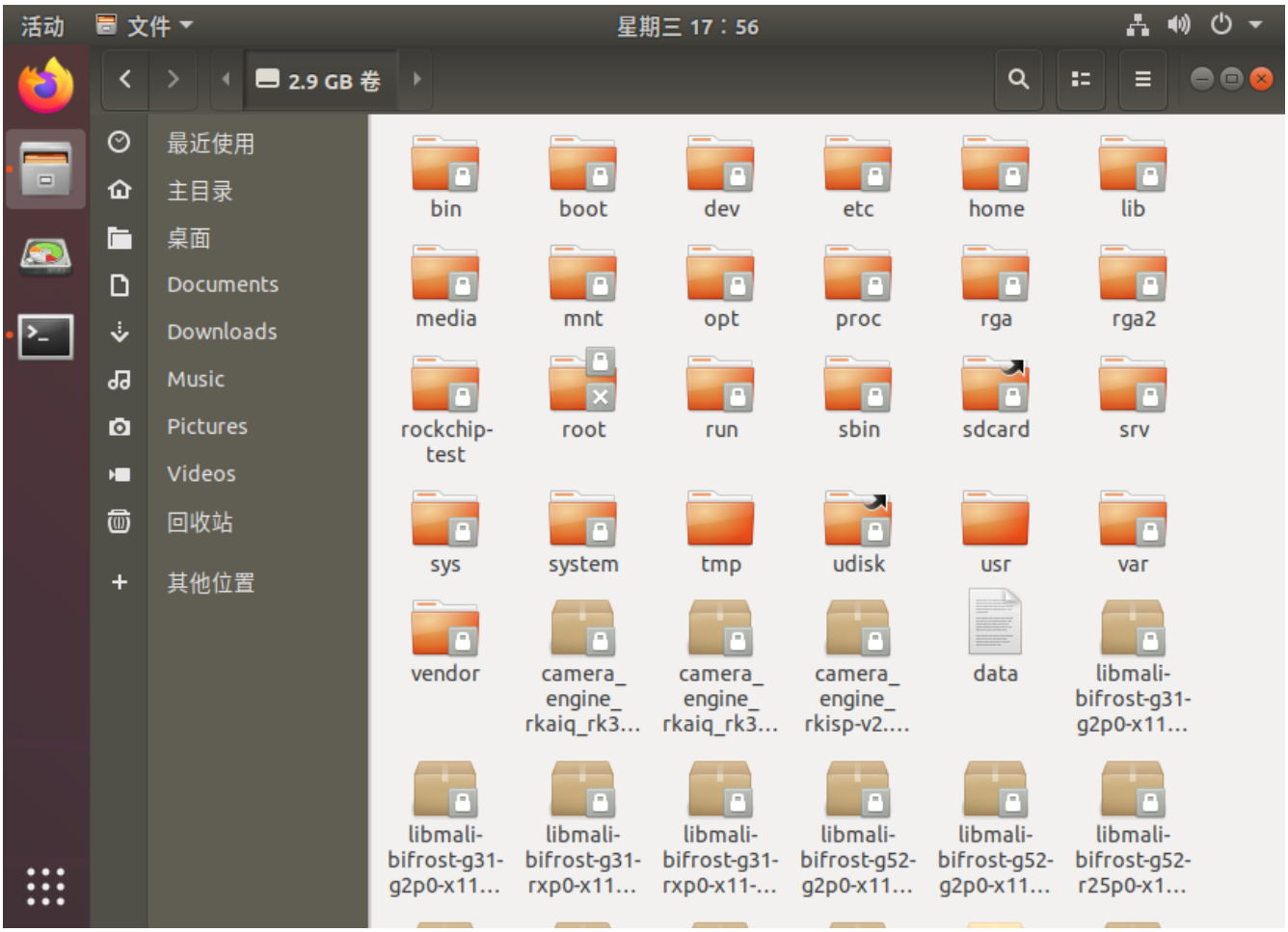


3. 将 rootfs.img 复制到 Linux PC 或虚拟机中，挂载镜像。

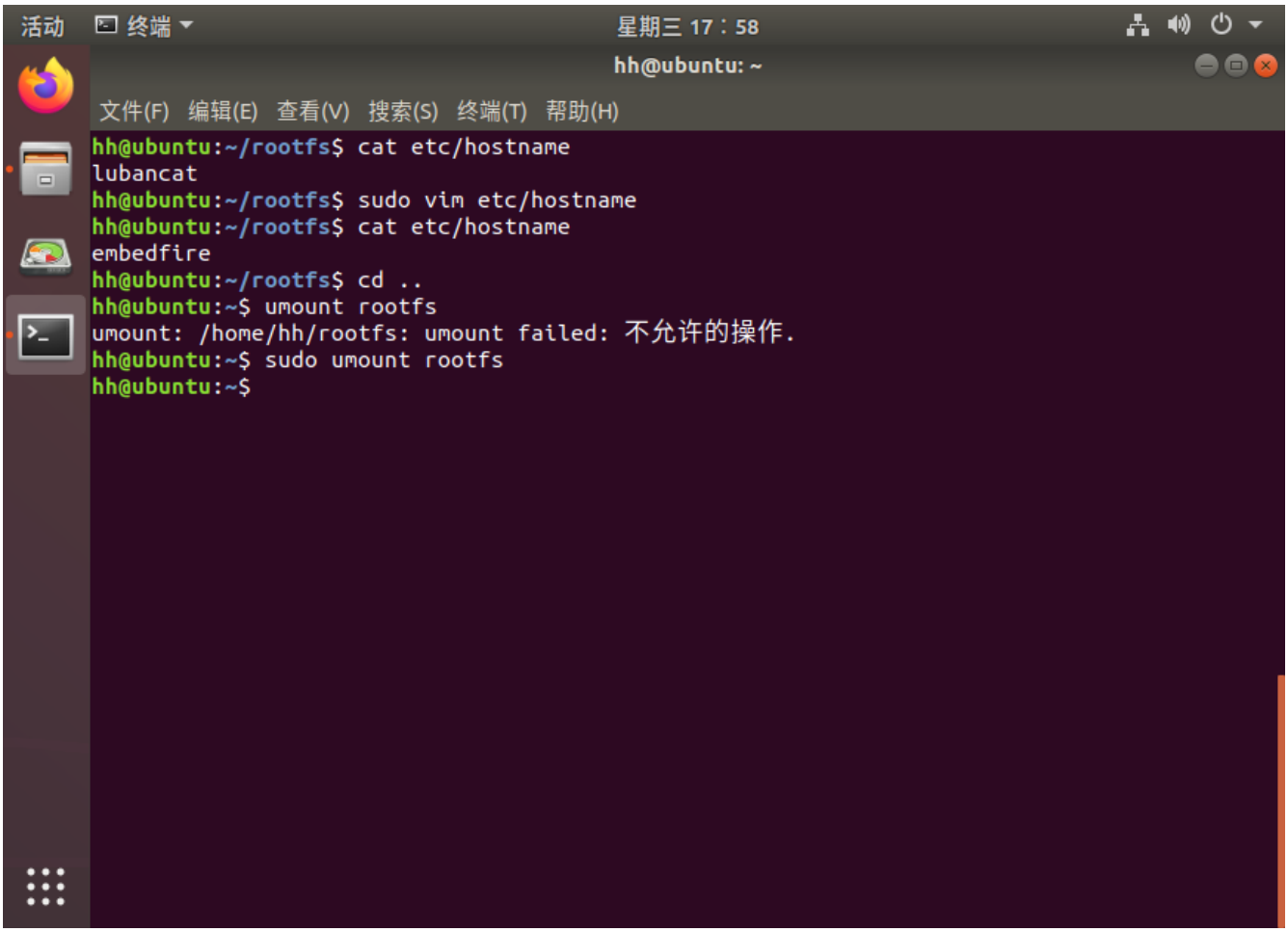
```
# 创建目录用来挂载 rootfs.img
mkdir rootfs

# 挂载
sudo mount rootfs.img rootfs/

# 查看是否挂载成功
df -h
```



4. 进入挂载目录修改文件



这里我们修改 etc/hostname 文件，将原本的 lubancat 修改为 embedfire。



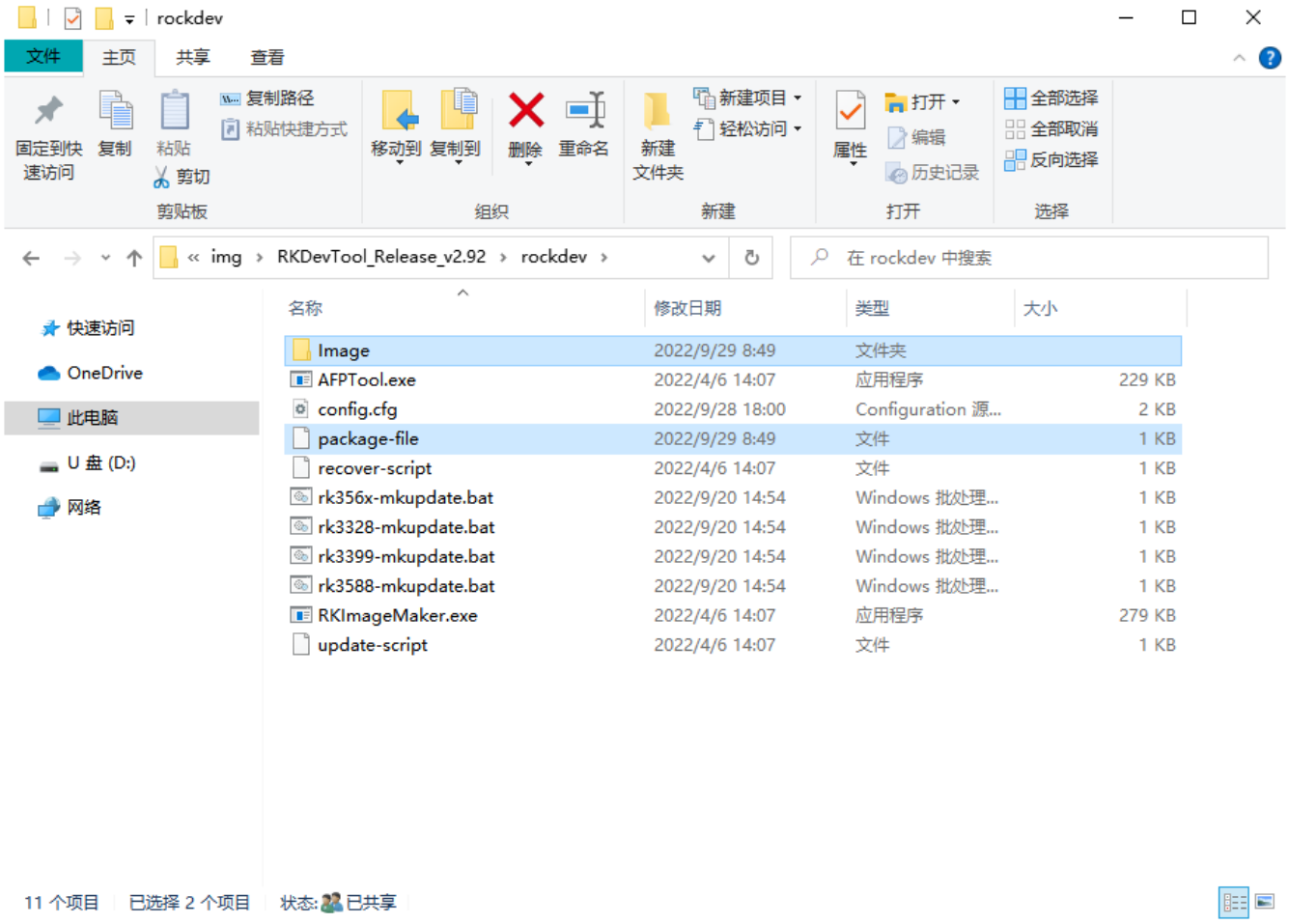
也可以根据需求修改其他的配置文件、新建或删除文件。

修改完成后我们取消挂载。

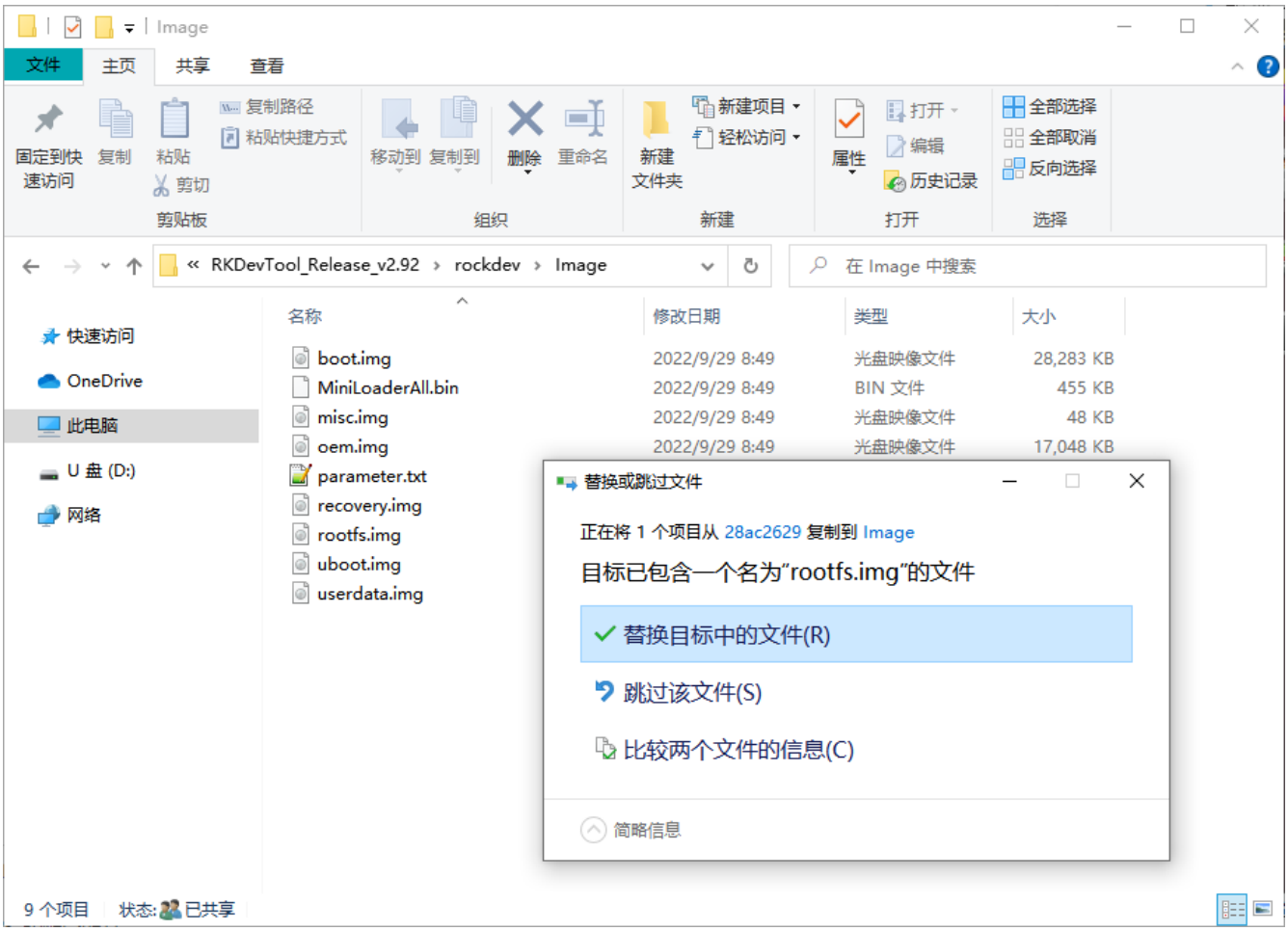
```
sudo umount rootfs/
```

1. 使用 RKDevTool_Release_v2.92 中的打包工具重新打包成完整镜像。

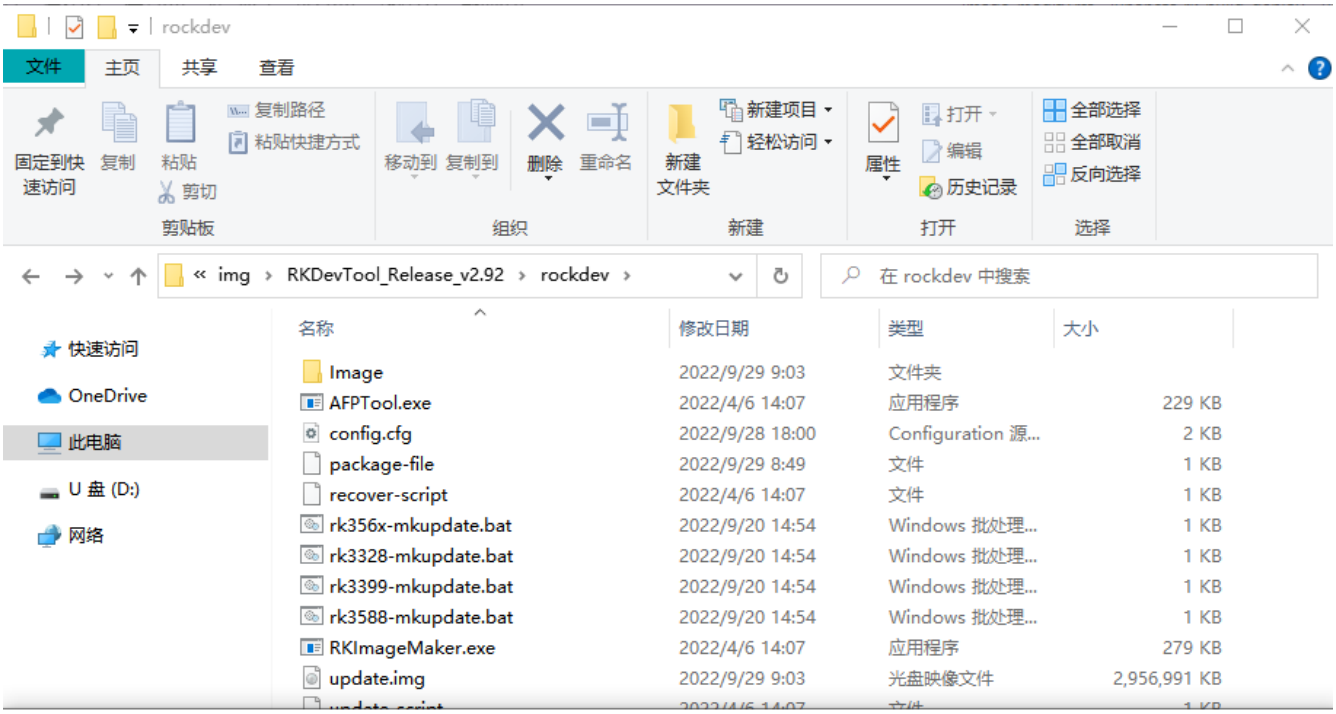
打开 RKDevTool_Release_v2.92\rockdev，并将之前解包的 Output\Android 下的所有文件复制到 rockdev 目录里。

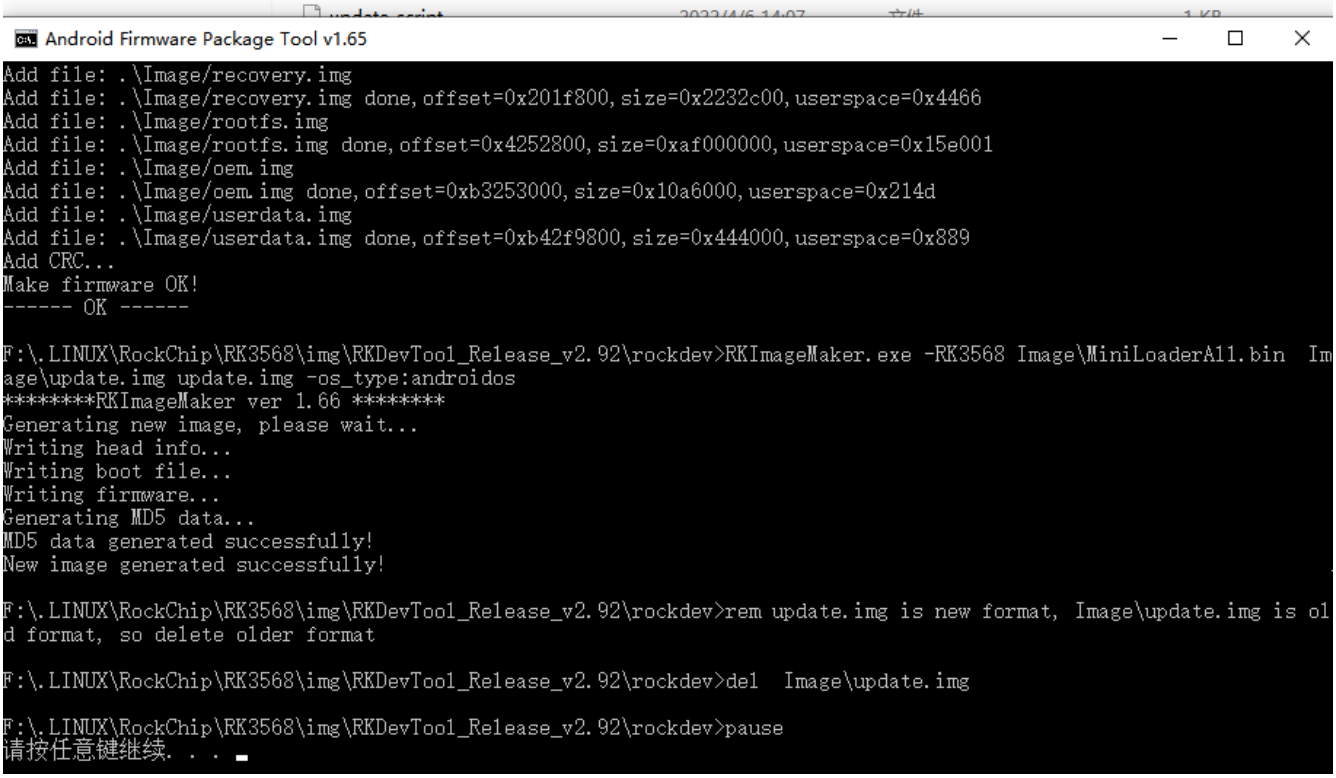


然后用修改过的 rootfs.img 文件替换现在 rockdev\Image 目录下的 rootfs.img 文件

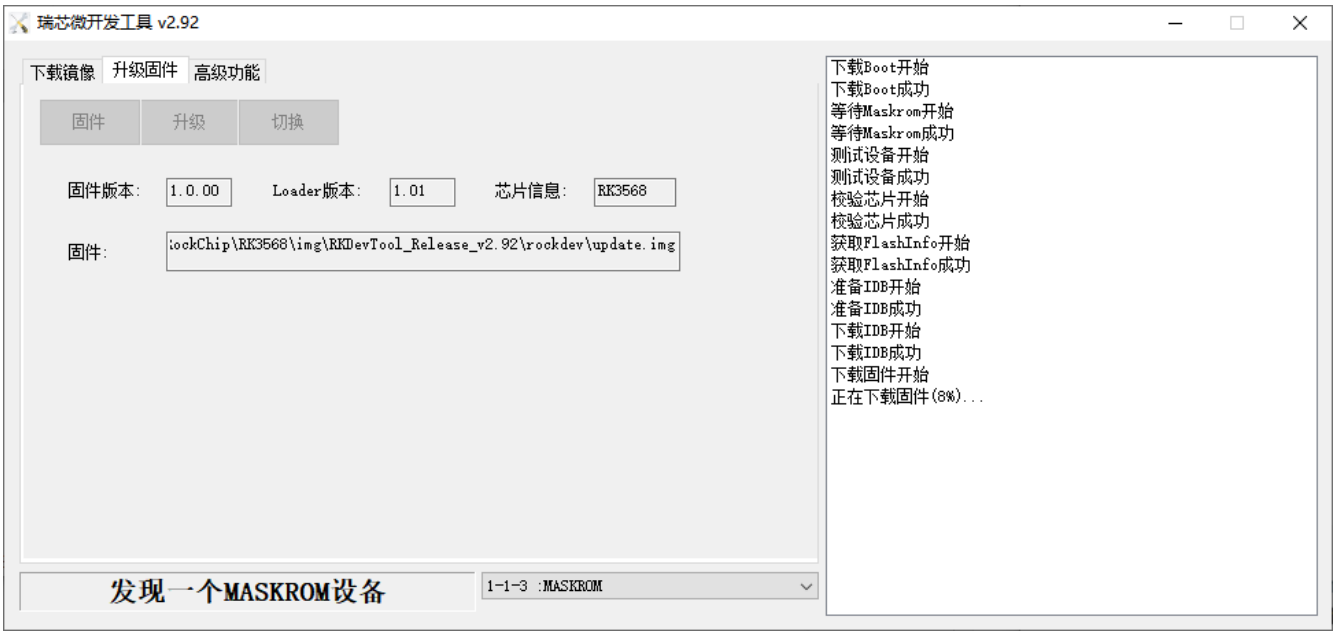


双击 rk356x-mkupdate.bat 开始打包镜像


rockdev
文件
主页
共享
查看
固定到快速访问
复制
粘贴
复制路径
粘贴快捷方式
剪切
剪贴板
移动到
复制到
删除
重命名
组织
新建项目
轻松访问
新建文件夹
新建
属性
打开
编辑
历史记录
打开
全部选择
全部取消
反向选择
选择
« img › RKDevTool_Release_v2.92 › rockdev ›
在 rockdev 中搜索
快速访问
OneDrive
此电脑
U 盘 (D:)
网络
名称
修改日期
类型
大小
Image 2022/9/29 9:03 文件夹
AFPTool.exe 2022/4/6 14:07 应用程序 229 KB
config.cfg 2022/9/28 18:00 Configuration 源... 2 KB
package-file 2022/9/29 8:49 文件 1 KB
recover-script 2022/4/6 14:07 文件 1 KB
rk356x-mkupdate.bat 2022/9/20 14:54 Windows 批处理... 1 KB
rk3328-mkupdate.bat 2022/9/20 14:54 Windows 批处理... 1 KB
rk3399-mkupdate.bat 2022/9/20 14:54 Windows 批处理... 1 KB
rk3588-mkupdate.bat 2022/9/20 14:54 Windows 批处理... 1 KB
RKImageMaker.exe 2022/4/6 14:07 应用程序 279 KB
update.img 2022/9/29 9:03 光盘映像文件 2,956,991 KB


Android Firmware Package Tool v1.65
Add file: .\Image/recovery.img
Add file: .\Image/recovery.img done, offset=0x201f800, size=0x2232c00, userspace=0x4466
Add file: .\Image/rootfs.img
Add file: .\Image/rootfs.img done, offset=0x4252800, size=0xaf000000, userspace=0x15e001
Add file: .\Image/oem.img
Add file: .\Image/oem.img done, offset=0xb3253000, size=0x10a6000, userspace=0x214d
Add file: .\Image/userdata.img
Add file: .\Image/userdata.img done, offset=0xb42f9800, size=0x444000, userspace=0x889
Add CRC...
Make firmware OK!
------ OK ------
F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>RKImageMaker.exe -RK3568 Image\MiniLoaderAll.bin Image\update.img update.img -os_type:androidos
********RKImageMaker ver 1.66 ********
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!
F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>rem update.img is new format, Image\update.img is old format, so delete older format
F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>del Image\update.img
F:\.LINUX\RockChip\RK3568\img\RKDevTool_Release_v2.92\rockdev>pause
请按任意键继续. . .

6. 烧录重新打包的镜像

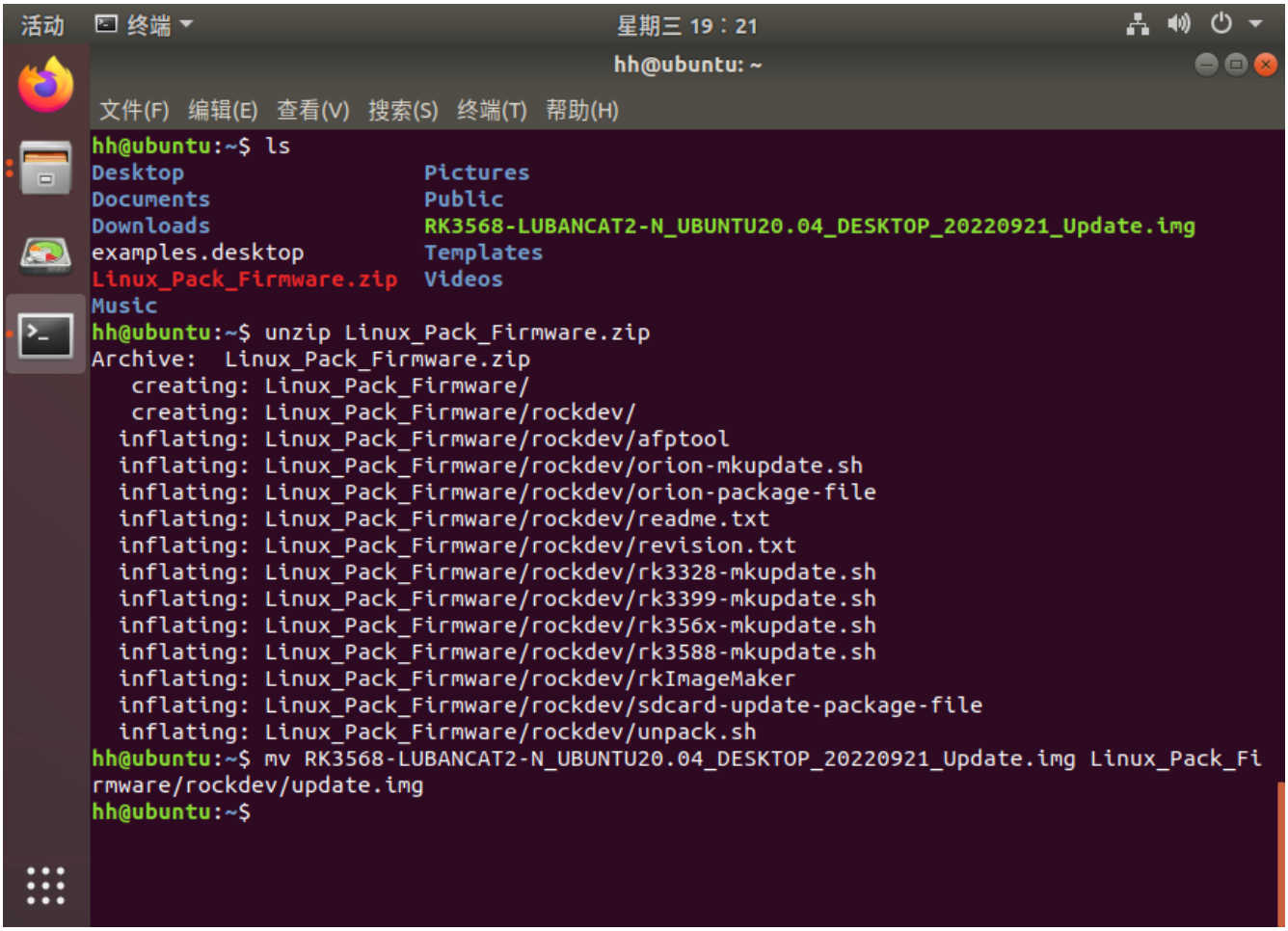



```
Debian GNU/Linux 10 embedfire ttyFIQ0

[username:password] root:root cat:temppwd

Modify information : /etc/issue

embedfire login: root
Password:
Last login: Thu Feb 14 18:12:45 CST 2019 on ttyFIQ0

Welcome to Debian GNU/Linux 10 (buster) (GNU/Linux 4.19.232 aarch64)

 * Documentation:  http://doc.embedfire.com
 * Management:     http://www.embedfire.com
Linux embedfire 4.19.232 #6 SMP Fri Sep 23 09:13:13 CST 2022 aarch64

System information as of Thu Sep 29 09:09:16 CST 2022

System load:   1.30 0.69 0.27   Up time:       1 min
Memory usage:  8 % of 3900MB    IP:           192.168.103.119
CPU temp:      49°C             GPU temp:     48°C
Usage of /:    46% of 5.9G

root@embedfire:~# cat /etc/hostname
embedfire
root@embedfire:~#
```

27.2 Linux_Pack_Firmware 解包、修改、打包 (Linux)

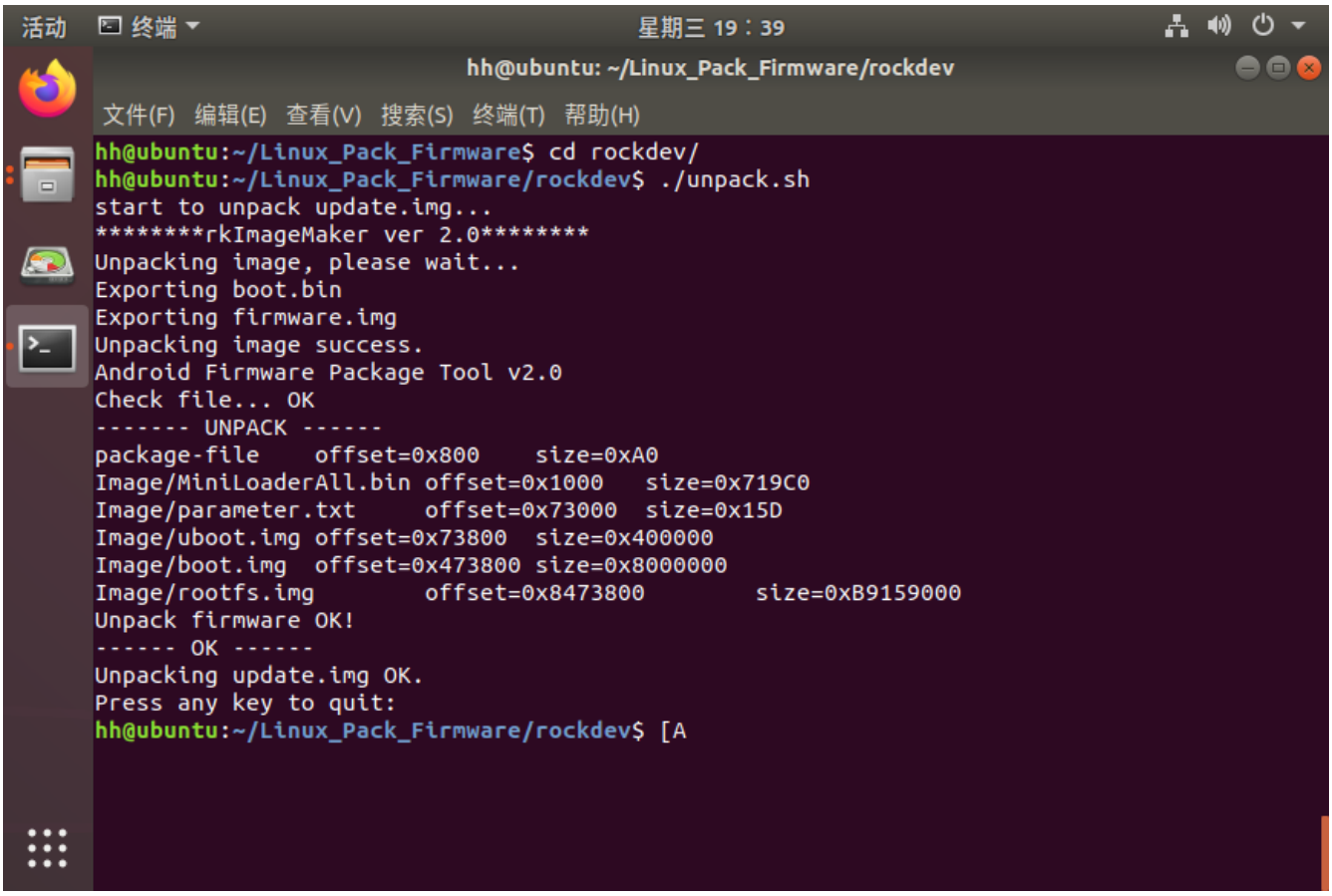
1. 将从网盘获取的镜像解压成.img 格式并重命名为 update.img，也可以直接使用 SDK 生成的 update.img 镜像。

将镜像 update.img 并复制到 Linux_Pack_Firmware/rockdev 目录下

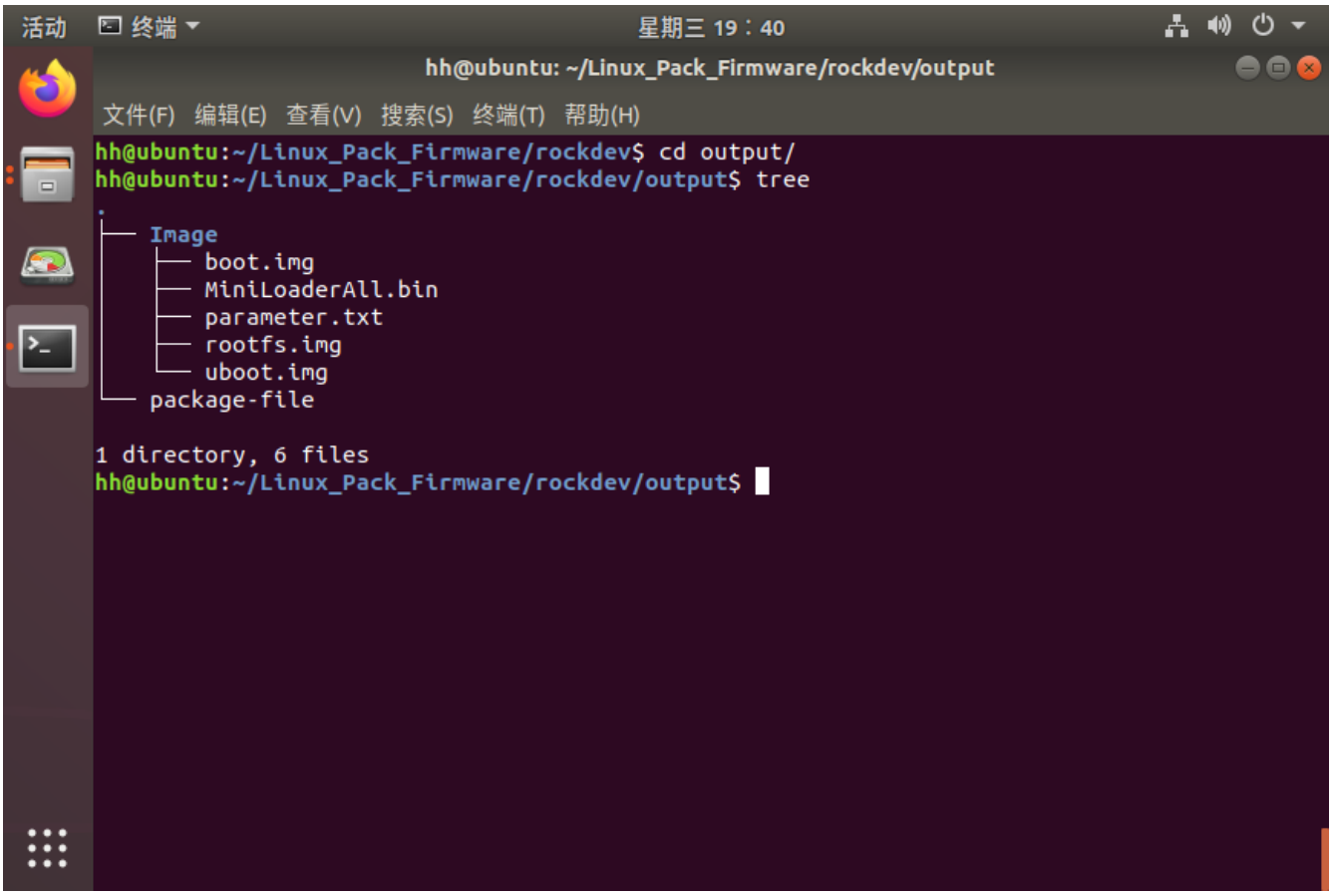


2. 运行 unpack.sh 脚本进行解包

```
unpack.sh
```



解包后的文件保存在 output 目录中，文件内容与 RKDevTool 解包后的内容相同。



output 目录中有一个文件一个文件夹

- package-file：分区与分区镜像名的对应关系
- Image：firmware.img 展开后的内容，也就是解包后的分区镜像。

进入 Image 目录，根据不同操作系统镜像，有以下三类文件

- 分区表：parameter.txt
- loader 文件：MiniLoaderAll.bin
- 分区镜像：boot.img、uboot.img 等以 img 结尾的分区镜像文件

3. 将生成的 rootfs.img 挂载并按需修改内容

```
1 # 将根文件系统挂载到/mnt
2 sudo mount Image/rootfs.img  /mnt
```

(下页继续)

(续上页)

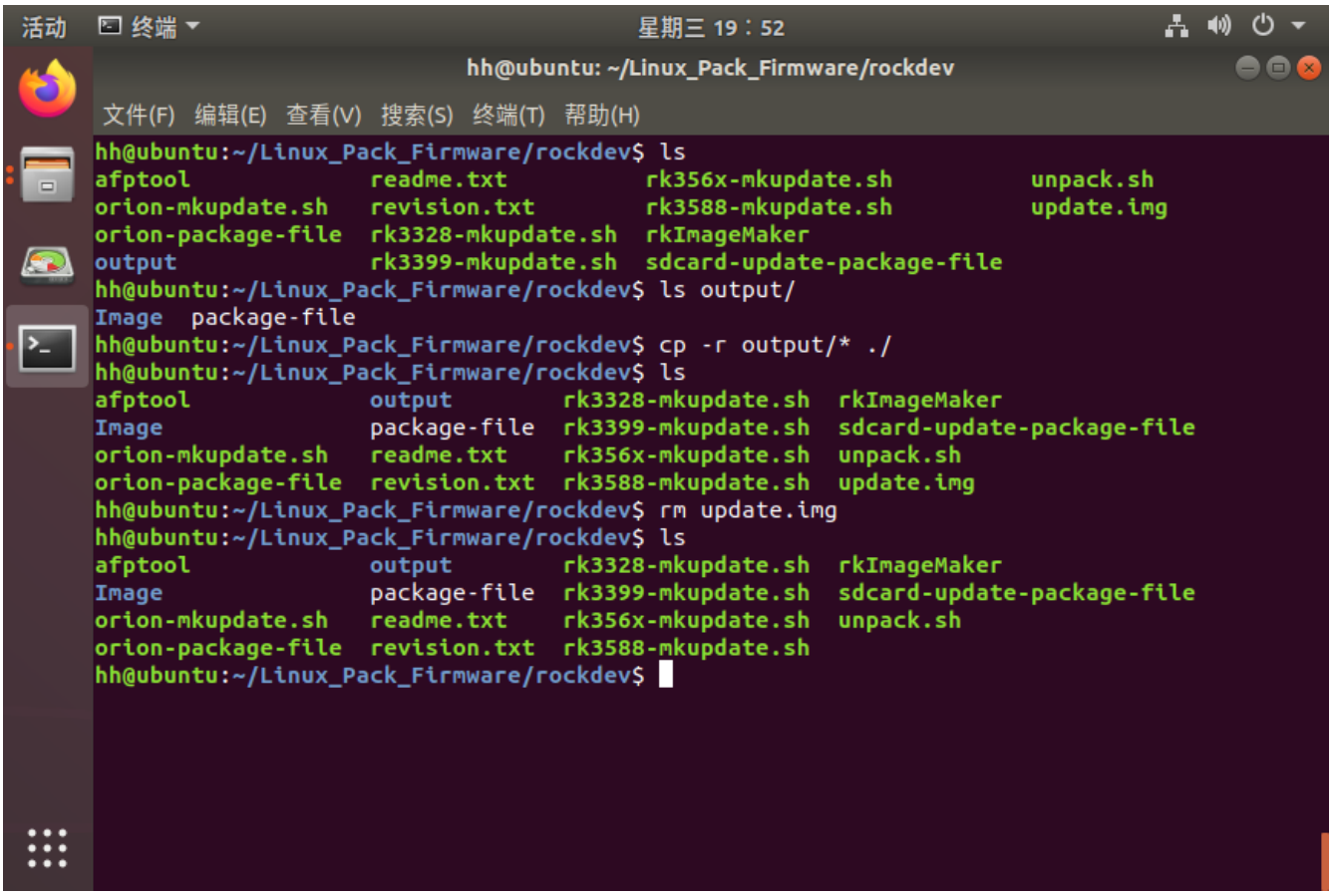
```

# 修改完成后进行卸载
sudo umount /mnt
```

```
lubancat@lubancat-vm:~/test/Linux_Pack_Firmware/rockdev/output$ ls Image/
boot.img  MiniLoaderAll.bin  parameter.txt  rootfs.img  uboot.img  update.img
lubancat@lubancat-vm:~/test/Linux_Pack_Firmware/rockdev/output$ sudo mount Image/rootfs.img  /m
nt
[sudo] lubancat 的密码:
lubancat@lubancat-vm:~/test/Linux_Pack_Firmware/rockdev/output$ ls /mnt/
bin   dev   lib         mnt   rknpu2-v1.5.0.tar  run     snap  tmp   var
boot  etc   lost+found  opt   rockchip-test      sbin    srv   udisk
data  home  media       proc  root               sdcard  sys   usr
lubancat@lubancat-vm:~/test/Linux_Pack_Firmware/rockdev/output$
```

4. 打包

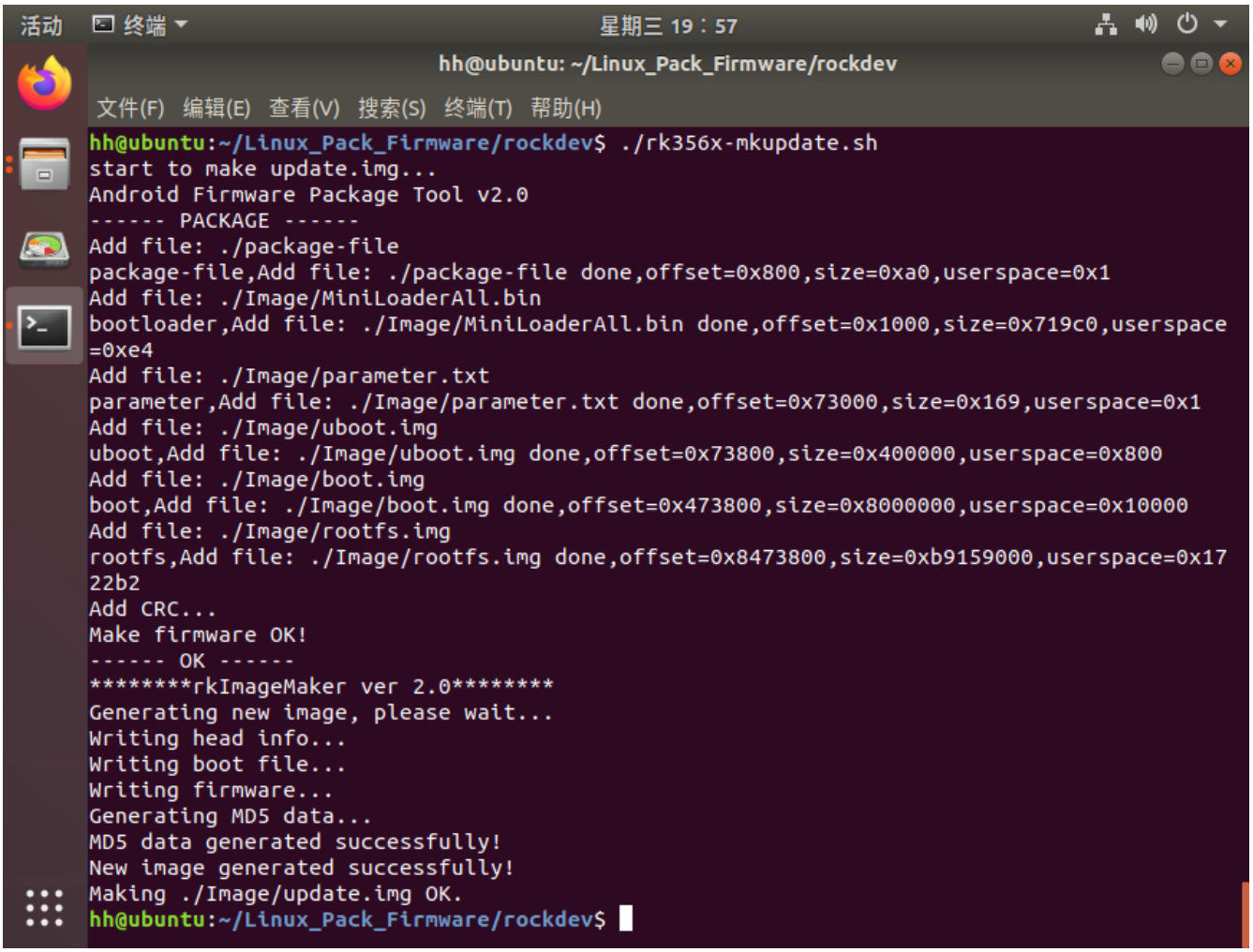
在前面的解包过程中，我们得到了 output 文件夹的内容。

首先我们将 rockdev/output 目录下的内容复制到 rockdev 目录下，删除之前用于解包的 update.img 镜像

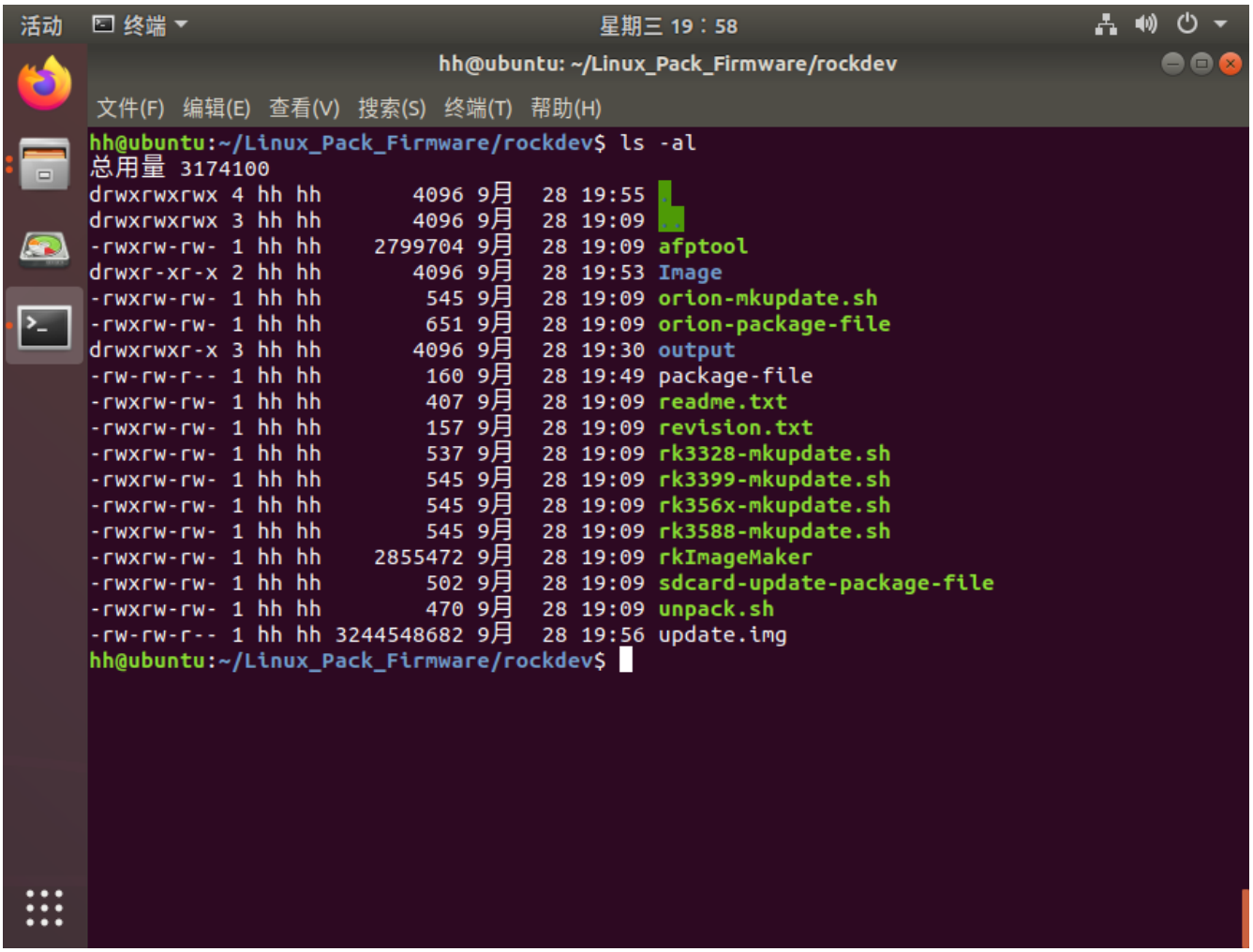


如果我们对某一分区镜像文件做出了改变，在打包时要注意镜像文件名正确，与 package-file 文件中的命名及路径一致。

LubanCat-1、2、Zero 都使用了 rk356x 主芯片，运行对应的打包脚本。



打包完成后，当前文件夹内出现了 update.img 文件，这就是打包后的完整镜像文件



我们使用烧录工具，将打包好的镜像烧录到板卡中，可以正常启动。

```
         Starting Network Manager Script Dispatcher Service...
[  OK  ] Started Network Manager Script Dispatcher Service.
         Starting Time & Date Service...
[  OK  ] Started Time & Date Service.
[  OK  ] Started Bluetooth management mechanism.

Ubuntu 20.04.4 LTS GNU/Linux lubancat ttyFIQ0

[username:password] root:root cat:temppwd

Modify information : /etc/issue

lubancat login: [   11.359201] rk-pcie 3c0000000.pcie: PCIe Link Fail
[   11.359296] rk-pcie 3c0000000.pcie: failed to initialize host

lubancat login: root
Password:

 _          _                  ____      _
| |   _   _| |__   __ _ _ __  / ___|__ _| |_
| |  | | | | '_ \ / _` | '_ \| |   / _` | __|
| |__| |_| | |_) | (_| | | | | |__| (_| | |_
|_____\__,_|_.__/ \__,_|_| |_|\____\__,_|\__|

Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 4.19.232 aarch64)

 * Documentation:  http://doc.embedfire.com
 * Management:     http://www.embedfire.com

System information as of Wed Aug 31 23:27:53 CST 2022

System load:   2.37 0.56 0.19   Up time:       0 min
Memory usage:  7 % of 3900MB    IP:            192.168.103.119
CPU temp:      57°C             GPU temp:      57°C
Usage of /:    9% of 29G


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@lubancat:~#
```

第 28 章　引导系统从固态硬盘启动

本章节将介绍如何制作固态启动镜像包、如何烧录镜像到固态硬盘并从固态硬盘启动系统。

与从 eMMC 或 SD 卡启动系统相比，从固态硬盘启动系统有以下好处：

- 更高的读写速度
- 更大的存储空间
- 更好的耐用性
- 更高的性能稳定性
- 软件运行更流畅

28.1 实现机制

由于瑞芯微的芯片不支持直接从芯片内部的 BootRom 初始化、识别固态硬盘，需要 BootRom 先从 SPI NOR Flash/eMMC/SD 中读取固件启动运行，包含 DDR 初始化和 SPL，SPL 完成后再启动 uboot，而在 uboot 阶段可初始化 PCIE/SATA 外设。

因此，需要将固件烧录到 SPI NOR Flash/eMMC/SD 中，启动到 uboot 阶段，再初始化固态硬盘，最后引导系统从固态硬盘启动。

由于鲁班猫板卡默认都没有 SPI NOR Flash 或有预留接口没有贴芯片，因此，只介绍如何通过 eMMC/SD 引导系统从固态硬盘中启动。

28.2 引导系统从 msata 固态启动

对于使用 rk3566、rk3576、rk3588s 主控芯片的鲁班猫系列板卡，没有 M.2 接口，无法使用 M.2 固态硬盘，但是有 MINI PCIE 接口，而 MINI PCIE 接口信号可复用为 sata 信号，可接 msata 固态硬盘。

对于使用 rk3568、rk3588 主控芯片的鲁班猫系列板卡，有 M.2 接口，因此不考虑使用 msata 固态，而使用 PCIE 协议的 M.2 固态。

目前已支持从 msata 固态启动的板卡：

- LubanCat-1
- LubanCat-1IO
- LubanCat-1H
- LubanCat-3
- LubanCat-3IO
- LubanCat-4

提示： 如果某鲁班猫板卡不在以上列表，但希望添加支持从 msata 固态启动系统，请联系淘宝技术支持进行反馈。

28.2.1 制作固态启动镜像包

28.2.1.1 重新编译 uboot

由于默认配置文件是没有支持 sata 相关配置的，因此发布的通用镜像默认不能在 uboot 阶段初始化 msata 固态硬盘。如果需要支持初始化 msata 固态硬盘需要使用对应的 msata 配置文件重新编译 uboot。

注意： 编译 uboot 源码需要借助 SDK 源码，请根据前面构建系统镜像章节获取对应板卡的 SDK 源码。

28.2.1.1.1 获取 uboot 源码

野火 uboot 源码地址：https://github.com/LubanCat/u-boot/tree/main

支持 msata 启动的分支为 main 分支，sdk 源码中的 uboot 源码不一定使用该分支，因此需要替换 uboot 源码。

```
# 进入 sdk 源码顶层文件夹，重命名原 uboot 源码
mv u-boot u-boot-old

# 获取 main 分支的 uboot 源码
git clone --depth=1 -b main https://github.com/LubanCat/u-boot.git
```

板卡配置文件位于 uboot 源码/configs 目录下，对应的配置文件如下：

- rk3566 系列：lubancat-rk3566-msata_defconfig
- rk3576 系列：lubancat-rk3576-msata_defconfig
- rk3588s 系列：lubancat-4-rk3588s-msata_defconfig

28.2.1.1.2 获取编译工具链

SDK 自带编译工具链，但是通过 uboot 源码中的 make.sh 脚本手动编译 uboot 时，脚本中指定了以下变量：

```
CROSS_COMPILE_ARM64=../prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-
↪2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-

CROSS_COMPILE_ARM64=$(cd `dirname ${CROSS_COMPILE_ARM64}`; pwd)"/aarch64-
↪linux-gnu-"
```

因此，需要使用 6.3.1 版本的交叉编译工具链，但是不是每个 SDK 都带这个工具链，SDK 中的工具链位于 SDK 源码/prebuilts/gcc/linux-x86/aarch64/ 目录下，如果没有 6.3.1 版本的交叉编译工具链，可以执行以下命令获取工具链，或者修改 make.sh 脚本指定使用 SDK 源码中带有的工具链。

```
# 从 SDK 顶层文件夹进入工具链目录
cd prebuilts/gcc/linux-x86/aarch64/

# 获取 6.3.1 版本工具链
git clone --depth=1 https://github.com/LubanCat/gcc-linaro-6.3.1-2017.05-
↪x86_64_aarch64-linux-gnu.git
```

28.2.1.1.3 编译 uboot 源码

```
# 进入 uboot 源码顶层文件夹
cd u-boot

# 清除旧编译输出
make clean

# 根据板卡配置文件名称进行编译，配置文件名称为 xxx_defconfig，则编译为./make.sh xxx
# 以 rk3566 系列为例
./make.sh lubancat-rk3566-msata

# 以 rk3576 系列为例
./make.sh lubancat-rk3576-msata

# 以 rk3588s 系列为例
./make.sh lubancat-4-rk3588s-msata
```

编译成功后在 uboot 源码顶层文件夹生成名为 uboot.img 的文件，该文件就是我们需要的支持 msata 的 uboot 镜像文件。

```
guest@dev107:~/LubanCat_Linux_rk356x_SDK/u-boot$ ls
api                 config.mk       include         rk356x_spl_loader_v1.18.112.bin  u-boot.cfg.configs
arch                configs         Kbuild          scripts                          u-boot.dtb
bl31_0x00040000.bin disk            Kconfig         snapshot.commit                  u-boot-dtb.bin
bl31_0x00069000.bin doc             lib             spl                              uboot.img
bl31_0x0006b000.bin Documentation   Licenses        System.map                       u-boot.lds
bl31_0xfdcc1000.bin drivers         MAINTAINERS     tee.bin                          u-boot.map
bl31_0xfdcce000.bin dtoverlay       Makefile        test                             u-boot-nodtb.bin
bl31_0xfdcd0000.bin dts             make.sh         tools                            u-boot.srec
bl31.elf            env             net             tpl                              u-boot.sym
board               examples        post            u-boot                           usb_update.txt
cmd                 fit             PREUPLOAD.cfg   u-boot.bin
common              fs              README          u-boot.cfg
```

28.2.1.2 重新编译内核设备树

由于内核主设备树默认配置 MINI PCIE 接口的功能为 PCIE，不能识别 msata 固态硬盘，因此需要切换为 SATA 功能。

28.2.1.2.1 开启 sata 节点

- rk3566 系列的 MINI PCIE 接口对应设备树节点为：`pcie2x1`、`sata2`
- rk3576 系列的 MINI PCIE 接口对应设备树节点为：`pcie0`、`sata0`
- rk3588 系列的 MINI PCIE 接口对应设备树节点为：`pcie2x1l2`、`sata0`

以 LubanCat-1 板卡为例，对应主设备树名称为 `rk3566-lubancat-1.dts`，修改 sdk 源码/`kernel/arch/arm64/boot/dts/rockchip/rk3566-lubancat-1.dts`

在主设备树找到 pcie2x1 节点，修改内容如下：

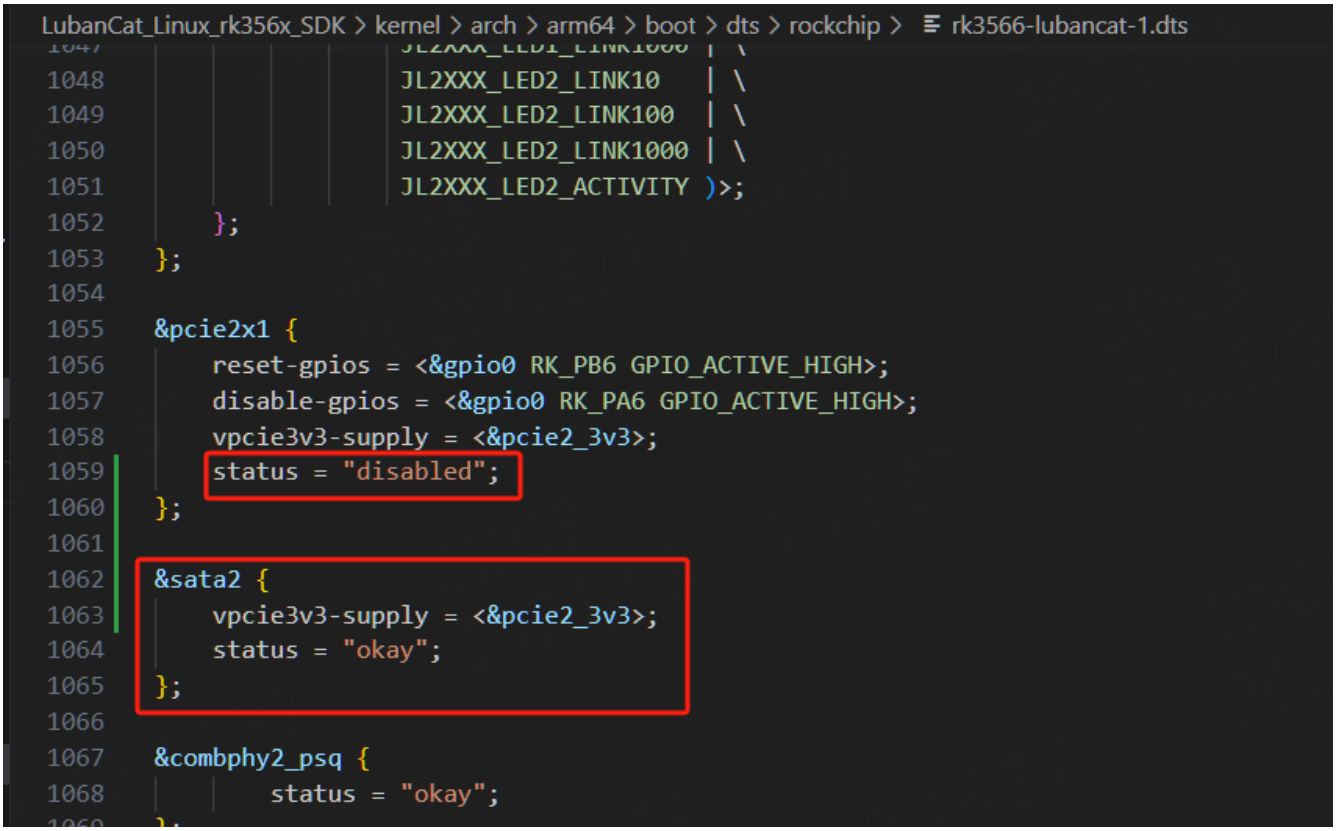
```
1 &pcie2x1 {
2     reset-gpios = <&gpio0 RK_PB6 GPIO_ACTIVE_HIGH>;
3     disable-gpios = <&gpio0 RK_PA6 GPIO_ACTIVE_HIGH>;
4     vpcie3v3-supply = <&pcie2_3v3>;
5     status = "disabled";
6 };
7
8 &sata2 {
9     vpcie3v3-supply = <&pcie2_3v3>;
```

(下页继续)

(续上页)

```
10      status = "okay";
11  };
```

也就是将 pcie2x1 节点 disabled，然后添加 sata2 节点使能，其中 sata2 节点的 vpcie3v3-supply 引用的电源要和 pcie2x1 节点的一致，修改完成后如下图所示：



28.2.1.2.2 编译设备树

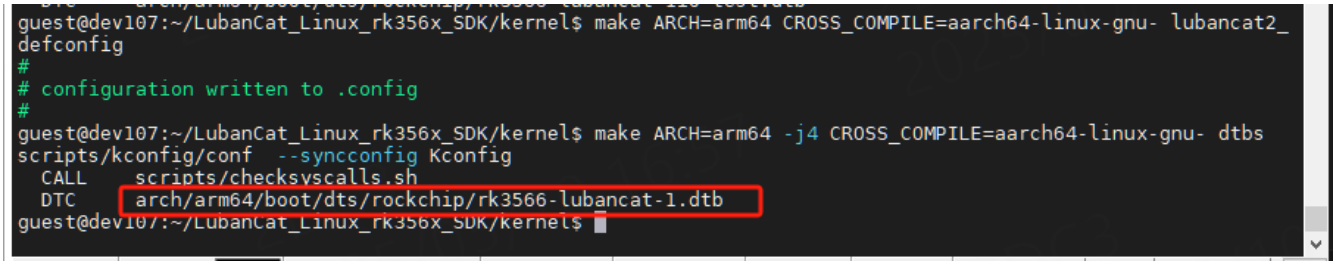
借助 LubanCat-SDK 提供的编译环境，我们在编译 kernel 镜像的同时也会编译对应的设备树。

在 LubanCat-SDK 顶层文件夹下运行以下命令编译内核：

```
1  # 编译 kerneldeb 文件
2  ./build.sh kerneldeb
```

也可以参照驱动手册的 使用内核的构建脚本编译设备树 章节单独编译设备树。

编译得到的与设备树源码相同目录下，相同名称以.dtb 结尾的文件就是我们需要的设备树 dtb 文件。



28.2.1.3 解包发布镜像并重新打包

为了方便我们不重新编译镜像，而是使用野火发布的通用镜像进行解包，解包后替换并修改一些组件，再重新打包成完整镜像，从而得到支持固态启动镜像包。

28.2.1.3.1 解包镜像

> **注意：** 解包镜像需要在 Linux 环境使用 Linux_Pack_Firmware 根据进行解包，不要在 windows 解包。

访问资料网盘/6-开发软件/，获取 Linux_Pack_Firmware_v2.29.zip

将 Linux_Pack_Firmware_v2.29.zip 解压到 Linux PC 或虚拟机中。

```
# 解压
unzip Linux_Pack_Firmware_v2.29.zip
```

访问资料网盘获取需要的系统镜像，传到 Linux_Pack_Firmware/rockdev 目录下。

以 LubanCat-1 的 Debian xfce 镜像为例：

```
# 进入工具目录
cd Linux_Pack_Firmware/rockdev/

# 解压镜像，根据实际压缩包名称而定
7z x lubancat-rk3566-debian10-xfce-20250303_update.7z

# 重命名镜像为 update.img，根据实际镜像名称而定
mv lubancat-rk3566-debian10-xfce-20250303_update.img update.img

# 解包镜像
./unpack.sh

# 解包镜像的到 output 目录
# 查看 output 目录
ls output
# 信息输出如下
boot.img  MiniLoaderAll.bin  package-file  parameter.txt  rootfs.img  uboot.
↪img
```

28.2.1.3.2 替换和修改镜像组件

替换 uboot.img

将前面编译 uboot 源码生成的 uboot.img 替换解包生成 uboot.img

```
# 替换 uboot.img，根据自己的实际路径确定
cp -f  ~/LubanCat_Linux_rk356x_SDK/u-boot/uboot.img ~/Linux_Pack_Firmware/
↪rockdev/output/
```

修改 boot.img

由于通用镜像默认没有支持引导启动 msata 配置，因此需要进行修改，引导配置位于 boot.img 中的 boot.scr，又因为 boot.scr 是由 boot.cmd 生成的，所以我们要修改的是 boot.img 中的 boot.cmd，然后生成 boot.scr。

```
# 进入解包输出目录
cd Linux_Pack_Firmware/rockdev/output/

# 挂载 boot.img
sudo mount boot.img /mnt

# 修改 boot.cmd
sudo vi /mnt/boot.cmd
```

在 boot.cmd 的中添加或修改为以下高亮部分：

列表 1: boot.cmd

```
echo [boot.cmd] run boot.cmd scripts ...;

sata_boot=1

if test "$sata_boot" = "1"; then

    setenv devnum_old ${devnum}

    scsi scan

    if test "$board" = "evb_rk3588" -o "$board" = "evb_rk3576"; then
        setenv devnum 1
    else
        setenv devnum 0
    fi
```

(下页继续)

(续上页)

```
16
17      scsi device ${devnum}
18
19      if test $? -eq 0; then
20          setenv devtype scsi
21          setenv rootfspart /dev/sda3
22
23          if test ! -e ${devtype} ${devnum}:${distro_bootpart} /rk-kernel.dtb;
    ↪ then
24              rockusb 0 scsi ${devnum}
25          fi
26
27      else
28          setenv devnum ${devnum_old}
29          setenv rootfspart /dev/mmcblk${devnum}p3
30      fi
31
32  else
33      setenv rootfspart /dev/mmcblk${devnum}p3
34  fi
35
36  if test -e ${devtype} ${devnum}:${distro_bootpart} /uEnv/uEnv.txt; then
37
38      echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_
    ↪r} /uEnv/uEnv.txt ...;
39      load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_r} /uEnv/uEnv.
    ↪txt;
40
41      echo [boot.cmd] Importing environment from ${devtype} ...
42      env import -t ${env_addr_r} 0x8000
43
44      setenv bootargs ${bootargs} root=${rootfspart} boot_part=${distro_
    ↪bootpart} ${cmdline}
```

(下页继续)

(续上页)

```
    printenv bootargs

    echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r} /initrd-${uname_r} ...
    load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r} /initrd-${uname_r}

    echo [boot.cmd] loading ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} /Image-${uname_r} ...
    load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} /Image-${uname_r}

    echo [boot.cmd] loading default rk-kernel.dtb
    load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} /rk-kernel.dtb

    fdt addr  ${fdt_addr_r}
    fdt set /chosen bootargs

    echo [boot.cmd] dtoverlay from /uEnv/uEnv.txt
    setenv dev_bootpart ${devnum}:${distro_bootpart}
    dtfile ${fdt_addr_r} ${fdt_over_addr}  /uEnv/uEnv.txt ${env_addr_r}

    echo [boot.cmd] [${devtype} ${devnum}:${distro_bootpart}] ...
    echo [boot.cmd] [booti] ...
    booti ${kernel_addr_r} ${ramdisk_addr_r} ${fdt_addr_r}
fi

echo [boot.cmd] run boot.cmd scripts failed ...;

# Recompile with:
# mkimage -C none -A arm -T script -d /boot/boot.cmd /boot/boot.scr
```

以上修改的意思是：

- 第 3-5 行，判断 sata_boot 变量是否为 1，如果为 1，则支持从 msata 启动，如果为 0 则恢复默认启动方式从 eMMC/SD 启动。
- 第 7 行，devnum 是启动介质的设备号，先记录原来的 devnum，如果没有扫描到 msata 设备还需要更改回原来的值。
- 第 8 行，扫描 sata 设备。
- 第 11-15 行，判断是否是 3588 或者 3576，如果是则设置 devnum 为 1，否则设置为 0
- 第 17-21 行，如果扫描到 sata 设备则设置启动类型为 scsi，设备号为 0，文件系统分区为/dev/sda3
- 第 23-25 行，判断 msata 硬盘 boot 分区是否存在 rk-kernel.dtb，如果不存在则说明 msata 固态中没有镜像，引导进入烧录模式，可将镜像直接烧录到 msata 固态。
- 第 44 行：引用 rootfspart 的变量设置根文件系统挂载分区。

修改完成 boot.cmd 和保存退出，然后执行以下命令生成 boot.scr

```
sudo mkimage -C none -A arm -T script -d /mnt/boot.cmd /mnt/boot.scr
```

由于 uboot 启动读取的内核设备树是 boot.img 的 `rk-kernel.dtb`，默认是通用设备树 (没有开启 sata 接口)，系统第一次启动后会软连接到 dtb 目录下实际的设备树。

因此，需要将前面编译设备树源码生成的 dtb 替换 `rk-kernel.dtb` 和 dtb 目录下实际的设备树，从而启用 sata 接口，以下以 LubanCat-1 为例：

```
# 前面已经将 boot.img 挂载到/mnt

# 替换 rk-kernel.dtb，根据实际路径和板卡实际设备树而定
sudo cp -f ~/LubanCat_Linux_rk356x_SDK/kernel/arch/arm64/boot/dts/rockchip/
↪rk3566-lubancat-1.dtb /mnt/rk-kernel.dtb

# 替换 dtb 目录下实际的设备树，根据实际路径和板卡实际设备树而定
sudo cp -f ~/LubanCat_Linux_rk356x_SDK/kernel/arch/arm64/boot/dts/rockchip/
↪rk3566-lubancat-1.dtb /mnt/dtb/
```

(下页继续)

(续上页)

```
8
9  # 替换完成后卸载/mnt 即可
10 sudo umount /mnt/
```

修改 rootfs.img

由于 rk356x 系列的 4.19 内核对应的 Debian10 镜像默认没有支持 msata 系统初始化，因此需要进行修改，系统初始化文件位于 rootfs.img 的 /etc/init.d/boot_init.sh。

```
1  # 进入解包输出目录
2  cd Linux_Pack_Firmware/rockdev/output/
3
4  # 挂载 rootfs.img
5  sudo mount rootfs.img /mnt
6
7  # 修改 boot_init.sh
8  sudo vi /mnt/etc/init.d/boot_init.sh
```

修改内容如下：

```
1  #Boot_Part="${Root_Part::-2}"p2
2  Boot_Part="${Root_Part::-1}"2
```

修改完成如下图所示：

```
if [ ! -e "/boot/boot_init" ] ; then
        if [ ! -e "/dev/disk/by-partlabel/userdata" ] ; then
                if [ ! -L "/boot/rk-kernel.dtb" ] ; then
                        for x in $(cat /proc/cmdline); do
                                case $x in
                                root=*)
                                        Root_Part=${x#root=}
                                        #Boot_Part="${Root_Part::-2}"p2
                                        Boot_Part="${Root_Part::-1}"2
                                        ;;
                                esac
                        done

                        mount "$Boot_Part" /boot
                        echo "$Boot_Part  /boot  auto  defaults  0 2" >> /etc/fstab
                fi

                service lightdm stop || echo "skip error"

                apt install -fy --allow-downgrades /boot/kerneldeb/* || true
                apt-mark hold linux-headers-$(uname -r) linux-image-$(uname -r) || true

-- INSERT --                                                    238,43-57     94%
```

修改完成后卸载 rootfs.img

```
# 卸载前面挂载 rootfs.img
sudo umount /mnt
```

28.2.1.3.3 打包镜像

替换 uboot.img、修改 boot.img 和 rootfs.img 后即可打包成支持固态启动镜像包。

```
# 进入输出目录
cd Linux_Pack_Firmware/rockdev/output

# 打包镜像，根据实际芯片名称而定

#rk356x 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3568 MiniLoaderAll.bin update_tmp.img update.img -os_
↪type:androidos || pause

#rk3576 打包执行
../afptool -pack ./ update_tmp.img || pause
```

(下页继续)

(续上页)

```
../rkImageMaker -RK3576 MiniLoaderAll.bin update_tmp.img update.img -os_type:androidos || pause

#rk3588/rk3588s 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3588 MiniLoaderAll.bin update_tmp.img update.img -os_type:androidos || pause
```

打包后在 output 当前目录生成 `update_tmp.img` 和 `update.img` 镜像文件，其中 `update.img` 就是最终打包生成的支持引导 msata 固态启动的镜像。

28.2.2 烧录镜像

制作完成支持引导 msata 固态启动的 `update.img` 镜像后，可将 `update.img` 烧录至 eMMC/SD 以及 msata 固态。

28.2.2.1 烧录镜像到 eMMC/SD

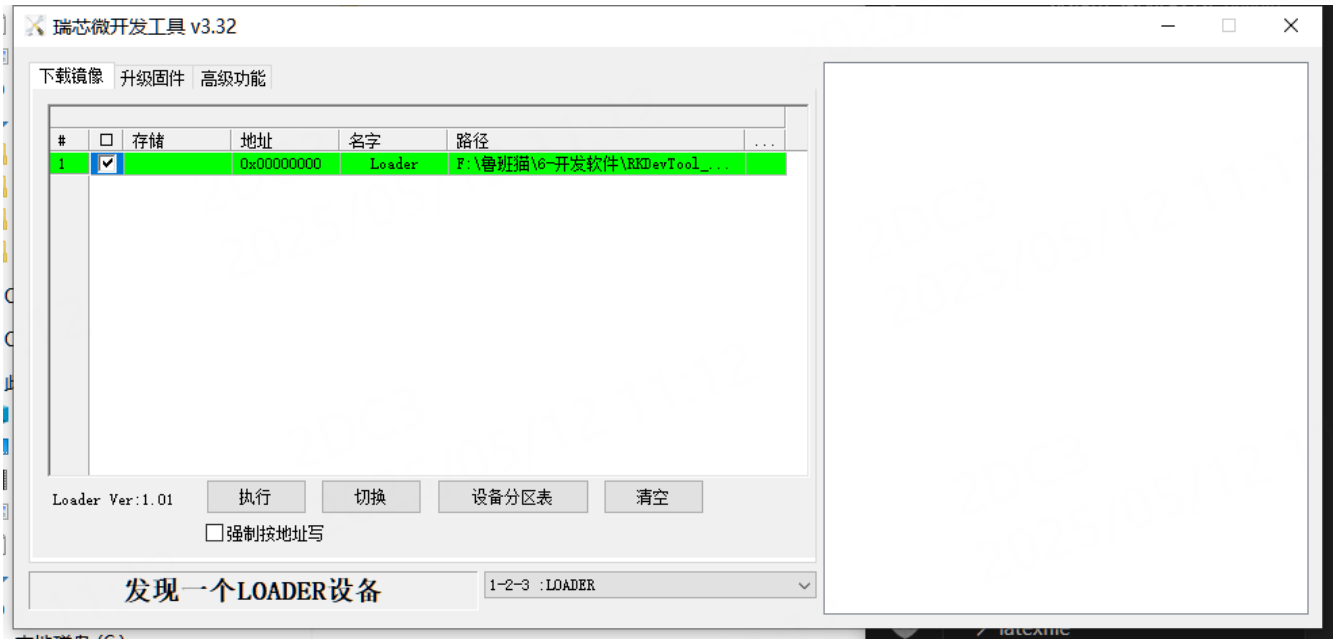
因为需要 eMMC/SD 作为引导，因此需要先将镜像烧录到 eMMC/SD，如何烧录系统到 eMMC/SD 参考以下文档：

- eMMC 镜像烧录参考：完整镜像烧录 以及各板卡的 快速使用手册
- SD 镜像烧录参考：SD 卡启动镜像烧录 ，注意选择功能模式勾选的是 **SD 启动**。

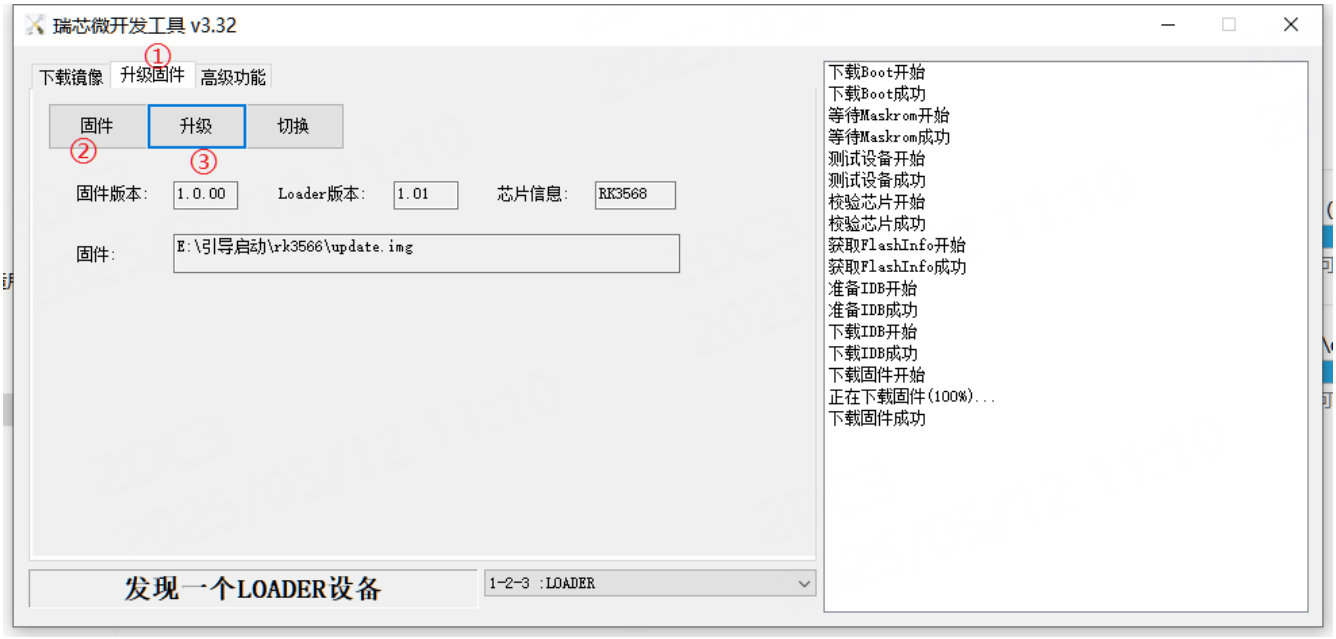
28.2.2.2 烧录镜像到 msata 固态

将镜像烧录到 eMMC 或 SD 卡后，再接入 msata 固态硬盘，启动时会扫描 sata 设备并判断 sata 设备中是否存在镜像，如果 sata 设备中的 boot 分区不存在 rk-kernel.dtb，则判断为不存在镜像，会引导进入烧录模式。

当 msata 硬盘不存在镜像时，usb 烧录口接入 usb 线会识别到 `LOADER` 设备，如下图所示：



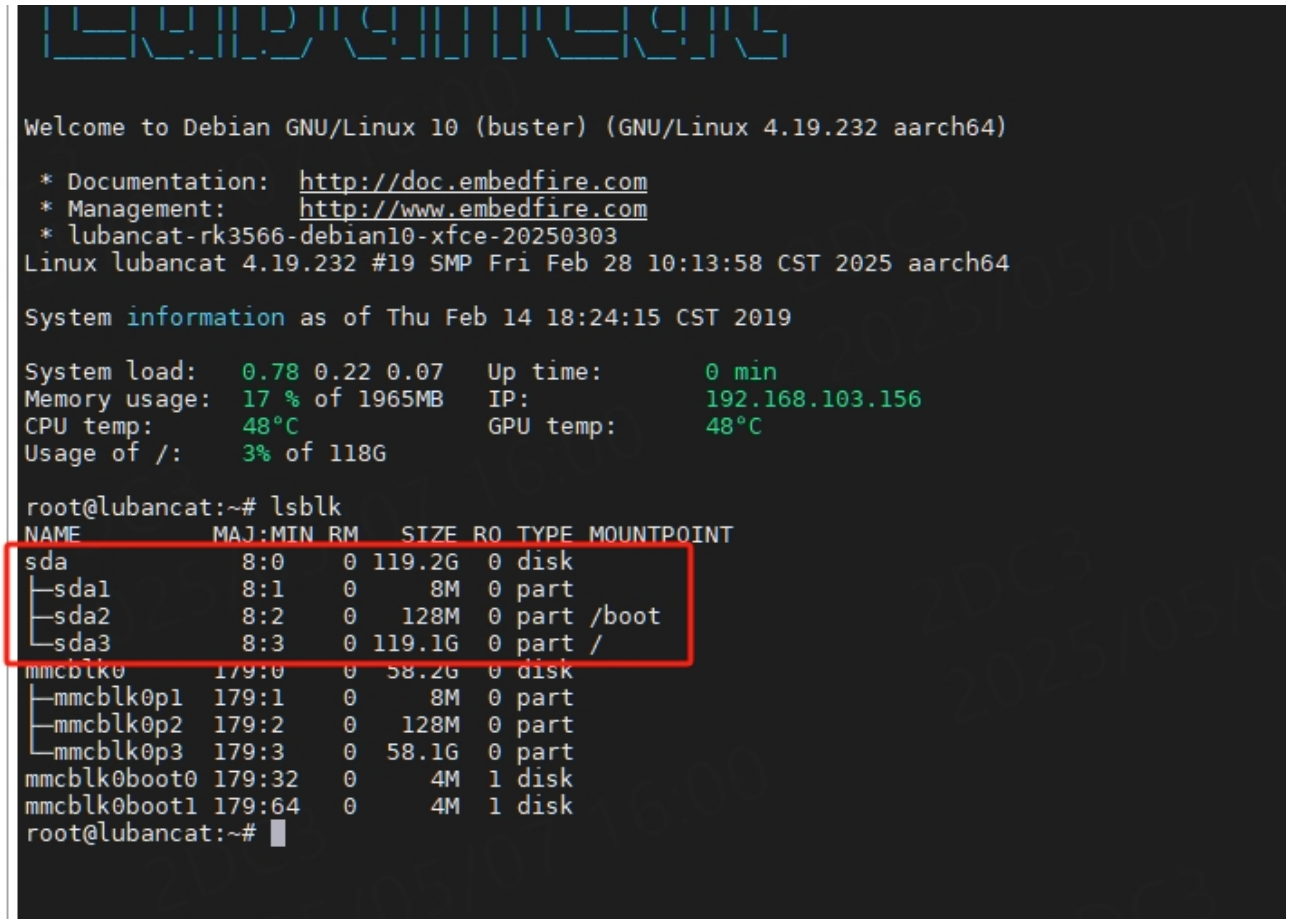
选择 升级固件界面，固件选择前面生成的 `update.img` ，最后点击烧录即可，如下图所示：



烧录完成后，会自动重启，系统启动成功后，执行以下命令查看是否成功从 msata 固态启动：

```
# 查看系统挂载
lsblk
```

引导成功效果如下图所示，其中/dev/sda2 为 msata 固态的 boot 分区，/dev/sda3 为 msata 固态的 rootfs 分区。



28.2.2.3 重新烧录镜像到 msata 固态

msata 硬盘带镜像时不会进入烧录模式，如果需要重新烧录 msata 硬盘，此时需要删除判断文件或者手动引导至烧录模式。

28.2.2.3.1 通过删除判断文件方式

因为启动会判断 sata 固态中 boot 分区是否存在 rk-kernel.dtb，如果不存在 rk-kernel.dtb，则会引导进入烧录模式。

因此，如果需要重新烧录 msata 硬盘，可删除 rk-kernel.dtb，msata 启动系统后执行以下命令删除：

```
# 删除判断文件
sudo rm /boot/rk-kernel.dtb

# 重启
sudo reboot
```

USB 烧录口连接电脑，识别到 LOADER 设备后即可重新烧录镜像到 msata 固态。

28.2.2.3.2 通过手动引导方式

在接入 msata 硬盘后，板卡 Debug 接口接串口模块，然后板卡上电，在上电过程中快速按下 ctrl + c 进入 uboot 命令行模式，在 uboot 命令行模式输入以下命令：

```
# 扫描设备
scsi scan

# 进入烧录模式
rockusb 0 scsi 0
```

如下图所示，USB 烧录口连接电脑，识别到 LOADER 设备后即可重新烧录镜像到 msata 固态。

```
  aclk_top_low 400000 KHz
  hclk_top 150000 KHz
  pclk_top 100000 KHz
  aclk_perimid 300000 KHz
  hclk_perimid 150000 KHz
  pclk_pmu 100000 KHz
Net:   eth1: ethernet@fe010000
Hit key to stop autoboot('CTRL+C'):  0
=> <INTERRUPT>
=> <INTERRUPT>
=> <INTERRUPT>
=> scsi scan
scanning bus for devices...
Target spinup took 0 ms.
AHCI 0001.0300 32 slots 1 ports 6 Gbps 0x1 impl SATA mode
flags: ncq stag pm led clo only pmp fbss pio slum part ccc apst
  Device 0: (0:0) Vendor: ATA Prod.: Kingchuxing 128G Rev: W022
            Type: Hard Disk
            Capacity: 122104.3 MB = 119.2 GB (250069680 x 512)
=> rockusb 0 scsi 0
RKUSB: LUN 0, dev 0, hwpart 0, sector 0x0, count 0xee7c2b0
\usb device is high-speed
/
```

28.3 引导系统从 M.2 固态启动

对于使用 rk3566、rk3576、rk3588s 主控芯片的鲁班猫系列板卡，没有 M.2 接口，无法使用 M.2 固态硬盘，但是有 MINI PCIE 接口，而 MINI PCIE 接口信号可复用为 sata 信号，可接 msata 固态硬盘。

对于使用 rk3568、rk3588 主控芯片的鲁班猫系列板卡，有 M.2 接口，因此不考虑使用 msata 固态，而使用 PCIE 协议的 M.2 固态。

目前已支持从 M.2 固态启动的板卡：

- LubanCat-2
- LubanCat-2IO
- LubanCat-5
- LubanCat-5IO

注意： 其中 LubanCat-RK3588 系列 M.2 固态启动系统需要使用 6.1 内核，在 5.10 内核启动会出现崩溃情况。另外，如需使用 Ubuntu22 系统，请参考 `资料网盘/3-Linux 镜像/LubanCat-rk3588/Ubuntu 系统镜像/通用系统镜像-适用于所有板卡/Ubuntu22.04/m2 固态启动`说明进行操作，Ubuntu22 系统不适用本文档。

提示： 如果某鲁班猫板卡不在以上列表，但希望添加支持从 M.2 固态启动系统，请联系淘宝技术支持进行反馈。

28.3.1 制作固态启动镜像包

28.3.1.1 重新编译 uboot

由于默认配置文件是没有支持 PCIE 相关配置的，因此发布的通用镜像默认不能在 uboot 阶段初始化 M.2 固态硬盘。如果需要支持初始化 M.2 固态硬盘需要使用对应的 PCIE 配置文件重新编译 uboot。

注意： 编译 uboot 源码需要借助 SDK 源码，请根据前面构建系统镜像章节获取对应板卡的 SDK 源码。

28.3.1.1.1 获取 uboot 源码

野火 uboot 源码地址：https://github.com/LubanCat/u-boot/tree/main

支持 PCIE 启动的分支为 main 分支，sdk 源码中的 uboot 源码不一定使用该分支，因此需要替换 uboot 源码。

```
1 # 进入 sdk 源码顶层文件夹，重命名原 uboot 源码
2 mv u-boot u-boot-old
```

(下页继续)

(续上页)

```

# 获取 main 分支的 uboot 源码
git clone --depth=1 -b main https://github.com/LubanCat/u-boot.git
```

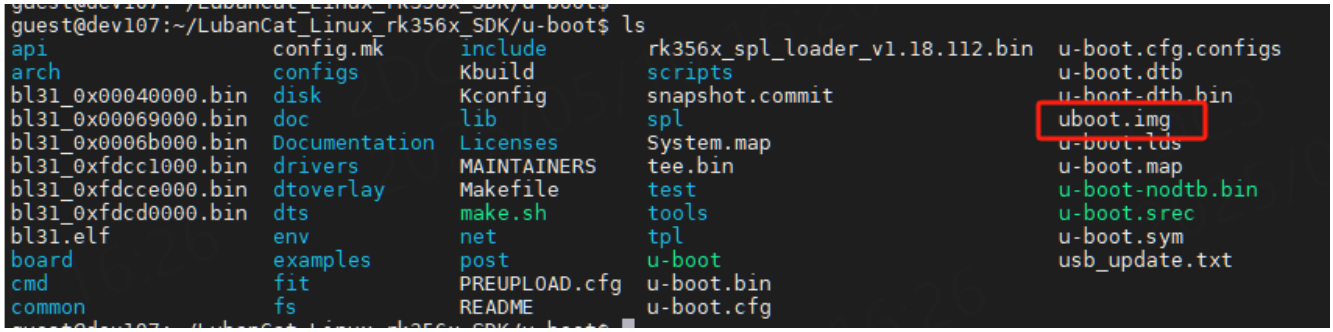
板卡配置文件位于 uboot 源码/configs 目录下，对应的配置文件如下：

- rk3568 系列：lubancat-rk3568-pcie_defconfig
- LubanCat5：lubancat-5-rk3588-pcie_defconfig
- LubanCat5IO：lubancat-5io-rk3588-pcie_defconfig

28.3.1.1.2 编译 uboot 源码

```
# 进入 uboot 源码顶层文件夹
cd u-boot

# 清除旧编译输出
make clean

# 根据板卡配置文件名称进行编译，配置文件名称为 xxx_defconfig，则编译为./make.sh xxx
# 以 rk3568 系列为例
./make.sh lubancat-rk3568-pcie
```

编译成功后在 uboot 源码顶层文件夹生成名为 `uboot.img` 的文件，该文件就是我们需要的支持 PCIE 的 uboot 镜像文件。



28.3.1.2 解包发布镜像并重新打包

为了方便我们不重新编译镜像，而是使用野火发布的通用镜像进行解包，解包后替换并修改一些组件，再重新打包成完整镜像，从而得到支持固态启动镜像包。

28.3.1.2.1 解包镜像

> **注意：** 解包镜像需要在 Linux 环境使用 Linux_Pack_Firmware 根据进行解包，不要在 windows 解包。

访问资料网盘/6-开发软件/，获取 Linux_Pack_Firmware_v2.29.zip

将 Linux_Pack_Firmware_v2.29.zip 解压到 Linux PC 或虚拟机中。

```
# 解压
unzip Linux_Pack_Firmware_v2.29.zip
```

访问资料网盘获取需要的系统镜像，传到 Linux_Pack_Firmware/rockdev 目录下。

以 LubanCat-2 的 Debian xfce 镜像为例：

```
# 进入工具目录
cd Linux_Pack_Firmware/rockdev/

# 解压镜像，根据实际压缩包名称而定
7z x lubancat-rk3568-debian10-xfce-20250303_update.7z

# 重命名镜像为 update.img，根据实际镜像名称而定
mv lubancat-rk3568-debian10-xfce-20250303_update.img update.img

# 解包镜像
./unpack.sh

```

(下页继续)

(续上页)

```
# 解包镜像的到 output 目录
# 查看 output 目录
ls output
# 信息输出如下
boot.img  MiniLoaderAll.bin  package-file  parameter.txt  rootfs.img  uboot.
↪img
```

28.3.1.2.2 替换和修改镜像组件

替换 uboot.img

将前面编译 uboot 源码生成的 uboot.img 替换解包生成 uboot.img

```
# 替换 uboot.img，根据自己的实际路径确定
cp -f  ~/LubanCat_Linux_rk356x_SDK/u-boot/uboot.img ~/Linux_Pack_Firmware/
↪rockdev/output/
```

修改 boot.img

由于通用镜像默认没有支持引导启动 PCIE 配置，因此需要进行修改，引导配置位于 boot.img 中的 boot.scr，又因为 boot.scr 是由 boot.cmd 生成的，所以我们要修改的是 boot.img 中的 boot.cmd，然后生成 boot.scr。

```
# 进入解包输出目录
cd Linux_Pack_Firmware/rockdev/output/

# 挂载 boot.img
sudo mount boot.img /mnt

# 修改 boot.cmd
sudo vi /mnt/boot.cmd
```

在 boot.cmd 的中添加或修改为以下高亮部分：

列表 2: boot.cmd

```
1  echo [boot.cmd] run boot.cmd scripts ...;
2
3  pcie_boot=1
4
5  if test "$pcie_boot" = "1"; then
6      pci enum
7      nvme scan
8      nvme device 0
9
10     if test $? -eq 0; then
11         setenv devtype nvme
12         setenv devnum 0
13         setenv rootfspart /dev/nvme0n1p3
14
15         if test ! -e ${devtype} ${devnum}:${distro_bootpart} /rk-kernel.dtb;
   ↪ then
16             rockusb 0 nvme 0
17         fi
18
19     else
20         setenv rootfspart /dev/mmcblk${devnum}p3
21     fi
22 else
23     setenv rootfspart /dev/mmcblk${devnum}p3
24 fi
25
26 if test -e ${devtype} ${devnum}:${distro_bootpart} /uEnv/uEnv.txt; then
27
28     echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_
   ↪r} /uEnv/uEnv.txt ...;
```

(下页继续)

(续上页)

```
29      load ${devtype} ${devnum}:${distro_bootpart} ${env_addr_r} /uEnv/uEnv.
    ↪txt;
30
31      echo [boot.cmd] Importing environment from ${devtype} ...
32      env import -t ${env_addr_r} 0x8000
33
34      setenv bootargs ${bootargs} root=${rootfspart} boot_part=${distro_
    ↪bootpart} ${cmdline}
35      printenv bootargs
36
37      echo [boot.cmd] load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_
    ↪addr_r} /initrd-${uname_r} ...
38      load ${devtype} ${devnum}:${distro_bootpart} ${ramdisk_addr_r} /initrd-$
    ↪{uname_r}
39
40      echo [boot.cmd] loading ${devtype} ${devnum}:${distro_bootpart} $
    ↪{kernel_addr_r} /Image-${uname_r} ...
41      load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} /Image-$
    ↪{uname_r}
42
43      echo [boot.cmd] loading default rk-kernel.dtb
44      load ${devtype} ${devnum}:${distro_bootpart} ${fdt_addr_r} /rk-kernel.
    ↪dtb
45
46      fdt addr  ${fdt_addr_r}
47      fdt set /chosen bootargs
48
49      echo [boot.cmd] dtoverlay from /uEnv/uEnv.txt
50      setenv dev_bootpart ${devnum}:${distro_bootpart}
51      dtfile ${fdt_addr_r} ${fdt_over_addr}  /uEnv/uEnv.txt ${env_addr_r}
52
53      echo [boot.cmd] [${devtype} ${devnum}:${distro_bootpart}] ...
```

(下页继续)

(续上页)

```
54      echo [boot.cmd] [booti] ...
55      booti ${kernel_addr_r} ${ramdisk_addr_r} ${fdt_addr_r}
56  fi
57
58  echo [boot.cmd] run boot.cmd scripts failed ...;
59
60  # Recompile with:
61  # mkimage -C none -A arm -T script -d /boot/boot.cmd /boot/boot.scr
```

以上修改的意思是：

- 第 3-5 行，判断 pcie_boot 变量是否为 1，如果为 1，则支持从 pcie 启动，如果为 0 则恢复默认启动方式从 eMMC/SD 启动。
- 第 6-8 行，扫描 pcie 设备。
- 第 10-13 行，如果扫描到 pcie 设备则设置启动类型为 nvme，设备号为 0，文件系统分区为/dev/nvme0n1p3
- 第 15-17 行，判断 pcie 硬盘 boot 分区是否存在 rk-kernel.dtb，如果不存在则说明 pcie 固态中没有镜像，引导进入烧录模式，可将镜像直接烧录到 pcie 固态。

修改完成 boot.cmd 和保存退出，然后执行以下命令生成 boot.scr

```
1  sudo mkimage -C none -A arm -T script -d /mnt/boot.cmd /mnt/boot.scr
```

由于 uboot 启动读取的内核设备树是 boot.img 的 `rk-kernel.dtb`，默认是通用设备树 (可能没有开启 PCIE 接口)，系统第一次启动后会软连接到 dtb 目录下实际的设备树。

因此，需要将 dtb 目录下实际的设备树替换 `rk-kernel.dtb` 从而使第一次启动时启用 PCIE 接口，以下以 rk3568-lubancat-2-v3 为例：

```
1  # 前面已经将 boot.img 挂载到/mnt
2
3  # 替换 rk-kernel.dtb，根据板卡实际设备树而定
4  sudo cp -f /mnt/dtb/rk3568-lubancat-2-v3.dtb /mnt/rk-kernel.dtb
```

(下页继续)

(续上页)

```
5
6 # 替换完成后卸载/mnt 即可
7 sudo umount /mnt/
```

修改 rootfs.img

由于 rk356x 系列的 4.19 内核对应的 Debian10 镜像默认没有支持 pcie 系统初始化，因此需要进行修改，系统初始化文件位于 rootfs.img 的 `/etc/init.d/boot_init.sh`。

```
1 # 进入解包输出目录
2 cd Linux_Pack_Firmware/rockdev/output/
3
4 # 挂载 rootfs.img
5 sudo mount rootfs.img /mnt
6
7 # 修改 boot_init.sh
8 sudo vi /mnt/etc/init.d/boot_init.sh
```

修改内容如下：

```
1 #Boot_Part="${Root_Part::-2}"p2
2 Boot_Part="${Root_Part::-1}"2
```

修改完成如下图所示：

```
if [ ! -e "/boot/boot_init" ] ; then
        if [ ! -e "/dev/disk/by-partlabel/userdata" ] ; then
                if [ ! -L "/boot/rk-kernel.dtb" ] ; then
                        for x in $(cat /proc/cmdline); do
                                case $x in
                                root=*)
                                        Root_Part=${x#root=}
                                        #Boot_Part="${Root_Part::-2}"p2
                                        Boot_Part="${Root_Part::-1}"2
                                        ;;
                                esac
                        done

                        mount "$Boot_Part" /boot
                        echo "$Boot_Part  /boot  auto  defaults  0 2" >> /etc/fstab
                fi

                service lightdm stop || echo "skip error"

                apt install -fy --allow-downgrades /boot/kerneldeb/* || true
                apt-mark hold linux-headers-$(uname -r) linux-image-$(uname -r) || true

-- INSERT --                                                    238,43-57      94%
```

修改完成后卸载 rootfs.img

```
# 卸载前面挂载 rootfs.img
sudo umount /mnt
```

28.3.1.2.3 打包镜像

替换 uboot.img、修改 boot.img 和 rootfs.img 后即可打包成支持固态启动镜像包。

```
# 进入输出目录
cd Linux_Pack_Firmware/rockdev/output

# 打包镜像，根据实际芯片名称而定

#rk356x 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3568 MiniLoaderAll.bin update_tmp.img update.img -os_
↪type:androidos || pause

#rk3576 打包执行
../afptool -pack ./ update_tmp.img || pause
```

(下页继续)

(续上页)

```
../rkImageMaker -RK3576 MiniLoaderAll.bin update_tmp.img update.img -os_type:androidos || pause

#rk3588/rk3588s 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3588 MiniLoaderAll.bin update_tmp.img update.img -os_type:androidos || pause
```

打包后在 output 当前目录生成 `update_tmp.img` 和 `update.img` 镜像文件，其中 `update.img` 就是最终打包生成的支持引导 M.2 固态启动的镜像。

28.3.2 烧录镜像

制作完成支持引导 M.2 固态启动的 `update.img` 镜像后，可将 `update.img` 烧录至 eMMC/SD 以及 M.2 固态。

28.3.2.1 烧录镜像到 eMMC/SD

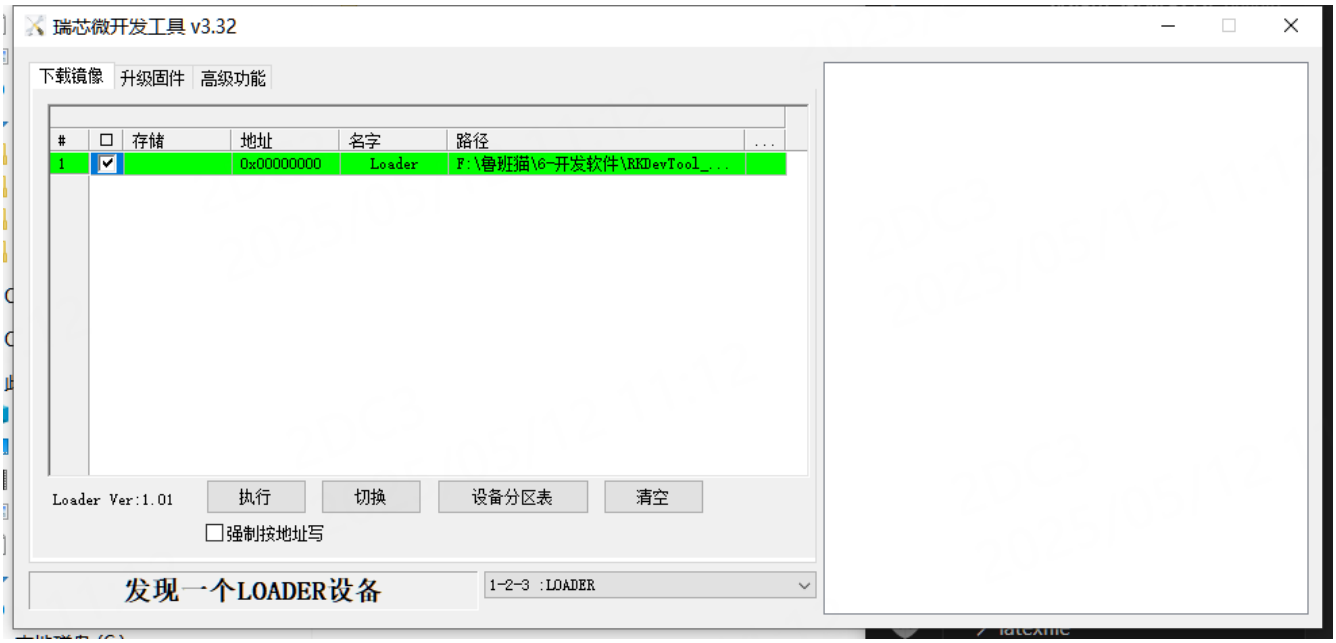
因为需要 eMMC/SD 作为引导，因此需要先将镜像烧录到 eMMC/SD，如何烧录系统到 eMMC/SD 参考以下文档：

- eMMC 镜像烧录参考：完整镜像烧录 以及各板卡的 快速使用手册
- SD 镜像烧录参考：SD 卡启动镜像烧录 ，注意选择功能模式勾选的是 **SD 启动**。

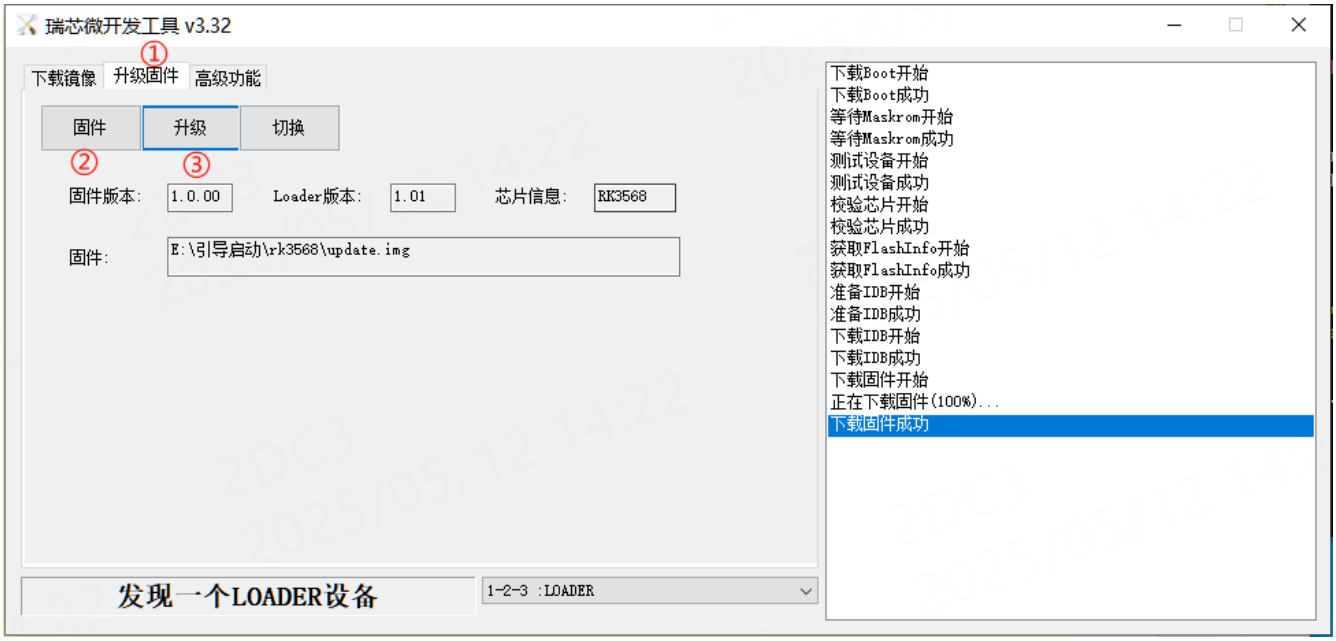
28.3.2.2 烧录镜像到 M.2 固态

将镜像烧录到 eMMC 或 SD 卡后，再接入 M.2 固态硬盘，启动时会扫描 pcie 设备并判断 pcie 设备中是否存在镜像，如果 pcie 设备中的 boot 分区不存在 rk-kernel.dtb，则判断为不存在镜像，会引导进入烧录模式。

当 M.2 硬盘不存在镜像时，usb 烧录口接入 usb 线会识别到 `LOADER` 设备，如下图所示：



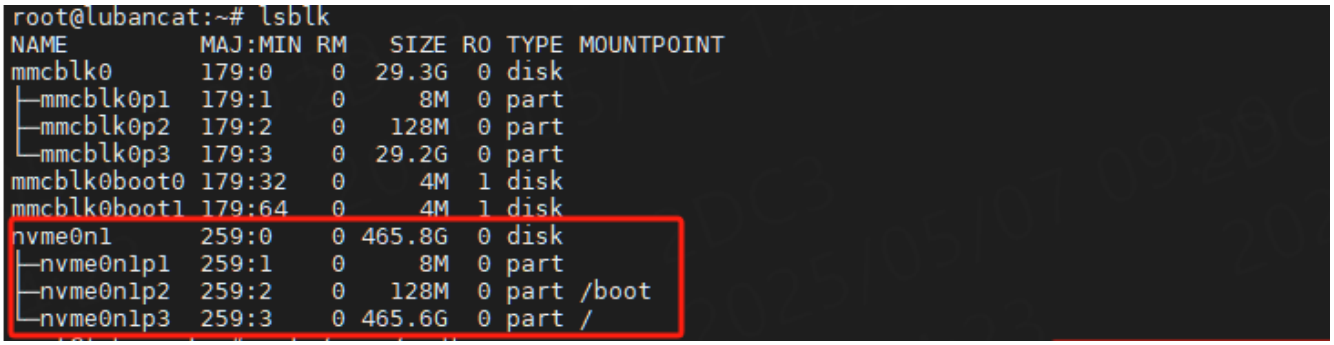
选择 升级固件界面，固件选择前面生成的 `update.img` ，最后点击烧录即可，如下图所示：



烧录完成后，会自动重启，系统启动成功后，执行以下命令查看是否成功从 M.2 固态启动：

```
# 查看系统挂载
lsblk
```

引导成功效果如下图所示，其中/dev/nvme0n1p2 为 M.2 固态的 boot 分区，/dev/nvme0n1p3 为 M.2 固态的 rootfs 分区。



28.3.2.3 重新烧录镜像到 M.2

M.2 硬盘带镜像时不会进入烧录模式，如果需要重新烧录 M.2 硬盘，此时需要删除判断文件或者手动引导至烧录模式。

28.3.2.3.1 通过删除判断文件方式

因为启动会判断 M.2 固态中 boot 分区是否存在 rk-kernel.dtb，如果不存在 rk-kernel.dtb，则会引导进入烧录模式。

因此，如果需要重新烧录 M.2 硬盘，可删除 rk-kernel.dtb，M.2 启动系统后执行以下命令删除：

```
# 删除判断文件
sudo rm /boot/rk-kernel.dtb

# 重启
sudo reboot
```

USB 烧录口连接电脑，识别到 LOADER 设备后即可重新烧录镜像到 M.2 固态。

28.3.2.3.2 通过手动引导方式

在接入 M.2 硬盘后，板卡 Debug 接口接串口模块，然后板卡上电，在上电过程中快速按下 ctrl + c 进入 uboot 命令行模式，在 uboot 命令行模式输入以下命令：

```
# 扫描设备
pci enum
nvme scan

# 进入烧录模式
rockusb 0 nvme 0
```

USB 烧录口连接电脑，识别到 LOADER 设备后即可重新烧录镜像到 M.2 固态。

28.4 制作精简的引导镜像

前面小节打包出一个完整的镜像，可用于烧录 sd/emmc 或固态硬盘，但 sd/emmc 仅用做引导使用，还烧录完全镜像就显得十分多余且费时，如果不考虑固态硬盘出现问题也能靠 sd/emmc 继续正常运行系统，那么可以裁剪出一个精简镜像烧录到 sd/emmc 用作引导。

以 rk3588 的 Debian12 系统解包为例：

将解包出的 Linux_Pack_Firmware/rockdev/output/package-file 删除 rootfs，修改完成如下：

```
package-file    package-file
parameter    parameter.txt
bootloader    MiniLoaderAll.bin
uboot    uboot.img
boot    boot.img
```

将解包出的 Linux_Pack_Firmware/rockdev/output/parameter.txt 删除 rootfs，修改完成如下：

```
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3588
```

(下页继续)

(续上页)

```
MACHINE_ID: 007
MANUFACTURER: RK3588
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
GROW_ALIGN: 0
CMDLINE: mtdparts=:0x00004000@0x00004000(uboot),
↪0x00040000@0x00008000(boot:bootable)
```

修改完成后重新打包镜像

```
# 进入输出目录
cd Linux_Pack_Firmware/rockdev/output

# 打包镜像，根据实际芯片名称而定

#rk356x 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3568 MiniLoaderAll.bin update_tmp.img update.img -os_
↪type:androidos || pause

#rk3576 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3576 MiniLoaderAll.bin update_tmp.img update.img -os_
↪type:androidos || pause

#rk3588/rk3588s 打包执行
../afptool -pack ./ update_tmp.img || pause
../rkImageMaker -RK3588 MiniLoaderAll.bin update_tmp.img update.img -os_
↪type:androidos || pause
```

打包后在 output 当前目录生成 `update_tmp.img` 和 `update.img` 镜像文件，其中 `update.img` 就是最终打包生成去掉了根文件系统的精简的引导镜像。

当然，如果熟悉 uboot 可直接修改 uboot 源码，在 uboot 源码中添加引导固态启动的内容，编译得到的 uboot.img 往往只有几 MB，仅需烧录 uboot.img 到 sd/emmc，本教程不作展开。

版权说明