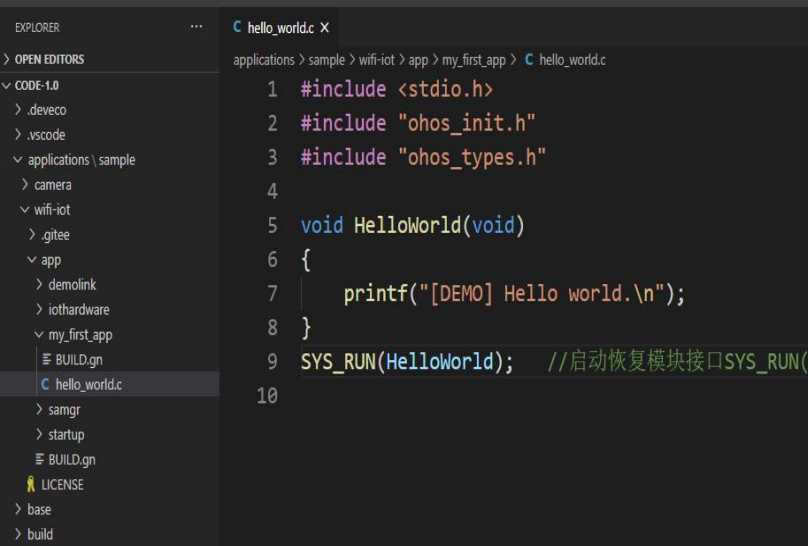


# Harmony OS 智能硬件 入门系列课程 <快速上手>

快速掌握Hi3861开发板基础开发技巧

## 第 2 讲：Hi3861上开发第一个Hello world程序

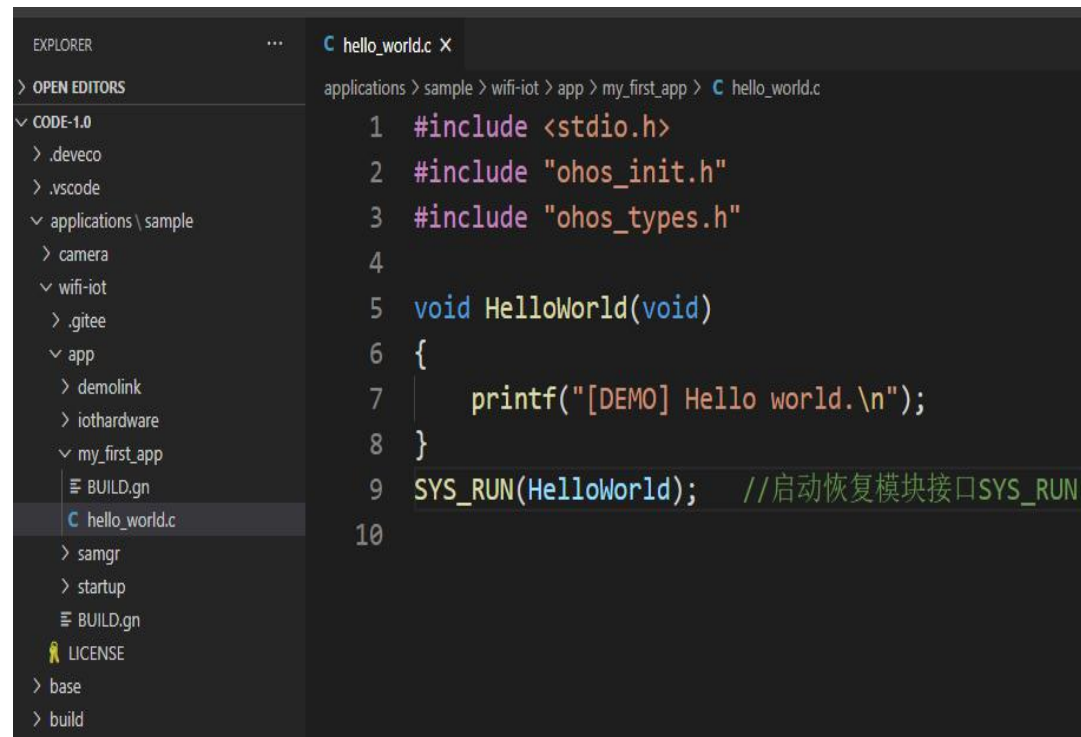


```
EXPLORER
...
C hello_world.c X
applications > sample > wifi-iot > app > my_first_app > C hello_world.c

1  #include <stdio.h>
2  #include "ohos_init.h"
3  #include "ohos_types.h"
4
5  void HelloWorld(void)
6  {
7      printf("[DEMO] Hello world.\n");
8  }
9  SYS_RUN(HelloWorld); //启动恢复模块接口SYS_RUN(
10
```

# 本讲内容

- 第1节：VS code中导入鸿蒙源码
- 第2节：编写第一个Hello world程序
- 第3节：完成HelloWorld程序编译、烧写、运行
- 第4节：启动流程分析



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project tree with the following structure:

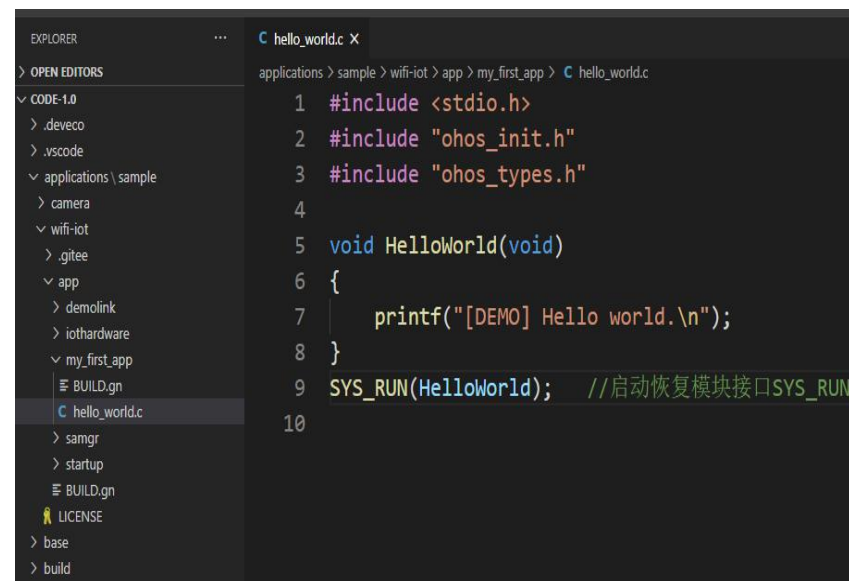
- EXPLORER
  - OPEN EDITORS
  - CODE-1.0
    - .devcco
    - .vscode
    - applications \ sample
      - camera
      - wifi-iot
        - .gitee
        - app
          - demolink
          - iothardware
          - my\_first\_app
            - BUILD.gn
            - hello\_world.c (selected)
          - samgr
          - startup
          - BUILD.gn
          - LICENSE
          - base
          - build

The main editor area shows the content of 'hello\_world.c' with the following code:

```
1 #include <stdio.h>
2 #include "ohos_init.h"
3 #include "ohos_types.h"
4
5 void HelloWorld(void)
6 {
7     printf("[DEMO] Hello world.\n");
8 }
9 SYS_RUN(HelloWorld); //启动恢复模块接口SYS_RUN()
10
```

# 本讲目标

- 1、掌握VS code中导入鸿蒙源码方法
- 2、能编写第一个Hello world程序
- 3、掌握HelloWorld程序编写、编译、烧写、运行、测试
- 4、了解鸿蒙系统启动流程

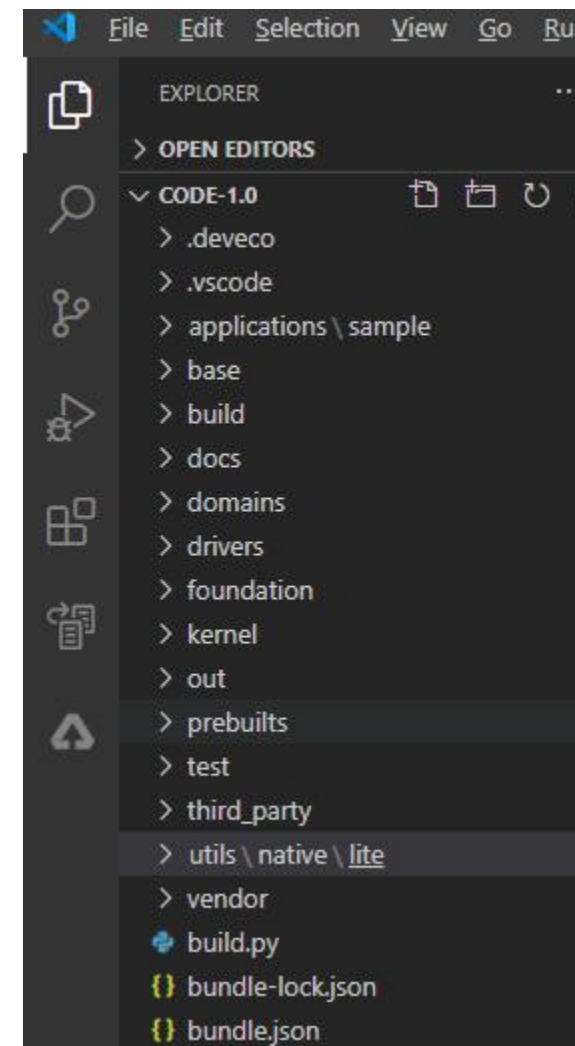


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project tree with folders like 'CODE-1.0', 'applications', and 'my\_first\_app'. The file 'hello\_world.c' is selected. The main editor area shows the content of 'hello\_world.c' with the following code:

```
1 #include <stdio.h>
2 #include "ohos_init.h"
3 #include "ohos_types.h"
4
5 void HelloWorld(void)
6 {
7     printf("[DEMO] Hello world.\n");
8 }
9 SYS_RUN(HelloWorld); //启动恢复模块接口SYS_RUN()
10
```

# 第1节：VS code中导入鸿蒙源码

- 知识点1：映射HarmonyOS源码
- 知识点2：导入源码
- 知识点3：配置工程



## 知识点1 【映射HarmonyOS源码】

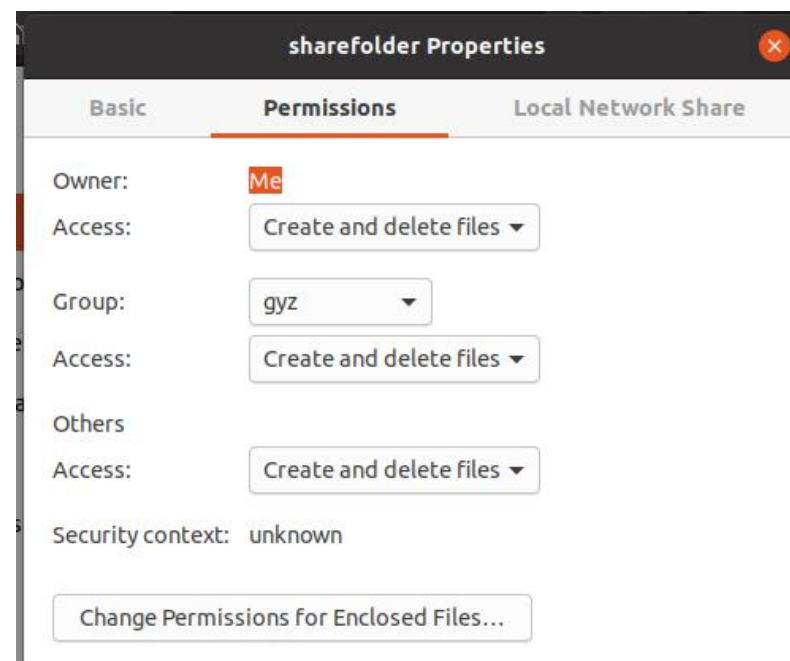
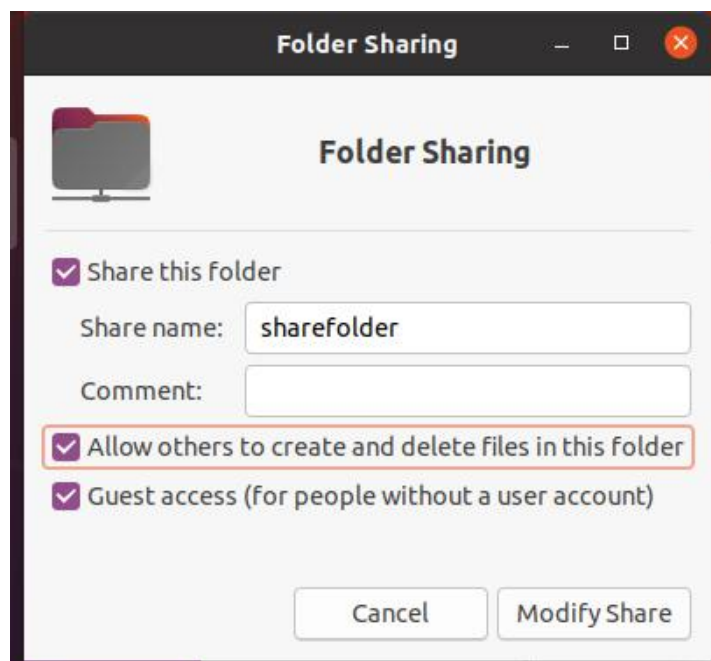
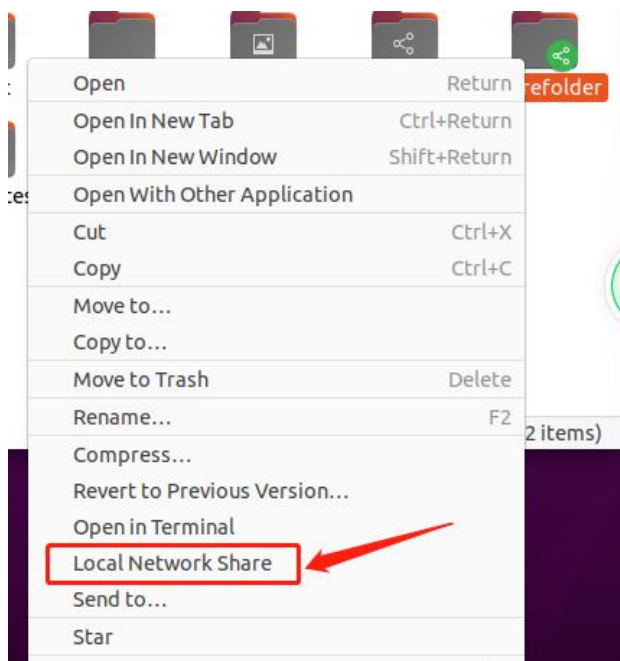
实现步骤：

1. 在Linux上，设置HarmonyOS源码所在文件夹为共享文件夹
2. 在Windows上，鼠标右键单击“此电脑”，选择“映射网络驱动器（N）”
3. 选择一个未使用的驱动器，并设置HarmonyOS源码所在的路径

# 知识点1 【映射HarmonyOS源码】

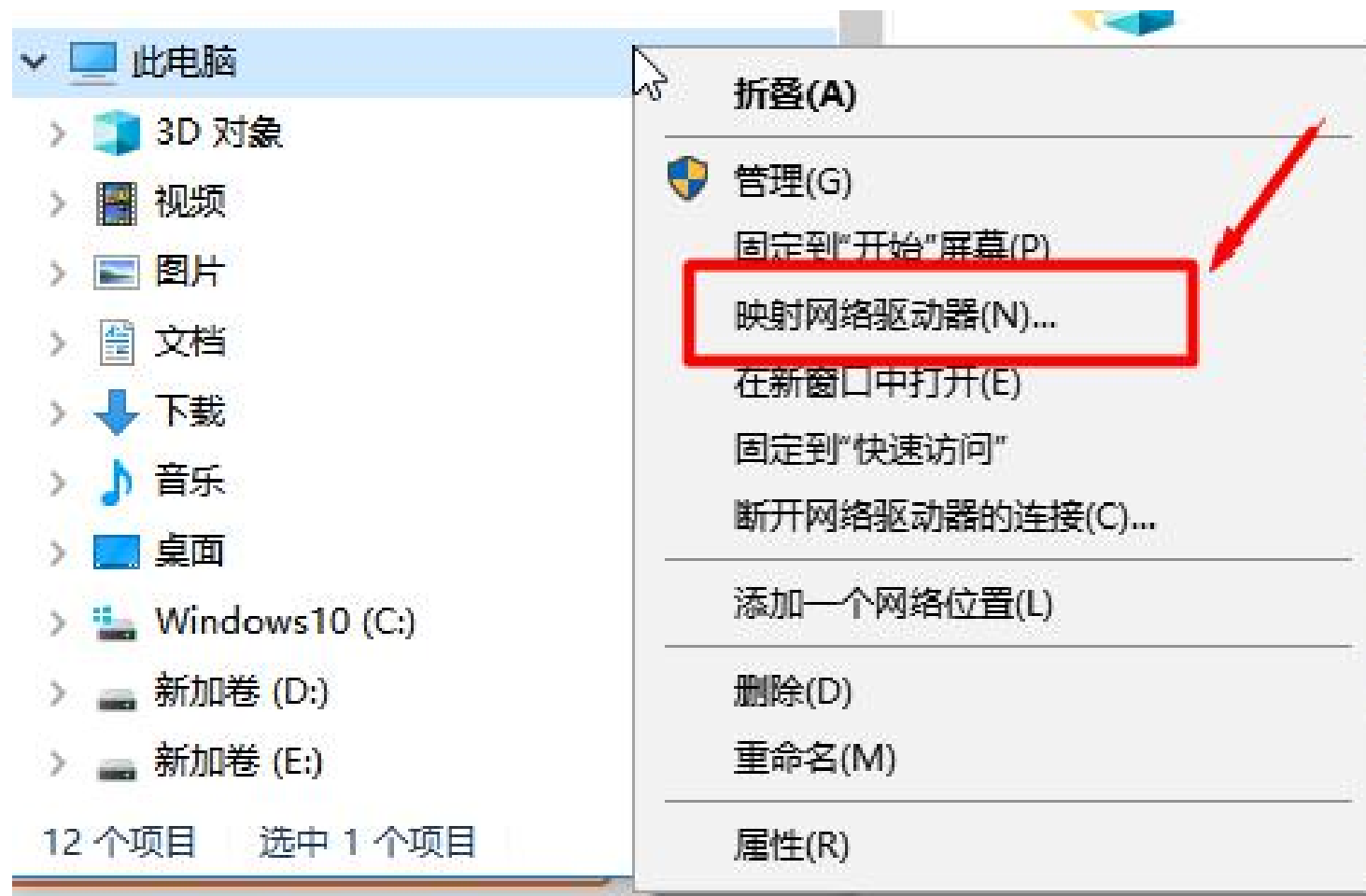
步骤1：在Linux上，设置HarmonyOS源码所在文件夹为共享文件夹

1. 如果没有安装samba服务，安装samba服务： `sudo apt-get install samba`
2. 设置共享文件夹、
3. 设置共享文件夹权限为：所有都为create and delete files ， 否则会报无创建及修改文件权限错误



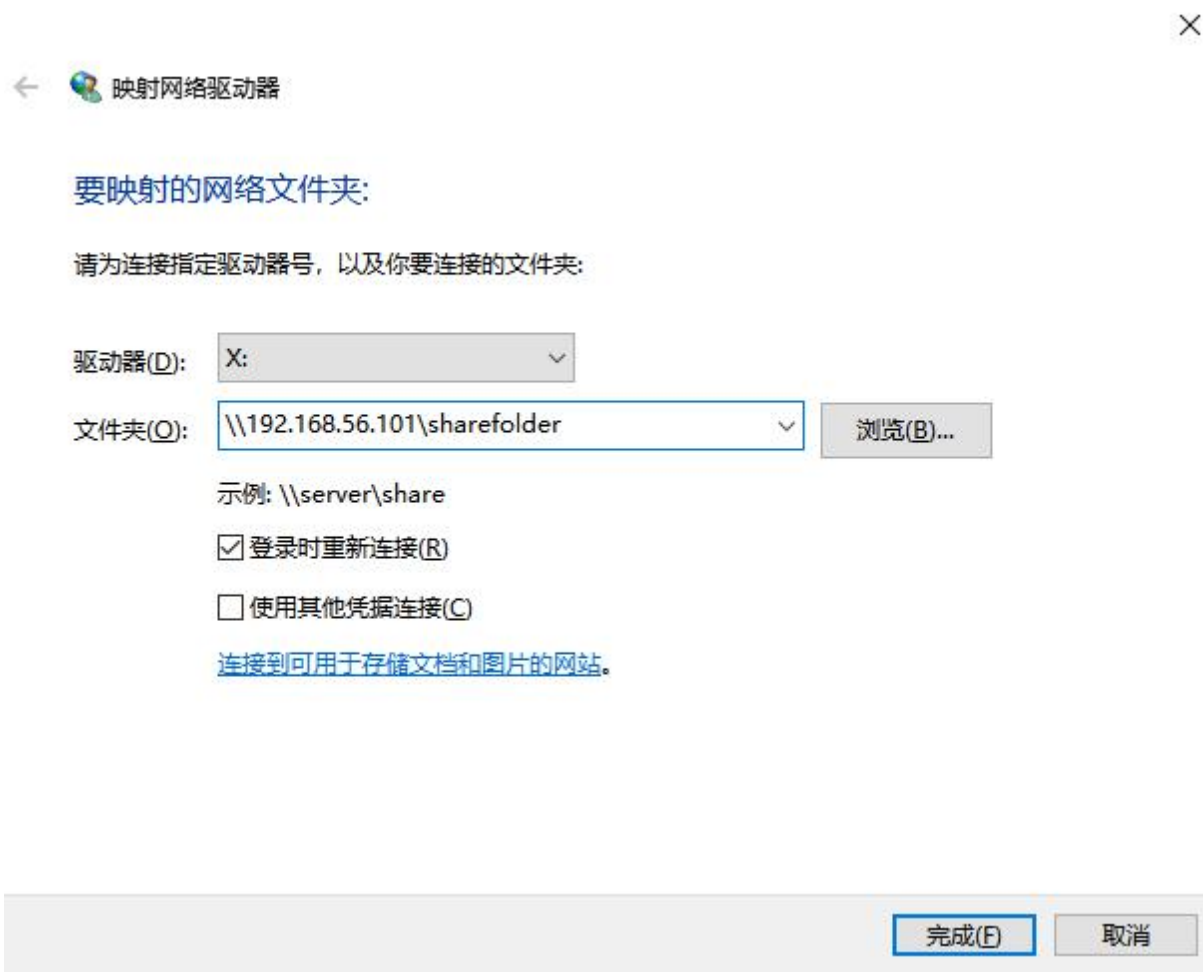
## 知识点1 【映射HarmonyOS源码】

步骤2：在Windows上，鼠标右键单击“此电脑”，选择“映射网络驱动器（N）”



## 知识点1 【映射HarmonyOS源码】

步骤3：选择一个未使用的驱动器，并设置HarmonyOS源码所在的路径





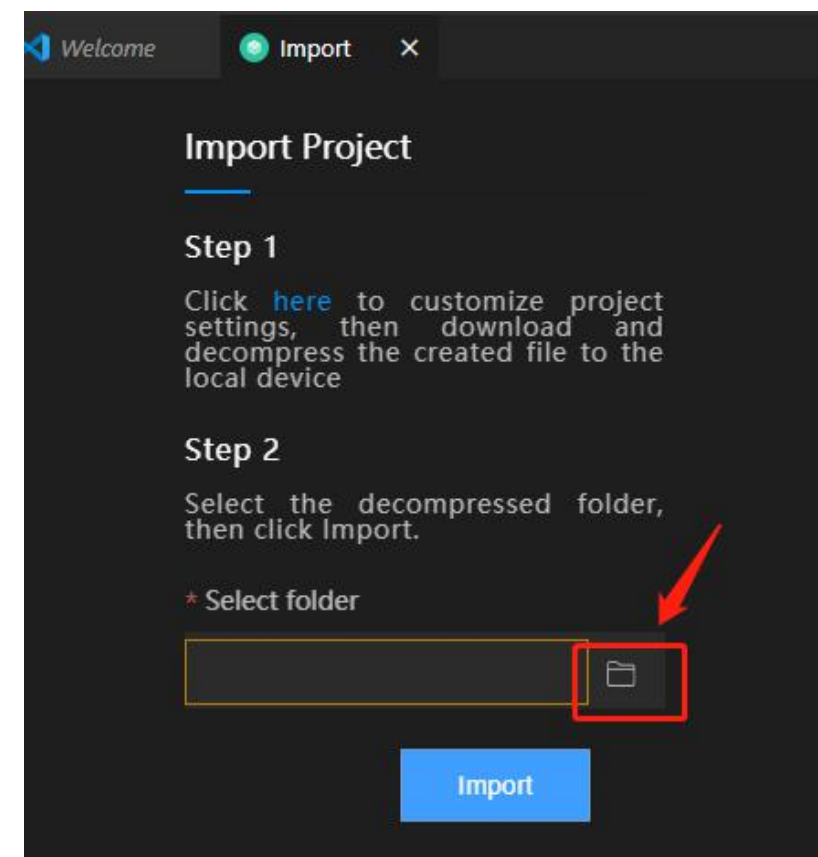
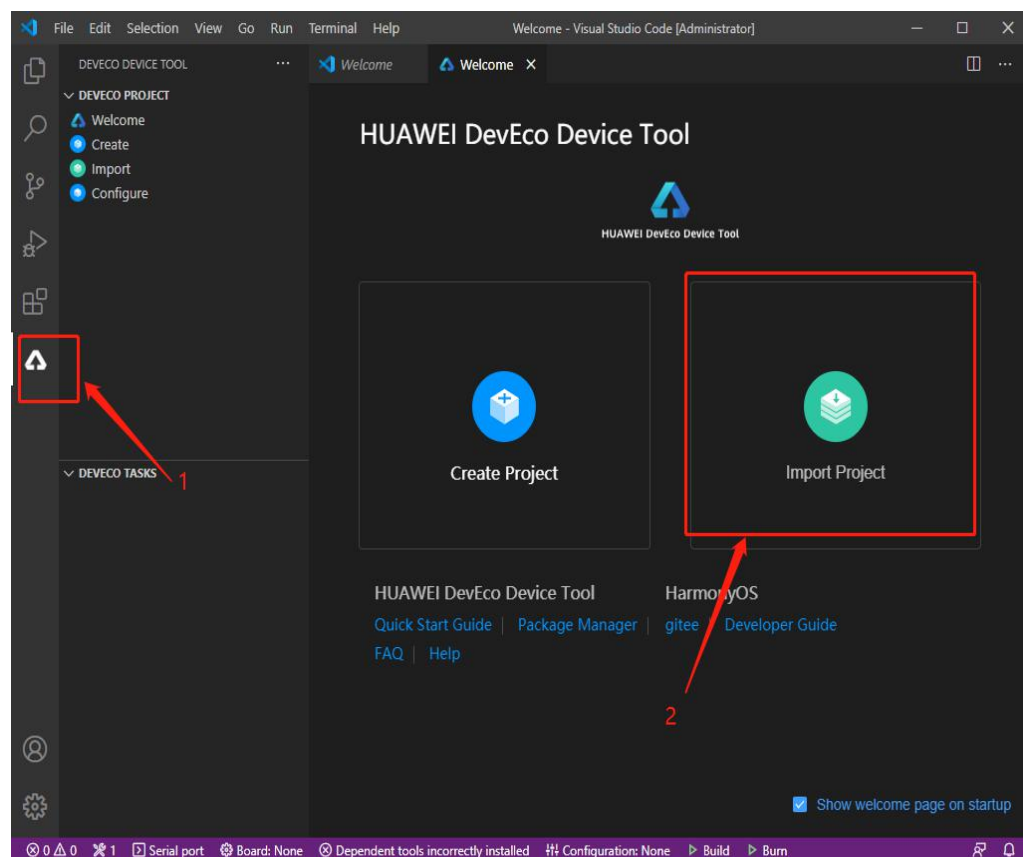
## 知识点2 【导入源码】

HarmonyOS暂不支持Windows系统源码，需从映射的Linux服务器的共享路径，导入源码实现步骤：

1. 在Visual Studio Code中，点击DevEco Device Tool插件按钮图标，然后点击 “Import” 按钮
2. 选择本地映射的HarmonyOS源码文件夹，然后点击 “Import” 导入工程

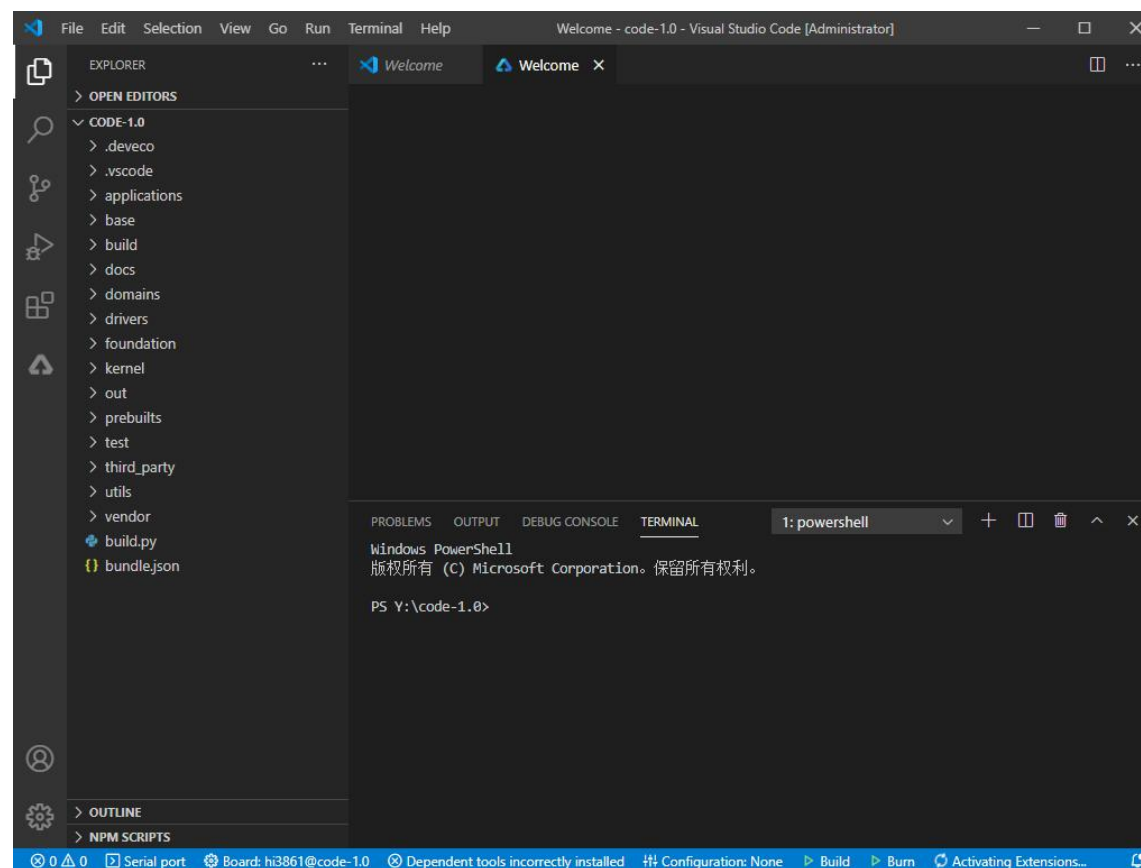
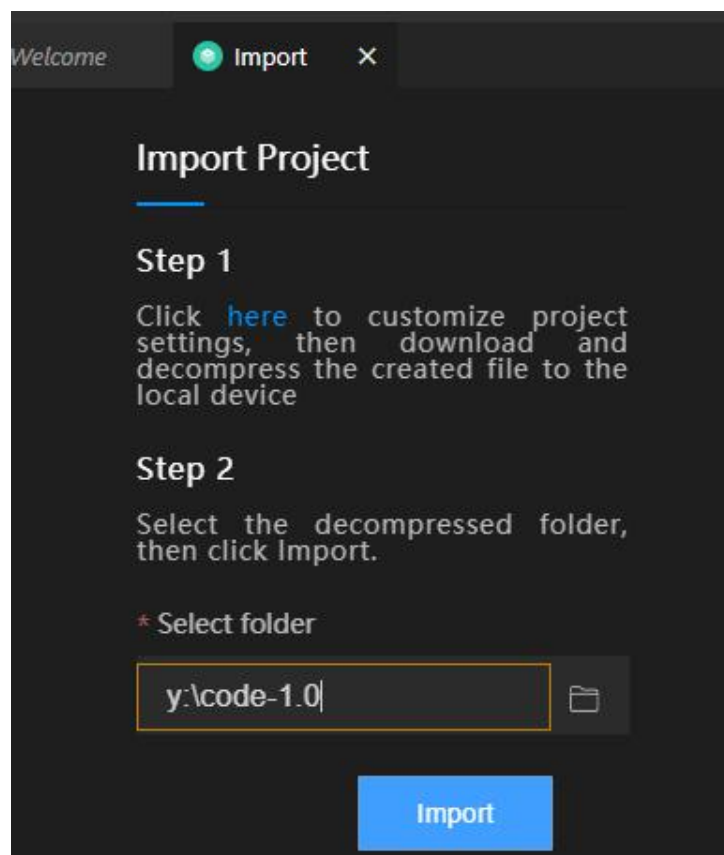
## 知识点2 【导入源码】

步骤1：在Visual Studio Code中，点击DevEco Device Tool插件按钮图标，然后点击 “Import” 按钮



## 知识点2 【导入源码】

步骤2：选择本地映射的HarmonyOS源码文件夹，然后点击“Import”导入工程



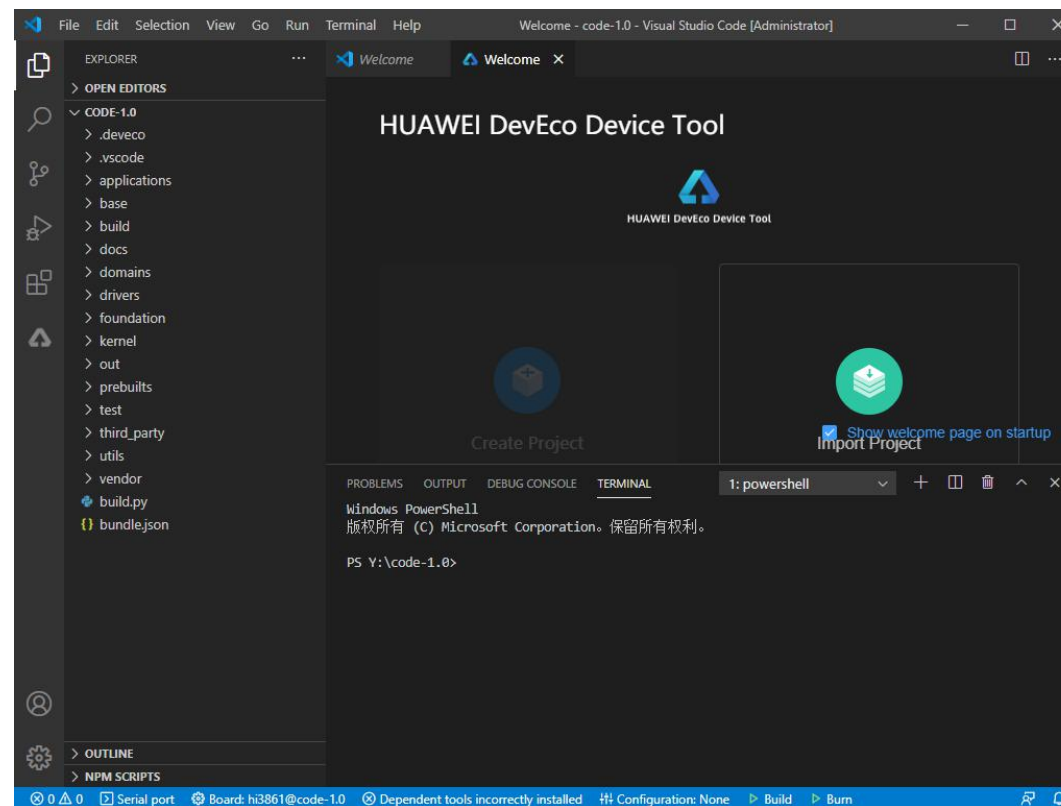
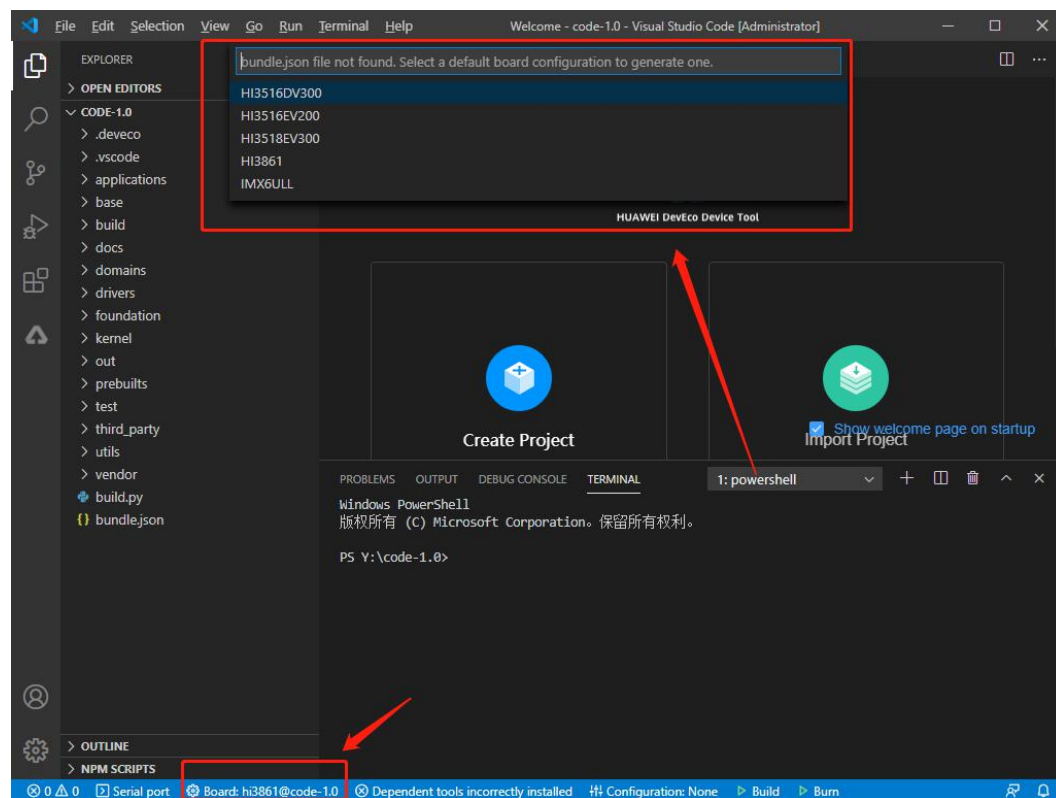
## 知识点3 【配置工程】

导入工程源码文件后，请对工程进行配置，如工程烧录和调试依赖的工具等信息，实现步骤：

1. 点击底部工具栏中的“Board”按钮，选择对应开发板的配置模板，例如：Hi3516DV300
2. 在命令行工具中，分别执行如下命令，下载烧录依赖工具

## 知识点3 【配置工程】

步骤1：点击底部工具栏中的“Board”按钮，选择对应开发板的配置模板



## 知识点3 【配置工程】

步骤2：在命令行工具中，分别执行如下命令，下载烧录依赖工具：

- `npm install -g tftp`
- `npm install -g serialport`
- `npm install -g @serialport/parser-readline`
- `npm install -g usb`
- `npm install -g crc`

说明：“`npm install -g usb`” 命令仅用于Hi3516/Hi3518系列开发板使用USB烧录时需要执行。

## 知识点3 【配置工程】

操作演示



## 本节小结

本讲所学知识点有：

- 知识点1：映射HarmonyOS源码
- 知识点2：导入源码
- 知识点3：配置工程



## 第2节：编写第一个Hello world程序

### ■ 知识点1：Hello world程序编写



## 知识点1 【Hello world程序编写】

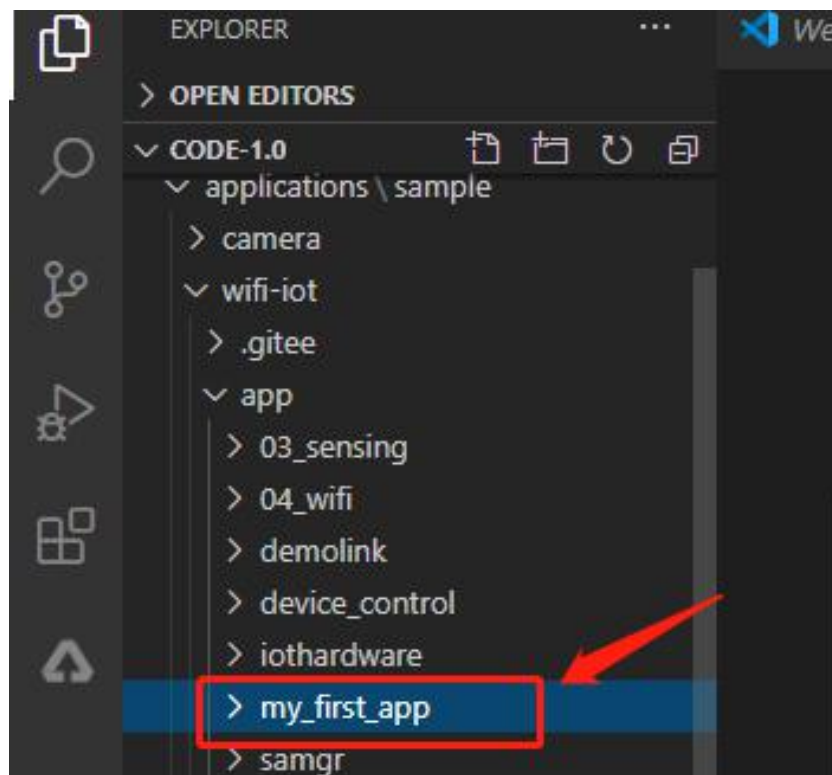
实现步骤：

1. 在./applications/sample/wifi-iot/app下创建工程目录。
2. 编写业务代码
3. 编写用于将业务构建成静态库的BUILD.gn文件。
4. 编写模块BUILD.gn文件，指定需参与构建的特性模块

## 知识点1 【Hello world程序编写】

步骤1：在./applications/sample/wifi-iot/app下创建工程目录

先在./applications/sample/wifi-iot/app路径下新建一个目录（或一套目录结构）chapter\_02，用于存放业务源码文件。



## 知识点1 【Hello world程序编写】

### 步骤2：编写业务代码

在chapter\_02目录下，新建hello\_world.c，在hello\_world.c中新建业务入口函数HelloWorld，并实现业务逻辑。  
并在代码最下方，使用HarmonyOS启动恢复模块接口SYS\_RUN()启动业务。（SYS\_RUN定义在ohos\_init.h文件中）

```
#include <stdio.h>
#include "ohos_init.h"
#include "ohos_types.h"

void HelloWorld(void)
{
    printf("[DEMO] Hello world.\n");
}

SYS_RUN(HelloWorld); //启动恢复模块接口SYS_RUN()
```

## 知识点1 【Hello world程序编写】

步骤3：编写用于将业务构建成静态库的BUILD.gn文件。

新建./applications/sample/wifi-iot/app/my\_first\_app下的BUILD.gn文件，并完成如下配置。

如步骤1所述，BUILD.gn文件由三部分内容（目标、源文件、头文件路径）构成，需由开发者完成填写。。

```
# static_library中指定业务模块的编译结果.为静态库文件libmyapp.a,开发者根据实际情况完成填写。  
#sources中指定静态库.a所依赖的.c文件及其路径，若路径中包含"/"则表示绝对路径（此处为代码根路径），若不包含"/"则表示相对路径。  
#include_dirs中指定source所需要依赖的.h文件路径。  
static_library("myapp") {  
    sources = [  
        "hello_world.c"  
    ]  
    include_dirs = [  
        "../utils/native/lite/include"  
    ]  
}
```

## 知识点1 【Hello world程序编写】

步骤4：编写模块BUILD.gn文件，指定需参与构建的特性模块

配置./applications/sample/wifi-iot/app/BUILD.gn文件，在features字段中增加索引，使目标模块参与编译。

features字段指定业务模块的路径和目标，以my\_first\_app举例，features字段配置如下。

```
#my_first_app是相对路径，指向./applications/sample/wifi-iot/app/my_first_app/BUILD.gn。  
#myapp是目标，指向./applications/sample/wifi-iot/app/my_first_app/BUILD.gn中的static_library("myapp")。
```

```
import("//build/lite/config/component/lite_component.gni")
```

```
lite_component("app") {  
    features = [  
        "my_first_app:myapp",  
    ]  
}
```

# 知识点1 【Hello world程序编写】

操作演示



## 本节小结

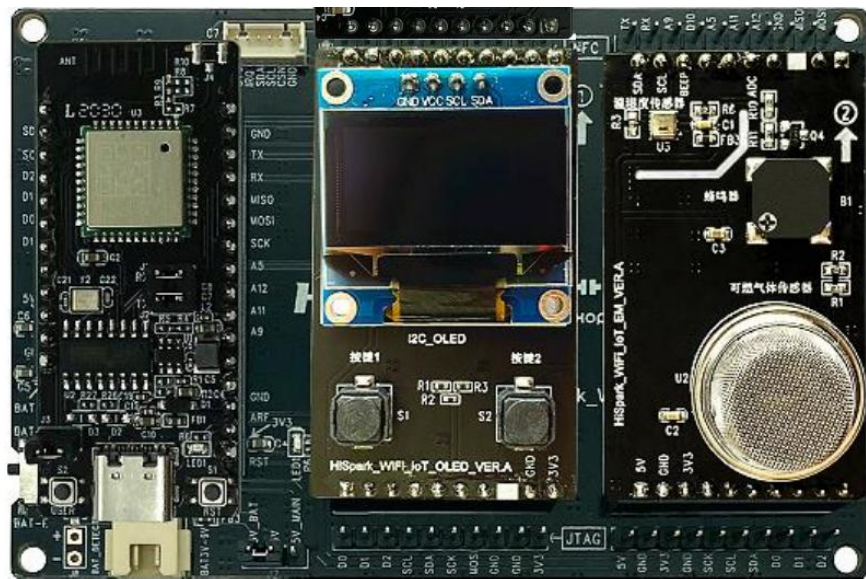
本讲所学知识点有：

- 知识点1：Hello world程序编写



## 第3节：完成HelloWorld程序编译、烧写、运行

- 知识点1：编译源码
- 知识点2：烧写
- 知识点3：运行



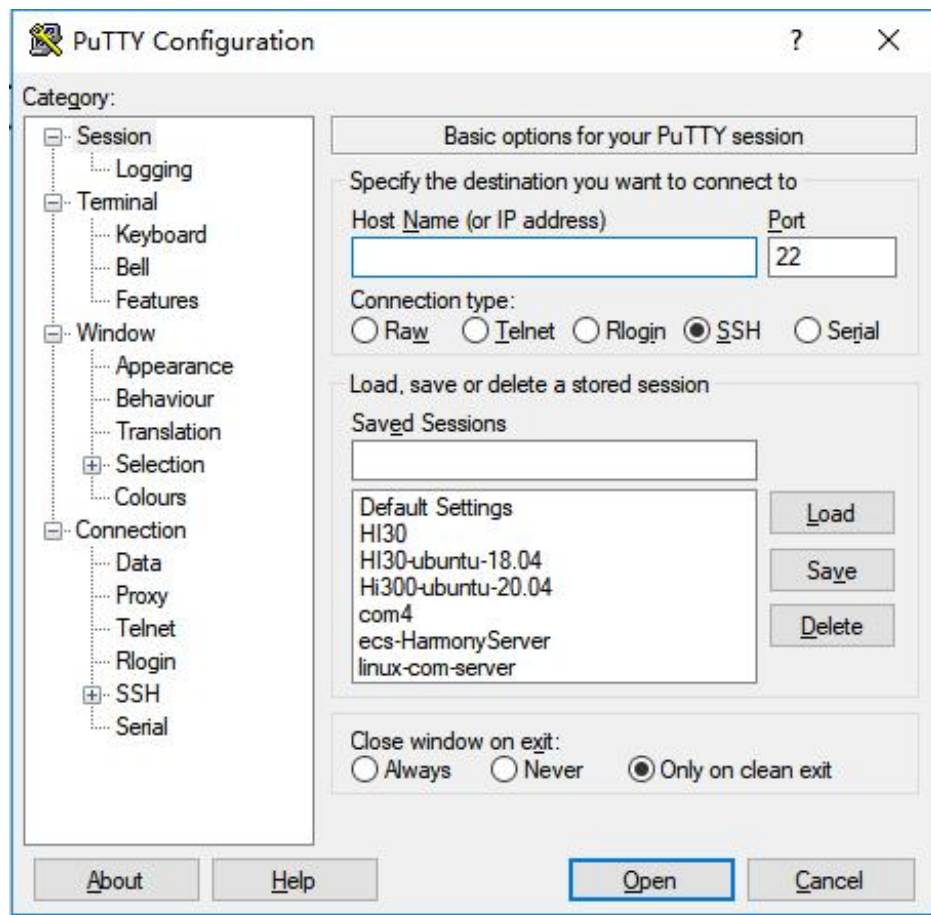
## 知识点1 【编译源码】

实现步骤：

1. 打开PuTTY工具。
2. 加载连接配置
3. 输入密码。
4. 编译

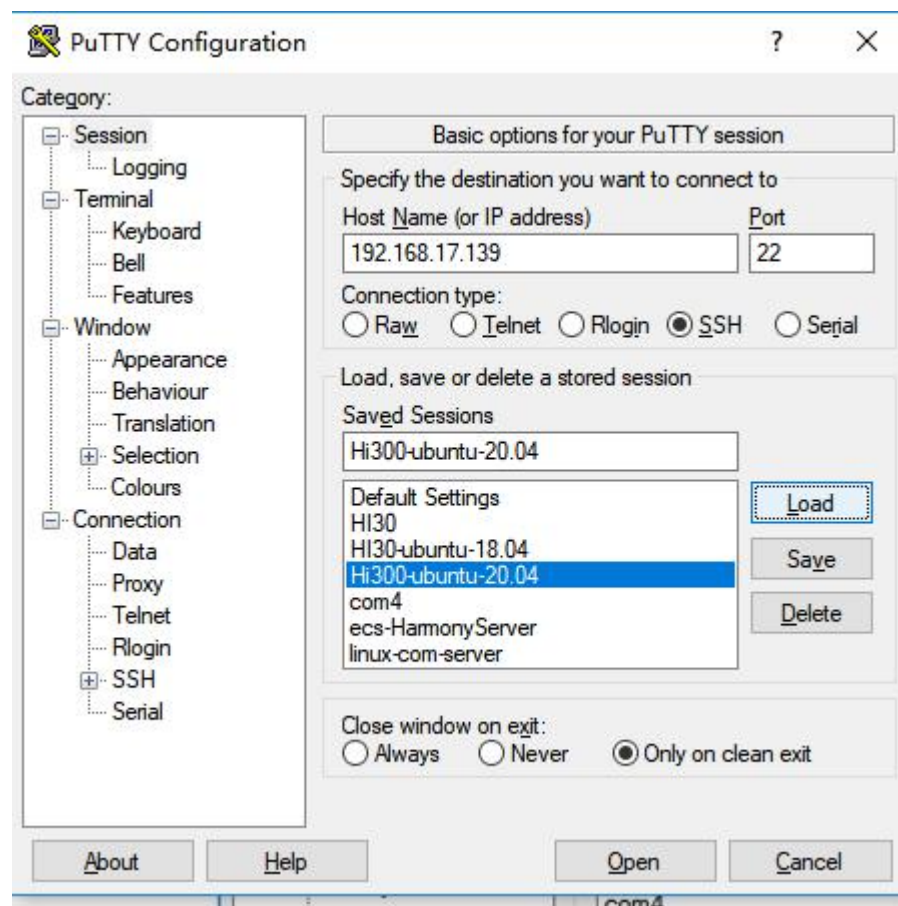
## 知识点1 【编译源码】

### 步骤1：打开PuTTY工具



## 知识点1 【编译源码】

步骤2：加载连接配置，点击open



## 知识点1 【编译源码】

步骤3：输入密码，并进行源码根目录

```
192.168.17.139 - PuTTY
Using username "gyz".
gyz@192.168.17.139's password: 
```

```
gyz@ubuntu: ~/sharefolder/code-1.0
Using username "gyz".
gyz@192.168.17.139's password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

22 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
*** 需要重启系统 ***
Last login: Sun Nov 15 17:25:39 2020 from 192.168.17.1
gyz@ubuntu:~$ cd sharefolder/code-1.0/
gyz@ubuntu:~/sharefolder/code-1.0$ 
```

# 知识点1 【编译源码】

步骤3：编译，输入： python build.py wifiiot 进行系统源码编译，出现BUILD SUCCESS 编译成功

```
gyz@ubuntu: ~/sharefolder/code-1.0
Using username "gyz".
gyz@192.168.17.139's password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

22 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
*** 需要重启系统 ***
Last login: Sun Nov 15 17:25:39 2020 from 192.168.17.1
gyz@ubuntu:~$ cd sharefolder/code-1.0/
gyz@ubuntu:~/sharefolder/code-1.0$ python build.py wifiiot
```

```
gyz@ubuntu: ~/sharefolder/code-1.0
[common sign][01]=[0x21]
[common sign][30]=[0x2f]
[common sign][31]=[0x82]
[section sign][00]=[0x3]
[section sign][01]=[0x99]
[section sign][30]=[0xe6]
[section sign][31]=[0xec]
[image_id=0x3c78961e][struct_version=0x0]
[hash_alg=0x0][sign_alg=0x3f][sign_param=0x0]
[section_count=0x1]
[section0_compress=0x0][section0_offset=0x3c0][section0_len=0x5f60]
[section1_compress=0x0][section1_offset=0x0][section1_len=0x0]
-----output/bin/Hi3861_wifiiot_app_flash_boot_ota.bin image info print e
nd-----

< ~~~~~ >
< BUILD SUCCESS >
< ~~~~~ >

See build log from: /home/gyz/sharefolder/code-1.0/vendor/hisi/hi3861/hi3861/bui
ld/build_tmp/logs/build_kernel.log
[197/197] STAMP obj/vendor/hisi/hi3861/hi3861/run_wifiiot_scons.stamp
ohos wifiiot build success!
gyz@ubuntu:~/sharefolder/code-1.0$
```



## 知识点1 【编译源码】

操作演示



## 知识点2 【烧写】

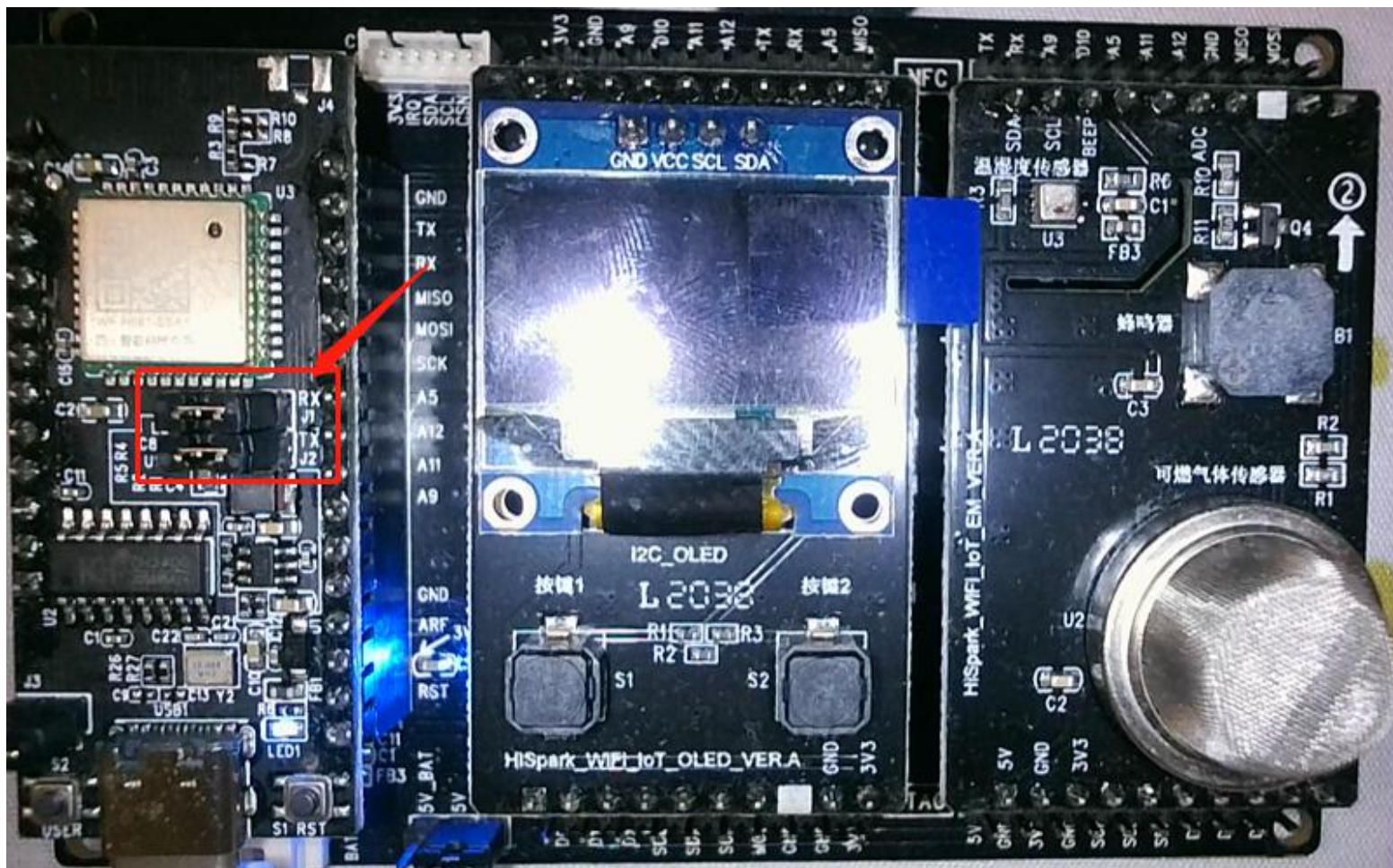
实现步骤：

1. Type-C接口线与开发板进行连，同时，需要将开发板上的J1、J2跳线进行连接
2. 打开电脑的设备管理器，查看并记录对应的串口号
3. 点击“Configure > Burn”，进入烧录配置界面，设置RISC-V系列芯片烧录信息
4. 设置烧录参数，最后保存
5. 在DevEco Device Tool中，点击Burn后的按钮开始烧录
6. 输出控制台会提示“Please reset board”，按下开发板上的RST键，重启开发板
7. 重启开发板后，请等待烧录完成，当控制台输出如下信息时，表示烧录成功。



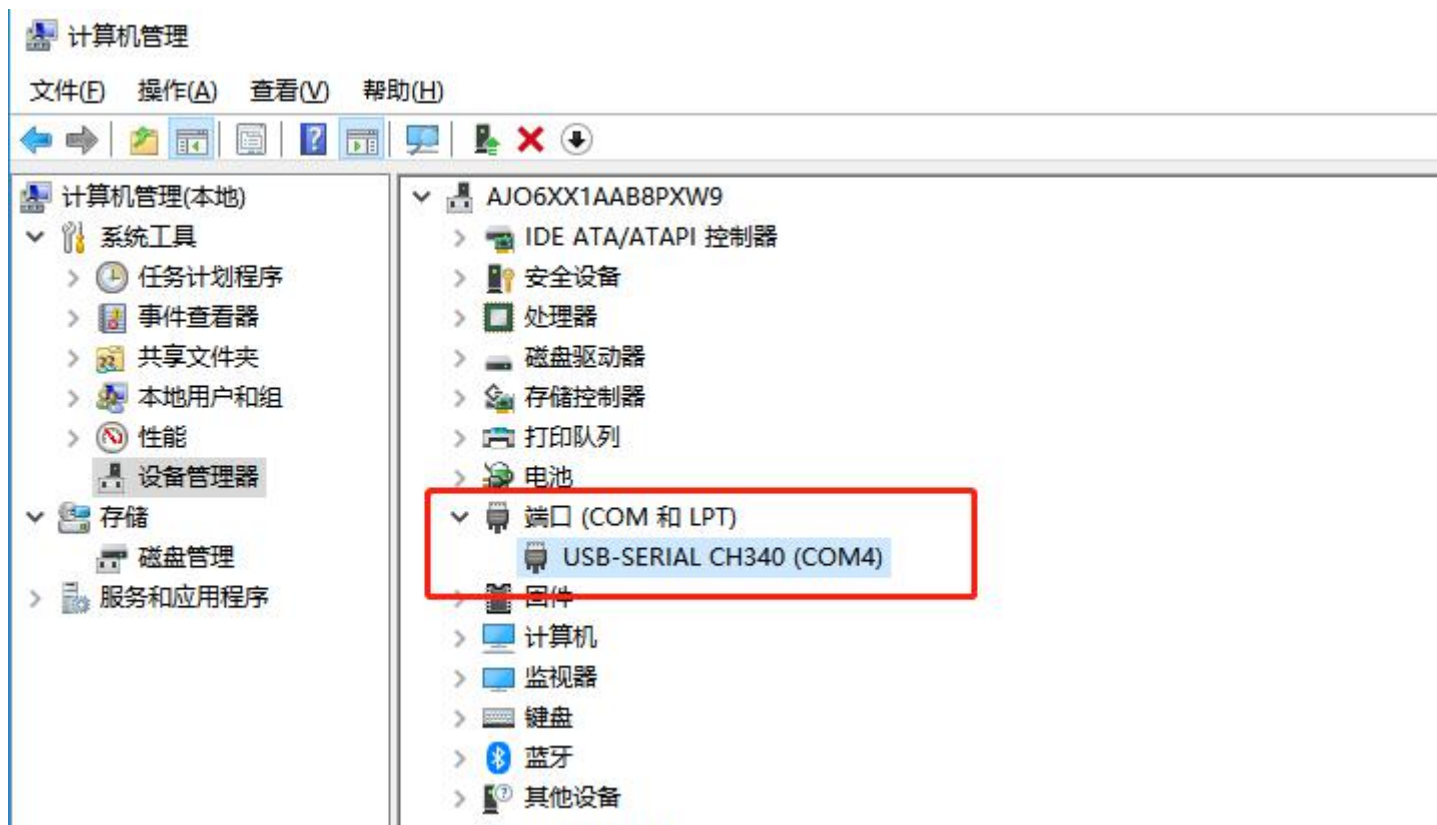
## 知识点2【烧写】

步骤1：Type-C接口线与开发板进行连，同时，需要将开发板上的J1、J2跳线进行连接



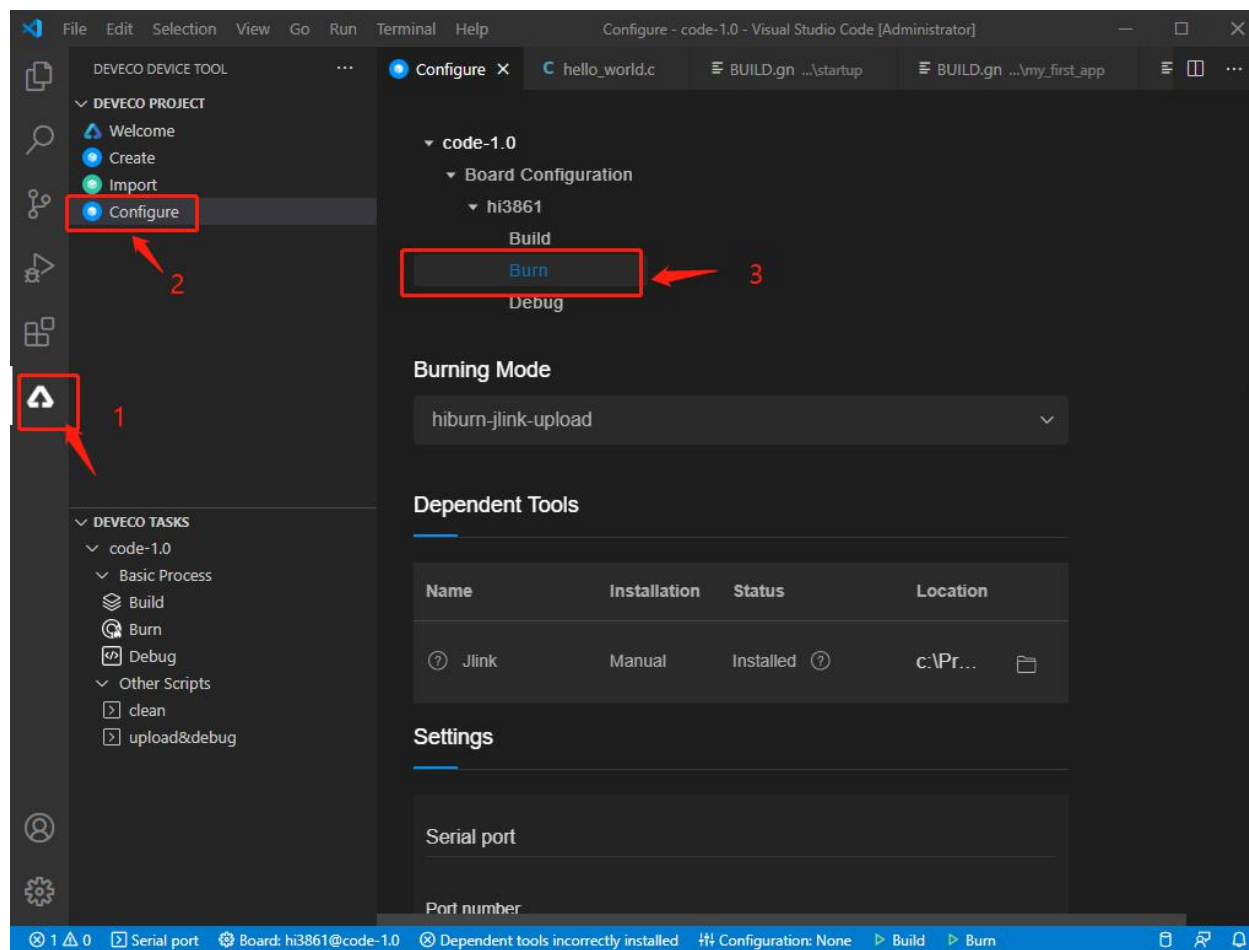
## 知识点2【烧写】

步骤2：打开电脑的设备管理器，查看并记录对应的串口号



## 知识点2【烧写】

步骤3：点击 “Configure > Burn”，进入烧录配置界面，设置RISC-V系列芯片烧录信息



## 知识点2【烧写】

### 步骤4：设置烧录参数，最后保存

- 设置串口信息（Serial port）：设置Port number，请选择2中查询的串口号；Baud Rate和Data Bits参数，已根据开发板进行适配，保持默认值即可。
- 设置烧录文件信息：如果需要烧录多个程序到不同的设备分区，可以点击“New”进行添加。
- Enable：烧录操作时，是否烧录该文件。
- File Name：选择待烧录的程序文件，选择编译生成的“out\wifiiot\Hi3861\_wifiiot\_app\_allinone.bin”文件。
- Mode：指定烧录工具，固定选择Hiburn。

**Burn Files**

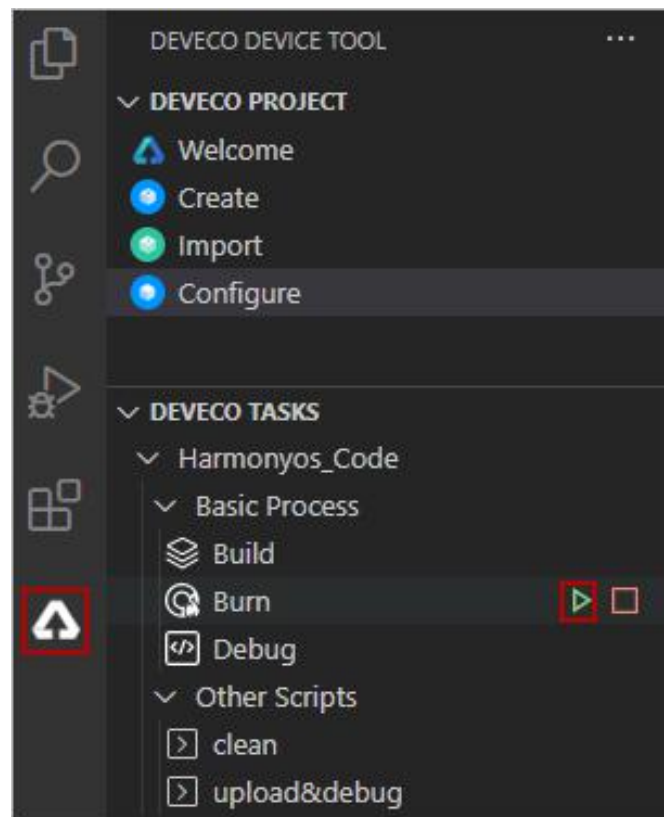
Enable	File Name	Mode
<input checked="" type="checkbox"/>	z:\code-1.0\out\wifiiot\Hi386...	Hiburn

New

Save

## 知识点2【烧写】

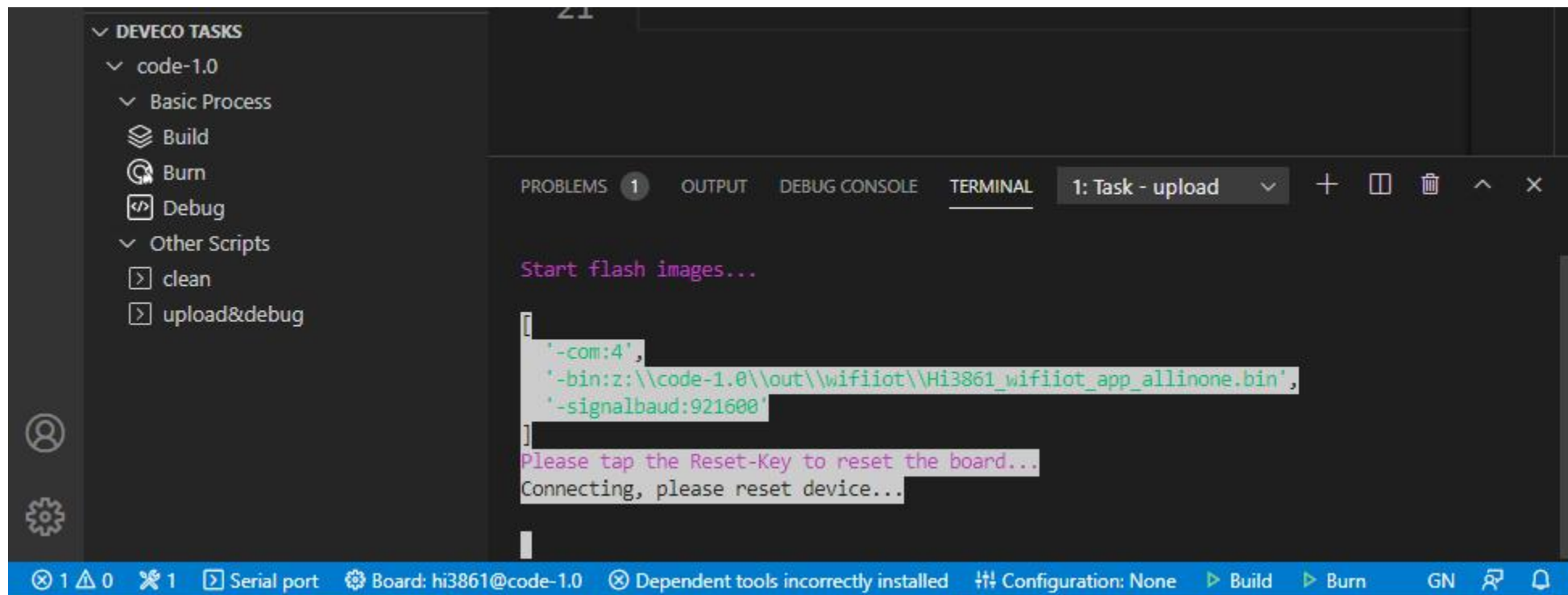
步骤5：在DevEco Device Tool中，点击Burn后的按钮开始烧录





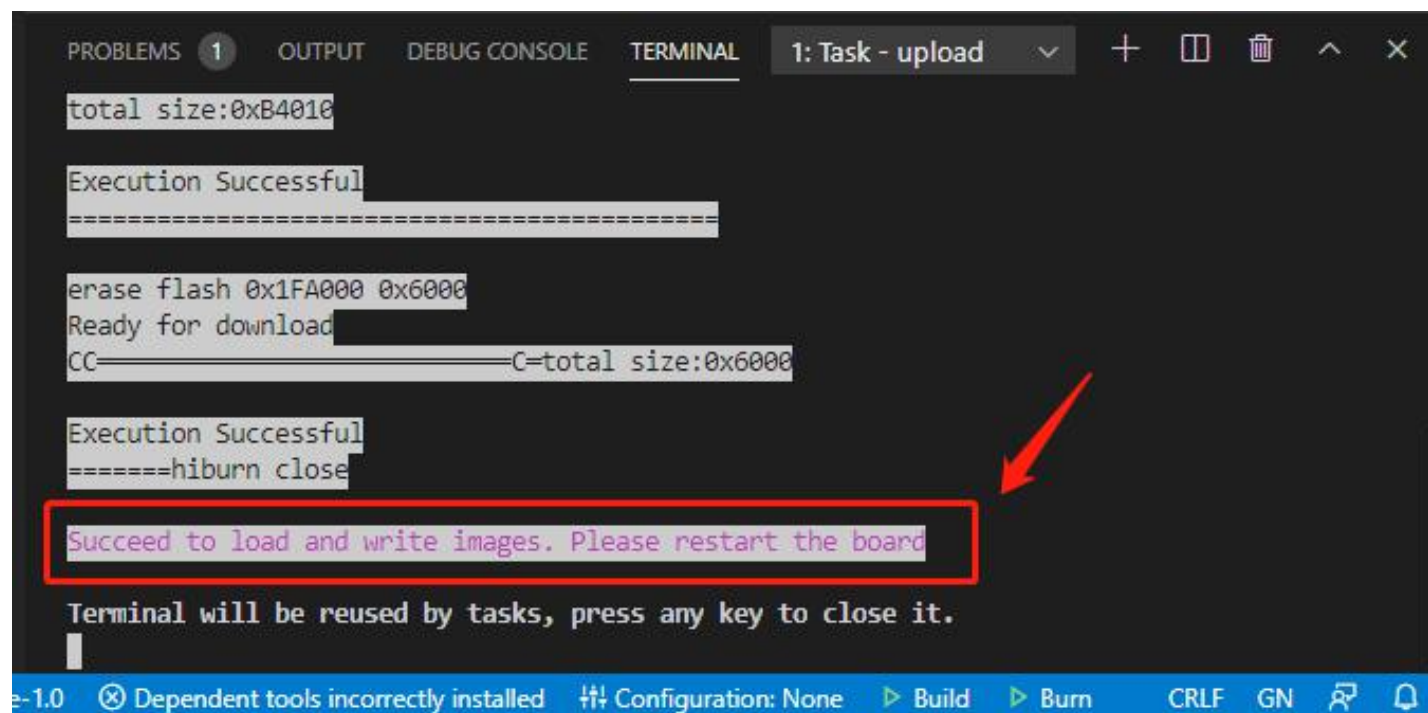
## 知识点2【烧写】

步骤6：输出控制台会提示 “Please reset board” ，按下开发板上的RST键，重启开发板



## 知识点2【烧写】

步骤7：重启开发板后，请等待烧录完成，当控制台输出如下信息时，表示烧录成功



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: Task - upload
total size:0xB4010
Execution Successful
=====
erase flash 0x1FA000 0x6000
Ready for download
CC=====C=total size:0x6000
Execution Successful
=====hiburn close
Succeed to load and write images. Please restart the board
Terminal will be reused by tasks, press any key to close it.
```

## 知识点2【烧写】

### 操作演示





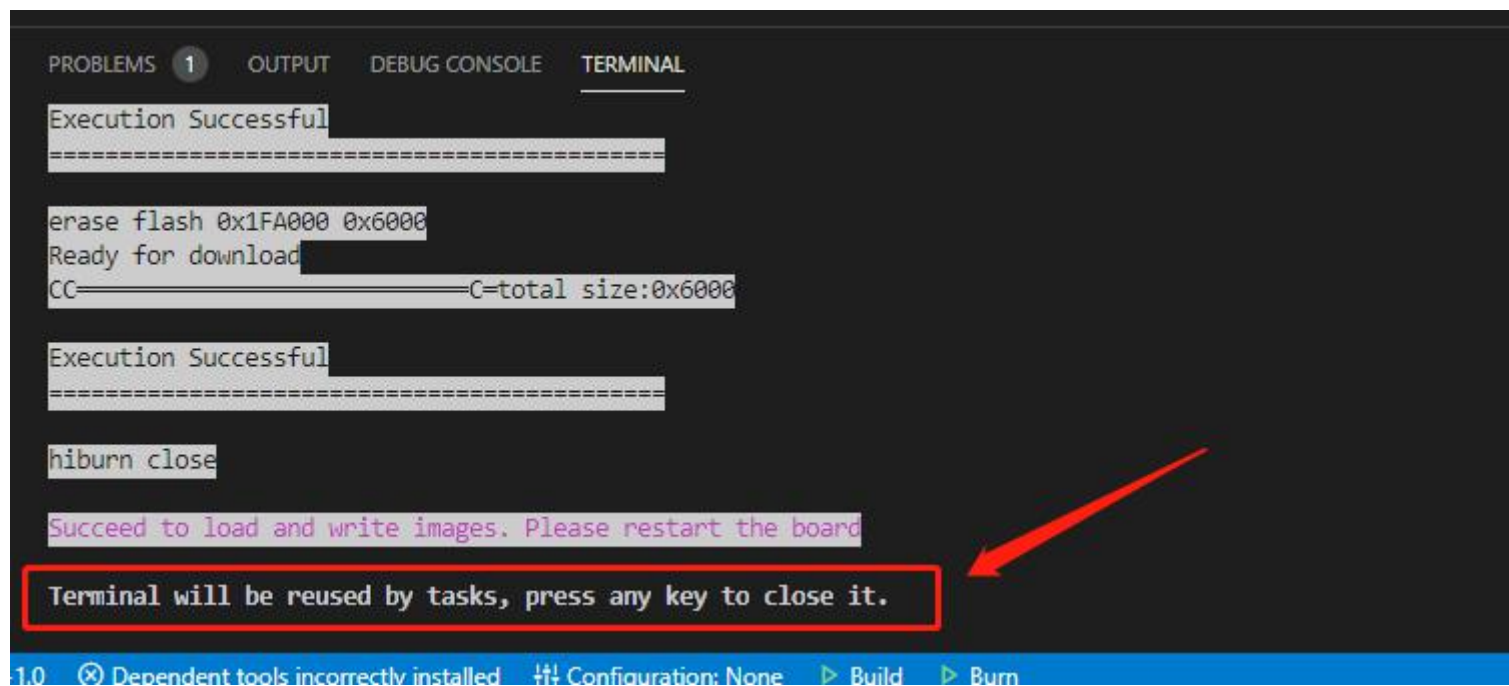
## 知识点3 【运行】

实现步骤：

1. 烧写完成后，点击任意退出连接
2. 启动串口连接
3. 重启开发板

## 知识点3 【运行】

步骤1：烧写完成后，点击任意退出连接



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
Execution Successful
=====
erase flash 0x1FA000 0x6000
Ready for download
CC=====C=total size:0x6000
Execution Successful
=====
hiburn close
Succeed to load and write images. Please restart the board
Terminal will be reused by tasks, press any key to close it.
1.0 (X) Dependent tools incorrectly installed # Configuration: None Build Burn
```

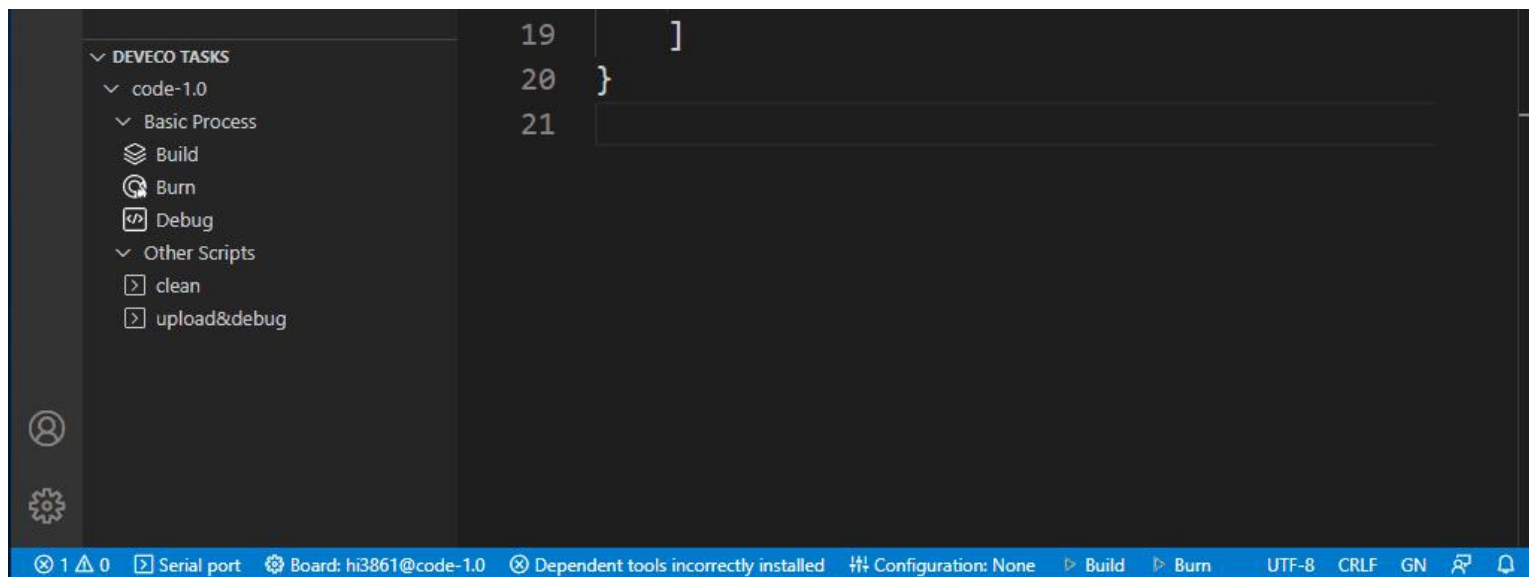
## 知识点3 【运行】

### 步骤2：启动串口连接

a) 点击DevEco Device Tool工具栏中的Serial port按钮。

b) 设置串口连接信息：

- 串口号：请输入对应的串口号，点击Enter按钮。
- 波特率：默认波特率为115200，直接点击Enter按钮。
- 数据位：默认数据位为8位，直接点击Enter按钮。
- 停止位：默认停止位为1位，直接点击Enter按钮。
- 是否加入换行符：默认为0，不需要换行。Hi3861系列开发板需要输入“1”，再点击Enter按钮。



## 知识点3 【运行】

### 步骤3：重启开发板

```
File Edit Selection View Go Run Terminal Help
hello_world.c - code-1.0 - Visual Studio Code [Administrator]

EXPLORER
OPEN EDITORS
CODE-1.0
.vscode
applications \ sample
camera
wifi-iot
.git
app
demolink
iohardware
my_first_app
BUILD.gn
C hello_world.c 1
samgr
startup
BUILD.gn
LICENSE
base
build
docs
domains
drivers
foundation
kernel
out
prebuilts
test
third_party
utils
OUTLINE
NPM SCRIPTS

C hello_world.c X BUILD.gn ...\startup BUILD.gn ...\my_first_app BUILD.gn ...\app
applications > sample > wifi-iot > app > my_first_app > C hello_world.c > ...
1 #include <stdio.h>
2 #include "ohos_init.h"
3 #include "ohos_types.h"
4
5 void HelloWorld(void)
6 {
7     printf("[DEMO] Hello world.\n");
8 }
9 SYS_RUN(HelloWorld); //启动恢复模块接口SYS_RUN()
10

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
1: Task - serial-0
[DEMO] Hello world.
00 00:00:00 0 132 D 0/HIVIEW: hilog init success.
00 00:00:00 0 132 D 0/HIVIEW: log limit init success.
00 00:00:00 0 132 I 1/SAMGR: Bootstrap core services(count:3).
00 00:00:00 0 132 I 1/SAMGR: Init service:0x4ae53c TaskPool:0xfa1e4
00 00:00:00 0 132 I 1/SAMGR: Init service:0x4ae560 TaskPool:0xfa854
00 00:00:00 0 132 I 1/SAMGR: Init service:0x4ae670 TaskPool:0xfaa14
00 00:00:00 0 164 I 1/SAMGR: Init service 0x4ae560 <time: 0ms> success!
00 00:00:00 0 64 I 1/SAMGR: Init service 0x4ae53c <time: 0ms> success!
00 00:00:00 0 8 D 0/HIVIEW: hiview init success.
00 00:00:00 0 8 I 1/SAMGR: Init service 0x4ae670 <time: 0ms> success!
00 00:00:00 0 8 I 1/SAMGR: Initialized all core system services!
00 00:00:00 0 64 I 1/SAMGR: Bootstrap system and application services(count:0).
00 00:00:00 0 64 I 1/SAMGR: Initialized all system and application services!
00 00:00:00 0 64 I 1/SAMGR: Bootstrap dynamic registered services(count:0).
```

## 知识点2【烧写】

### 操作演示



## 本节小结

本讲所学知识点有：

- 知识点1：编译源码
- 知识点2：烧写
- 知识点3：运行

## 第4节：启动流程分析

- 知识点1：启动流程分析
- 知识点2：测试验证

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: node serialterminal.js <

The port name:
COM4
Pick the one for opening: com4
What is the Baud Rate? The default is 115200. Enter:
What is the Data Bits? The default is 8. Enter:
What is the Stop Bits? The default is 1. Enter:
Set endline characters as "\r\n"? The default is 0, Enter 1 to set. Enter: 1
Open serial port: com4

ready to OS start
sdk ver:Hi3861V100R001C00SPC025 2020-09-03 18:10:00
FileSystem mount ok.
wifi init success!
[DEMO] Hello world.

00 00:00:00 0 132 D 0/HIVIEW: hilog init success.
00 00:00:00 0 132 D 0/HIVIEW: log limit init success.
00 00:00:00 0 132 I 1/SAMGR: Bootstrap core services(count:3).
00 00:00:00 0 132 I 1/SAMGR: Init service:0x4ae53c TaskPool:0xfa1e4
00 00:00:00 0 132 I 1/SAMGR: Init service:0x4ae560 TaskPool:0xfa854
00 00:00:00 0 132 I 1/SAMGR: Init service:0x4ae670 TaskPool:0xfaa14
00 00:00:00 0 164 I 1/SAMGR: Init service 0x4ae560 <time: 0ms> success!
00 00:00:00 0 64 I 1/SAMGR: Init service 0x4ae53c <time: 0ms> success!
00 00:00:00 0 8 D 0/HIVIEW: hiview init success.
00 00:00:00 0 8 I 1/SAMGR: Init service 0x4ae670 <time: 0ms> success!
00 00:00:00 0 8 I 1/SAMGR: Initialized all core system services!
00 00:00:00 0 64 I 1/SAMGR: Bootstrap system and application services(count:0).
00 00:00:00 0 64 I 1/SAMGR: Initialized all system and application services!
00 00:00:00 0 64 I 1/SAMGR: Bootstrap dynamic registered services(count:0).
```

## 知识点1 【启动流程】

启动步骤:

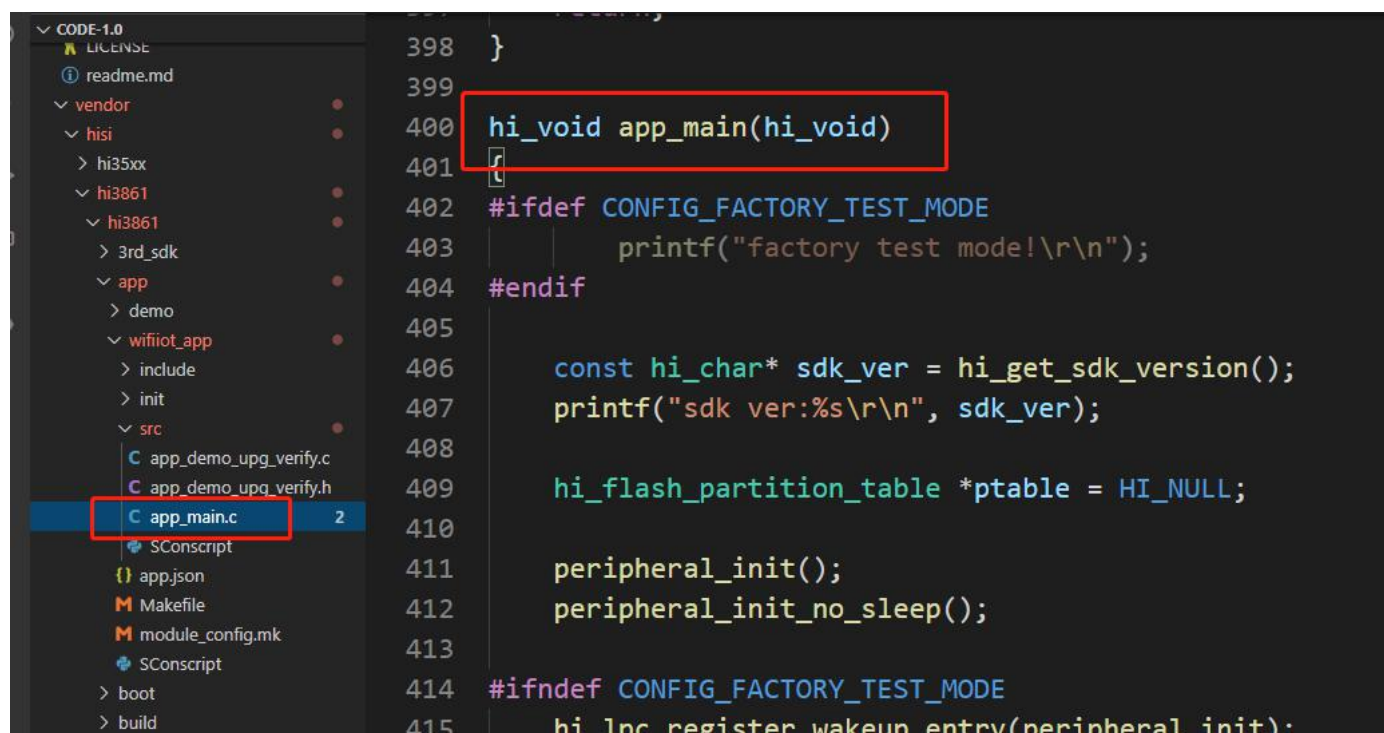
1. 内核启动后，第一个入口函数hi\_void app\_main(hi\_void)
2. 调用鸿蒙系统初始化方法：HOS\_SystemInit()
3. 调用MODULE\_INIT(run);
4. 调用SYS\_RUN>HelloWorld);



# 知识点1 【启动流程】

步骤1：内核启动后，第一个入口函数hi\_void app\_main(hi\_void)

➤ 路径：vendor\hisi\hi3861\hi3861\app\wifiot\_app\src\app\_main.c

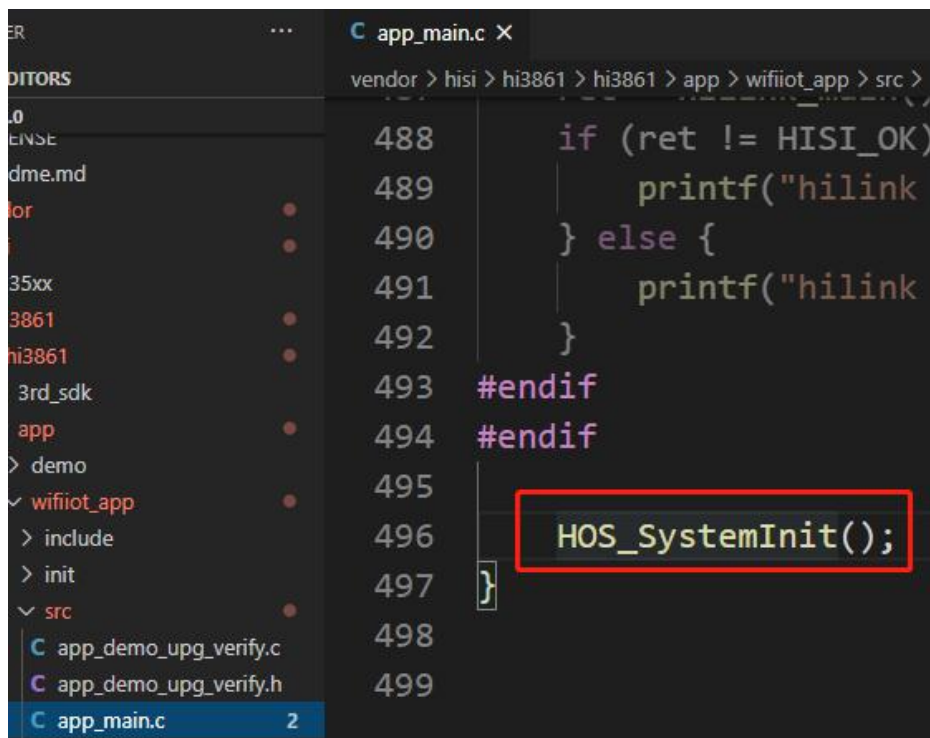


```
398 }
399
400 hi_void app_main(hi_void)
401 {
402     #ifdef CONFIG_FACTORY_TEST_MODE
403         printf("factory test mode!\r\n");
404     #endif
405
406     const hi_char* sdk_ver = hi_get_sdk_version();
407     printf("sdk ver:%s\r\n", sdk_ver);
408
409     hi_flash_partition_table *ptable = HI_NULL;
410
411     peripheral_init();
412     peripheral_init_no_sleep();
413
414     #ifndef CONFIG_FACTORY_TEST_MODE
415         hi_lpc_register_wakeup_entry(peripheral_init);
```

## 知识点1 【启动流程】

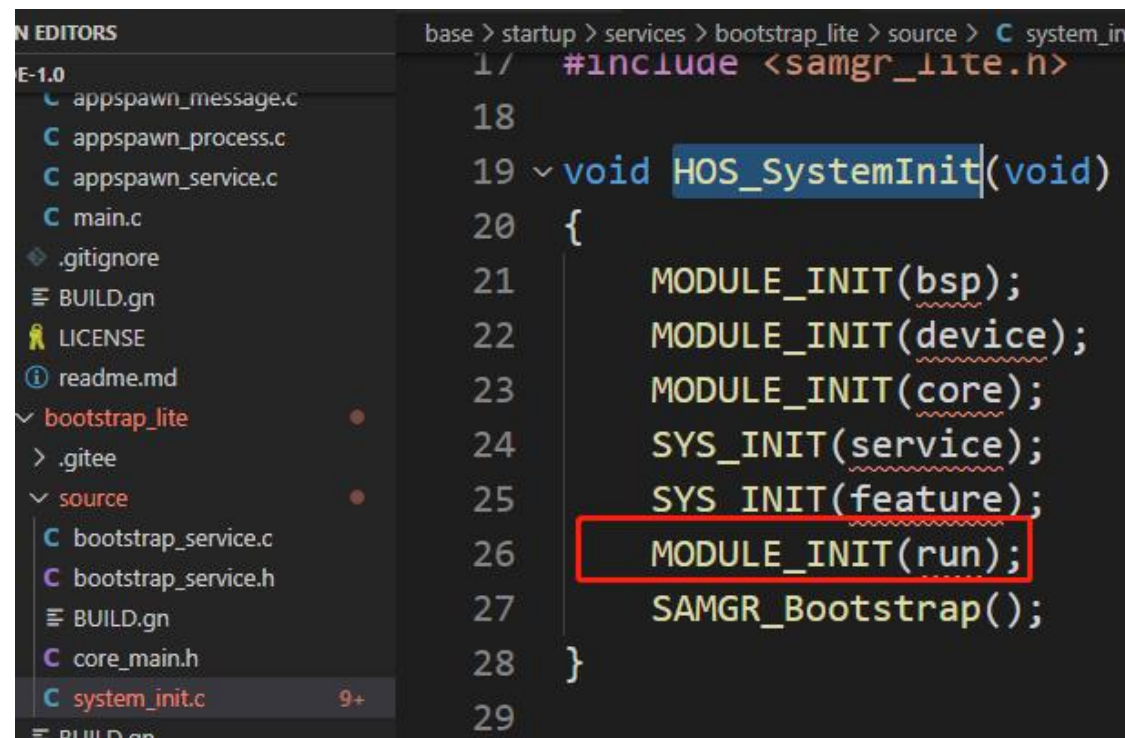
步骤2：调用鸿蒙系统初始化方法：HOS\_SystemInit()

➤ 路径：base/startup/services/bootstrap\_lite/source/system\_init.c



The screenshot shows a code editor with a file explorer on the left. The file explorer shows a directory structure: vendor > hisi > hi3861 > app > wifiot\_app > src. The main editor displays the file app\_main.c. The code shows a conditional execution block where HOS\_SystemInit() is called. The function call is highlighted with a red rectangle.

```
488     if (ret != HISI_OK)
489     {
490         printf("hilink
491     } else {
492         printf("hilink
493     }
494 #endif
495 #endif
496 HOS_SystemInit();
497 }
498
499
```



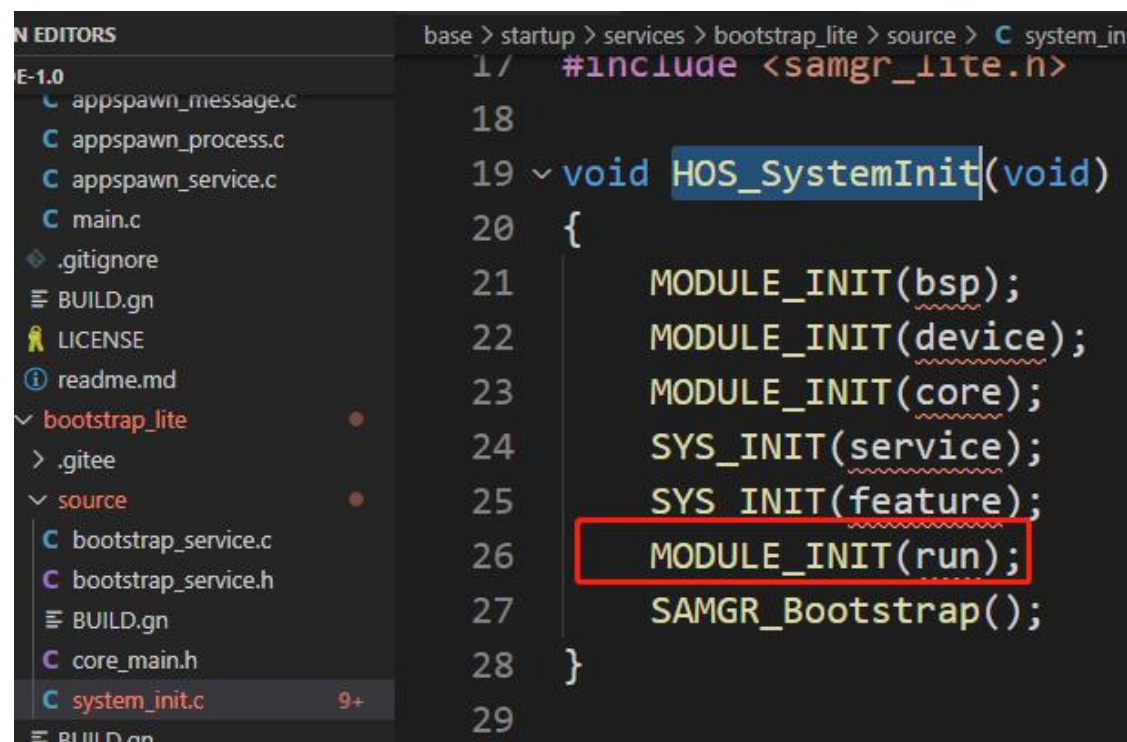
The screenshot shows a code editor with a file explorer on the left. The file explorer shows a directory structure: base > startup > services > bootstrap\_lite > source. The main editor displays the file system\_init.c. The code shows the definition of the HOS\_SystemInit() function. The function name is highlighted with a blue box, and the MODULE\_INIT(run); line is highlighted with a red rectangle.

```
17 #include <samgr_lite.h>
18
19 void HOS_SystemInit(void)
20 {
21     MODULE_INIT(bsp);
22     MODULE_INIT(device);
23     MODULE_INIT(core);
24     SYS_INIT(service);
25     SYS_INIT(feature);
26     MODULE_INIT(run);
27     SAMGR_Bootstrap();
28 }
29
```

## 知识点1 【启动流程】

步骤3：调用MODULE\_INIT(run);

路径：base/startup/services/bootstrap\_lite/source/system\_init.c



```
base > startup > services > bootstrap_lite > source > C system_init.c
1/ #include <samgr_lite.h>
18
19 void HOS_SystemInit(void)
20 {
21     MODULE_INIT(bsp);
22     MODULE_INIT(device);
23     MODULE_INIT(core);
24     SYS_INIT(service);
25     SYS_INIT(feature);
26     MODULE_INIT(run);
27     SAMGR_Bootstrap();
28 }
29
```

## 知识点1 【启动流程】

步骤4：调用SYS\_RUN>HelloWorld);;

路径：base/startup/services/bootstrap\_lite/source/system\_init.c

```
app_main.c  C hello_world.c X  C system_init.c  C ohos_init.h ●
applications > sample > wifi-iot > app > my_first_app > C hello_world.c > ...
1  #include <stdio.h>
2  #include "ohos_init.h"
3  #include "ohos_types.h"
4
5  void HelloWorld(void)
6  {
7      printf("[DEMO] Hello world.\n");
8  }
9  SYS_RUN(HelloWorld); //启动恢复模块接口SYS_RUN()
10
```

## 知识点2 【测试验证】

步骤:

1. 在hi\_void app\_main(hi\_void)中插入测试语句
2. 在HOS\_SystemInit()中插入测试语句
3. 重新编译后烧录。查看启动打印信息

## 知识点2 【测试验证】

步骤1：在hi\_void app\_main(hi\_void)中插入测试语句

```
app_main.c × hello_world.c system_init.c ohos_init.h
vendor > hisi > hi3861 > hi3861 > app > wifiot_app > src > app_main.c > app_main(hi_void)

493 #endif
494 #endif
495     printf("%s  %d\n", __FILE__, __LINE__);
496     HOS_SystemInit();
497     printf("%s  %d\n", __FILE__, __LINE__);
498 }
499
500
```



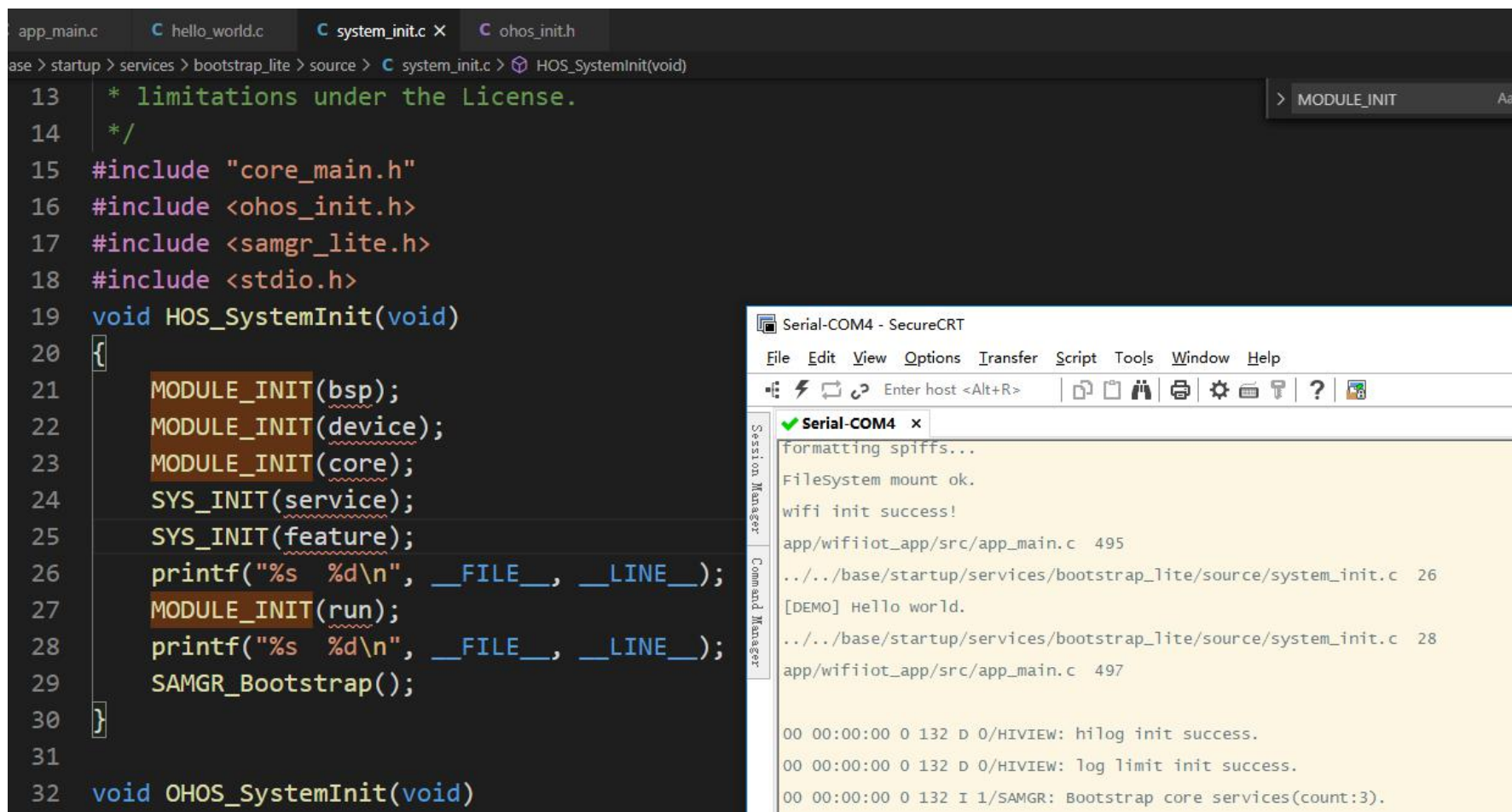
## 知识点2 【测试验证】

步骤2：在HOS\_SystemInit()中插入测试语句，（需添加stdio.h）

```
base > startup > services > bootstrap_lite > source > C system_init.c > ...
13  * limitations under the License.
14  */
15  #include "core_main.h"
16  #include <ohos_init.h>
17  #include <samgr_lite.h>
18  #include <stdio.h>
19  void HOS_SystemInit(void)
20  {
21      MODULE_INIT(bsp);
22      MODULE_INIT(device);
23      MODULE_INIT(core);
24      SYS_INIT(service);
25      SYS_INIT(feature);
26      printf("%s %d\n", __FILE__, __LINE__);
27      MODULE_INIT(run);
28      printf("%s %d\n", __FILE__, __LINE__);
29      SAMGR_Bootstrap();
30  }
```

## 知识点2 【测试验证】

步骤3：重新编译后烧录。查看启动打印信息



The image shows a development environment with two windows. The left window is a code editor showing the implementation of `HOS_SystemInit(void)` in `system_init.c`. The code includes headers for `core_main.h`, `ohos_init.h`, `samgr_lite.h`, and `stdio.h`. It defines `MODULE_INIT` macros for `bsp`, `device`, `core`, `run`, `service`, and `feature`, and calls `SYS_INIT` for `service` and `feature`. It also prints the file and line numbers and calls `SAMGR_Bootstrap()`. The right window is a SecureCRT terminal window titled "Serial-COM4 - SecureCRT" showing the boot logs. The logs indicate successful formatting of spiffs, mounting of the file system, successful WiFi initialization, and successful initialization of the HiView and log limit modules. The SAMGR module also reports successful bootstrap of core services.

```
13  * limitations under the License.
14  */
15  #include "core_main.h"
16  #include <ohos_init.h>
17  #include <samgr_lite.h>
18  #include <stdio.h>
19  void HOS_SystemInit(void)
20  {
21      MODULE_INIT(bsp);
22      MODULE_INIT(device);
23      MODULE_INIT(core);
24      SYS_INIT(service);
25      SYS_INIT(feature);
26      printf("%s %d\n", __FILE__, __LINE__);
27      MODULE_INIT(run);
28      printf("%s %d\n", __FILE__, __LINE__);
29      SAMGR_Bootstrap();
30  }
31
32  void OHOS_SystemInit(void)
```

Serial-COM4 - SecureCRT

File Edit View Options Transfer Script Tools Window Help

Enter host <Alt+R>

Serial-COM4 x

formatting spiffs...

FileSystem mount ok.

wifi init success!

app/wifiiot\_app/src/app\_main.c 495

../base/startup/services/bootstrap\_lite/source/system\_init.c 26

[DEMO] Hello world.

../base/startup/services/bootstrap\_lite/source/system\_init.c 28

app/wifiiot\_app/src/app\_main.c 497

00 00:00:00 0 132 D 0/HIVIEW: hilog init success.

00 00:00:00 0 132 D 0/HIVIEW: log limit init success.

00 00:00:00 0 132 I 1/SAMGR: Bootstrap core services(count:3).



## 知识点2 【测试验证】

### 操作演示



## 本节小结

本讲所学知识点有：

- 知识点1：启动流程分析
- 知识点2：测试验证

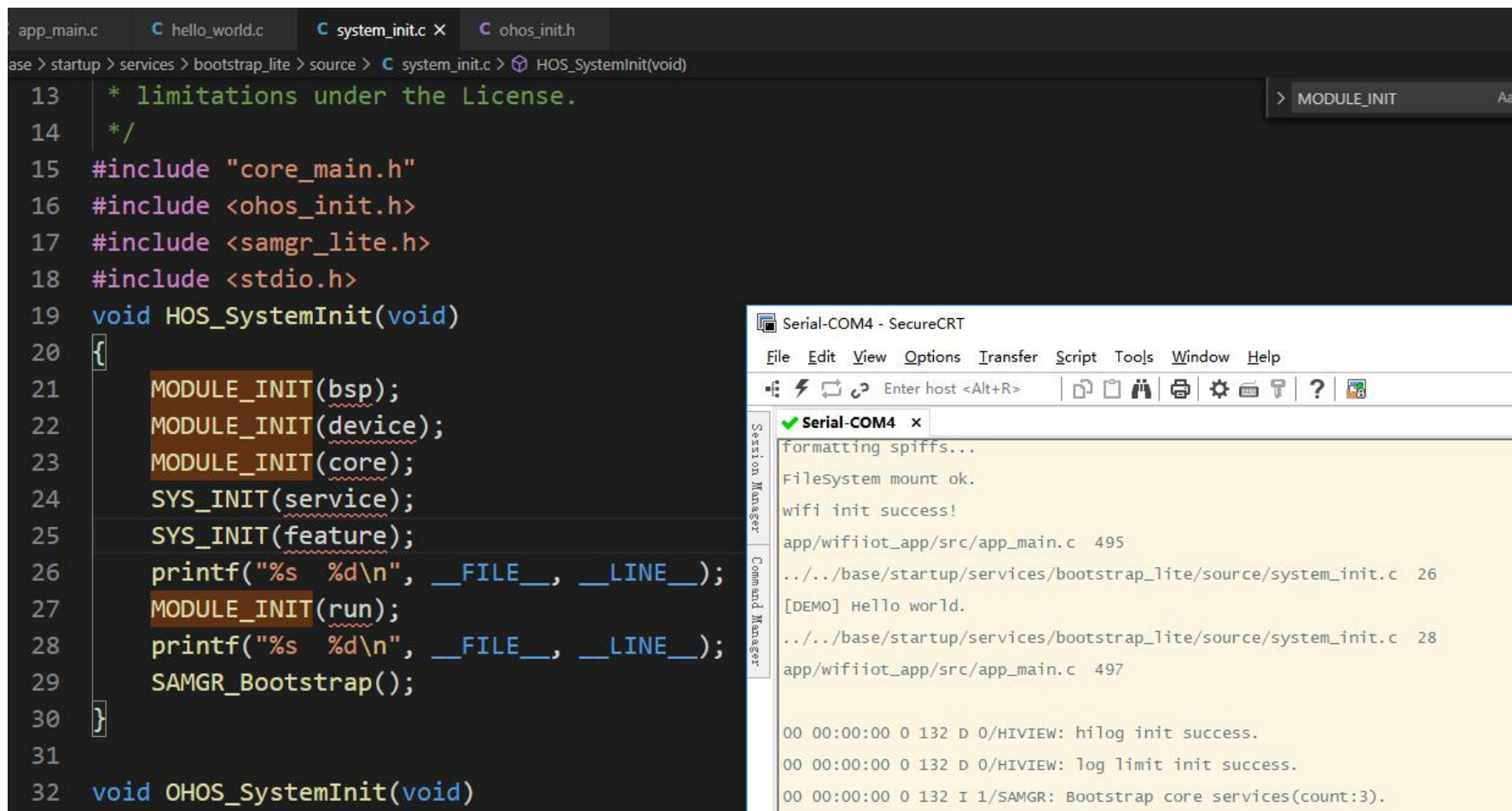
## 本章总结

本章所学内容有：

- 第1节：VS code中导入鸿蒙源码
- 第2节：编写第一个Hello world程序
- 第3节：完成HelloWorld程序编译、烧写、运行
- 第4节：启动流程分析

# 任务挑战

挑战任务：借助PPT和视频完成本章内容



The image shows a code editor window with a dark theme. The active file is `system_init.c`. The code contains several `MODULE_INIT` and `SYS_INIT` calls, along with `printf` statements for logging. The `HOS_SystemInit` function is defined, and the `OHOS_SystemInit` function is also visible. A search bar at the top right shows the term `MODULE_INIT`.

```
13  * limitations under the License.
14  */
15  #include "core_main.h"
16  #include <ohos_init.h>
17  #include <samgr_lite.h>
18  #include <stdio.h>
19  void HOS_SystemInit(void)
20  {
21      MODULE_INIT(bsp);
22      MODULE_INIT(device);
23      MODULE_INIT(core);
24      SYS_INIT(service);
25      SYS_INIT(feature);
26      printf("%s %d\n", __FILE__, __LINE__);
27      MODULE_INIT(run);
28      printf("%s %d\n", __FILE__, __LINE__);
29      SAMGR_Bootstrap();
30  }
31
32  void OHOS_SystemInit(void)
```

Below the code editor is a serial terminal window titled "Serial-COM4 - SecureCRT". It shows the output of the program, including file system mounting, WiFi initialization, and a "Hello world" message. The terminal also displays log messages from the system, such as "hilog init success" and "log limit init success".

```
formatting spiffs...
Filesystem mount ok.
wifi init success!
app/wifiiot_app/src/app_main.c 495
../../base/startup/services/bootstrap_lite/source/system_init.c 26
[DEMO] Hello world.
../../base/startup/services/bootstrap_lite/source/system_init.c 28
app/wifiiot_app/src/app_main.c 497

00 00:00:00 0 132 D 0/HIVIEW: hilog init success.
00 00:00:00 0 132 D 0/HIVIEW: log limit init success.
00 00:00:00 0 132 I 1/SAMGR: Bootstrap core services(count:3).
```

# THANKS

更多学习视频，关注宅客学院.....

