实验案例二：编译构建子系统基础

实验案例二：编译构建子系统基础

- 一、实验简介
- 二、实验内容及要求
  - 任务一:个人信息打印程序
- 三、实验原理
  - 1.OpenHarmony编译构建工具
    - 1.1 GN与Ninja简介
    - 1.2 GN与Ninja构建流程
  - 2. OpenHarmony系统组织架构
    - 2.1 概述
    - 2.2 源码分析
      - 2.2.1 产品
      - 2.2.2 子系统
      - 2.2.3 组件与模块
  - 3. OpenHarmony编译流程
    - 3.1 hb set [args]
    - 3.2 hb build [args]
- 四、实验步骤
  - 1. 任务一：个人信息打印程序
    - 1.1 流程
    - 1.2 实验提示
- 五、参考资料

一、实验简介

统一OS，弹性部署是OpenHarmony的一大特性，整个OpenHarmony系统就是以**系统-子系统-部件/模块**的层级构造而成，通过采用组件化的设计方案，将整个操作系统分为一个个的组件，使得开发者可根据设备的资源能力和业务特征灵活裁剪，满足不同形态终端设备对操作系统的要求。而在这其中，OpenHarmony中的编译构建子系统就扮演着一个重要的角色，帮助开发者实现裁剪与挑选OpenHarmony组件的需求。

事实上，编译构建子系统并不属于OpenHarmony的成品的一部分，但是它却是OpenHarmony必不可少的基础。虽然并不构成其一部分，但其将各个不同的子系统黏合组织在一起，并最终帮助开发者编译构造出最终的成品。因此了解与学习编译构建子系统是学习OpenHarmony必不可少的一部分。

本次实验即需要同学们通过OpenHarmony的编译构建子系统，为OpenHarmony创建一个简单的子系统和组件，在其中实现一些简单的功能，以加深同学们对OpenHarmony的了解。

> 需要注意的是，OpenHarmony编译子系统是以GN和Ninja构建为基座，对构建和配置粒度进行部件化抽象、对内建模块进行功能增强、对业务模块进行功能扩展的系统，因此在这个实验中可能需要同学们了解一点GN和Ninja。

二、实验内容及要求

本次实验需要同学们依次完成下列实验要求，并提交最终在qemu中运行的结果截图。

任务一:个人信息打印程序

本节实验需要同学们完成一个“个人信息打印”的程序——**print_information**，具体要求如下：

1. 完成**print_information**组件的创建，其中包含程序的实现模块。
2. 完成**sysu**子系统的创建，其中包含**print_information**组件。
3. 打开qemu，进入ohos环境之后，能在/bin目录下找到`print_information`程序，运行程序将打印个人的学号与姓名。示例如下：

```
OHOS:/$ ./bin/print_information

***********************************************

        张三 22xxxxxx!

***********************************************
```

三、实验原理

1.OpenHarmony编译构建工具

OpenHarmony主要使用GN与Ninja实现编译构建子系统，本节将对GN和Ninja进行简单地介绍，同学们在实验过程中如果有需要也可以自行进行相关的学习。

1.1 GN与Ninja简介

不同于我们接触过的小型项目，其中的编译也许只需要处理几个文件，OpenHarmony作为目前一个源码规模达到50g的操作系统项目，其中包含着数以千记的文件。这其中更是包括着许多不同的模块，各个模块之间有着不同的依赖关系，其中有些需要编译成静态库，有些则需要编译为动态库或是可执行文件，面对如此大规模的项目，简单地通过命令行方式来进行处理就显得不太现实。也正因此，构建工具就应运而生，例如GNU Make，帮助我们管理项目中各个文件模块的依赖构建关系。

然而随着项目规模的进一步扩大，尤其是面对有着多平台需求的项目时，就连编写GNU Make中的Makefile代码也同样地显得困难复制，容易出错。也因此在编译构建系统的基础之上，就出现了帮助生成Makefile的工具，例如cmake、AutoMake等等，它们也被称为元构建系统。

而在本节中所要介绍的GN与Ninja则是在Chromium开源项目中所使用的构建工具，分别属于元构建系统与构建系统的范畴，两者间的关系也就类似于cmake与Make，其也被使用在如今的OpenHarmony系统项目之上。

1.2 GN与Ninja构建流程

在使用GN与Ninja构建项目时，编译脚本会依次的调用GN和Ninja程序执行两步操作，分别对应着命令`gn cmd args`与`ninja cmd args`的执行，其中**cmd**为执行的操作、**args**为传递的参数。这两个命令的执行流程合在一起即构建了完整的GN和Ninja构建流程。

`gn cmd args`命令执行过程中具体包含着以下的6个步骤：

1. 加载构建入口**.gn**文件。首先在执行命令的当前目录下搜索**.gn**文件，若当前目录下不存在，则递归地沿着上一级目录进行查找，直到找到**.gn**文件为止。

   若找到**.gn**文件，则会将文件所处的目录设置为构建过程之中默认的根目录(source root)，任意构建过程中的文

件都可以通过`//`来访问根目录。并且**.gn**文件中还需要定义`buildconfig`参数，用于指定构建过程中所使用的编译配置文件(**BUILDCONFIG.gn**)。

若直到文件系统的根目录`/`下都找不到**.gn**文件的话，命令则会报错，编译失败。

2. 运行`buildconfig`参数所指定的**BUILDCONFIG.gn**文件，并根据其中的配置设置全局变量与默认的编译工具链，其中的全局变量和参数默认会对整个构建过程所涉及的文件都有效。
3. 加载**.gn**文件下`root`参数指定的目录下的**BUILD.gn**文件。若**.gn**文件中没有进行配置，则默认加载编译根目录下的**BUILD.gn**文件，即`//BUILD.gn`。
4. 根据**BUILD.gn**递归地评估依赖关系，并加载相应路径下的**BUILD.gn**文件。
5. 在递归评估构建目标依赖关系的过程中，每解决一个构建目标的依赖关系就生成对应目标的**.ninja**文件。
6. 当完成所有构建目标的依赖关系处理后，就会生成一个**build.ninja**文件。

`ninja cmd args`命令则会运行Ninja程序，默认从当前目录下查找**build.ninja**文件，并根据其描述使用编译工具构建所有的目标。在大部分情况下，整个编译过程中我们只需要学习GN的语法，了解如何编写gn文件，而ninja文件则会由GN程序自动生成。

了解上述的GN与Ninja的构建流程能够帮助我们理解在OpenHarmony中整个项目的编译构建流程，以便我们学习如何实现OpenHarmony中各个子系统的裁剪。但是在我们这次的实验之中，同学们还需要编写简单的**BUILD.gn**文件，因此还需要同学们自行学习GN程序的语法。不过不需要担心，本次实验所使用的gn文件很简单，只需要同学们学习最基本的语法即可完成。

2. OpenHarmony系统组织架构

2.1 概述

在OpenHarmony的官方文档中介绍了以下概念：

- 平台：开发板和内核的组合，不同平台支持的子系统和部件不同。
- 产品：产品是包含一系列部件的集合，编译后产品的镜像包可以运行在不同的开发板上。
- 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（详见OpenHarmony技术架构）。系统功能按照“系统 > 子系统 > 部件”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子系统或部件。子系统是一个逻辑概念，它具体由对应的部件构成。
- 部件：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。需要注意的是下文中的芯片解决方案本质是一种特殊的部件。
- 模块：模块就是编译子系统的一个编译目标，部件也可以是编译目标。
- 特性：特性是部件用于体现不同产品之间的差异。

在后文中会对上述概念进行更详细的介绍，同学们也会对其有更深入地认知。

OpenHarmony事实上包括着各种各样的子系统，而每个子系统中又包含着多个的部件(部件并非最小单位，其中仍然包含着不同的模块)，整个系统都是按照“系统 > 子系统 > 部件”逐级进行展开。OpenHarmony的一大特性**统一OS，弹性部署**，正是来源于这样的架构设计，如图1所示。

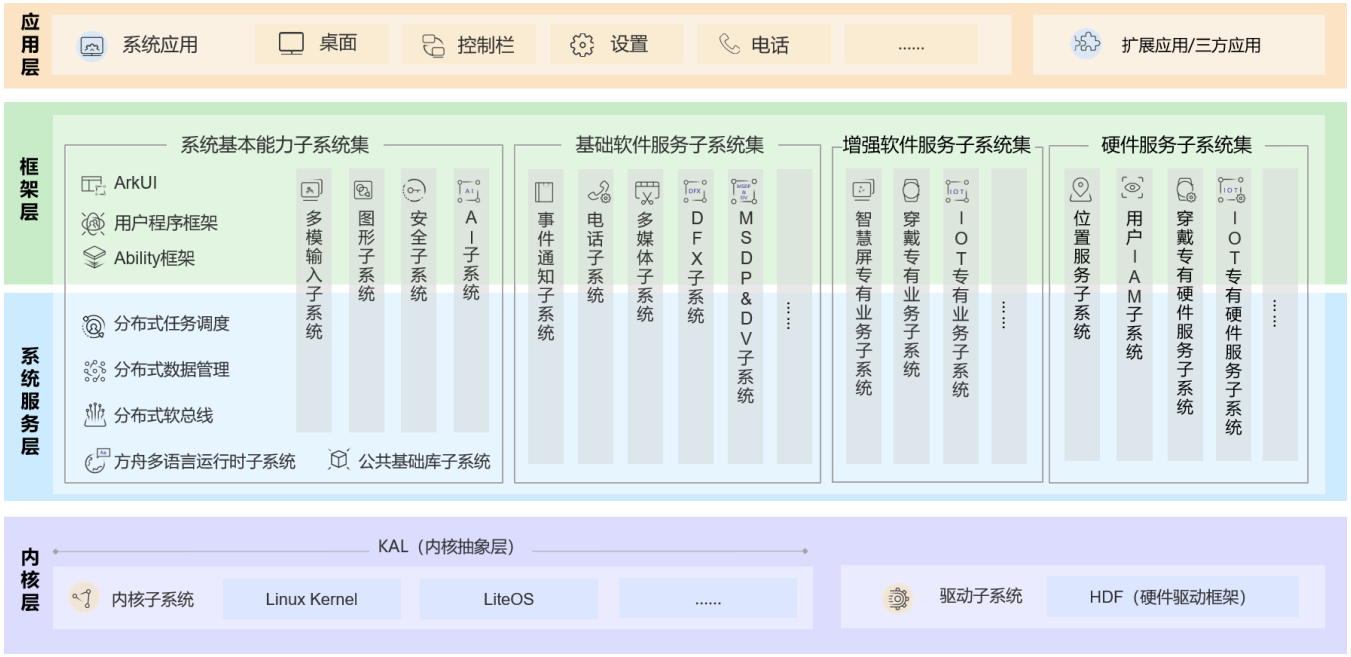

图1.OpenHarmony技术结构[1]

作为一个开源的系统，任何人都可以根据自身的需求自由地裁剪选择OpenHarmony中的不同子系统与部件，从而每个人都能以OpenHarmony作为基础，编译出自己独特的系统，也就是官方文档中所说的“产品”，图2中展示了其中产品、子系统、部件和模块之间的关系。

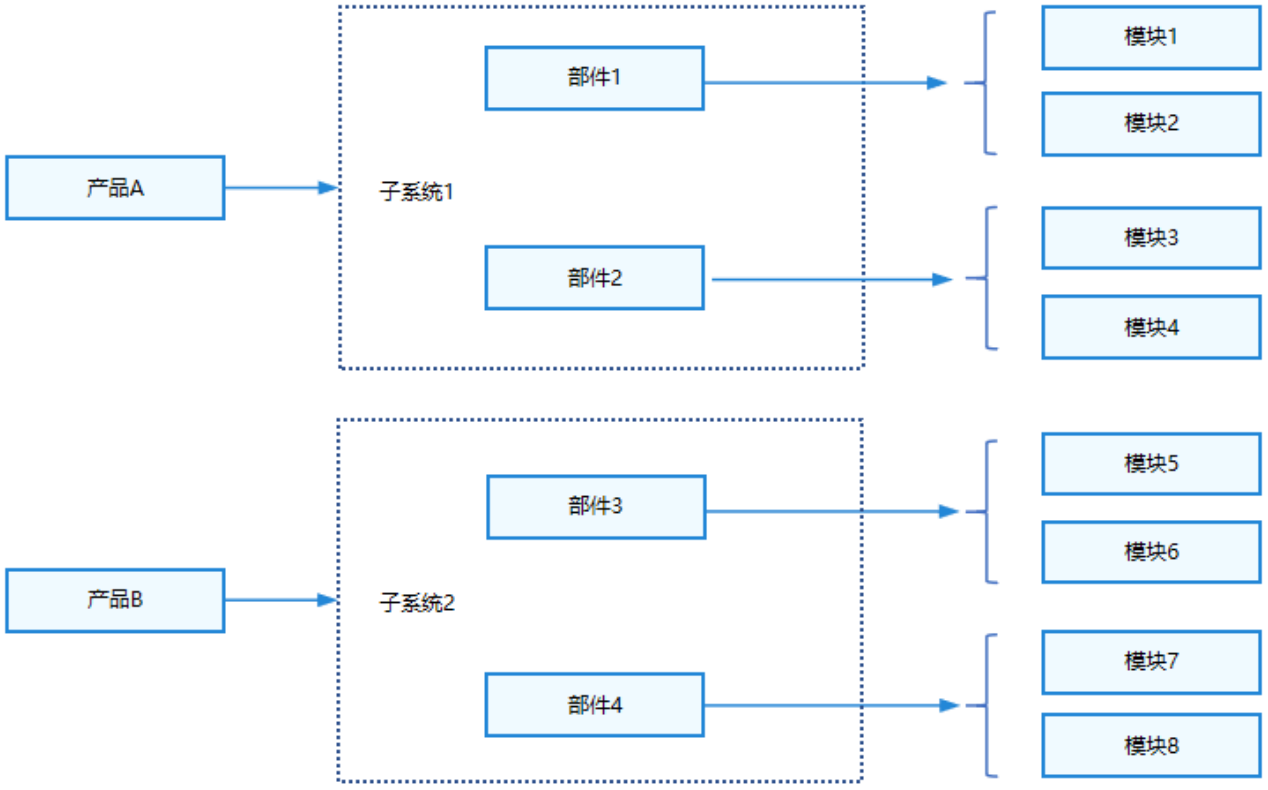

图2.产品、子系统、部件和模块关系[2]

2.2 源码分析

OpenHarmony 4.1版本的源码目录大致信息如下表所示，接下来将会主要介绍编译构建相关的几个部分。

目录名	描述
applications	应用程序样例，包括camera等
arkcompiler	方舟编译器，主要分成两个部分：编译工具链与运行时
base	基础软件服务子系统集&硬件服务子系统集
build	组件化编译、构建和配置脚本
commonlibrary	公共基础类库存放OpenHarmony通用的基础组件
developtools	研发工具链子系统,主要为开发人员提供了开发时用于调试的命令行以及追踪性能轨迹、查看性能的工具
device	第三方平台厂商部分，芯片方案商基于开发板的解决方案
docs	说明文档
domains	增强软件服务子系统集
drivers	驱动子系统
foundation	系统基础能力子系统集
kernel	内核子系统
napi_generator	NAPI框架生成工具
prebuilts	编译器及工具链子系统
productdefine	公共产品形态配置,主要定义与芯片无关的通用系统组件形态配置。
test	测试子系统
third_party	开源第三方组件
vendor	第三方平台厂商部分，厂商提供的软件

2.2.1 产品

产品解决方案为基于开发板的完整产品，也就是**vendor**目录下的内容，主要包含产品对OS的适配、部件拼装配置、启动配置和文件系统配置等。具体到源码中来看，产品解决方案的源码路径规则为：**vendor/{产品解决方案厂商}/{产品名称}**。具体目录结构如下：

```
vendor
└── company                          # 产品解决方案厂商
    ├── product                      # 产品名称
    |   ├── init_configs
    |   |     ├── etc                # init进程启动配置（可选，仅linux内核需要）
    |   |     └── init.cfg           # 系统服务启动配置
    |   ├── hals                     # 产品解决方案OS适配
    |   ├── BUILD.gn                 # 产品编译脚本
    |   └── config.json              # 产品配置文件
    |   └── fs.yml                   # 文件系统打包配置
    └── ......
```

如果厂商需要创建自己的产品，只需仿照其中的结构，根据需要编写配置文件即可。在上面展示的目录结构中**config.json**文件定义着相应的产品中所包含的子系统、组件以及特性，以我们实验所使用**ohemu**厂商的**qemu_small_system_demo**产品为例，其中的**config.json**文件内容如下：

```
{
    "product_name": "qemu_small_system_demo",
    "ohos_version": "OpenHarmony 1.0",
    "version": "3.0",
    "type": "small",
    "device_company": "qemu",
    "board": "arm_virt",
    "kernel_type": "liteos_a",
    "kernel_version": "3.0.0",
    "subsystems": [
      ...
      {
        "subsystem": "kernel",
        "components": [
          { "component": "liteos_a", "features":[] }
        ]
      }
      ...
    ],
    "third_party_dir": "//third_party",
    "product_adapter_dir": "//vendor/ohemu/qemu_small_system_demo/hals"
  }
```

我们暂时只关注其中的**subsystems**字段，其中定义着当前产品所裁剪使用的子系统。其中描述着我们所实验的**qemu_small_system_demo**产品使用着**"liteos_a"**内核，也因此其中使用了内核子系统**"kernel"**中的**"liteos_a"**组件。

2.2.2 子系统

在**build**目录下的**subsystem_config.json**文件中，定义OpenHarmony中所有子系统的配置规则，其中内容如下：

```
{
  "arkui": {
    "path": "foundation/arkui",       # 路径
```

```
    "name": "arkui"                    # 子系统名
  },
  "ai": {
    "path": "foundation/ai",
    "name": "ai"
  },
  "account": {
    "path": "base/account",
    "name": "account"
  },
  "distributeddatamgr": {
    "path": "foundation/distributeddatamgr",
    "name": "distributeddatamgr"
  },
  "applications": {
    "path": "applications",
    "name": "applications"
  },
  ...
}
```

可以看到，其中的每个字段都代表着一个子系统，定义着子系统所处的路径与名称。这种方式使得为OpenHarmony扩展新的子系统非常简便。

2.2.3 组件与模块

每个子系统都包含着多个组件，而一个组件中又包含着多个模块，以**application**子系统为例，其目录结构如下：

```
application
└── sample
|     ├── camera
|     |   ├── cameraApp
|     |   |     ├── cameraApp
|     |   |     └── BUILD.gn          # cameraApp模块构建文件
|     |   ├── launcher
|     |   |     ├── launcher
|     |   |     ├── cert
|     |   |     └── BUILD.gn          # launcher模块构建文件
|     |   ├── ...
|     |   ├── bundle.json             # camera_sample_app组件描述文件
|     |   └── ...
|     └── wifi-iot
└── standard
```

其中的**bundle.json**文件就定义着组件的配置信息，该文件除了需要放置于子系统目录下，没有固定的存放位置要求。而模块作为组件的组成部分没有配置描述文件，每一个BUILD.gn中定义的target都是一个模块，也是编译的最小单位。以launcher模块为例，其中的`shared_library("launcher")`正定义着GN中的一个"launcher"的target。

```
import("//build/lite/config/hap_pack.gni")

shared_library("launcher") {
  sources = [
    "launcher/src/main/cpp/view_group_page.cpp",
    ...
  ]
  ...
}
...
```

而组件配置文件**bundle.json**内容如下,其中"component"字段下的"name"定义着部件的名称，"build"字段下的"sub_component"定义该部件所包含的组件。

```
{
    ...
    "component": {
        "name": "camera_sample_app",          # 部件名称
        "subsystem": "applications",          # 部件所属子系统
        "syscap": [],                         # 部件系统能力
        "features": [],                       # 部件特征
        "adapted_system_type": [              # 适用的子系统
            "small"
        ],
        "deps": {                             # 依赖的其它部件
            "components": [
                "utils_base",
                ...
            ],
            ...
        },
        "build": {                            # 部件包含的模块
            "sub_component": [
                "//applications/sample/camera/launcher:launcher_hap",    # launcher模块编译文件
                "//applications/sample/camera/cameraApp:cameraApp_hap", # cameraApp模块编译文
件
                ...
            ],
        }
    }
}
```

根据上面所展示的内容，若想要为子系统创建新的部件和模块，只需要参照上面的方式编写**bundle.json**配置文件与相应模块的BUILD.gn编译文件即可。

3. OpenHarmony编译流程

为了方便实现OpenHarmony项目的编译构建，OpenHarmony还提供了命令行工具hb，用于执行编译构建命令。其中提供了如下几项命令：

- **hb set [args]**：设置要编译的产品。

- **hb env [args]**：查看当前设置信息
- **hb build [args]**：编译产品、部件、模块或芯片解决方案。
- **hb clean [args]**：清除out目录对应产品的编译产物

图1中展示了OpenHarmony的大致编译流程。

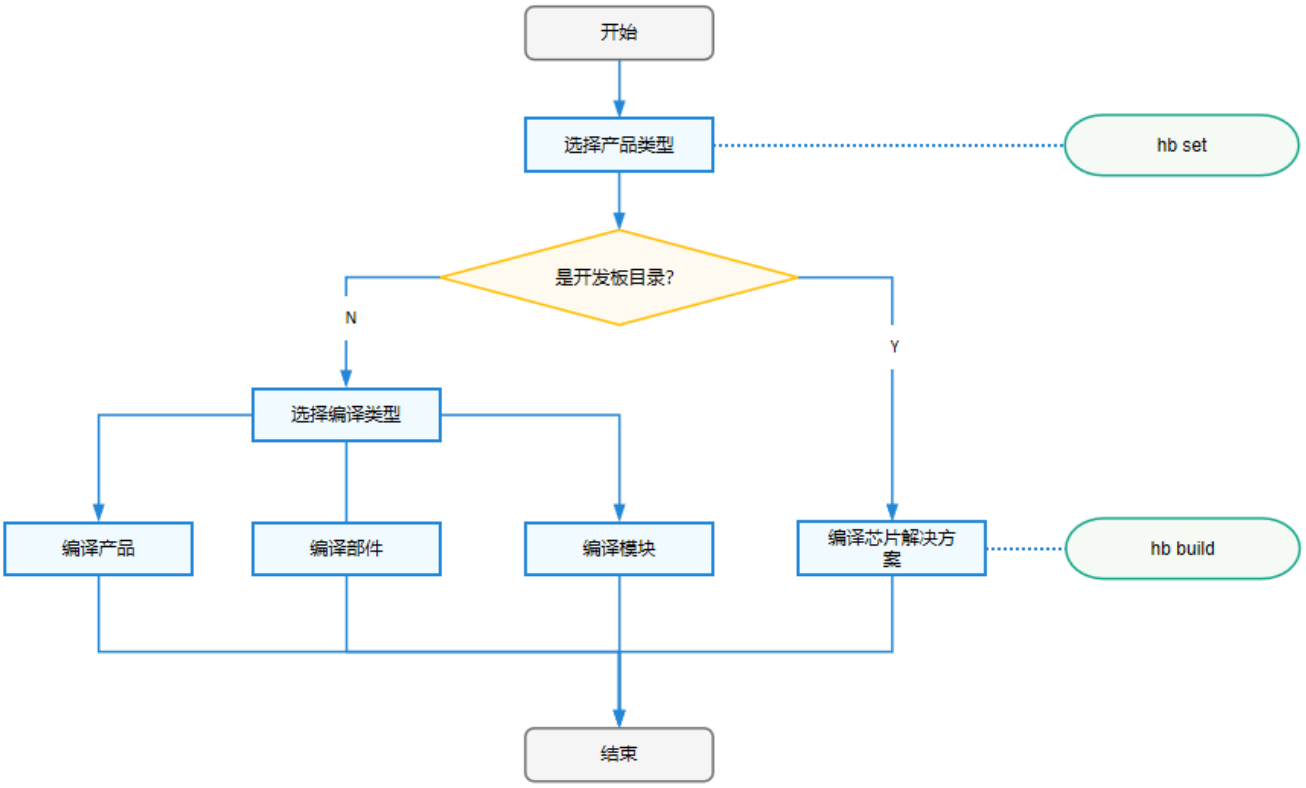

图3.OpenHarmony编译流程[3]

通过hb命令编译之后将在OpenHarmony根目录下生成out目录，并将编译过程中生成的文件放置其中，以编译产品**qemu_small_system_demo**为例，生成的out目录结构如下：

```
out
└── arm_virt/qemu_small_system_demo              # 产品编译文件
|     ├── build_config
|     └── ...
└── preloader                                    # Openharmony配置文件
└── ohos_config.json                             # 全局编译参数
```

3.1 hb set [args]

`hb set`命令主要负责产品配置参数的设置与更新，以便编译工具能够知道我们需要编译的产品，并根据所选择的产品类型设置OpenHarmony编译时的参数，并保存至`out/config.json`文件之中。

3.2 hb build [args]

`hb build`命令的工作大致可以分为四个阶段，分别对应着preloader、loader、generate_ninja(gn)、ninja。

preloader阶段主要负责将OpenHarmony中各个子系统、组件和部件整理工作。正如前文所述，OpenHarmony可扩展性很强，只需要添加一些简单的配置描述文件，就能为OpenHarmony添加新的子系统、组件和模块。但是这些简单的配置文件显然无法很好地指引项目的编译，因此需要在正式编译前做一些预处理工作。因此在preloader阶段，hb脚本文件将会根据我们编写的配置描述文件进行进一步地处理，扫描子系统文件夹并整理信息，生成更详细的配置文件将其存储在`out/preloader`文件夹之中。

loader阶段主要作用在于负责补全gn的目标依赖关系。由于不同产品中可能包含着不同地子系统和组件，GN的产品编译文件(BUILD.gn)不可能预先写好，因此在正式的开始编译gn文件评估目标依赖关系之前，hb工具需要预先根据preloader阶段生成的配置文件信息动态地生成产品依赖BUILD.gn文件，搭建出一条完整的目标依赖路径，最终的结果就将放置在`out/{board}/{product}/build_config`目录下。命令`gn path target1 target2`能够生成gn中两个编译目标之间的依赖关系路径，若是探究OpenHarmony的起始编译目标与一个模块编译目标间依赖关系，可以得到如下结果：

```
user@ubuntu:~/OpenHarmony/OpenHarmony-v4.1-Release/OpenHarmony$ gn path
out/arm_virt/qemu_small_system_demo/ //sysu/practice/helloworld:helloworld
//build/core/gn:images
//build/core/gn:images --[private]-->
//build/ohos/images:make_images --[private]-->
//build/ohos/packages:packer --[private]-->
//build/ohos/packages:make_packages --[private]-->
//build/ohos/packages:phone_parts_list --[private]-->
//build/ohos/common:merge_all_parts --[private]-->
//build/ohos/common:generate_host_info --[private]-->
//out/arm_virt/qemu_small_system_demo/build_configs/sysu/sysu_practice_app:sysu_practice_app
--[private]-->
//out/arm_virt/qemu_small_system_demo/build_configs/sysu/sysu_practice_app:sysu_practice_app
_info --[private]-->
//sysu/practice/helloworld:helloworld
```

可以看到在依赖路径之间，需要使用到在本阶段中所动态生成的编译目标。

generate_ninja和ninja阶段则按照本章第一节的流程开始构建与编译我们的产品项目，以OpenHarmony源码根目录下的**.gn**文件作为入口，依次评估目标依赖关系构建编译项目，最终生成的结果将放置在`out/{board}/{product}`目录之下。

上述只是hb工具的大致工作流程介绍，其中还有许多细节在这里没有介绍，若同学们感兴趣，可以自行查看`build/hb`目录下的hb工具源码。

四、实验步骤

1. 任务一：个人信息打印程序

1.1 流程

本节具体步骤如下：

1. 创建子系统：在OpenHarmony源码目录下新建**sysu**目录，并且修改`build/subsystem_config.json`配置文件，在其中添加sysu子系统条目。
2. 创建组件：在子系统目录下新建**print_information**目录，并且在其中新建**bundle.json**组件配置描述文件。
3. 创建模块: 在组件目录下新建模块目录，在其中编写c程序与**BUILD.gn**编译文件。
4. 修改产品配置：修改`vendor/ohemu/qemu_small_system_demo/config.json`文件，在其中添加我们新建立的组件。
5. 编译运行：使用hb工具重新编译与运行项目，进入ohos系统之后输入命令`./bin/print_information`运行程序。

1.2 实验提示

1. 组件的目录结构建议配置如下

```
组件
├── 模块1
|   ├── BUILD.gn
|   ├── include
|   |   └── helloworld1.h
|   └── src
|       └── helloworld1.cpp
├── 模块2
|   ├── BUILD.gn
|   ├── include
|   |   └── helloworld2.h
|   └── src
|       └── helloworld2.cpp
└── bundle.json
```

2. 编写BUILD.gn文件时，OpenHarmony建议我们不要使用gn的原始模板，而是以编译子系统提供的模板代替。所谓gn原生模板，是指source_set，shared_library, static_library, action, executable，group这六个模板，它们和编译子系统提供的模板之间的对应关系如下:

编译子系统提供的模板	**原生模板**
ohos_shared_library	shared_library
ohos_source_set	source_set
ohos_executable	executable
ohos_static_library	static_library
action_with_pydeps	action
ohos_group	group

由于编译子系统提供的模板定义在`//build/ohos.gni`文件中，因此在使用前需要在**BUILD.gn**文件的起始处添加`import("//build/ohos.gni")`以导入模板定义。

五、参考资料

[1]. OpenHarmony技术架构:https://gitee.com/openharmony/docs/blob/master/zh-cn/OpenHarmony-Overview_zh.md

[2]. 产品、子系统、部件和模块关系:https://gitee.com/openharmony/docs/blob/OpenHarmony-4.1-Release/zh-cn/device-dev/subsystems/subsys-build-all.md

[3]. OpenHarmony编译流程:https://gitee.com/openharmony/docs/blob/OpenHarmony-4.1-Release/zh-cn/device-dev/subsystems/subsys-build-all.md

[4]. GN参考文档：https://gn.googlesource.com/gn/+/master/docs/reference.md

[5]. 编译子系统介绍：https://docs.openharmony.cn/pages/v4.1/zh-cn/device-dev/subsystems/subsys-build-all.md