OpenHarmony 基础实验

实验目的

- 了解开源鸿蒙整体框架并尝试使用开源鸿蒙
- 了解交叉编译的基本概念与用法
- 学会通过交叉编译和添加子系统的方式修改开源鸿蒙来提供相关功能

实验环境

- OS:
  - 烧录：Windows 10 / 11
  - 编译：Ubuntu 22.04.4 LTS
- Platform : VLab / VMware / VirtualBox

实验时间安排

> 注：此处为实验发布时的安排计划，请以课程主页和课程群内最新公告为准

- 4.26 晚实验课，讲解实验、检查实验
- 5.3 假期期间，无实验课及实验检查
- 5.10 晚实验课，检查实验
- 5.17 晚实验课，检查实验

友情提示/为什么要做这个实验?

- **本实验难度不高，目的是让大家体会当前开源国产操作系统的前沿发展成果，同时让大家尝试一下手动配置 OpenHarmony 的方法，并且为后续的鸿蒙选做实验打下基础。**
- 如果同学们遇到了问题，请先查询在线文档。在线文档地址：
  https://docs.qq.com/sheet/DU3FMRE5UQU5YRlZa

文档更新说明

2024年4月29日

- [ALL] 优化部分文档格式；
- [3.1.1] 修改了两处 `wget` 的参数错误；
- [3.1.1] 增加了更详细的代码下载说明；
- [3.1.3] 增加了关于重新安装 MiniConda 的说明；
- [3.1.3] 增加依赖库 `libstdc++-11-dev` ；
- [3.2.3] 增加关于 HDC 文件传输目录的说明；
- [3.3.5] 增加关于配置文件修改的说明；

第一部分 开源鸿蒙以及润和DAYU200开发板介绍

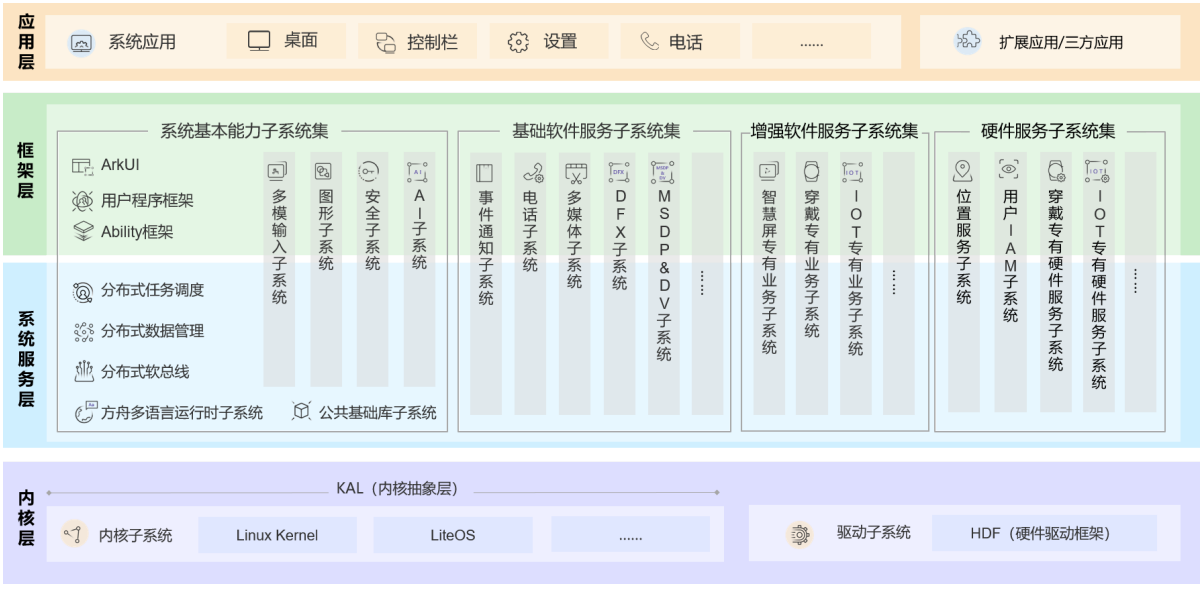
1.1 开源鸿蒙简介

1.1.1 整体简介

OpenHarmony是由开放原子开源基金会（OpenAtom Foundation）孵化及运营的开源项目，目标是面向全场景、全连接、全智能时代，基于开源的方式，搭建一个智能终端设备操作系统的框架和平台，促进万物互联产业的繁荣发展。

技术架构

OpenHarmony整体遵从分层设计，从下向上依次为：内核层、系统服务层、框架层和应用层。系统功能按照“系统 > 子系统 > 组件”逐级展开，在多设备部署场景下，支持根据实际需求裁剪某些非必要的组件。OpenHarmony技术架构如下所示：



内核层

- 内核子系统：采用多内核（Linux内核或者LiteOS）设计，支持针对不同资源受限设备选用适合的OS内核。内核抽象层（KAL，Kernel Abstract Layer）通过屏蔽多内核差异，对上层提供基础的内核能力，包括进程/线程管理、内存管理、文件系统、网络管理和外设管理等。
- 驱动子系统：驱动框架（HDF）是系统硬件生态开放的基础，提供统一外设访问能力和驱动开发、管理框架。

系统服务层

系统服务层是OpenHarmony的核心能力集合，通过框架层对应用程序提供服务。该层包含以下几个部分：

- 系统基本能力子系统集：为分布式应用在多设备上的运行、调度、迁移等操作提供了基础能力，由分布式软总线、分布式数据管理、分布式任务调度、公共基础库、多模输入、图形、安全、AI等子系统组成。
- 基础软件服务子系统集：提供公共的、通用的软件服务，由事件通知、电话、多媒体、DFX（Design For X）等子系统组成。
- 增强软件服务子系统集：提供针对不同设备的、差异化的能力增强型软件服务，由智慧屏专有业务、穿戴专有业务、IoT专有业务等子系统组成。

- 硬件服务子系统集：提供硬件服务，由位置服务、用户IAM、穿戴专有硬件服务、IoT专有硬件服务等子系统组成。

根据不同设备形态的部署环境，基础软件服务子系统集、增强软件服务子系统集、硬件服务子系统集内部可以按子系统粒度裁剪，每个子系统内部又可以按功能粒度裁剪。

框架层

框架层为应用开发提供了C/C++/JS等多语言的用户程序框架和Ability框架，适用于JS语言的ArkUI框架，以及各种软硬件服务对外开放的多语言框架API。根据系统的组件化裁剪程度，设备支持的API也会有所不同。

应用层

应用层包括系统应用和第三方非系统应用。应用由一个或多个FA（Feature Ability）或PA（Particle Ability）组成。其中，FA有UI界面，提供与用户交互的能力；而PA无UI界面，提供后台运行任务的能力以及统一的数据访问抽象。基于FA/PA开发的应用，能够实现特定的业务功能，支持跨设备调度与分发，为用户提供一致、高效的应用体验。

1.1.2 子系统

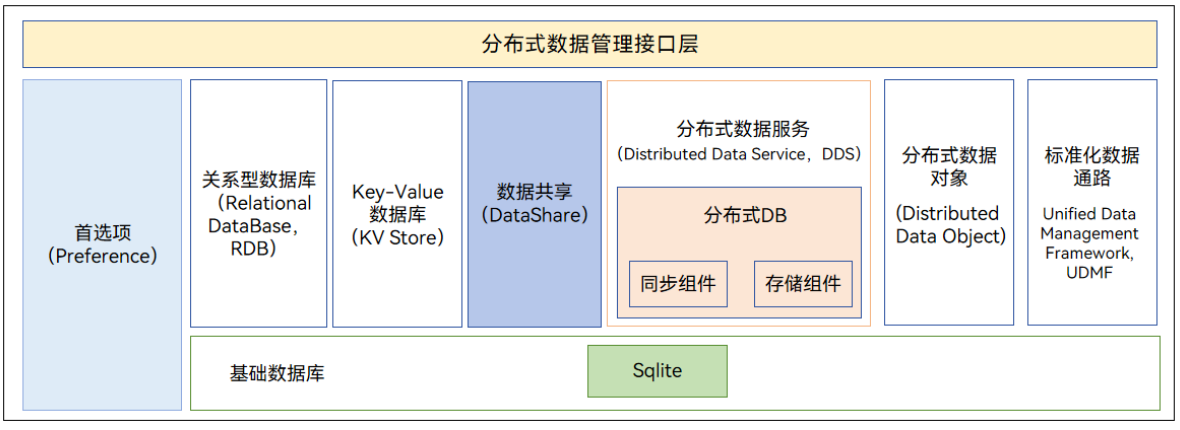
OpenHarmony整体遵从分层设计，从下向上依次为：内核层、系统服务层、框架层和应用层。系统功能按照“系统 > 子系统 > 组件”逐级展开，在多设备部署场景下，支持根据实际需求裁剪某些非必要的组件。子系统是一个逻辑概念，它具体由对应的组件构成。

这里用分布式数据管理子系统作为例子进行分析：

子系统介绍

分布式数据管理子系统支持单设备的各种结构化数据的持久化，以及跨设备之间数据的同步、共享功能。开发者通过分布式数据管理子系统，能够方便地完成应用程序数据在不同终端设备间的无缝衔接，满足用户跨设备使用数据的一致性体验。

子系统架构



子系统1-2层目录描述

```
distributeddatamgr/           # 子系统目录
├── data_object               # 分布式数据对象目录
└── data_share                # 数据共享目录
└── datamgr_service           # 数据服务目录
└── kv_store                  # Key-Value数据库目录
└── preferences               # 首选项目录
└── relational_store          # 关系型数据库目录
└── udmf                      # 标准化数据通路目录

third_party/                  # 开源软件目录
├── bounds_checking_function # bounds_checking_function代码目录
```

```
└──── cJSON                    # cJSON代码目录
└──── flatbuffers              # flatbuffers代码目录
└──── googletest               # googletest代码目录
└──── jsoncpp                  # jsoncpp代码目录
└──── icu                      # icu代码目录
└──── libuv                    # libuv代码目录
└──── openssl                  # openssl代码目录
└──── sqlite                   # SQLite代码目录
└──── zlib                     # zlib代码目录
```

1.1.3 组件

组件是对子系统的进一步拆分，可复用的软件单元，它包含源码、配置文件、资源文件和编译脚本；能独立构建，以二进制方式集成，具备独立验证能力的二进制单元。

这里同样用分布式数据管理子系统中的组件来说明，该包含以下组件（选列）：

- 分布式数据对象
- 数据共享
- 分布式数据服务
- 标准化数据通路
- Key-Value数据库
- ……

这个例子是想表明一个子系统由很多组件组成，各个组件代码所在目录一般为子系统所在目录的子目录。一个子系统调用下一层的接口来实现功能，并向上层提供对应的接口，比如分布式数据管理子系统调用内核层的接口，并提供接口给框架层使用。

1.2 DAYU200开发板

1.2.1 DAYU200开发板介绍

> 关于开发板的更多信息可以查看链接 **润和HH-SCDAYU200开发套件**

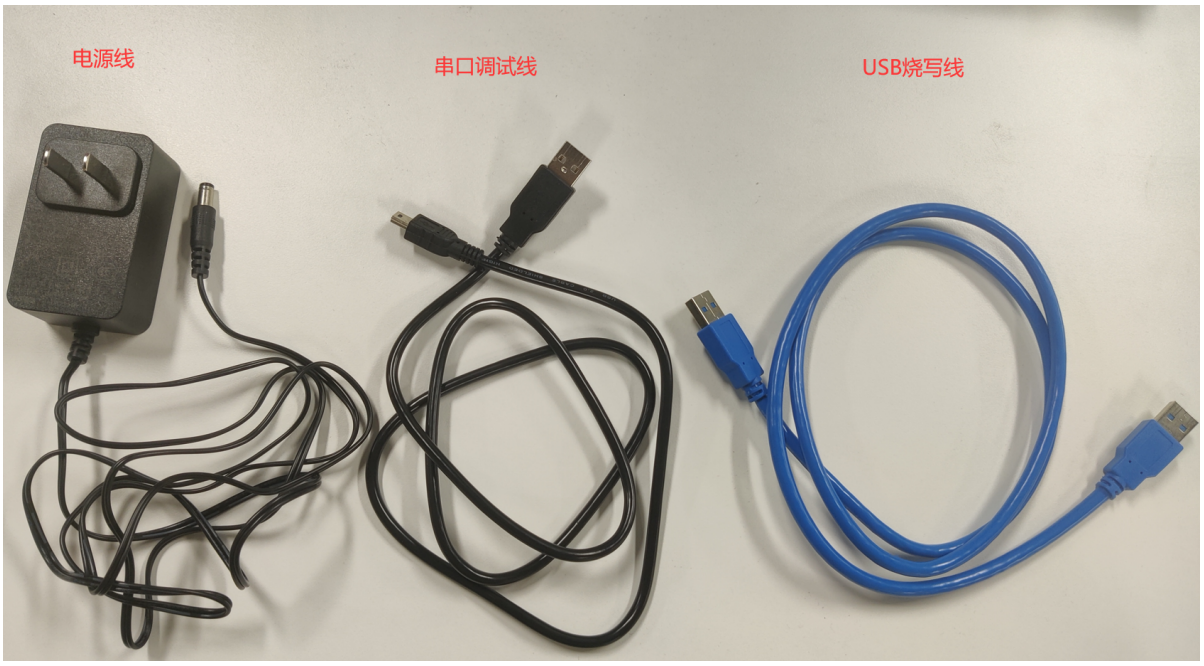
我们使用的开发板型号为DAYU200开发板，其基于Rockchip RK3568芯片，集成双核心架构GPU以及高效能NPU；板载四核64位Cortex-A55 处理器采用22nm先进工艺，主频高达2.0GHz；支持蓝牙、Wi-Fi、音频、视频和摄像头等功能，拥有丰富的扩展接口，支持多种视频输入输出接口；配置双千兆自适应RJ45以太网口，可满足NVR、工业网关等多网口产品需求。

RK3568芯片基于Rockchip，集成双核心架构GPU以及高效能NPU；搭载四核64位Cortex-A55处理器，采用22nm先进工艺，主频高达2.0GHz；支持蓝牙、Wi-Fi、音视频和摄像头等功能，有丰富的扩展接口，支持多种视频输入输出接口；配置双千兆 RJ45 以太网口，可以满足各类实际开发需求。

1.2.2 开发板管理和使用规范

开发板的使用采用分组负责人制度。即三人一组，由一人担任负责人并且保管开发板，发放和回收开发板向负责人进行。在发放的开发板套件中，包含以下物品，请注意保管:

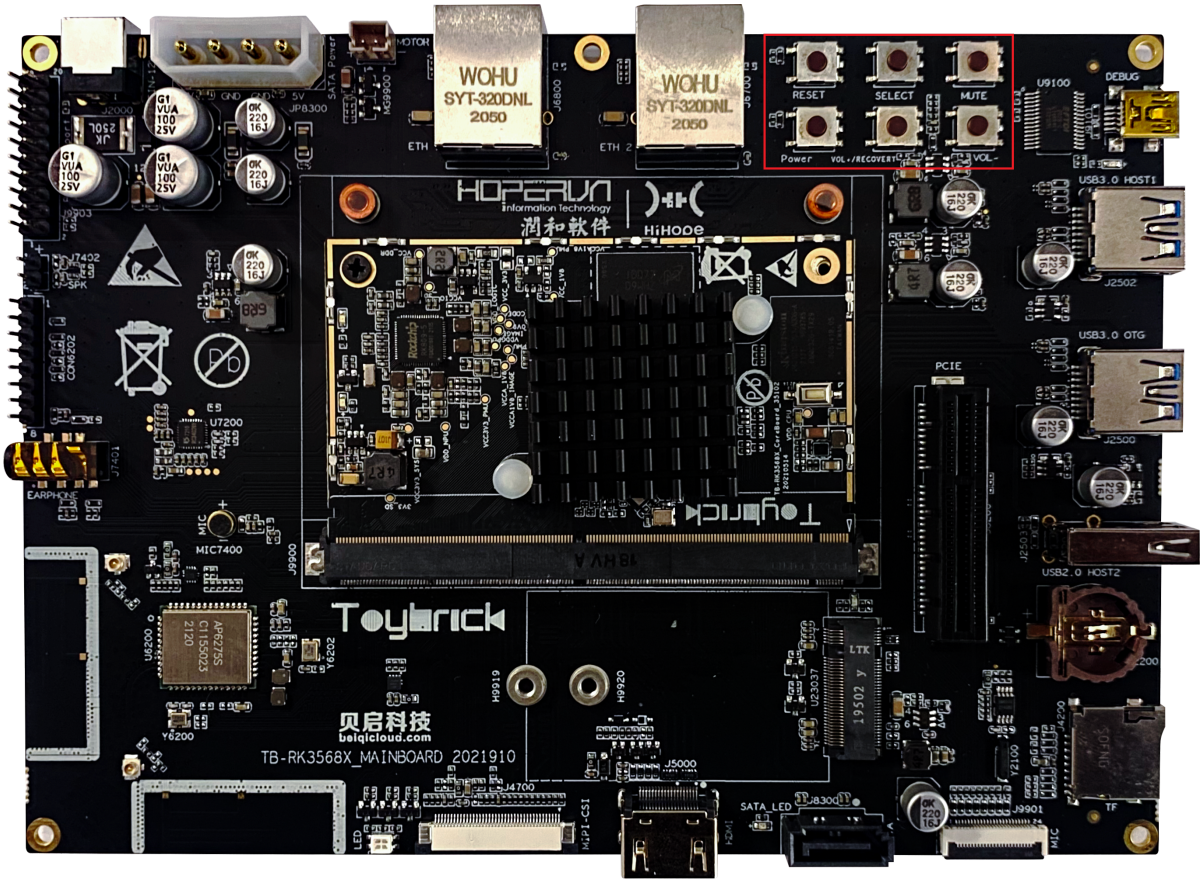
1. DAYU200开发板一块；
2. DAYU200电源适配线一条；
3. 公对公USB数据线一条；（用于烧写）；
4. mini USB B数据线一条（用于串口调试）；



1.2.2 开发板的使用

当接上电源后，开发板一般会自行启动，如果没有启动请查看开发板上的按钮，根据按钮的功能进行尝试打开。

当开发板开机后运行的是Openharmony3.0版本的全量系统，较为卡顿，后面实验我们将教大家如何烧录Openharmony4.0版本的全量系统。



第二部分 开源鸿蒙镜像的烧录

在这一部分，我们将介绍 OpenHarmony 镜像的烧录。为了方便大家体验开发版，助教给大家准备了已经编译好的完整版 OpenHarmony 镜像，可以直接烧录到开发板上。你也可以先完成下一部分的编译任务后，再将自己编译好的镜像烧录至开发板。

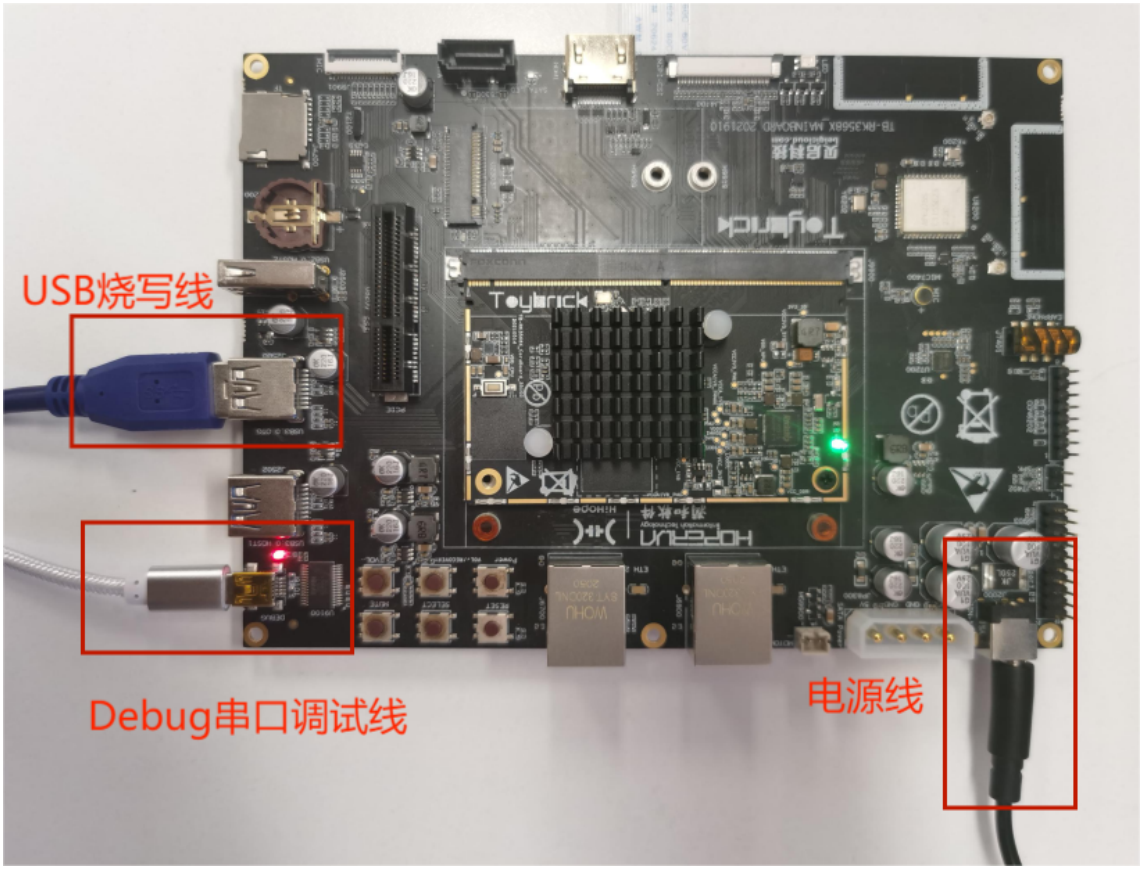
2.1 OH烧录

RK3568 开发板的烧录软件目前**只支持 Windows 系统**，请在 Windows 系统下进行本章节的步骤。

2.1.1烧录前准备

1.连接开发板

- 按照下图提示连接**电源线**、 **Debug串口调试线**和**USB烧写线**



2.下载烧录工具和驱动

- **DAYU200 烧录工具**
  下载后解压到文件夹即可

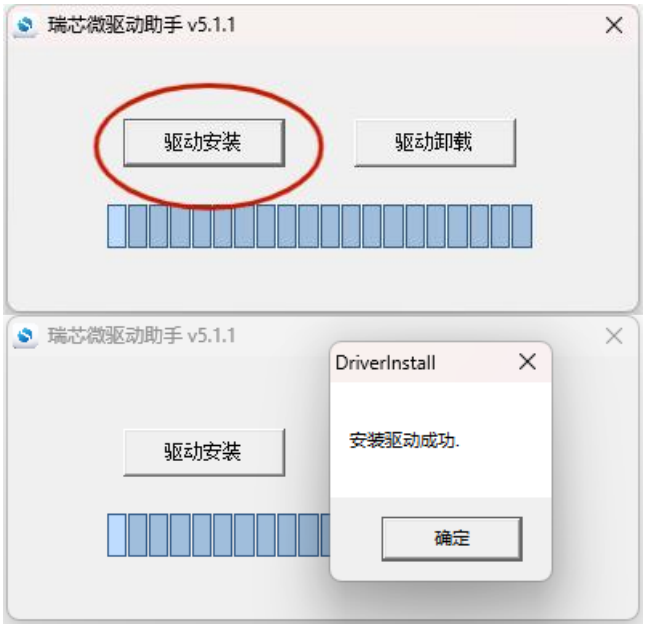
3.准备编译镜像

- 下载Openharmony4.0全量系统镜像。镜像下载地址：
  - 推荐链接： https://git.ustc.edu.cn/KONC/oh_lab/-/blob/main/RK3568_full_images.zip
  - 备用链接： https://rec.ustc.edu.cn/share/f6330c40-df7c-11ee-b2da-8d6fb2396f2a

2.1.2烧录步骤

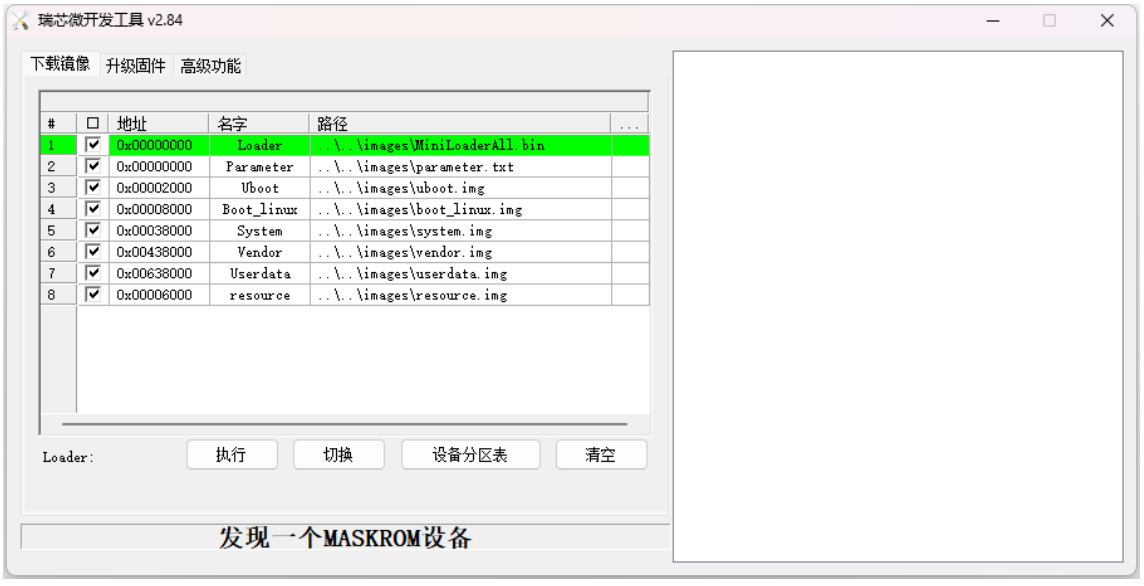
1. 安装USB驱动

- 双击烧录工具中 `windows\DriverAssitant_v5.1.1\DriverInstall.exe` 打开安装程序，点击下图所示驱动安装按钮:



2. 打开烧写工具

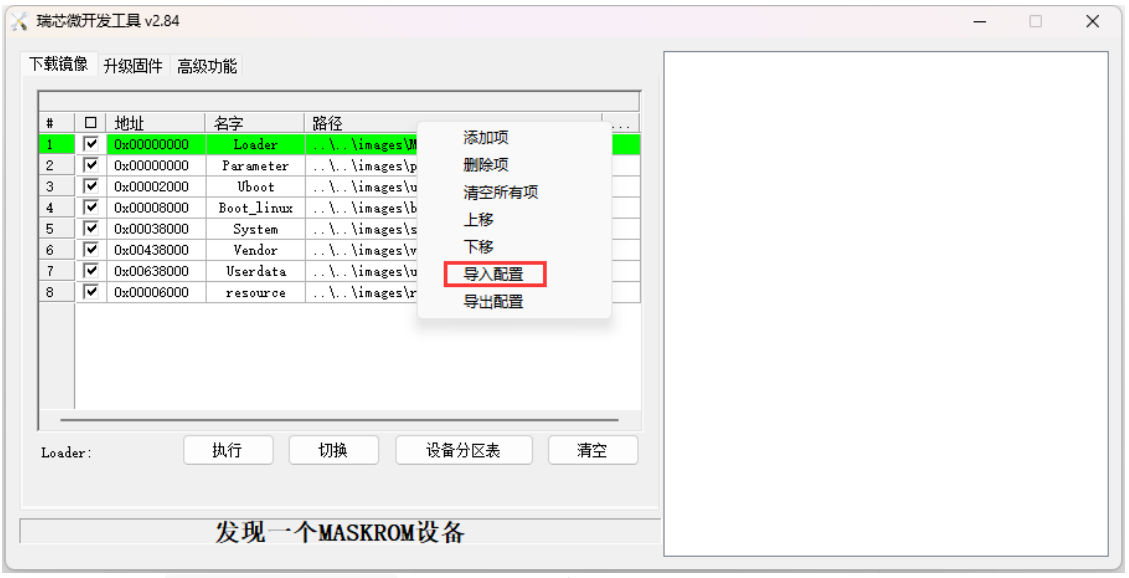
- 双击烧录工具中 `windows\RKDevTool.exe` 打开烧写工具， 如图所示， 默认是 Maskrom 模式:



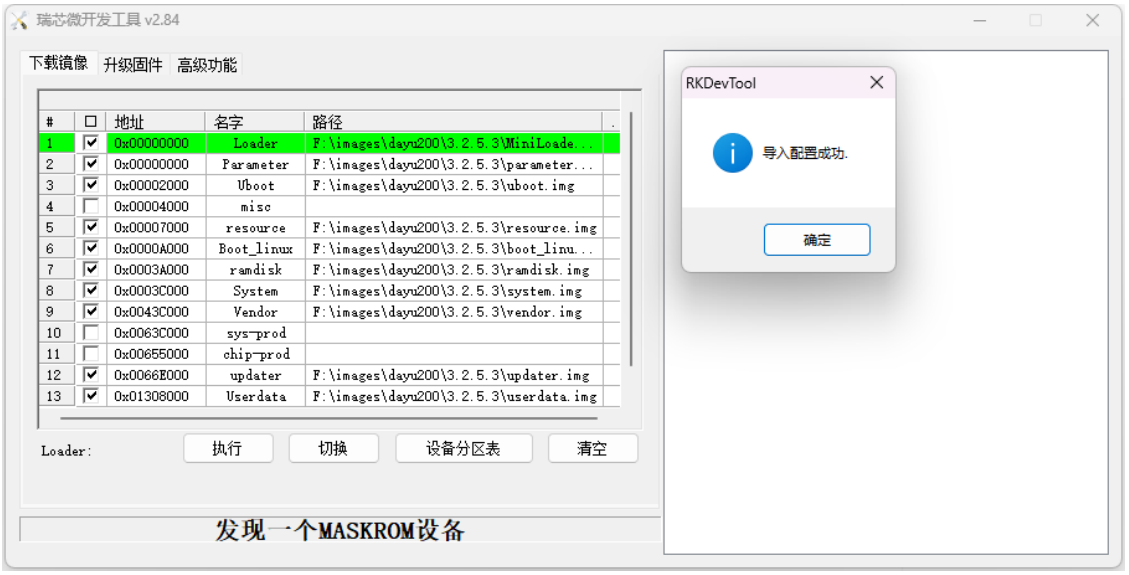
- 导入编译文件镜像包中的 `config.cfg` 配置，路径为 `images\config.cfg` (如果你下载的备用链接的镜像，则目录为 `rk3568_packages\phone\images\config.cfg`)

> 注意导入 `config.cfg` 配置为必须操作，尤其是后续重新编译镜像后，不能简单的替换文件夹的文件，否则会烧录失败

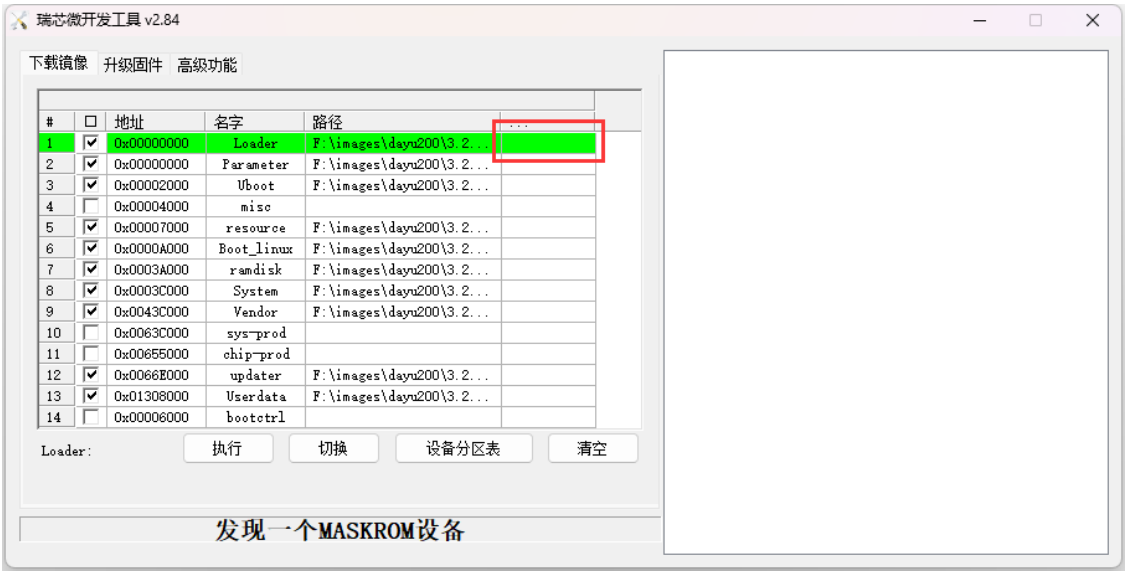
1. 右击导入配置

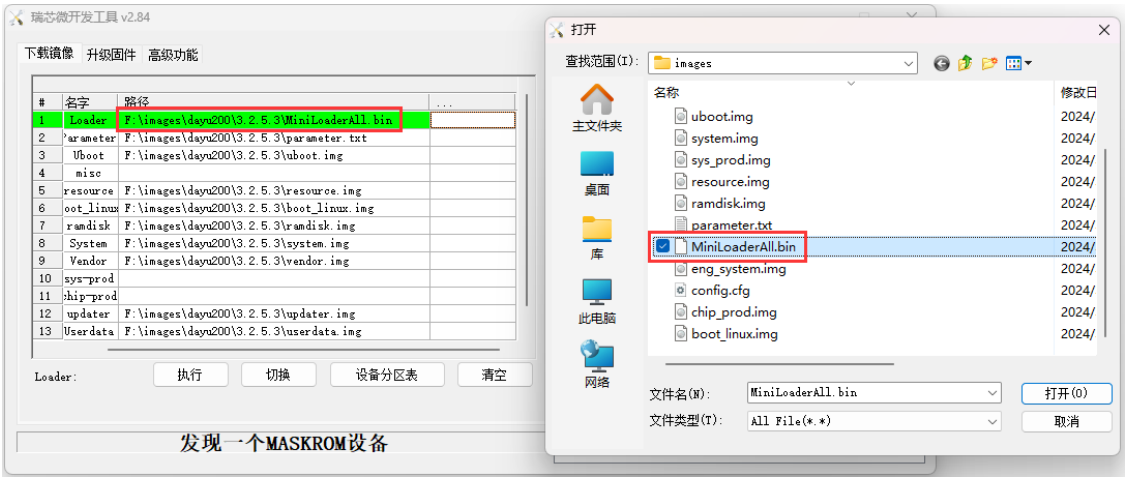


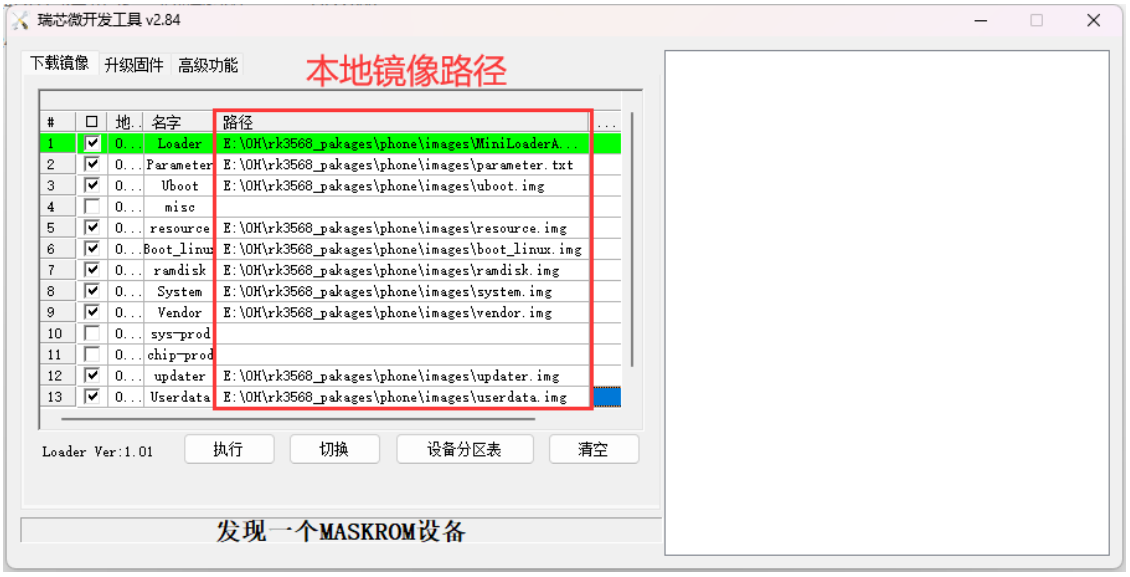
2. 选中配置文件 `images\config.cfg` (如果你下载的备用链接的镜像，则目录为 `rk3568_packages\phone\images\config.cfg`)



3. 点击空白栏，逐一配置对应镜像路径(也可以双击路径，手动修改路径)

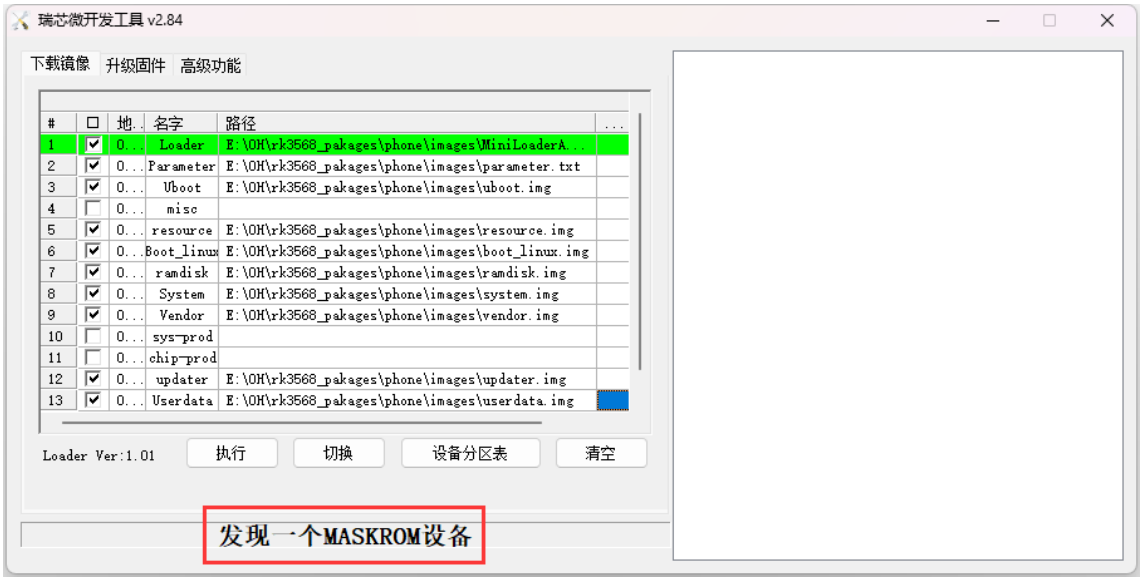




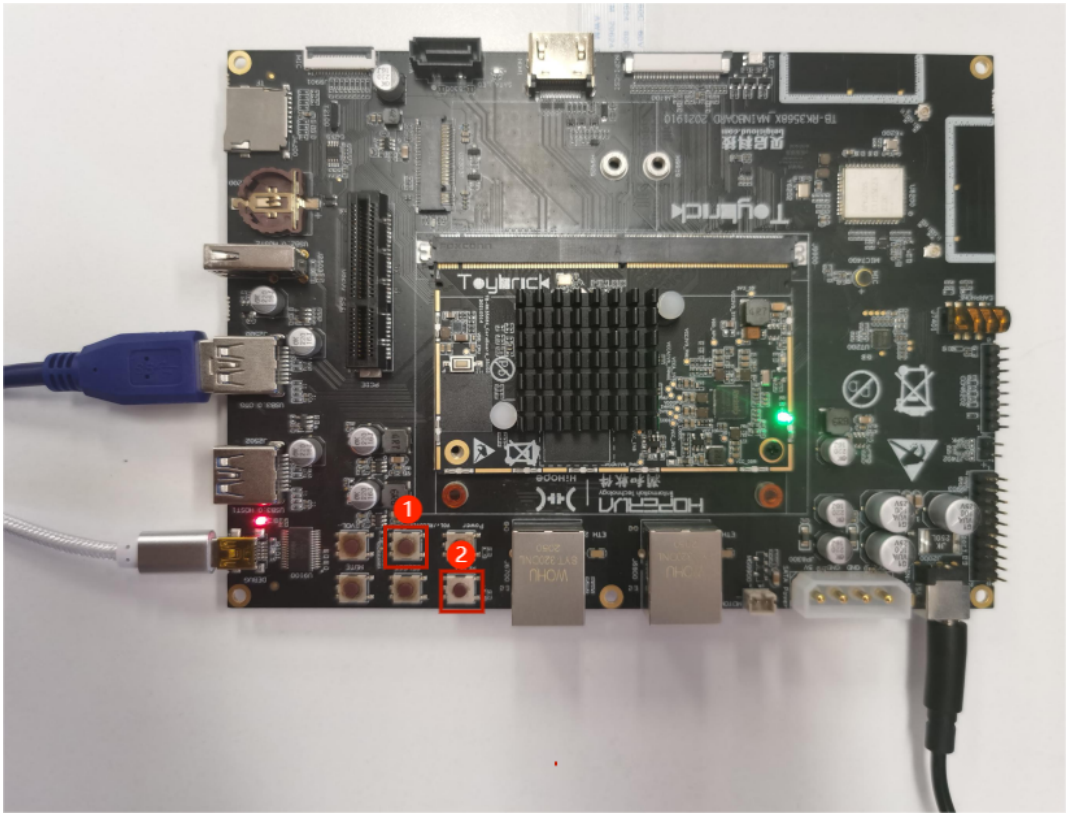


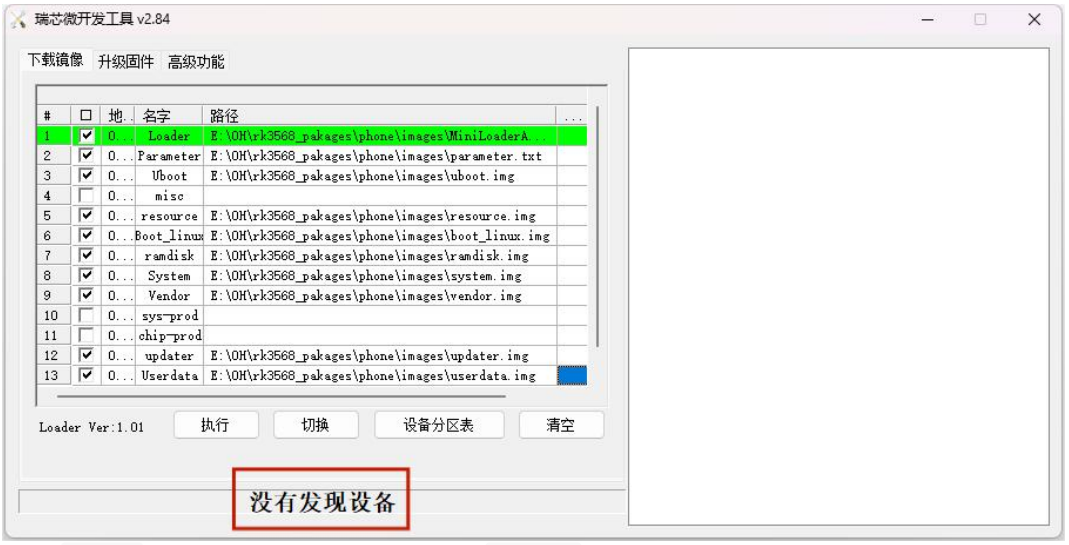
3. 进入LOADER烧写模式

- 默认烧写工具是 `Maskrom` 模式

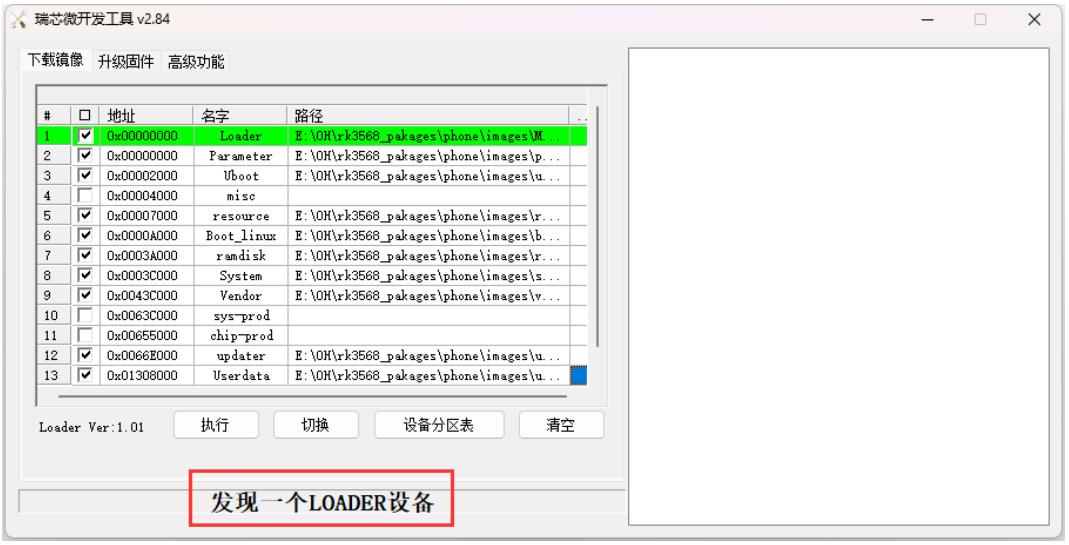


- 进入 `LOADER` 烧写模式
    1. 按住 `VOL-/RECOVERY` 按键（图中标注的①号键）和 `RESET` 按钮（图中标注的②号键）不松开，烧录工具此时显示“没有发现设备”

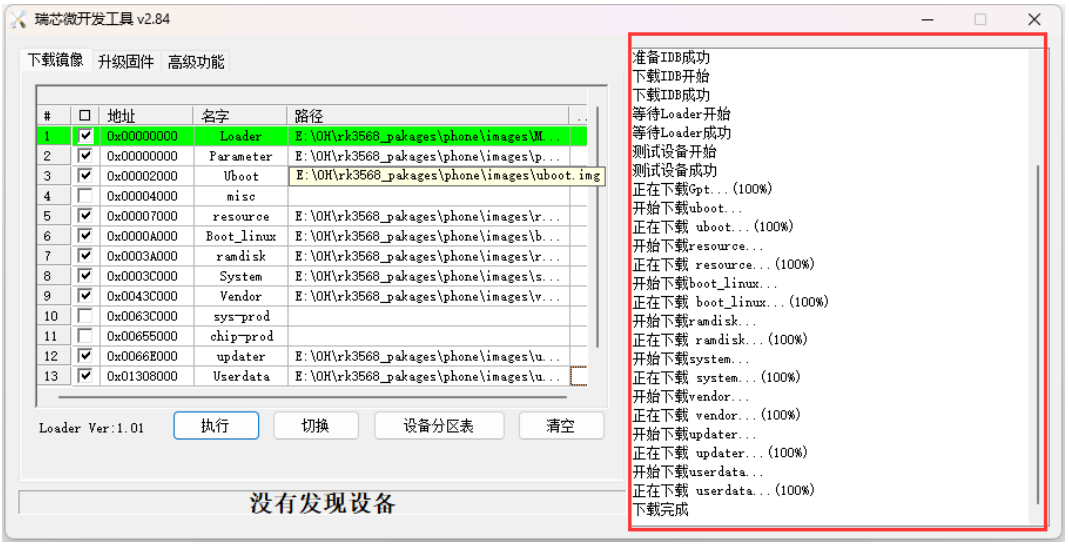




2. 松开 RESET 键， 烧录工具显示“发现一个 LOADER 设备”， 说明此时已经进入烧写模式



3. 松开按键， 稍等几秒后点击执行进行烧录，如果烧写成功， 在工具界面右侧会显示下载完成



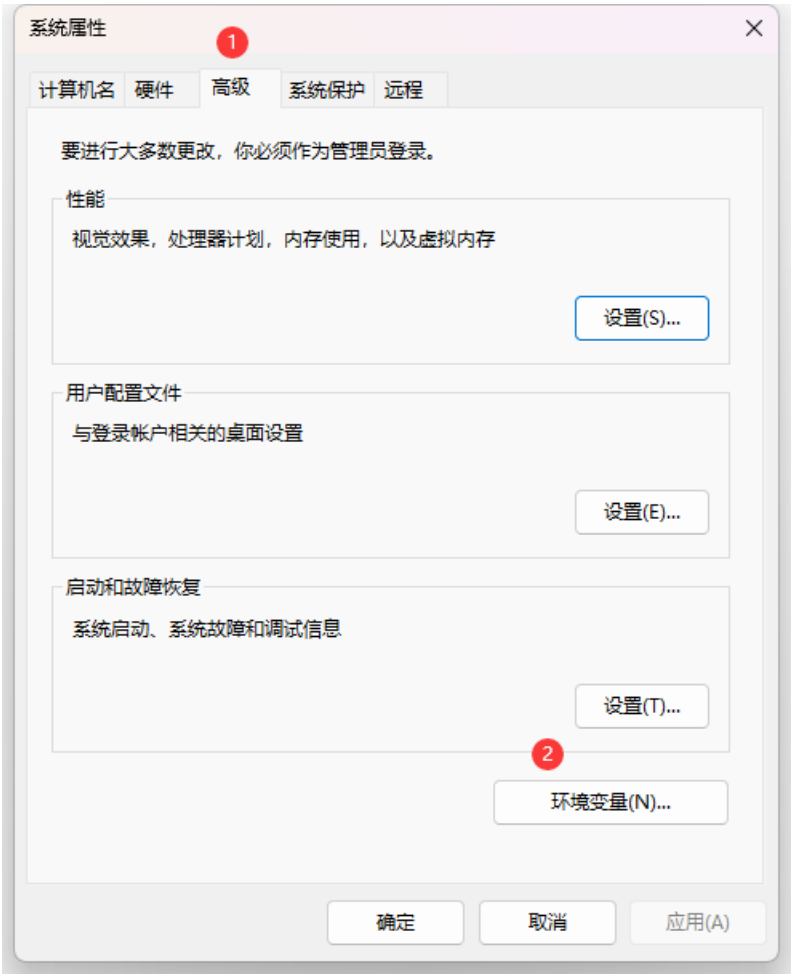
2.2 OH命令行工具hdc

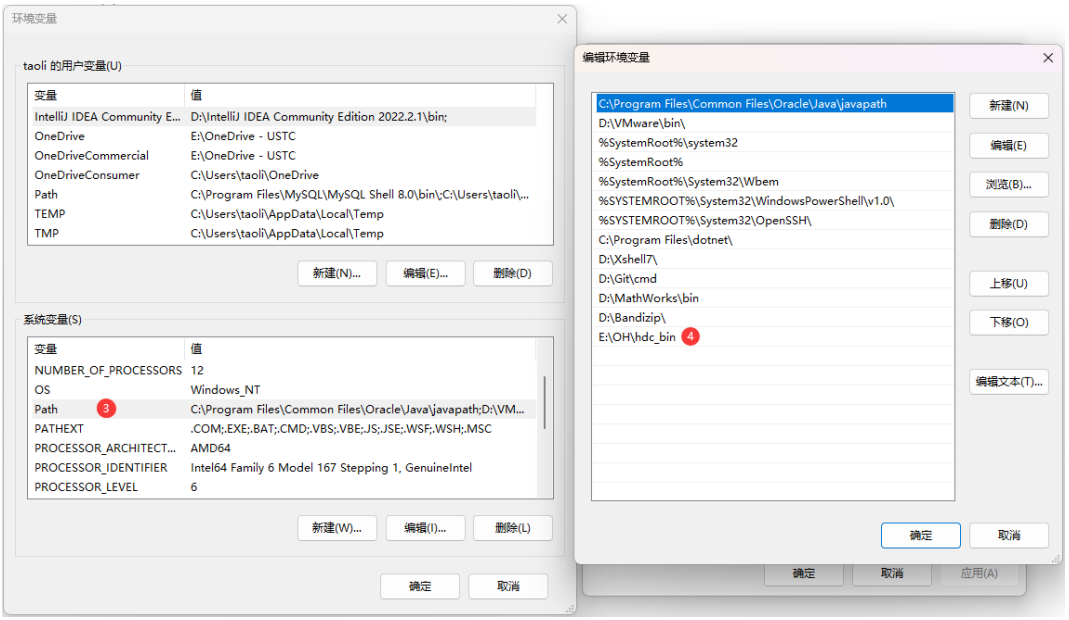
2.2.1 hdc概述

- `hdc`（OpenHarmony Device Connector）是为开发人员提供的用于设备连接调试的命令行工具，该工具需支持部署在Windows/Linux/Mac等系统上与OpenHarmony设备（或模拟器）进行连接调试通信。简言之，hdc是OpenHarmony提供的用于开发人员调试硬件、应用的命令行工具，用在电脑与开发板之间的交互。适用于OpenHarmony应用、硬件开发及测试人员,是每个开发人员的必备、入门工具。

2.2.2 环境准备

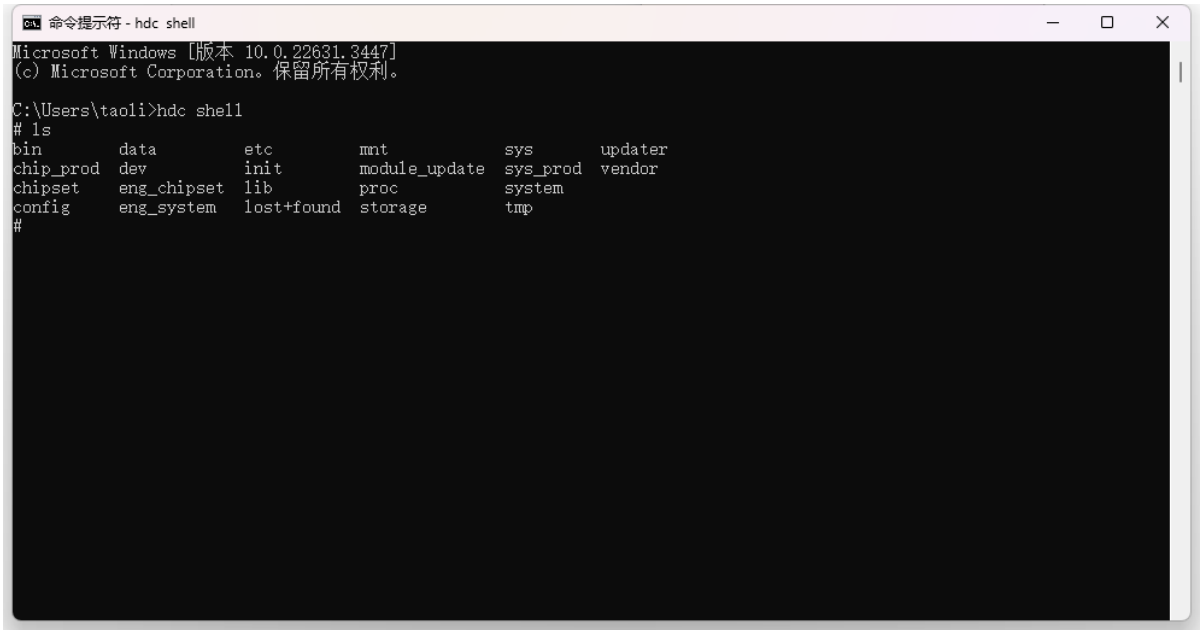
- 支持运行环境：`windows 10、ubuntu 18.04` 以上 `64bit` 版本（下面以Windows11为例）
- 安装USB设备驱动：**见烧录部分**
- `hdc` 工具下载
  1. 下载安装包 https://git.ustc.edu.cn/KONC/oh_lab/-/raw/main/HDC.zip
  2. 解压缩，并将其中的文件 `hdc.exe` 和 `libusb_shared.dll`，复制到文件夹 `E:\OH\hdc_bin\` 目录下（可自定义目录名）
  3. `hdc` 路径环境配置：在设置中找到高级系统设置，然后按照以下步骤操作。“高级系统设置”→高级→环境设置→系统环境变量中的 path→输入 `E:\OH\hdc_bin\` →一路“确定”





2.2.3 运行hdc

- 打开 `cmd` 窗口，执行 `hdc shell` 就进入了命令交互界面。
- 你也可以使用 PowerShell 或者其它 Windows 下的 Shell，用同样的方法使用 `hdc`。


命令提示符 - hdc shell
Microsoft Windows [版本 10.0.22631.3447]
(c) Microsoft Corporation。保留所有权利。
C:\Users\taoli>hdc shell
ls
bin data etc mnt sys updater
chip_prod dev init module_update sys_prod vendor
chipset eng_chipset lib proc system
config eng_system lost+found storage tmp
#

第三部分 编译开源鸿蒙

在这一部分，我们将尝试编译一个能在 RK3568 开发板上运行的最小 OpenHarmony 系统。OpenHarmony 目前只支持在 Linux 环境下编译，我们假设操作系统为 Ubuntu 22.04。本节介绍的 OpenHarmony 编译预计需要 **15GB磁盘空间**，**6GB内存占用**，在性能较低的 2个核心的 CPU 下需要运行大约**30分钟**，如果你的虚拟机不满足这些需求，请你提前分配更多的空间。如果你的电脑配置不能满足实验要求，也可以尝试使用 **VLAB 实验中心** 完成实验。

> VLab 上的虚拟机资源有限，请使用全新的 VLab 虚拟机，并严格按照实验步骤操作，否则可能因为空间不足而编译失败。如果你的自己的计算机拥有**超过 20GB 的剩余磁盘空间**，我们建议使用自己的计算机进行实验，以获得更好的体验。

3.1 下载源码、编译镜像

> 为防止表述混乱，本文档指定 `~/OpenHarmony/` 为本次实验使用的的目录。同学们也可以根据需要在其他 目录中完成本次实验。如果在实验中遇到“找不到 `~/oslab` ”的报错，请先创建相关目录。

3.1.1 下载源码

> OpenHarmoy 的完整源码较大（超过30GB），为了方便实验，我们基于 OpenHarmony-v4.0-Release 版本，预先裁剪了源码，制作了能够编译 RK3568 最小镜像的精简版本，该版本源码删去了版本管理、不需要的组件、占用空间较大的测试数据等，因此不支持除了 RK3568 开发板外的其它设备，也不支持编译带GUI的系统。对完整版系统感兴趣的同学，可以参考 OpenHarmony 的发布页面。例如 **OpenHarmony-v4.1-Release 版本发布页面 - 代码获取**

- 下载精简后的 OpenHarmony 源码，保存到 `~/oslab` 目录下（提示：使用 `wget` ）。
- 下载地址：
- **https://git.ustc.edu.cn/ldeng/ustc-os-resources/-/raw/main/oh4.0_mini_20240423.tar.zst**

```
wget -O oh4.0_mini.tar.zst \
https://git.ustc.edu.cn/ldeng/ustc-os-resources/-/raw/main/oh4.0_mini_20240423.tar.zst
# 文件大小：567M
# SHA256：43808b4a13afc285627437ab69178ef92cfd72957f1a8d75ad3bf3d8e70c36ee
```

- **如果你使用 vlab 完成实验**，请使用使用下面的地址下载更精简的压缩包，该包里不包含 linux kernel 源码，因此能够节约更多空间（但也意味着无法修改 OpenHarmony 所使用的 linux kernel）：
- 下载地址：
- **https://git.ustc.edu.cn/ldeng/ustc-os-resources/-/raw/main/oh4.0_mini_no_kernel_20240423.tar.zst**
- （由于下载地址过长，命令在 pdf 中有换行，直接复制下面的命令会出现问题，请手动从上面复制地址。并在检查命令无误后下载。）

```
wget -O oh4.0_mini.tar.zst \
https://git.ustc.edu.cn/ldeng/ustc-os-
resources/-/raw/main/oh4.0_mini_no_kernel_20240423.tar.zst
# 文件大小：453M
# SHA256：3243ee90de1eead40a70156a47992c07d657f5a7f739a955820a37a0e2370a5f
```

- 下载完成后，请确认目录下存在 `oh4.0_mini.tar.zst` 文件，并且大小和文档中提供的相符。你也可以通过校验文档中提供的 SHA256 值来确认文件是否正确下载（你可以自行查询 `sha256sum` 命令的用法）。
- **注意：如果之前下载出错，请删除当前目录下所有以 `oh4.0_mini` 开头的文件并重试。**
- 使用以下命令解压下载好的 OpenHarmony 源码（注意不要漏掉最后的 `-`）：

  ```
  unzstd -c oh4.0_mini.tar.zst | tar xvf -
  ```

  如果你更喜欢直接使用 `tar`，也可以直接使用以下命令：（不推荐用此命令是因为 tar 对 zstd 格式的自动补全似乎存在问题，无法自动识别 zst 文件）

  ```
  tar x --zstd -vf oh4.0_mini.tar.zst
  ```

- 确认当前目录下出现 `OpenHarmony-v4.0-Release` 目录，该目录下有名为 `OpenHarmony` 的目录。下文中，我们将 `~/oslab/OpenHarmony-v4.0-Release/OpenHarmony` 称为**OpenHarmony 源码路径。**

3.1.2 下载预编译内容

编译 OpenHarmony 需要使用一些特定版本的工具，这些工具已经被提前编译好，需要额外下载。下载这些工具的方式有两种，一种是通过 OH 源码中的 `./build/prebuilts_download.sh` 脚本下载，该脚本会从华为云的存储中直接将所需要的压缩包下载到本地。在网络环境不好时，该方法可能需要较长时间。所幸，脚本会校验已下载的文件的 Hash 值，来避免下载重复文件，因此，也可以选择使用以下命令来从校内仓库下载对应压缩包，并解压到合适位置：

```
# 注意：请在 ~/oslab 目录下运行以下命令，才能将压缩包解压至正确位置
# 文件大小：859M
# SHA256：957af04fab034d646c4a20aaf2e35f46ce96b219d5f8d824373907374ae38b21
wget https://git.ustc.edu.cn/ldeng/ustc-os-
resources/-/raw/main/oh4.0_mini_prebuilts_20240423.tar.gz
mkdir OpenHarmony-v4.0-Release/openharmony_prebuilts
tar xvf oh4.0_mini_prebuilts_20240423.tar.gz --directory=OpenHarmony-v4.0-
Release/openharmony_prebuilts
```

提前下载好所需工具后，再运行 `prebuilts_download.sh` 脚本，会自动将工具解压到源码的所需目录，并正确安装：
（如果不提前下载，直接运行该脚本，则会从华为云下载对应工具）

```
cd OpenHarmony-v4.0-Release/OpenHarmony
./build/prebuilts_download.sh
```

> 注：由于实验中使用的代码是不完整的，上述脚本运行结束后会提示错误，因此，脚本最后提示 `Error: directory '~/oslab/OpenHarmony-v4.0-Release/OpenHarmony/prebuilts/clang/ohos/windows-x86_64/llvm/bin' does not exist while creating lldb-mi.exe` 是正常情况。

如果你希望在 VLab 下完成实验，为了节省空间，运行上述命令解压后，请将 `oh4.0_mini_prebuilts_20240423.tar.gz` 删除以节约空间（否则会因 VLab上虚拟机空间不足而编译失败）。

3.1.3 编译环境和编译工具准备

3.1.3.1 安装 Miniconda

编译 OpenHarmony 需要特定的 Python 版本 （Python 3.8），而 Ubuntu 22.04 中，默认 Python 版本已经升级至 Python 3.10， 因此我们需要使用 Miniconda 来创建我们需要的 Python 环境。

> - Conda是一个开源的包管理器和环境管理器，主要用于安装、运行和升级数据科学和机器学习等领域的软件包和依赖。它支持多种编程语言，包括Python、R、Ruby、Lua、Scala、Java、JavaScript、C/C++等。
> - Anaconda是一个发行版本，它便捷地获取包并对包进行管理，同时对环境进行统一管理。Anaconda包含了conda、Python在内的超过180个科学包及其依赖项。
> - Miniconda是Anaconda分发版的一个更精简的版本。它是一个免费的包管理器和环境管理系统，提供了一个命令行工具conda，允许用户创建独立的环境，每个环境可以有不同的软件包和版本。使用Miniconda不必一开始就安装Anaconda分发版中的大量预装包。这使得Miniconda成为那些对系统空间有限制、或者倾向于仅安装需要的包的用户的理想选择。
>
> 详细的介绍可以参考： **https://blog.csdn.net/u011775793/article/details/134962741**

使用以下命令来下载 Miniconda 安装脚本并安装 （我校的镜像源不提供 Miniconda 镜像，因此我们选择从其它国内镜像中下载，因此下载速度会受限制）：

```
wget https://mirrors.bfsu.edu.cn/anaconda/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b
~/miniconda3/bin/conda init
source ~/.bashrc
```

安装后，命令行前出现 `(base)` 说明成功。命令行前的括号代表着当前的 Python 环境， `base` 是 Miniconda 默认的 Python 环境。通过 Miniconda，我们可以安装多个相互隔离的 Python 环境，并自由切换。

> 如果安装后，命令行前没有出现 `(base)` ，说明安装步骤中出现问题，请检查是否执行了 `~/miniconda3/bin/conda init` ,若执行命令后仍然没有出现，则考虑删除 `Miniconda3-latest-Linux-x86_64.sh` ,重新执行本步骤

3.1.3.2 建立 Python 环境

安装 Miniconda 后，使用以下命令建立一个名为 oh 的 Python 3.8 环境：

```
conda create -y --name oh python=3.8 --no-default-packages
conda activate oh # 激活对应的环境
```

如果在 `conda create` 一步，出现网络非常缓慢，或者无法连接的情况，可以尝试参考 **anaconda | 北京外国语大学开源软件镜像站** 中的介绍，切换 conda 源。

命令行前出现 `(oh)` 说明环境创建并切换成功，此时 `python -V` 显示 `3.8.X` （ `X` 为小版本号，具体数字不影响实验）。

创建好环境后，我们还需将该环境设置为 Shell 默认的 Python 环境，这样我们不需要每次打开 Shell 后都重新切换环境。你可以将以下命令添加到 `~/.bashrc` 文件的最后，并保存退出，以设置默认环境。

> `~/.bashrc` 实际上是个 Bash 脚本，当前用户每次打开 Bash 时，都会自动运行该脚本，因此我们实际上是在每次开启 Bash 后，自动激活了对应环境。另外， `.bashrc` 的文件名以 `.` 开头，在 linux 中，这意味着该文件是个“隐藏文件”，所以你无法直接在文件浏览器中看到它。你可以通过命令行，调用编辑器打开它。

```
# 请将以下两行添加至 ~/.bashrc 的末尾
conda activate oh
export PATH=~/.local/bin:$PATH # 将~/.local/bin 添加到 PATH 环境变量，下面会用到
```

必要时，重启你的 Shell ，或者运行 `source ~/.bashrc` 以使刚才的改动生效。

3.1.3.3 安装编译工具 hb

> hb是HarmonyOS生态中基于Python开发的编译构建工具。hb不仅简化了编译和构建的复杂性，还通过一系列的命令和参数，使得开发者能够精确控制构建过程。得益于Python的跨平台性和易用性，hb可以在不同的操作系统上无缝使用。同时，hb也充分考虑了编译构建的性能和效率，通过合理的优化和缓存机制，极大地提高了构建速度。除此之外，hb还提供了丰富的日志和错误信息输出，帮助开发者快速定位和解决构建过程中遇到的问题。详细了解可以参考：
> https://www.seaxiang.com/blog/78f1134ecc064e2c9c104f158d41c9f5

OpenHarmony 的编译需要使用 `hb` 命令，在 OpenHarmony 源码目录（即 `~/oslab/OpenHarmony-v4.0-Release/OpenHarmony` ）下，使用以下命令安装 `hb` （运行前，请确认你的 Python 环境是刚刚创建的 `oh` ）：

```
python3 -m pip install --user build/hb
```

`hb` 将被安装到 `.local/bin` 目录下，我们在上一步中，已经将该目录添加到 `PATH` 环境变量，因此安装完成后，应该可以直接使用 `hb` 命令。你可以尝试运行 `hb help` 来检查安装是否成功。

3.1.3.4 安装所需库

使用以下命令安装编译时会用到的库：

```
sudo apt update
sudo apt install build-essential flex bison ruby libgl1-mesa-dev libglib2.0-dev
libncurses5-dev libxcursor-dev libxrandr-dev libxinerama-dev libexpat1-dev default-jdk
libelf-dev ccache libssl-dev genext2fs liblz4-tool curl libstdc++-11-dev
```

3.1.4 编译 OpenHarmony

OpenHarmony 可以运行在不同硬件的设备上，为了编译 OpenHarmony 我们首先要指定需要编译的产品类型。使用 `hb set` 可以设置产品类型，也可以用 `-p` 参数直接指定产品类型。我们的开发板使用的是 `rk3568` 芯片组，而为了简化编译，我们将编译一个该芯片组上支持的最小 OpenHarmony 系统，该系统对应的产品类型是 `rk3568_mini_system` 。你可以使用以下命令设置产品类型：

```
hb set -p rk3568_mini_system
```

然后，使用以下命令即可编译 OpenHarmony。编译需要时间较长（VLab 上约30分钟），请耐心等待。

> 1. `--gn-args load_test_config=False` ：这个参数是用来设置GN（Generate Ninja）构建系统的参数的。GN是Chrome和其他一些项目使用的元构建系统，用于生成Ninja构建文件。`load_test_config=False` 可能是用来禁用或关闭某个与测试配置相关的加载。具体这个参数的作用取决于构建系统的配置和上下文。
> 2. `--ccache=False` ：这个参数用来禁用 `ccache` 。 `ccache` 是一个编译器缓存工具，用于加速C/C++编译过程。它缓存编译结果，这样在下次编译相同的代码时，可以直接使用缓存的结果，从而避免重复编译。设置为 `False` 表示不使用 `ccache` 进行编译。
>
> hb用法的详细解释可以参考： https://www.seaxiang.com/blog/78f1134ecc064e2c9c104f158d41c9f5

```
hb build --gn-args load_test_config=False --ccache=False
```

编译的日志可以在源码目录的 `out/rk3568/build.log` 文件里找到，编译出错时，可以查看该日志里的错误提示尝试解决。遇到无法解决的问题时，需要请教助教时，也请将该文件附上，以方便助教排查错误。

编译完成后，可以在源码目录下 `out/rk3568/packages/phone/images/` 找到编译好的镜像文件。你可以按照上一步

3.2 交叉编译应用过程

> ❗注意：本部分采用Hello World程序演示步骤，与实验检查要求需要编译的应用不一样，请仔细阅读实验评分标准。

本部分，我们将向大家介绍通过交叉编译+HDC的方式，在本机上完成HelloWorld程序的编写和交叉编译，并通过HDC传入开发板进行运行。

3.2.0 什么是交叉编译

之前我们接触的大多都是本地编译，比如我们实验二和实验三，我们在完成编译以后，就会直接在本机上运行该程序，这种在同一个平台编译和运行目标代码的编译就是本地编译。

而交叉编译，则是指在一个平台上生成另一个平台上的可执行代码。这在嵌入式领域是常见的编译方法，比如本次实验，我们需要在自己的PC上的linux环境下进行编译，但是我们的目标程序需要在开发板上运行。为了保证可以在开发板上运行，因此我们需要使用交叉编译来完成代码的编译。

3.2.1 编写应用程序

> 3.2.1和3.2.2在Linux环境下执行

首先创建文件 `hello.c` ，在该文件中编写一个简单的HelloWorld程序如下：

```
#include <stdio.h>

int main()
{
    printf("Hello World!");
    return 0;
}
```

3.2.2 交叉编译

实验开发板采用的是RK3568芯片，该芯片采用的是64位的ARM架构，因此需要使用交叉编译来完成HelloWorld程序的编译。

1. 使用apt下载交叉编译工具 `aarch64-linux-gnu-gcc` ：

   ```
   sudo apt update
   sudo apt install gcc-aarch64-linux-gnu
   ```

2. 使用交叉编译工具编译我们编写的程序（注意需要静态编译，加上 `-static` 参数）：

   ```
   aarch64-linux-gnu-gcc -static -o hello hello.c
   ```

   得到了交叉编译好的 `hello` 程序后，将该程序传输到宿主机上，方便后面操作。

   > 有些同学的虚拟机可能安装增强功能出现bug无法拖放文件，可以采取共享文件夹、睿客网盘、邮箱等方式传输文件。

3.2.3 HDC连接和运行

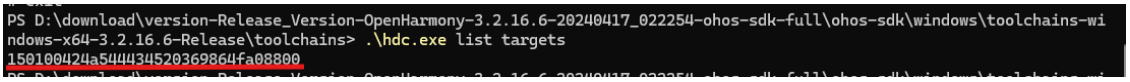
> 注意，从这一步开始将在宿主机上操作。

首先，请参照实验文档第二部分的 `OH命令行工具hdc` 一节下载和配置HDC。

1. 按照上述烧写要求将实验开发板连接到电脑。
2. 打开命令提示符，运行如下指令检查是否成功连接：

   ```
   hdc list targets
   ```

   如果出现了设备名字（一串由数字和字母组成的奇怪字符串），则说明连接成功。效果如下：

   > 如果你执行后没有出现设备，只显示 `[EMPTY]`，请确认镜像烧写是否成功，如果镜像烧写成功，请尝试插拔开发板充电器。

3. 将编译好的程序 `hello` 发送到开发板上，执行如下指令：

   ```
   hdc file send  [宿主机上hello的路径]  /data/local/tmp/hello
   ```

   注意把上面指令中的路径写上你自己存放 `hello` 程序的路径，比如 `hdc file send  D:\hello /data/local/tmp/hello`

4. 执行 `hdc shell` 进入shell交互模式。
5. 执行 `cd /data/local/tmp/hello` 进入hello所在目录（具体目录根据你的hdc file send来确定，比如在第3步中传输到 `/data/local/tmp/hello` ,则进入该目录）。
6. 执行 `chmod +x hello` 为 `hello` 程序添加执行权限。
7. 执行 `./hello` 执行程序得到输出 `Hello World!` ：

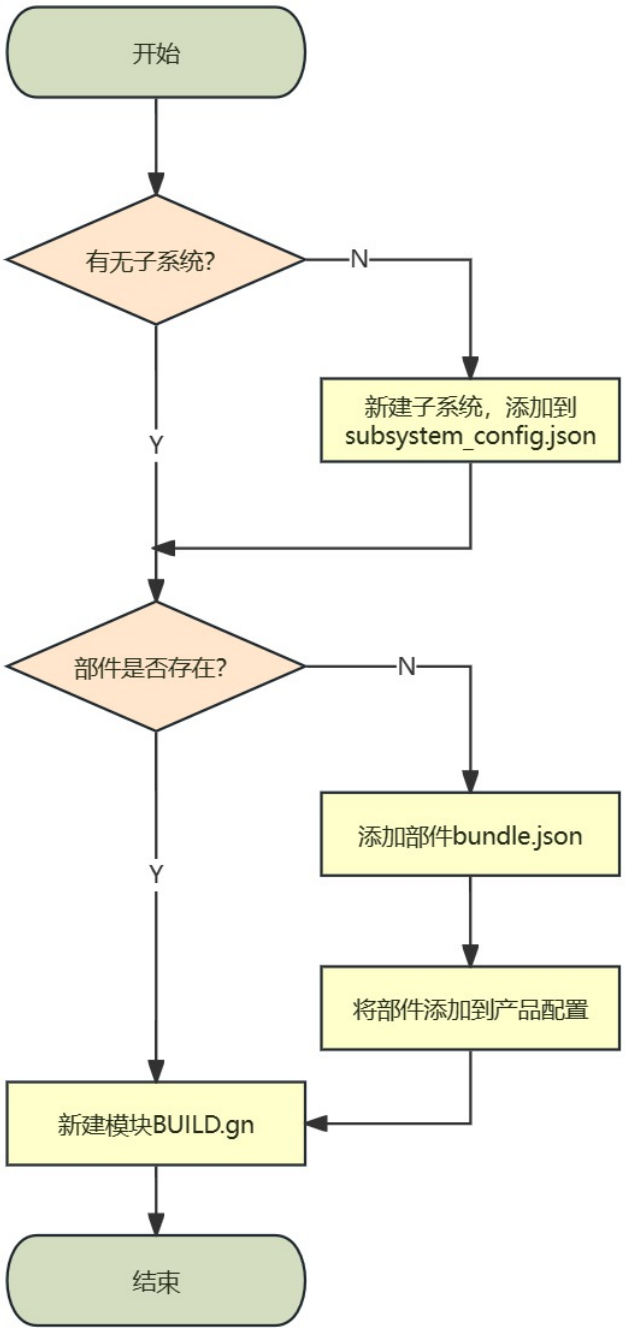
   

3.3 添加子系统

3.3.0 纲要

子系统是 Openharmony 最为明显的特征，他可以给系统添加定制化的功能，在多设备部署场景下，支持根据实际需求裁剪某些非必要的组件。子系统是一个逻辑概念，它具体由对应的组件构成。本节的目的是将lab2中实现的shell当作子系统添加到 Openharmony 中，并且将其添加到镜像中，最后验证子系统是否正常运行。

整体添加思路如下所示：



在添加模块时，需要先判断是否存在模块所需要的子系统，若不存在，则需要新建子系统目录并且将配置(子系统名称和目录)添加到 `build/subsystem_config.json` 文件中。下面是该文件的几个例子，例如有一个叫做 `arkui` 的子系统，其存放的目录为 `foundation/arkui` 。正如之前所说，子系统只是一个逻辑概念，它是由部件组成的，只是为了方便管理。

```
"arkui": {
  "path": "foundation/arkui",
  "name": "arkui"
},
"ai": {
  "path": "foundation/ai",
  "name": "ai"
},
"account": {
  "path": "base/account",
  "name": "account"
},
```

然后判断模块所属的部件是否存在，如果不存在则需要添加部件的配置信息 `bunlde.json` ，以 `OpenHarmony/developtools/hdc/bundle.json` 即hdc部件为例，其部分内容如下，该部件的功能即用于debug的串口工具。其源码的位置是 `developtools/hdc` ,所属的子系统为 `developtools` ,还依赖一些其他的部件例如 `init` 以及第三方开源软件 `libuv` (这些都可以在源码中找到),并且其包含两个模块，分别是 `hdc:hdcd_system` 和 `hdc:hdcd_updater` 。

```
{
  "name": "@ohos/hdc",
  "description": "Device debug connector that provides the device connection capability
and a command line tool",
  "segment": {
    "destPath": "developtools/hdc"
  },
  "component": {
    "name": "hdc",
    "subsystem": "developtools",
    "adapted_system_type": [
      "standard"
    ],
    "deps": {
      "components": [
        "init",
        ...
      ],
      "third_party": [
        "libuv",
        ...
      ]
    },
    "build": {
      "sub_component": [
        "//developtools/hdc:hdcd_system",
        "//developtools/hdc:hdcd_updater"
      ],
    }
  }
}
```

添加完部件信息后，需要将其添加到产品配置信息(即编译对应镜像时需要包含的子系统和部件),以rk3568 mini_system为例(`vendor/hihope/rk3568_mini_system/config.json`)。产品描述信息不再赘述，在inherit中为其继承的配置信息，而在subsystem中可以看到该产品所需要的子系统，里面如下面的 `rockchip_products` 子系统，并且需要该子系统下的 `rockchip_products` 部件

```
  "product_name": "rk3568_mini_system",
  "device_company": "rockchip",
  "device_build_path": "device/board/hihope/rk3568",
  "target_cpu": "arm",
  "type": "standard",
  "version": "3.0",
  "board": "rk3568",
  "enable_ramdisk": true,
  "build_selinux": false,
  "support_jsapi": false,
  "inherit": [ "productdefine/common/base/standard_system.json" ],
  "subsystems": [
```

```
    {
      "subsystem": "rockchip_products",
      "components": [
        {
          "component": "rockchip_products", "features":[ "is_support_boot_animation =
false" ]
        }
      ]
    },
```

在添加完部件后，最后一步即为添加模块配置信息，在该信息中，主要说明源代码与头文件的路径信息，主要是编译时所需要的信息。该部分不再详细说明。

3.3.1 创建子系统目录并添加到子系统配置文件中

> ❗注意：本部分采用Hello World程序演示步骤，与实验检查要求需要编译的应用不一样，请仔细阅读实验评分标准。

进入源码根目录，创建 `sample` 子系统文件夹，在该文件夹中创建 `hello` 部件目录，hello文件夹中创建 `hello` 源码目录，构建文件 `BUILD.gn` 及部件配置文件 `bundle.json` 。

在创建完目录后，你的文件目录树应该为下面例子所示

```
# tree sample
sample
└── hello
    ├── BUILD.gn
    ├── bundle.json

1 directories, 2 files
```

将子系统添加到 `build/subsystem_config.json` 文件,便于编译时查找子系统目录。

新增子系统的配置如下所示。

```
"sample": {
    "path": "sample",
    "name": "sample"
  },
```

3.3.2 编写业务代码

新建 `sample/hello/src/helloworld.c` 目录及文件。其中helloworld.h包含字符串打印函数HelloPrint的声明。

```
#include <stdio.h>
#include "helloworld.h"

int main(int argc, char **argv)
{
    HelloPrint();
    return 0;
}

void HelloPrint()
{
    printf("\n\n");
    printf("\n\t\tHello World!\n");
```

```
    printf("\n\n");
}
```

再添加头文件 `sample/hello/include/helloworld.h`，代码如下所示。

其中前两行的作用为头文件包含防护（include guard） `#ifndef HELLOWORLD_H` 检查是否已经定义了宏 `HELLOWORLD_H` 。如果没有定义（即第一次包含此头文件），则继续执行后续代码并定义该宏（ `#define HELLOWORLD_H` ）。这样可以防止同一头文件被多次包含而导致的重复定义错误。

3-7行用于处理C++编译器对C代码的兼容性问题。 `__cplusplus` 是C++编译器提供的预定义宏，当使用C++编译器编译时其值非零。(如果是c++编译器，进一步检查 `__cplusplus` 的值是否非零（通常情况下会是这样）。如果是，则进入 `extern "C"` 作用域。这告诉C++编译器按照C语言规则而非C++的命名修饰规则来处理接下来声明的函数，确保C++代码能正确链接到用C编写的库或函数。）

第9行为上面源代码中所使用的函数声明。

最后为结束C++编译器下的C语言兼容模式与关闭头文件防护。

```
#ifndef HELLOWORLD_H
#define HELLOWORLD_H
#ifdef __cplusplus
#if __cplusplus
extern "C" {
#endif
#endif

void HelloPrint();

#ifdef __cplusplus
#if __cplusplus
}
#endif
#endif
#endif // HELLOWORLD_H
```

3.3.3新建部件配置文件（部件）

编辑 `sample/hello/bundle.json` 文件，添加sample部件描述,内容如下所示

```
{
    "name": "@ohos/hello",
    "description": "Hello world example.",
    "version": "3.1",
    "license": "Apache License 2.0",
    "publishAs": "code-segment",
    "segment": {
        "destPath": "sample/hello"
    },
    "dirs": {},
    "scripts": {},
    "component": {
        "name": "hello",
        "subsystem": "sample",
        "syscap": [],
        "features": [],
        "adapted_system_type": [ "mini", "small", "standard" ],
        "rom": "10KB",
        "ram": "10KB",
```

```
        "deps": {
            "components": [],
            "third_party": []
        },
        "build": {
            "sub_component": [
                "//sample/hello:helloworld"
            ],
            "inner_kits": [],
            "test": []
        }
    }
}
```

其中"name","description","version","license","publishAs"如代码中注释所示，不再赘述。

- `name`:HPM部件英文名称，格式"@组织/部件名称"
- `segment`:当发布类型为code-setment时必填选项，用于定义部件的源码位置
- `dirs`:HPM包的目录结构，字段必填内容可以留空
- `scripts`:HPM包定义需要执行的脚本，字段必填，值非必填
- `component:name`:部件名称
- `component:subsystem`:部件所属子系统
- `component:adapted_system_type`:部件适配系统
- `component:deps`:部件依赖
- `component:build:sub_component`:部件编译入口

参考链接： **部件配置规则** Gitee.com

3.3.3新建编译组织文件（模块）

编辑`sample/hello/BUILD.gn`，由于我们编写的是C语言程序，故使用OHOS的C/C++模板，其类型为`ohos_executable`,即可执行模块。

```
import("//build/ohos.gni")
ohos_executable("helloworld") {
  sources = [
    "src/helloworld.c"
  ]
  include_dirs = [
    "include"
  ]
  cflags = []
  cflags_c = []
  cflags_cc = []
  ldflags = []
  configs = []
  deps =[]
  part_name = "hello"
  install_enable = true
}
```

- `ohos_executable("helloworld")`中的`helloworld`为模块名，与上面部件配置文件`bundle.json`文件中的`sub_component`所写内容对应，后续在运行过程中的程序名也为该名称。

- `sources`：模块源码位置
- `include_dirs`:模块依赖的头文件目录
- `cflags`：编译可选选项
- `deps`：模块内部依赖
- `part_name`：所属部件名称，必填
- `install_enable`：是否默认安装(缺省默认不安装)

参考链接： **模块配置规则** Gitee.com

3.3.5 修改产品配置文件

在 `vendor/hihope/rk3568_mini_system/config.json` 中添加对应的hello部件，直接添加到原有部件后即可(arkui是已经存在的)。

> 根据3.3.0所讲述的，产品配置文件指定了镜像所包含的子系统和内核版本，所以我们需要在想要生成的镜像(rk3568_mini_system)中添加我们新建的子系统和其部件(sample 和 hello)

```
...
{
  "subsystem": "arkui",
  "components": [
    {
      "component": "ui_lite",
      "features": []
    }
  ]
},
{
  "subsystem": "sample",
  "components": [
    {
      "component": "hello",
      "features": []
    }
  ]
},
```

3.3.6 编译、烧录、运行

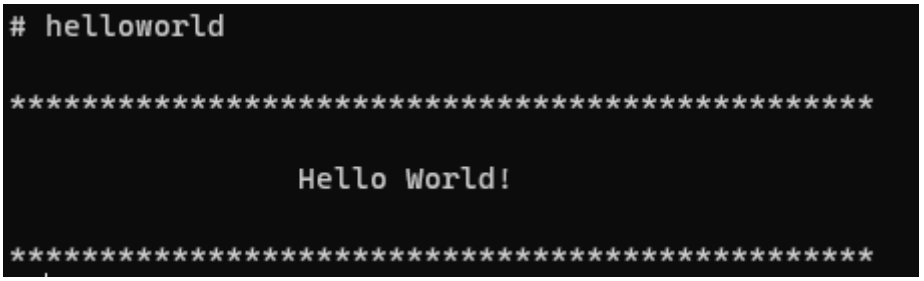
添加完上述所需要的文件后，你的整体的目录树应该为

```
# tree sample
sample
└── hello
    ├── BUILD.gn
    ├── bundle.json
    ├── include
    │   └── helloworld.h
    └── src
        └── helloworld.c

3 directories, 4 files
```

编译方法见 3.1，烧录部分见第二部分

编译烧录后，打开HDC连接后，在任意目录（以设备根目录为例）下输入命令helloworld后回车，界面打印“Hello World！”，程序运行成功。



3.4 任务目标

- 使用提供的代码，编译产品为 `rk3568_mini_system` 的 OpenHarmony 系统。
- 将编译好的系统烧录到开发板上并使用 HDC 连接开发板，并能打开 Shell。
- 通过**交叉编译**的方式，实现**输出自己学号的程序**，并在 OpenHarmony 中运行。
- 添加子系统的方式，**将 Lab2 实现的 Shell 添加为 OpenHarmony 的子系统，并成功运行**。我们要求修改 Shell，使 Shell 在运行开始时**输出你的学号**（如：`PB22000000`）。我们不要求你的子系统、部件、模块名称与上面相同，你可以自行定义，一个名称的例子如下：
  - 子系统：`sample`
  - 部件：`simple_shell`
  - 模块：`my_shell`

3.5 任务提示

1. 添加子系统时由于编译选项的原因，OpenHarmony 不允许任何 Warning 的产生，如果编译出现问题，请查看编译日志是否是代码中出现了 warning （在lab2中提供的TODO代码中存在warning，具体为 `split_string` 函数中 `string_dup[MAX_BUF_SIZE]` 的定义没有用到，以及最后while判断时缺少 `()` ）
2. 添加子系统时可能不需要 `include` ,可以不使用 `include` 的文件，具体方法自行判断
3. 子系统添加完成后通过 `ls` 可以查看到可执行文件的名字，通过 `hdc shell` 直接输入该名字即可正常执行
4. 由于shell环境的原因，删除键可能无法使用，属于正常情况

第四部分：实验内容与检查标准

4.1 实验内容

1. 【可跳过】将提供的 Openharmony4.0 全量标准系统烧录到开发板中，**体验完整版 OpenHarmony 系统**。
2. 编译 rk3568_mini_system 版本的系统并烧录。（参考 3.1 节）
3. 通过**交叉编译**的方式，实现**输出自己学号的程序**，并在 OpenHarmony 中运行。（参考 3.2 节）
4. 添加子系统的方式，**将 Lab2 实现的 Shell 添加为 OpenHarmony 的子系统**，并成功运行。（参考 3.4 节）

更具体实验要求请参考 3.4 任务目标 和 4.2 实验评分标准 。

4.2 实验评分标准

本次实验共 10 分，实验检查要求和评分标准如下：

- 编译 rk3568_mini_system（共3分）：
  - 可以将 rk3568_mini_system 成功编译，并且提供编译成功的输出信息。（3分）
- 烧录系统镜像（共3分）：
  - 将镜像烧录到开发板中，并成功启动（显示Logo）。（2分）
  - 能够通过 HDC 连接至开发板，并打开 Shell 。（1分）
- 交叉编译和添加子系统（共4分）：
  - 通过**交叉编译**的方式编写并运行 OpenHarmony 程序，能够**打印出自己的学号**，并且能够**讲解**实现过程 。（2分）
  - 通过**添加子系统**的方法添加 Shell 程序， **Shell 能成功运行**，并打印出自己的学号，能够**讲解**实现过程。（2分）

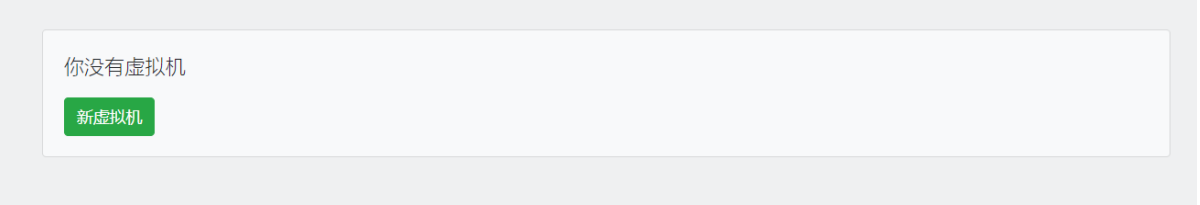
附录

1. VLAB 使用指南

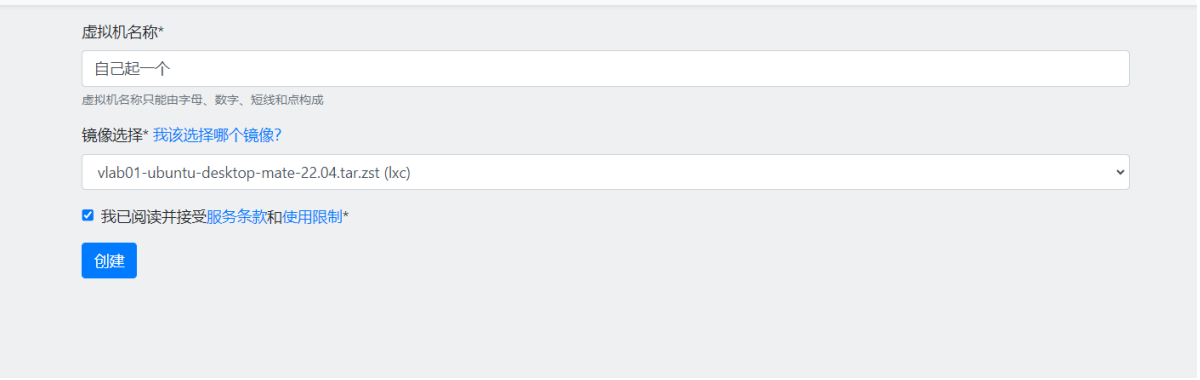
> 此处为vlab使用的简单介绍，有兴趣的同学也可以访问官方文档详细了解：
> https://vlab.ustc.edu.cn/docs/web/

1.1 虚拟机创建

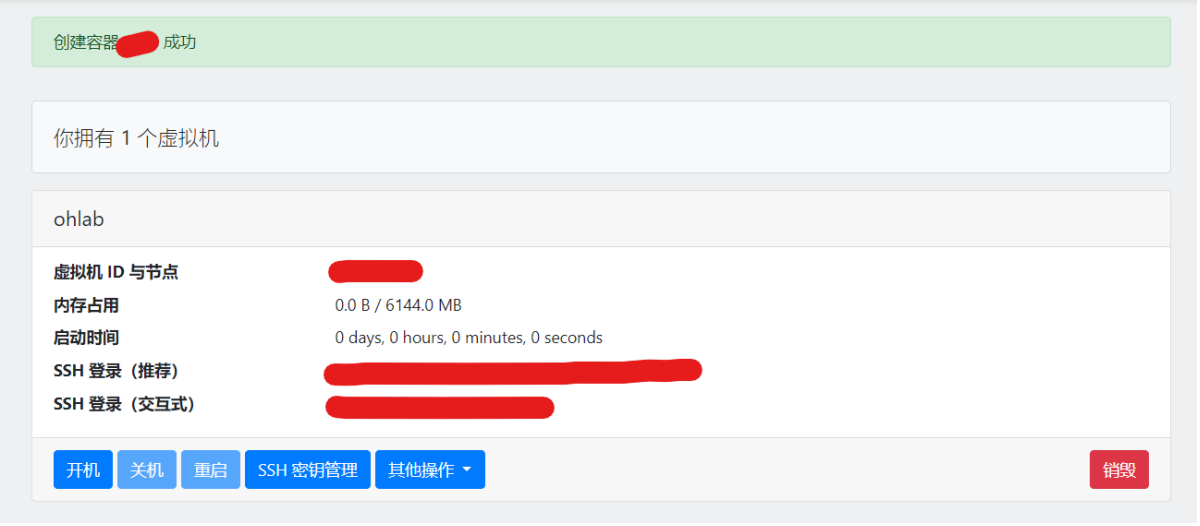
1. 访问 https://vlab.ustc.edu.cn ，点击 `虚拟机管理` ，使用你的学号和密码登录系统。
2. 点击 `新虚拟机` 按钮：



3. 镜像选择 `vlab01-ubuntu-desktop-mate-22.04.tar.zst(lxc)` ，点击 `创建`



4. 稍等一会儿刷新页面看到如下界面则创建成功！此处点击 `开机` 完成虚拟机的启动。



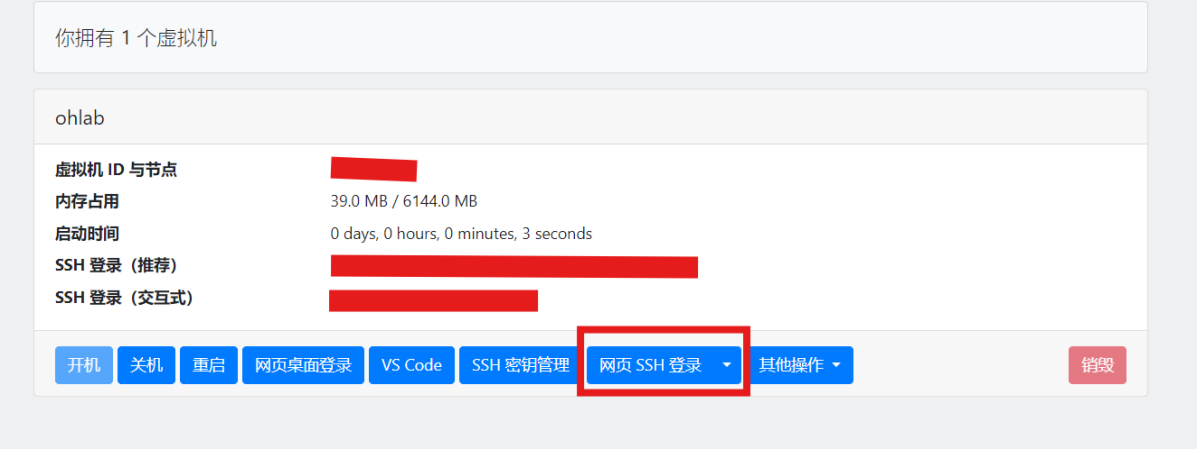
1.2 虚拟机使用

这里推荐两种登录方式（更多登录方式详见官方文档）：浏览器直接登录或ssh命令行登录。

1.2.1 浏览器直接登录

该方式非常简单，点击 `开机` 以后会出现如下界面，点击 `网页SSH登录` 按钮即可。

> 如果你需要使用图形化桌面，请点击 `网页桌面登录` 按钮。



PS：值得一提的是，网页SSH登录提供文件拖动上传和下载功能，具体使用方法如下：

首先需要在虚拟机内安装 `lrzsz` 软件包：

```
sudo apt install lrzsz
```

安装后从本地拖动文件到浏览器窗口中即可上传。如需下载文件，使用 `sz` 命令。例如下载 `yrcamp.jpg` 文件：

```
sz yrcamp.jpg
```

输入命令后，浏览器窗口中会出现一个对话框显示文件名与文件大小，点击 Download 即可。

1.2.2 ssh命令行登录

这里我们仅介绍Linux环境下SSH登录，其他系统平台请参考：
https://vlab.ustc.edu.cn/docs/login/ssh/

打开shell，输入命令：

```
ssh ubuntu@vlab.ustc.edu.cn
```

> 对于本教程使用的镜像，登录用户名为ubuntu。

如果遇到 Warning，请输入 `yes`，然后根据提示输入 Vlab 平台的用户名和密码，即可登录虚拟机。出于安全考虑，输入密码的时候没有回显。

2. SCP 命令使用指南

在使用虚拟机时，很常用的一项功能为在虚拟机和宿主机之间传输数据，在虚拟机平台上，我们可以使用平台自带的部分功能，比如共享文件夹，拖拽等功能。或者我们可以使用在线的云存储平台例如邮箱，睿课网盘等传输文件。本节将介绍一个命令行工具用于在宿主机和虚拟机之间传输数据，该工具为Windows和Linux自带工具，无需下载配置即可使用

2.1 SCP 介绍

scp 是 secure copy 的缩写, 是 linux 系统下基于 ssh 登陆进行安全的远程文件拷贝命令。在群文件中 `实验技巧.PDF` 中，我们介绍过如何通过ssh链接到虚拟机，但是没有说如何通过ssh在虚拟机和宿主机之间传输文件。下面的内容基于你已经了解了ssh如何连接虚拟机。

2.2 SCP 使用方法

scp可以在linux上使用，也可以在windows的命令行工具(cmd/powershell)中使用，使用 `man scp` 可以查看scp的具体用法。下面将在Windows上演示如何进行文件传输。

2.2.1 查看虚拟机ip地址(如果使用vlab，可以忽略该步骤)

使用 `ip addr` 查看ip地址,一般来说，以192.168开头的ip为虚拟机的ip地址。

在命令行(powershell/cmd)中使用 `ping 192.168.xx.xx -c 3` 可以查看是否可以连接到该地址，即是否可以连接到虚拟机.如果显示连接超时，则输入的ip地址不正确，请再次验证是否是该ip地址。

```
# ping 192.168.122.248 -c 3
PING 192.168.122.248 (192.168.122.248) 56(84) bytes of data.
64 bytes from 192.168.122.248: icmp_seq=1 ttl=64 time=0.650 ms
64 bytes from 192.168.122.248: icmp_seq=2 ttl=64 time=0.485 ms
64 bytes from 192.168.122.248: icmp_seq=3 ttl=64 time=0.498 ms

--- 192.168.122.248 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2055ms
rtt min/avg/max/mdev = 0.485/0.544/0.650/0.074 ms
```

2.2.2 使用scp进行文件传输

假设我们现在有一个文件名为 `helloworld.c` ,该文件在windows桌面上，那么我们需要先将windows命令行使用 `cd` 命令到达桌面(Desktop)文件夹，也可以直接在桌面或者你的文件所在的目录右键-> `从终端打开` 。

如果我们想把该文件传输到虚拟机上的家目录，则使用以下命令

```
scp ./helloworld.c user@192.168.xx.xx:~/
```

如果你想要传输一个文件夹（例如文件夹名称为 `helloworld` ）到家目录下的叫做 `sample` 文件夹中，则使用以下命令

```
scp -r ./helloworld user@192.168.xx.xx:~/sample/
```

反过来，如果我们想要从虚拟机上传输一个文件到宿主机，则将位置反过来即可,例如虚拟机家目录下有一个 `helloworld.c` 文件，我们想要传输到本地

```
scp user@192.168.xx.xx:~/hellowolrd.c ./
```

如果你使用的vlab，则可以不使用ip地址进行传输，直接将上面的 `user@192.168.xx.xx` 替换为 `[用户名]@vlab.ustc.edu.cn` ，并且在 `scp` 命令后添加 `-i vlab-vmxxxx.pem` 即可。（具体参数见自己的vlab网页显示）。当然如果使用vlab，更推荐使用vlab自带的文件传输功能。

SCP 总结

scp总结下来用法为

```
scp [-i vlab-vmxxxx.pem] [-r] file_source file_target
```