第3章

C语言鸿蒙OS设备程序开发方法和步骤

【本章学习目标】

（1）学会C语言鸿蒙OS设备开发编译环境的使用。

（2）熟练掌握C语言鸿蒙OS设备开发方法和开发步骤。

（3）掌握C语言鸿蒙OS设备开发程序项目的结构和组成内容。

本章以点亮一只LED灯的C语言设备开发程序案例，详细讲述利用本书第2章已经搭建好的C语言鸿蒙OS设备开发环境，开发设计C语言设备程序的方法和步骤。

3.1

3.1 C语言鸿蒙OS设备程序开发方法

如图3-1所示，C语言鸿蒙OS设备开发必须同时使用Windows开发环境与Linux编译环境，程序开发实验基于Hi3861芯片的开发板，大致开发过程分为几个步骤：代码编写，代码编译，镜像烧录，串口调试。程序员在Windows开发环境完成C语言程序源代码的编辑和修改，在Linux编译环境中将编辑好的C语言程序源代码编译成二进制的C语言程序可执行代码，然后再利用Windows开发环境中的烧录软件Hiburn，将Linux编译环境中编译好的C语言程序可执行代码，烧录到Hi3861开发板中，最后在Hi3861开发板中运行C语言程序可执行代码。

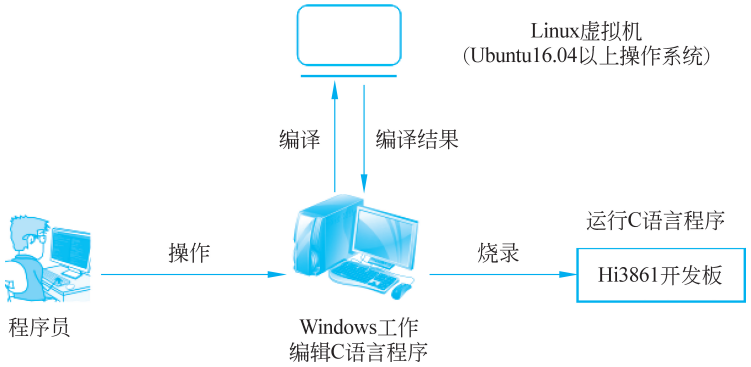

图3-1 C语言鸿蒙OS设备程序开发环境

其中，Linux 编译环境通过在 Windows 上搭建 Ubuntu 虚拟机的方式来实现。通过 samba 工具构建 Linux 本地虚拟机与 Windows 之间的共享文件夹，实现 Windows 环境与 Ubuntu 环境的资源共享。获取鸿蒙 OS 源码和支持开发板的源代码之后，将全部源码解压存放在共享文件夹 share 下（本书设定的共享文件夹是 share）。

开发方法：首先在共享文件夹下完成 C 语言程序源代码编辑；然后在 Ubuntu 虚拟机中完成 C 语言程序代码编译，编译生成的二进制可执行代码文件存放在源码目录下的 out 文件夹内，因为源码存放在共享文件夹下，所以编译生成的二进制可执行代码文件也存放在共享文件夹下；最后在 Windows 环境中获取编译生成的二进制可执行代码文件，进行镜像烧录与运行调试。

从图 3-1 可以看出，在程序的设备实验开发过程当中，软件的编译与执行不在同一设备上进行，编译在电脑端执行，编译产物在开发板上执行，这一过程称为交叉编译。在硬件设备的嵌入式开发当中，使用交叉编译的开发方式是极其必要的，这主要是因为嵌入式操作系统的硬件资源过少，无法完成源代码的编译工作，这就需要将编译的工作交由资源更多的设备进行，硬件设备只进行编译产物的执行工作。

3.2　C 语言鸿蒙 OS 设备程序项目结构和内容


3.2

通常情况下 C 语言程序不是由一个文件组成的，而是由一些与该程序密切相关的源程序代码文件，图形、图像、图标等资源文件，数据文件，配置文件，操作说明书等多个文件共同组成。所以准确地说，一个 C 语言程序通常被称为一个 C 语言程序项目或者 C 语言程序工程。当然，构成 C 语言程序项目的文件可视必要与否进行取舍。

本节以点亮一个 Hi3861 开发板上的 LED 灯的 C 语言程序项目为例，简要介绍一个 C 语言鸿蒙 OS 程序项目的结构和内容。

图 3-2 形象地展示了一个 C 语言鸿蒙 OS 程序项目的结构和内容，图中外带方框的 LED 和 SOURCE_LED 是文件夹，而且 SOURCE_LED 文件夹是 LED 文件夹下面的子文件夹。

图 3-2　点亮 LED 灯的 C 语言鸿蒙 OS 程序项目结构和内容示意图

图 3-2 表示将点亮一个 Hi3861 开发板上的 LED 灯的 C 语言程序项目放在 LED 主文件夹里面进行管理，在 LED 主文件夹里面存放了 BUILD.gn、config.json 两个文件和一个子文件夹 SOURCE_LED，在子文件夹 SOURCE_LED 里面存放了 BUILD.gn 和 LED.c 两个文件。其中：LED 主文件夹里面的 BUILD.gn 和 SOURCE_LED 子文件夹里面的 BUILD.gn 是两个与编译环境配置有关的文件，它们虽然名字相同，但是内容和作用却是不同的；LED 主文件夹里面的 config.json 也是一个编译环境配置文件；SOURCE_LED 子文件夹里面的 LED.c 是程序员编写的 C 语言程序源代码文件，它由实现程序功能所必需的

C 语言程序代码组成,这个文件的扩展名必须是.c(小写的 c)。

为便于组织和管理 C 语言鸿蒙 OS 程序项目,一般将 C 语言鸿蒙 OS 程序项目存放在特定的文件夹下,而且不同厂商的程序项目要存放在自己特定的文件夹下,这是因为一个 C 语言鸿蒙 OS 程序项目,既有程序员自己编写的 C 语言程序源代码文件 LED.c,也有程序员自己编写的项目配置文件 config.json 和 BUILD.gn,还有构成程序项目必不可少的鸿蒙 OS 操作系统文件和支持 C 语言鸿蒙 OS 开发的设备厂商提供的项目支撑文件。所有这些文件相互依赖,相互协作,共同构成 C 语言鸿蒙 OS 程序项目,因此放在特定的文件夹下,便于这种依赖关系的实施。

要想使用 C 语言编程来点亮一个 Hi3861 开发板上的 LED 灯,必须调用一些驱动开发板工作的 API(Application Programming Interface)函数,本例中用到的 API 函数如表 3-1 所示。

表 3-1 点亮一只 LED 灯项目用到的 API 函数一览表

API 函数	功 能 描 述
unsigned intIoTGpioInit(OUT_GPIO7)	初始化 GPIO 端口
IoTGpioSetDir(WifiIotGpioIdx id, WifiIotGpioDir dir);	设置 GPIO 引脚方向,id 参数用于指定引脚,dir 参数用于指定输入或输出
IoTGpioSetOutputVal (WifiIotGpioIdx id, WifiIotGpioValue val);	设置 GPIO 引脚的输出状态,id 参数用于指定引脚,val 参数用于指定高电平或低电平
IoTGpioSetFunc(WifiIotName id,unsigned char val);	设置 GPIO 引脚的功能,id 参数用于指定引脚,val 参数用于指定引脚功能
APP_FEATURE_INIT(LED);	用于开发板引导启动程序模块

点亮一个 Hi3861 开发板上的 LED 灯项目的各个文件的内容和作用分别介绍如下。

1. LED 主文件夹里面的 BUILD.gn 文件内容及作用

```
group("LED")
{
    deps = ["SOURCE_LED:LED",
            "//device/bossay/hi3861_l0/sdk_liteos:wifiiot_sdk",
             "../common/iot_wifi:iot_wifi",
    ]
}
```

(1) C 语言鸿蒙 OS 设备程序的编译结果可以是静态库(static_library)、动态库(dynamic_library)、可执行文件或者 group(组件),组件是鸿蒙 OS 系统最小的可复用、可配置、可裁剪的功能单元。组件具备目录独立、可并行开发、可独立编译、可独立测试的特征。

(2) LED 主文件夹里面的 BUILD.gn 文件是一个组件编译脚本文件,该文件的作用就是设置组件编译的配置,也就是在编译 C 语言鸿蒙 OS 设备程序时,要根据这个文件的内容来编译和生成编译目标结果。该文件中的 group("LED")用以设置编译组件的目标名称“LED”,就是组件的名称为“LED”,它和 C 语言程序的项目名称 LED 保持了一致,遵循了

编译目标名称和组件一致的原则。deps＝["SOURCE_LED:LED","//device/bossay/hi3861_l0/sdk_liteos:wifiiot_sdk","../common/iot_wifi:iot_wifi"]语句中：deps 是英文单词 depends 的缩写，汉语的意思是“依赖”，因此该句的含义为，编译构建 LED 组件，需要依靠 LED 文件夹下面 SOURCE_LED 文件夹下的内容，以及保存在/device/bossay/hi3861_l0/sdk_liteos:wifiiot_sdk 和 ../common/iot_wifi:iot_wifi 中的鸿蒙 OS 操作系统中有关无线网络的开发包中的内容。

2. SOURCE_LED 文件夹里面的 BUILD.gn 文件内容及作用

```
static_library("LED")
{
    sources = [ "LED.c", ]
    include_dirs = [
        "//utils/native/lite/include",
        "//base/iot_hardware/peripheral/interfaces/kits",
        "//device/bossay/hi3861_l0/iot_hardware_hals/include",
        "//device/bossay/hi3861_l0/sdk_liteos/include"
            ]
}
```

（1）SOURCE_LED 主文件夹里面的 BUILD.gn 文件也是一个组件编译脚本文件，该文件的作用是设置程序 LED.c 编译的配置，也就是在编译 C 语言程序 LED.c 时，要根据这个文件的内容来编译和生成目标结果。static_library("LED") 表明是要将 LED.c 编译成静态库。

（2）sources ＝ ["LED.c",]表明编译生成静态库的源代码文件来源于 LED.c。

（3）include_dirs ＝ [****]用于设置 LED.c 中 # include 包含语句的头文件的存储路径。

3. SOURCE_LED 主文件夹里面的 LED.c 文件内容及作用

```
#include <stdio.h>
#include "ohos_init.h"
#include "iot_gpio_ex.h"
#include "iot_gpio.h"
#define OUT_GPIO13 13
static void LED(void)                    //定义静态函数 LED
{
  IoTGpioInit(OUT_GPIO13);               //初始化 Hi3861 芯片 GPIO13 个引脚(GPIO 端口)
  //设置 Hi3861 芯片 GPIO13 引脚的功能
  IoTGpioSetFunc(OUT_GPIO13,IOT_GPIO_FUNC_GPIO_13_GPIO);
  IoTGpioSetDir(OUT_GPIO13,IOT_GPIO_DIR_OUT);     //设置写 Hi3861 芯片 GPIO13 引脚
  IoTGpioSetOutputVal(OUT_GPIO13, 1);      //设置写高电平到 Hi3861 芯片 GPIO13 引脚
}
APP_FEATURE_INIT(LED);                   //初始化且调用执行 LED 组件程序
```

Hi3861 芯片的 GPIO13 引脚连接着实验板上的一个 LED 灯（发光二极管），实际上实验板上有排列成五角形的 5 个 LED 灯，分别对应连接 Hi3861 芯片的 GPIO9、GPIO10、GPIO11、GPIO12、GPIO13 引脚。将上述代码中的 13 分别全部改成 9、10、11、12，然后编译生成可执行代码，写入实验板看看都有哪个灯被点亮。

4. LED 主文件夹里面的 config.json 文件内容及作用

```
{
  "product_name": "LED",
  "ohos_version": "鸿蒙 OS 3.0",
  "device_company": "bossay",
  "board": "hi3861_l0",
  "kernel_type": "liteos_m",
  "kernel_version": "",
  "subsystems":
    [
    {
      "subsystem": "iot_hardware",
      "components":
      [
        { "component": "iot_controller", "features":[] }
      ]
    },
    {
      "subsystem": "distributed_schedule",
      "components":
      [
         { "component": "samgr_lite", "features":[] }
      ]
    },
    {
      "subsystem": "security",
      "components":
      [
        { "component": "hichainsdk", "features":[] },
        { "component": "deviceauth_lite", "features":[] },
        { "component": "huks", "features":
        [
           "huks_config_file = \"hks_config_lite.h\"",
          "huks_mbedtls_path = \"//device/bossay/hi3861_l0/sdk_liteos/third_
          party/mbedtls/include/\""
      ]
    }
    ]
  },
  {
    "subsystem": "startup",
    "components":
    [
      { "component": "bootstrap_lite", "features":[] },
        { "component": "syspara_lite", "features":
          [
            "enable_ohos_startup_syspara_lite_use_thirdparty_mbedtls = false"
          ]
        }
```

```
        ]
      },

    {
      "subsystem": "utils",
      "components":
    [
      { "component": "file", "features":[] },
      { "component": "kv_store", "features":[] },
      { "component": "os_dump", "features":[] }
    ]
      }
  ],
    "third_party_dir": "//device/bossay/hi3861_l0/sdk_liteos/third_party",
    "product_adapter_dir": "//vendor/bossay/hi3861_l0/hals"
  }
```

LED 主文件夹下的 config.json 文件是一个组件编译配置文件，该文件的作用是设置在编译生成 LED 产品组件时，需要用到的鸿蒙 OS 操作系统源码和鸿蒙智联创新开发板的支持源码。因为鸿蒙 OS 操作系统是可裁剪的，在编译生成 LED 产品组件时，通过 config 文件配置那些必要的鸿蒙 OS 组件即可，例如，在该 config.json 中就配置了必须的鸿蒙 OS 的硬件子系统的"iot_controller"组件、分布式数据管理子系统的"samgr_lite 组件、安全子系统的部分组件和启动子系统的部分组件，以及鸿蒙智联创新开发板的部分支持代码。

3.3　点亮一只 LED 灯的 C 语言设备程序开发步骤

3.3

本节以点亮一只 LED 灯的 C 语言鸿蒙 OS 设备程序开发为例，详细讲述 C 语言鸿蒙 OS 设备程序开发的步骤。

1. 准备好 C 语言鸿蒙 OS 设备程序编译环境

说明：在开发 C 语言鸿蒙 OS 程序之前，要先准备好 C 语言鸿蒙 OS 程序编译环境，也就是要先在 Windows 工作台上运行虚拟机管理程序 VMware Workstation Pro，然后再在虚拟机管理程序中启动 Ubuntu 版 Linux 虚拟机 BossayUbuntu。

如果虚拟机管理程序 VMware Workstation Pro 和 Linux 虚拟机 BossayUbuntu 已经运行，则省略此步。否则如图 3-3 所示，要运行虚拟机管理程序 VMware Workstation Pro 并启动运行 Linux 虚拟机 BossayUbuntu，为 C 语言鸿蒙 OS 设备开发准备好编译环境。

2. 打开并运行 VS Code，通过 SSH 建立与虚拟机的连接

说明：VS Code 程序是 C 语言鸿蒙 OS 程序的编辑软件，C 语言鸿蒙 OS 程序项目的所有文件都是使用 VS Code 程序来编辑的。

如果 VS Code 程序已运行，则略过此步。否则在 Windows 工作台操作系统的桌面上

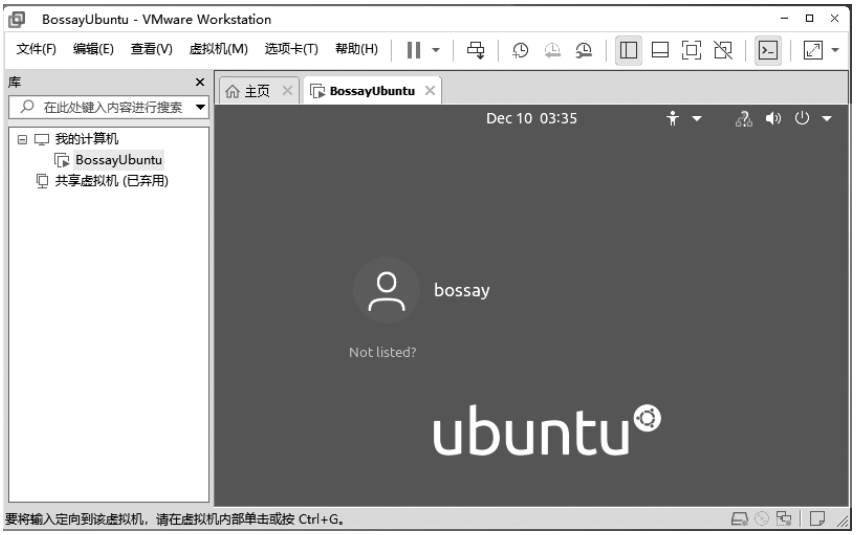

图 3-3 C 语言鸿蒙 OS 设备开发虚拟机

找到如图 3-4 所示 VS Code 程序的快捷图标，双击它运行该程序，如果一切正常会出现 VS Code 程序运行主窗口如图 3-5 所示。如果在启动 VS Code 程序的过程中出现如图 3-6 所示的“SSH 连接异常”窗口，意味着 SSH 没有建立与 IP 地址为 192.168.249.128 的虚拟机的连接，这有可能是由于虚拟机刚启动、其 SSH 服务还没有启动的情况下出现的问题，这时单击图 3-6 所示的窗口中的 Retry 按钮，尝试再次进行 SSH 连接，如果连接成功，则出现如图 3-5 所示的窗口，VS Code 程序完成正常启动。如果尝试 Retry 几次问题还得不到解决，极有可能是由于 SSH 连接设置存在问题，需要重新检查并解决 SSH 连接设置存在的问题。

图 3-4 VS Code 程序的快捷图标

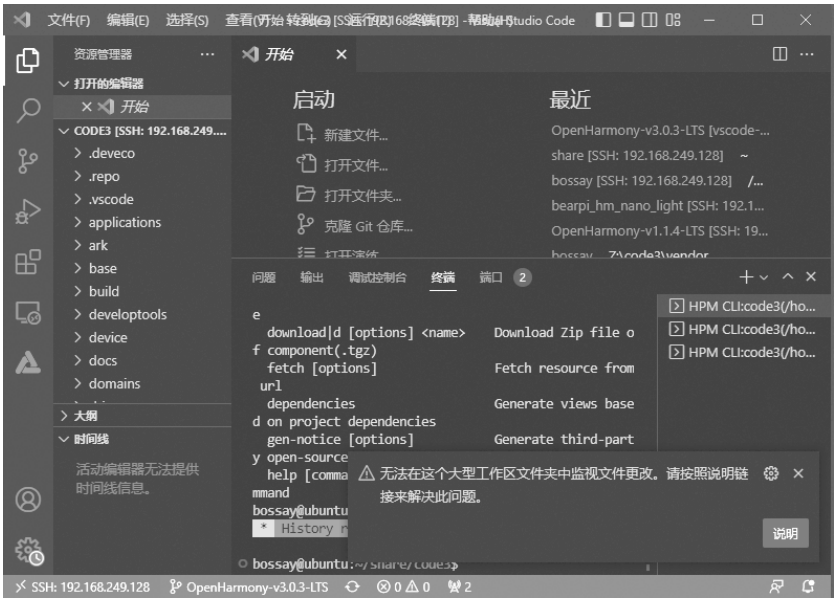

图 3-5 建立了 SSH 连接的 VS Code 程序的主窗口

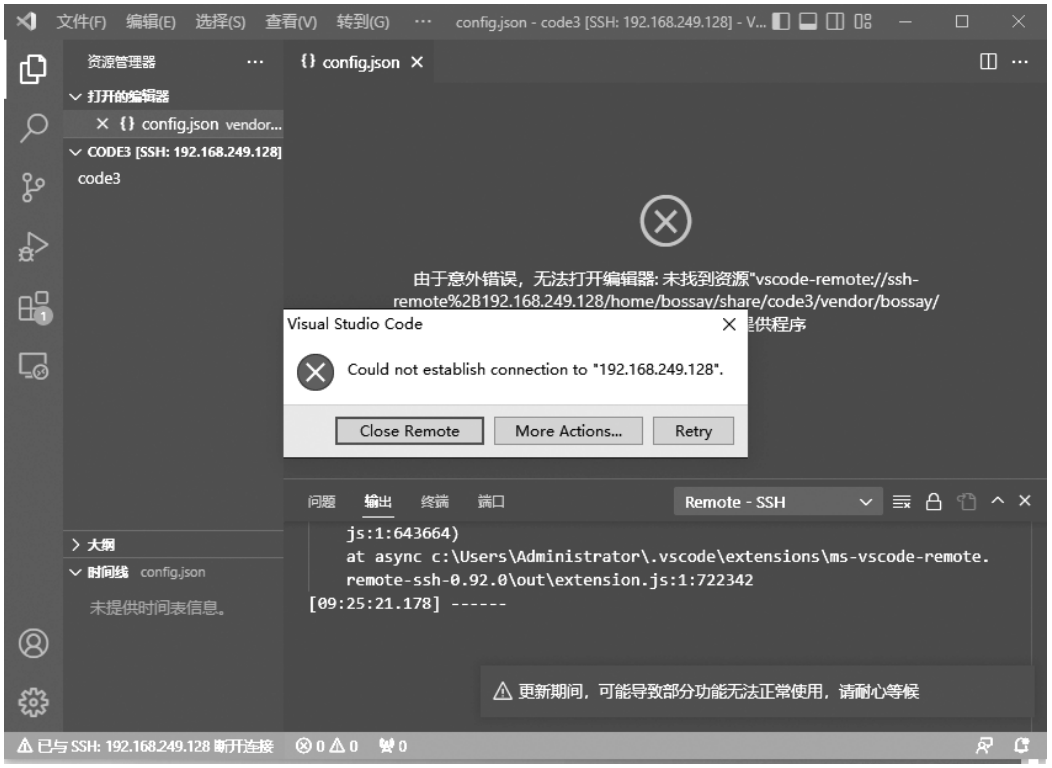

图 3-6　SSH 连接异常

3. 在 VS Code 程序中打开资源管理工具，使用 DevEco 添加新产品

在如图 3-7 所示的窗口中，单击窗口左侧列表中最上方的资源管理器图标打开资源管理器工具，然后单击资源管理器下方的 CODE3 项目，找到并且单击打开 vendor，然后右击 vendor 下面的 bossay，这时会在右侧弹出菜单，接着单击弹出菜单最下方的“[DevEco]开源鸿蒙”，在其右侧会弹出子菜单“添加新产品”，单击“添加新产品”，弹出“产品创建向导”如图 3-8 所示。

4. 使用 DevEco 产品创建向导，设定产品基础信息

在如图 3-8 所示的窗口中，用键盘输入的方式在“供应商名称”下方的文本输入框中输入“bossay”，在“产品名称”下方的文本输入框中输入点亮一只 LED 灯的项目名称“LED”，单击“开发板名称”下方列表框，在弹出的列表中选择“hi3861_IO”，产品名称保持“无”，设定好的新产品的基础信息如图 3-9 所示，设定好产品基础信息后，单击窗口下方的“确定”键，回到如图 3-10 所示的窗口。要注意，刚回到该窗口时如果看不到新建的“LED”新产品项目，这时必须单击“刷新资源管理器”图标后才能在 vendor 下面的 bossay 中见到新建的产品项目“LED”。注意以后每次新建项目、文件夹、文件后，如果在 bossay 文件夹下看不到它们都需要单击一下刷新图标，才能在列表中看到它。

5. 检查一下新创建的 LED 项目的基础内容

在如图 3-10 所示的窗口中，单击“LED”项目，在其下方列出新建“LED”项目的两个文件 BUILD.gn 和 config.json 如图 3-11 所示。单击 BUILD.gn 文件，可以在右侧文本编辑器中编辑 BUILD.gn 文件内容，如图 3-12 所示；单击 config.json 文件，就可以在右侧文本编辑

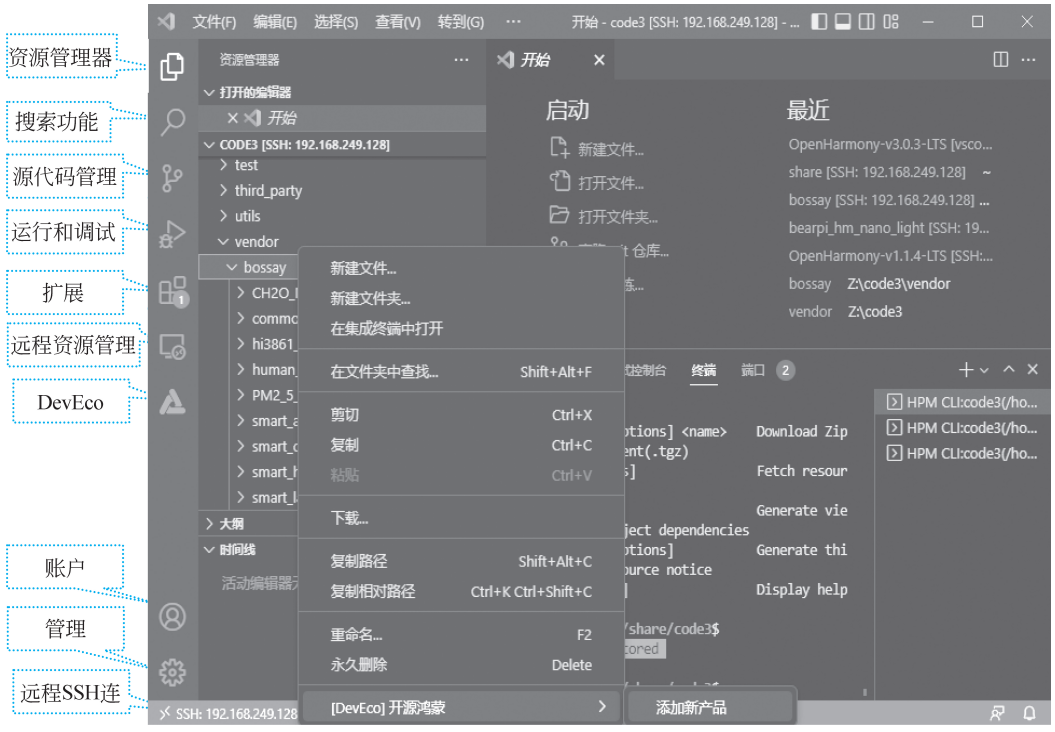

图 3-7　VS Code 程序运行的主窗口

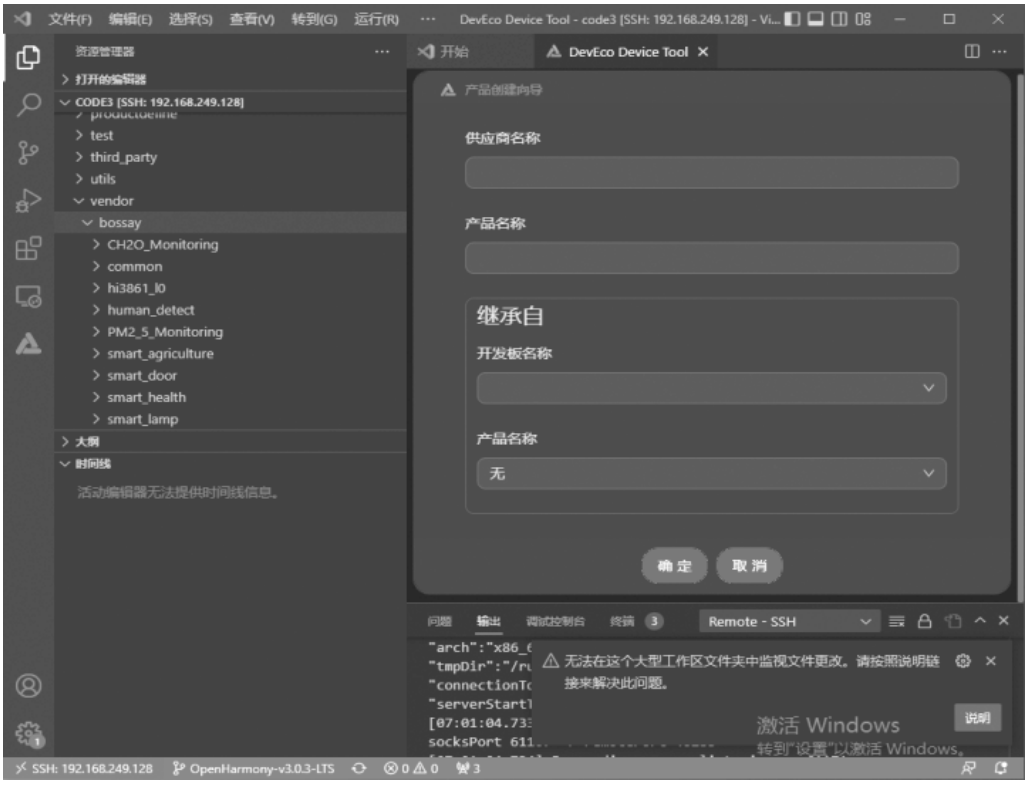

图 3-8　产品创建向导

图 3-9　产品基础信息

图 3-10　新建 LED 项目

图 3-11　新建 LED 项目的基础内容

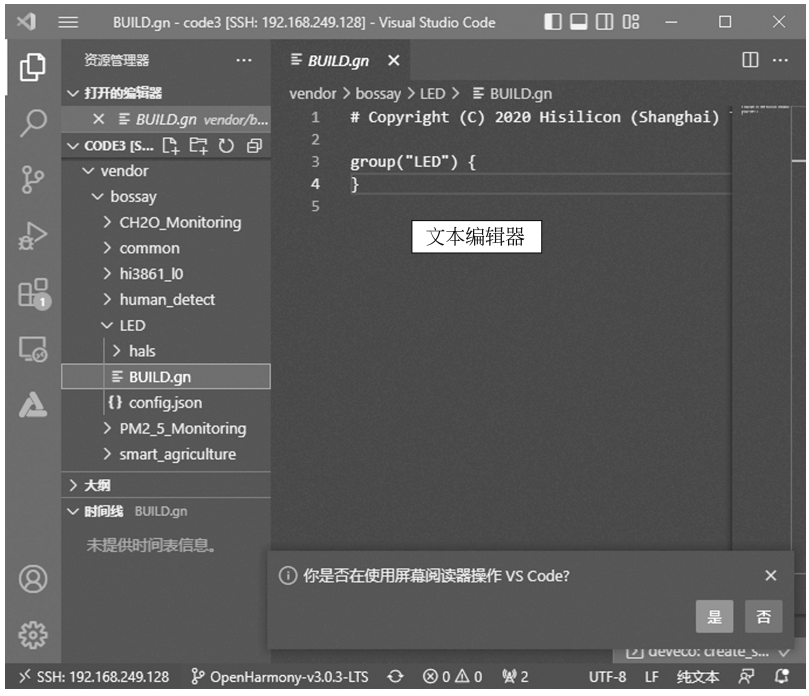

图 3-12　编辑器中编辑文件 BUILD.gn

器中编辑 config.json 文件内容，如图 3-13 所示。如此操作，想编辑任何一个文件，只要单击文件名，该文件就会显示在编辑器中任由编辑了。

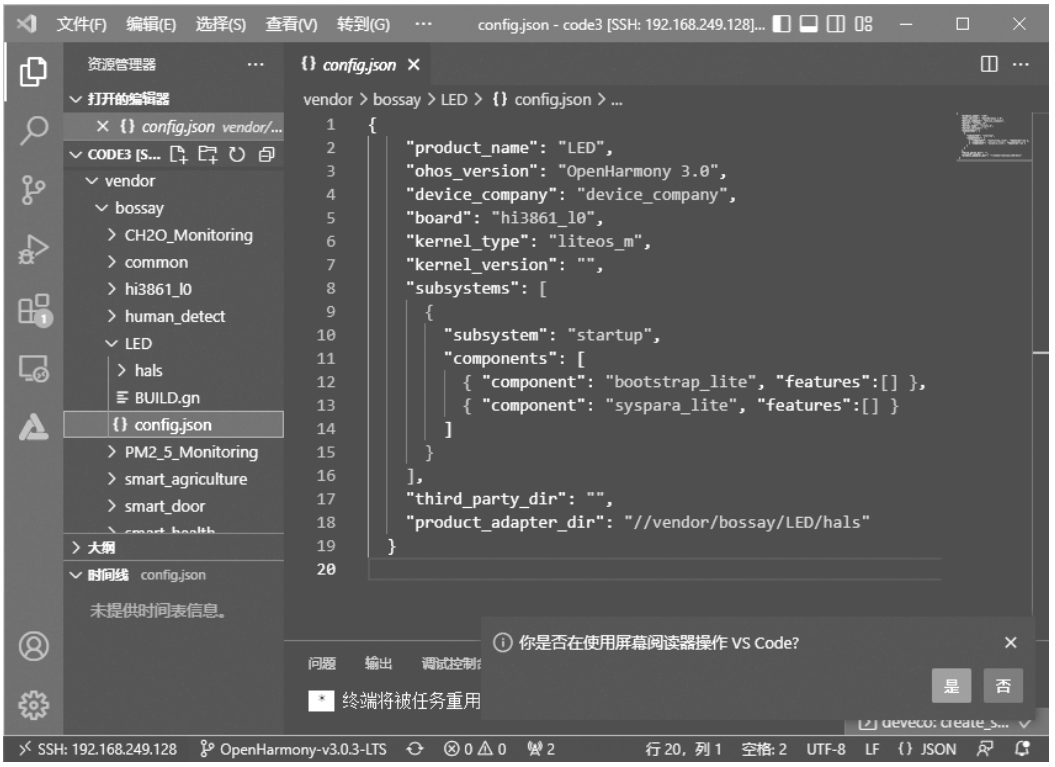

图 3-13　编辑器中编辑文件 config.json

6. 编辑保存 LED 项目根目录下的 BUILD.gn 文件

说明：这一步是程序员编辑 LED 项目根目录下的 BUILD.gn 文件，用键盘输入文件的内容，然后保存该文件。

本步操作由以下 3 步完成。

1）在文本编辑器中打开 LED 文件夹下的 BUILD.gn 文件

在如图 3-11 所示的窗口中，单击打开窗口左侧列表中文件夹 LED，然后单击 LED 文件夹下面的文件 BUILD.gn，将其在编辑器中打开如图 3-12 所示。

2）使用文本编辑器编辑 LED 文件夹下的 BUILD.gn 文件

如图 3-14 所示，在确保内容、格式严格符合 C 语言程序规范的前提下，采用键盘输入的方式，将本书 3.2 节 LED 主文件夹下的 BUILD.gn 代码输入，完成 BUILD.gn 文件内容的编辑。

3）保存 LED 文件夹下的 BUILD.gn 文件

如图 3-15 所示，编辑完文件 BUILD.gn 的内容后，单击“文件”菜单，然后在弹出的子菜单中单击“保存”，将 BUILD.gn 的内容保存到磁盘上。凡是输入和修改后的文件内容，都需要照此方法进行及时的保存。

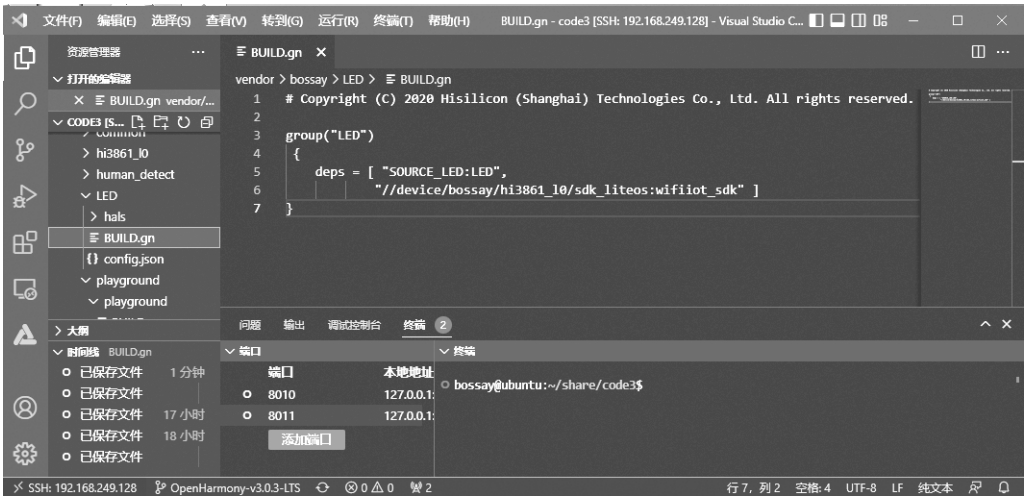

图 3-14 编辑器中编辑 LED 文件夹的 BUILD.gn

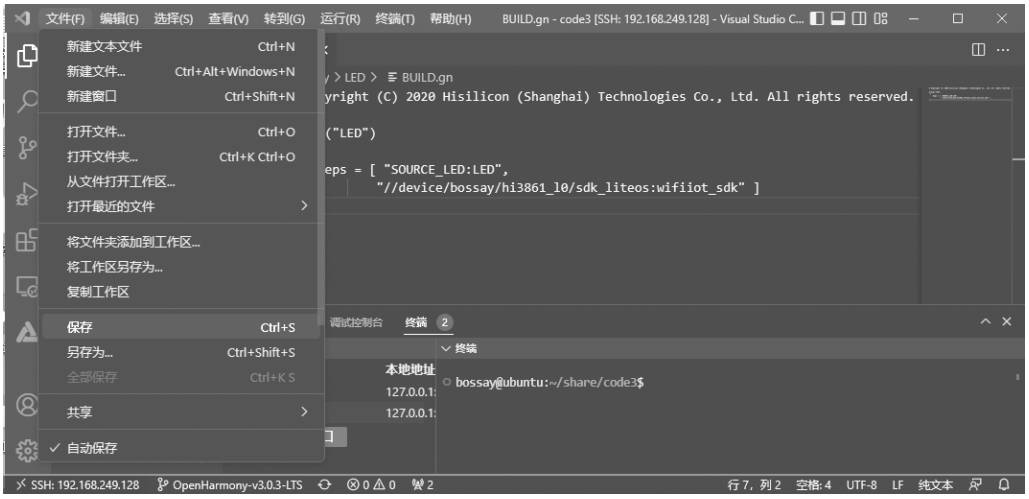

图 3-15 保存文件 BUILD.gn 的内容

7. 编辑保存 LED 项目根目录下的 config.json 文件

本步操作由以下 3 步完成。

1) 在文本编辑器中打开 LED 文件夹下的 config.json 文件

在如图 3-11 所示的窗口中，单击打开窗口左侧列表中文件夹 LED，然后单击 LED 文件夹下面的文件 config.json，将其在编辑器中打开，如图 3-13 所示。

2) 使用文本编辑器编辑 LED 文件夹下的 config.json 文件

编辑 config.json 文件内容有以下两种方法。

(1) 使用键盘输入的方式编辑 config.json 文件。

如图 3-13 所示，在确保内容、格式严格符合 C 语言程序规范的前提下，采用键盘输入的方式，修改 config.json 文件，将本书 3.2 节 LED 主文件夹下的 config.json 代码输入，完成 config.json 文件内容的编辑。

（2）使用复制和粘贴的方式编辑 config.json 文件。

因为 config.json 文件的内容不但有点多，而且文件内容对于初学 C 语言的人来说也难以读懂。用键盘输入编辑修改整个文件的内容，对于初学者来说有点困难，有时候还难免出现错误，所以，初学者可以用复制粘贴的方式来输入 config.json 文件的内容。与本书配套的网站提供了本书中所有的程序项目内容，每一个案例程序的文件都可以从网上下载，每个案例项目中都有一个 config.json 文件。初学者可以采用复制粘贴的方式来输入编辑 config.json 文件的内容。

3）保存 LED 文件夹下的 config.json 文件

编辑完文件 config.json 的内容后，单击 VS Code 的“文件”菜单，然后在弹出的子菜单中单击“保存”，将 config.json 的内容保存到磁盘上。

8. 在 LED 项目的文件夹 LED 下新建子文件夹 SOURCE_LED

说明：这一步是程序员在 LED 文件夹下创建一个子文件夹 SOURCE_LED，在子文件夹 SOURCE_LED 下存放程序项目的源代码文件和编译配置文件，文件夹的名字由程序员自己决定，但这个文件夹要创建在项目文件夹 LED 下。

在 LED 文件夹下新建 SOURCE_LED 文件夹有以下 2 种方法。

（1）在如图 3-16 窗口中，右击 LED 文件夹弹出菜单，然后单击菜单中的“新建文件夹”，在弹出的文本输入框中输入“SOURCE_LED”，输入完成后按回车键就在 LED 文件夹下建立了 SOURCE_LED 文件夹，如图 3-18 所示。

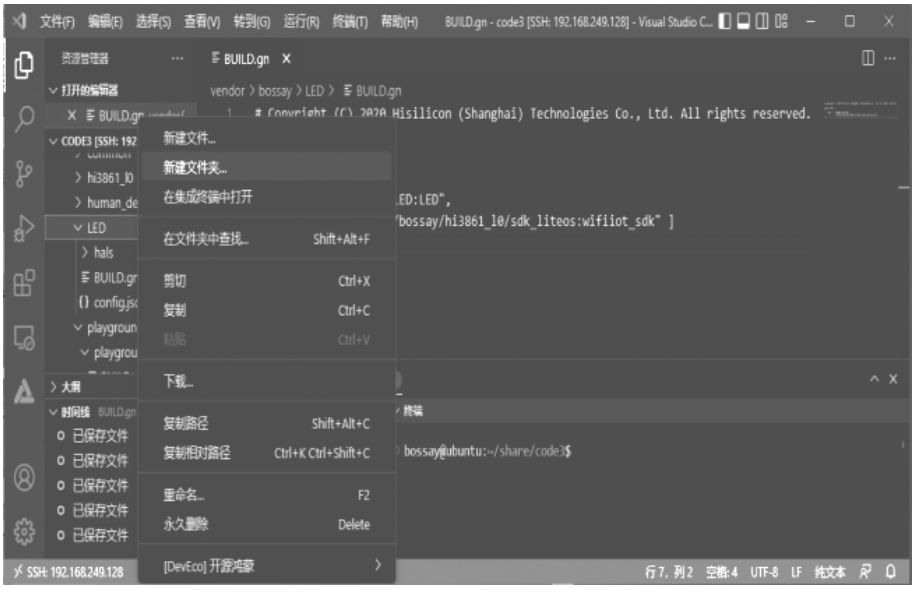

图 3-16　使用菜单在文件夹 LED 下创建子文件夹

（2）如图 3-17 所示，首先单击打开 LED 文件夹，然后单击“新建文件夹”按钮，接着在弹出的文件夹名称输入框中输入“SOURCE_LED”，输入完成后按回车键就在 LED 文件夹下建立了 SOURCE_LED 子文件夹，如图 3-18 所示。

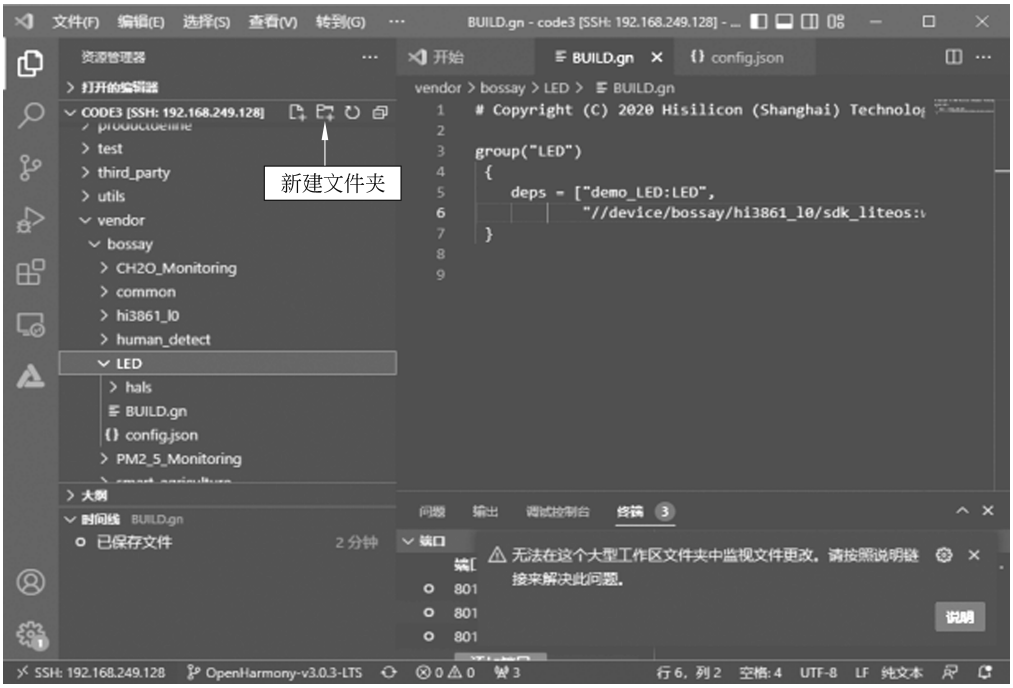

图 3-17　使用图标按钮在 LED 文件夹下新建文件夹

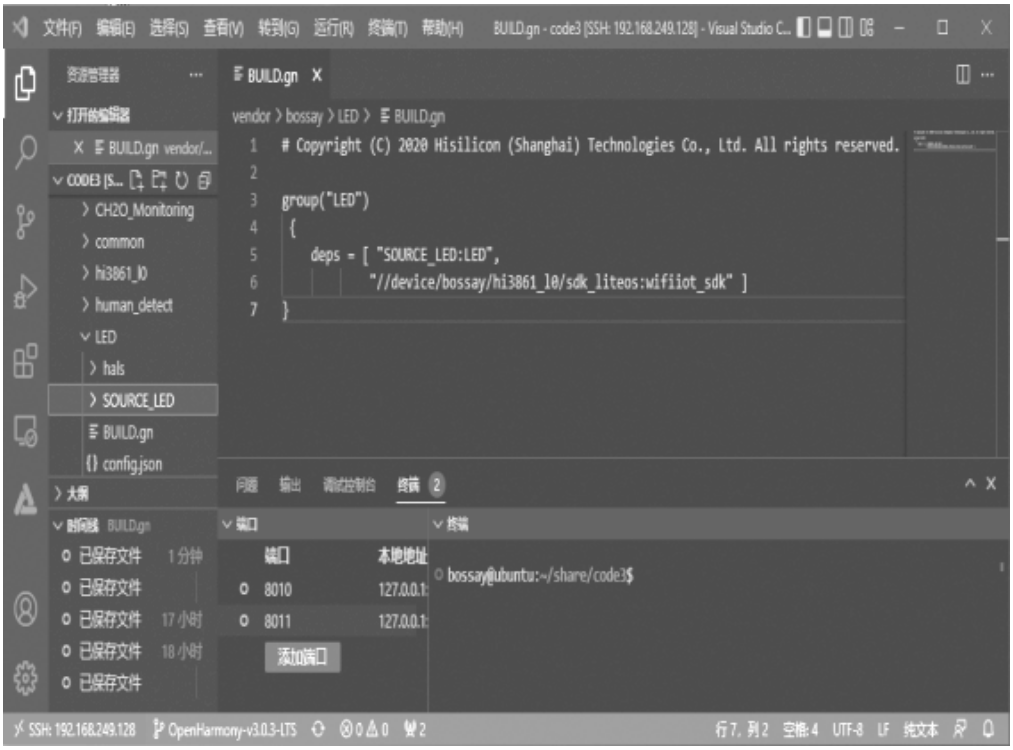

图 3-18　在 LED 文件夹下创建 SOURCE_LED 文件夹

9. 在 LED 项目的子文件夹 SOURCE_LED 下，新建编译配置文件 BUILD.gn

说明：这一步是程序员在 LED 文件夹下边的子文件夹 SOURCE_LED 中，创建程序项目的编译配置文件 BUILD.gn，包括：创建这个文件的文件名、利用键盘输入文件的内容以及文件内容输入完成后保存这个文件。要注意的是，这个文件和前面第 6 步编辑的文件虽然名称相同，但是它们的内容不相同，作用也不一样，而且存放在不同的文件夹下。

本步操作由以下 3 步完成。

1）在 SOURCE_LED 文件夹下新建 BUILD.gn 文件

在 SOURCE_LED 文件夹下新建 BUILD.gn 文件有以下 2 种方法。

（1）在如图 3-19 所示的窗口中，右击 SOURCE_LED 文件夹会弹出菜单，然后单击弹出菜单中的“新建文件”，在弹出的文件名称输入框中输入“BUILD.gn”，输入完成后按回车键就在 SOURCE_LED 文件夹下建立了文件 BUILD.gn，如图 3-19 所示。

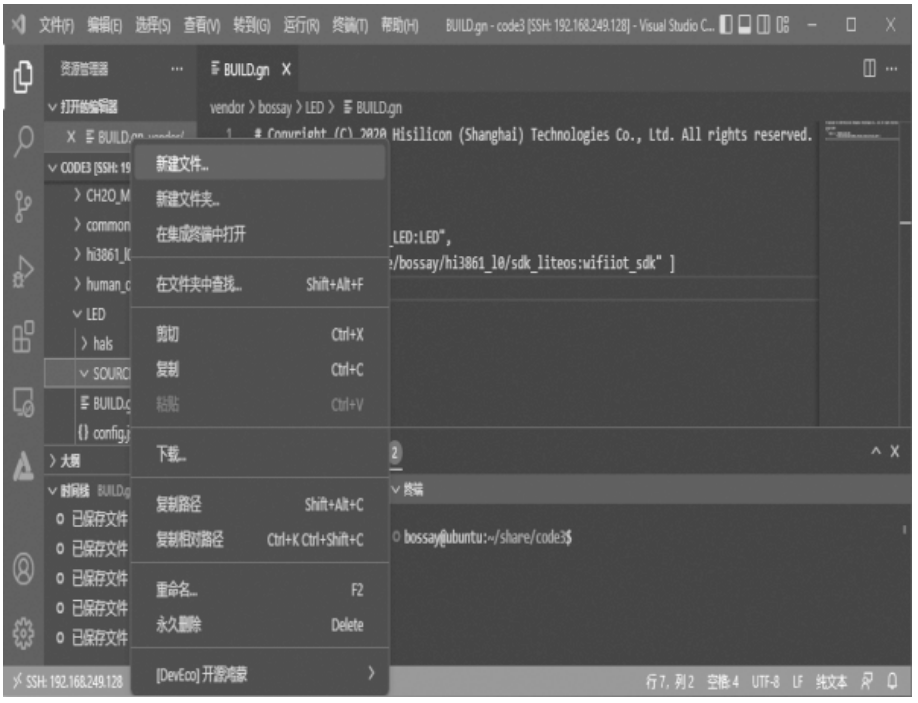

图 3-19　使用菜单在文件夹 SOURCE_LED 下新建文件

（2）如图 3-20 所示，首先单击打开 SOURCE_LED 文件夹，然后单击“新建文件”按钮，接着在弹出的文件名称输入框中输入“BUILD.gn”，输入完成后按回车键就在 SOURCE_LED 文件夹下建立了 BUILD.gn 文件，如图 3-21 所示。

2）编辑 SOURCE_LED 文件夹下新建的 BUILD.gn 文件

在如图 3-21 所示的窗口中，保持文件 BUILD.gn 处于打开编辑状态，然后用键盘在窗口右侧文本编辑器中输入本书 3.2 节中 SOURCE_LED 文件夹下的文件 BUILD.gn 的内容，如图 3-22 所示。需要说明的是，该文件内容的编辑，也可以用复制粘贴方式去完成。

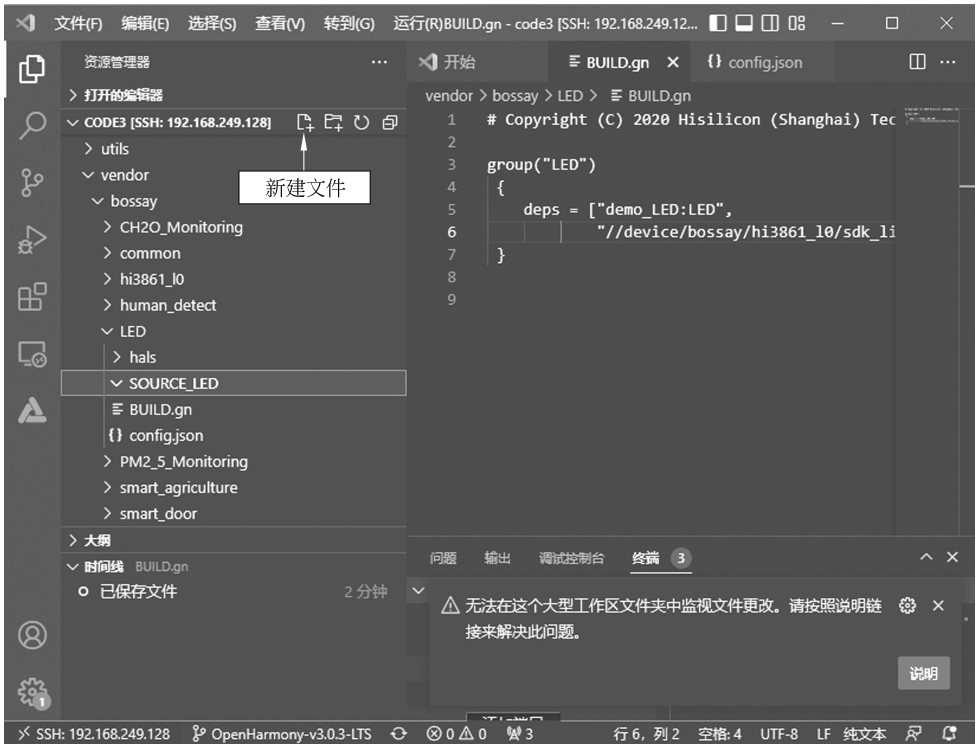

图 3-20　使用图标按钮在 SOURCE_LED 文件夹新建文件

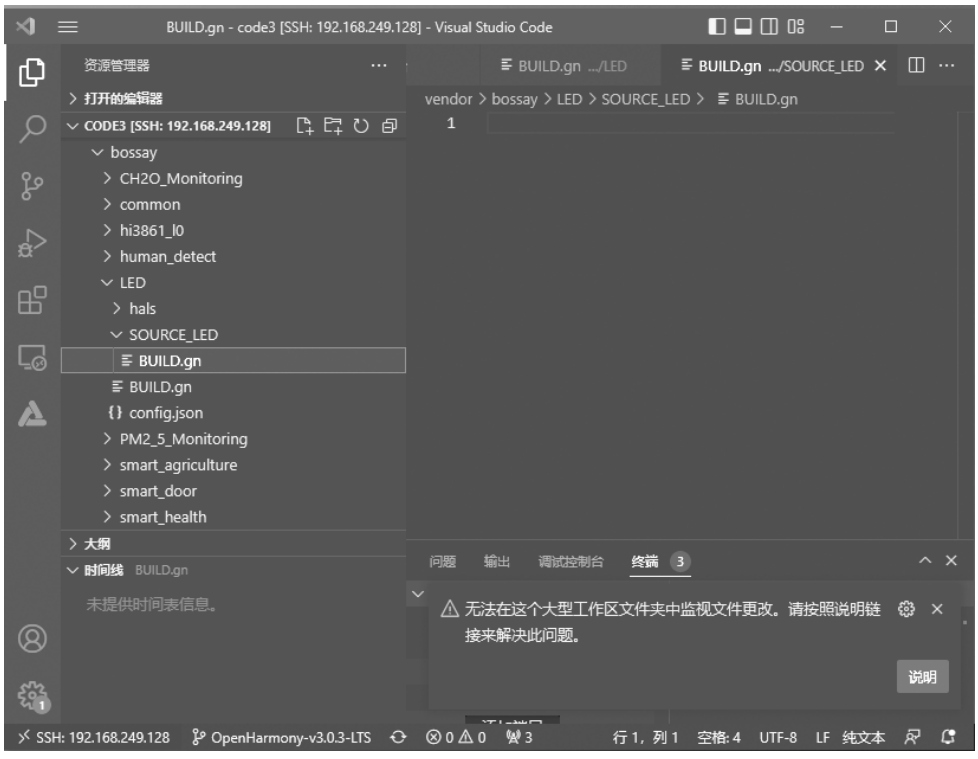

图 3-21　在 SOURCE_LED 文件夹新建文件 BUILD.gn

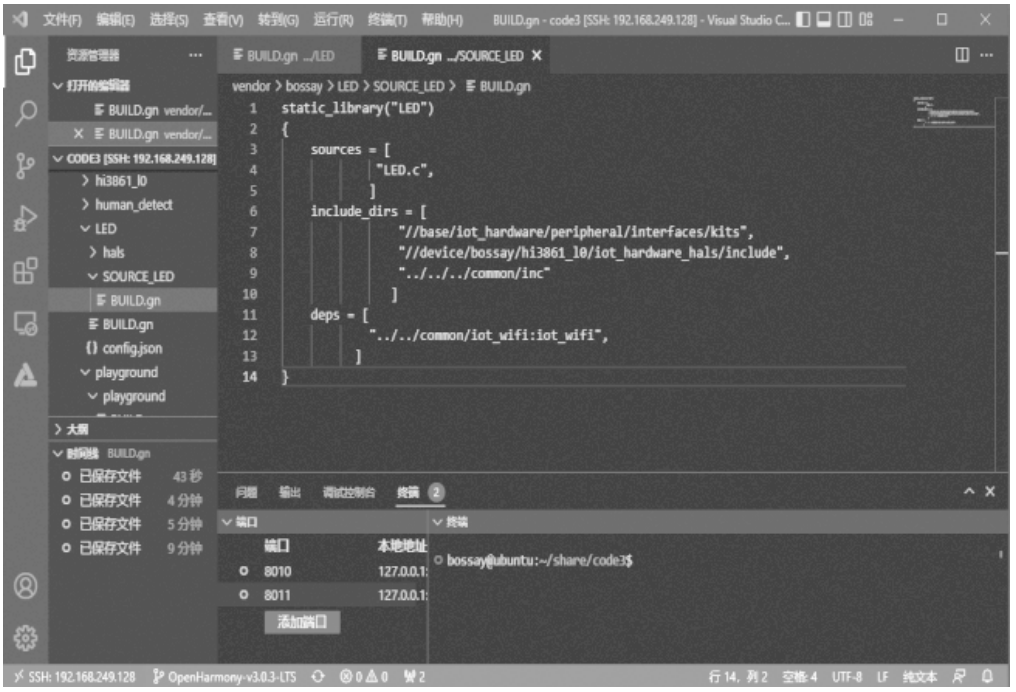

图 3-22　编辑文件 BUILD.gn 文件

3）保存 SOURCE_LED 文件夹下编辑好的 BUILD.gn 文件

如图 3-23 所示，编辑好文件 BUILD.gn 的内容后，要及时单击 VS Code 程序的“文件”菜单，然后在弹出的子菜单中单击“保存”，及时将 BUILD.gn 的内容保存到磁盘上。

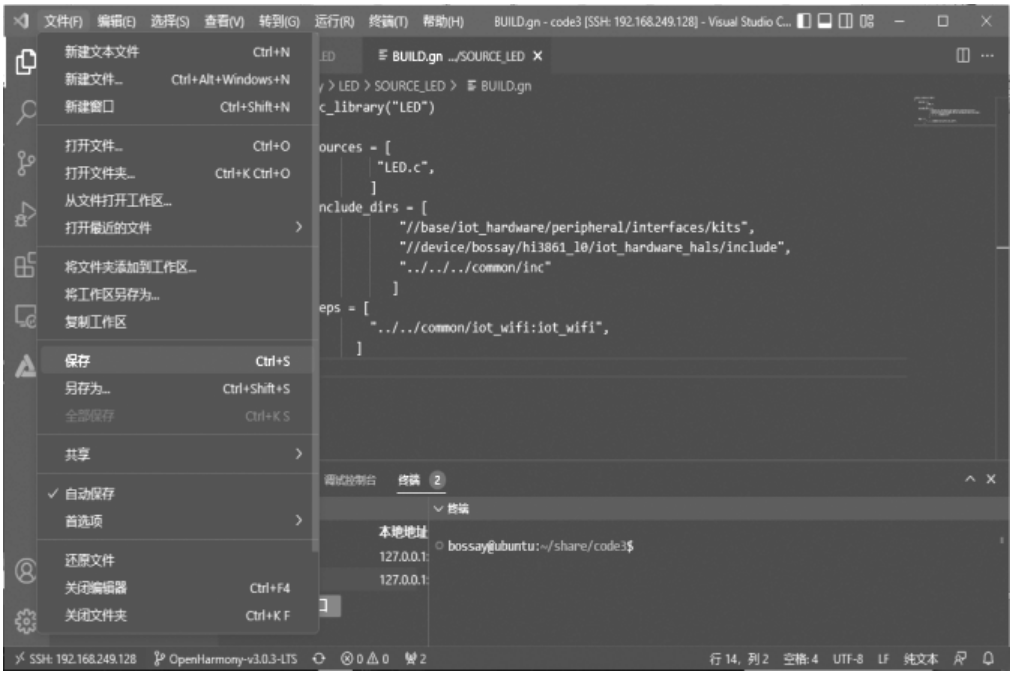

图 3-23　保存 SOURCE_LED 文件夹下的 BUILD.gn 文件

10. 在 LED 项目的子文件夹 SOURCE_LED 下新建 C 语言程序源代码文件 LED.c

说明：这一步是在 LED 文件夹下边的子文件夹 SOURCE_LED 里面，创建程序项目的 C 语言程序代码文件 LED.c，包括：创建这个文件的文件名、利用键盘输入文件的内容以及文件内容编辑输入完成后保存这个文件。需要说明的是，这个文件是由程序员根据程序的功能要求，自己设计编写的 C 语言程序文件。

本步操作由以下 3 步完成。

1）在 SOURCE_LED 文件夹下新建 LED.c 文件

在 SOURCE_LED 文件夹下新建 LED.c 文件有以下 2 种方法。

（1）在如图 3-24 所示的窗口中，右击 SOURCE_LED 文件夹弹出菜单，然后单击“新建文件”，在弹出的文件名称输入框中输入文件名称“LED.c”，输入完成后按回车键就在 SOURCE_LED 文件夹下新建了 LED.c 文件，如图 3-26 所示。

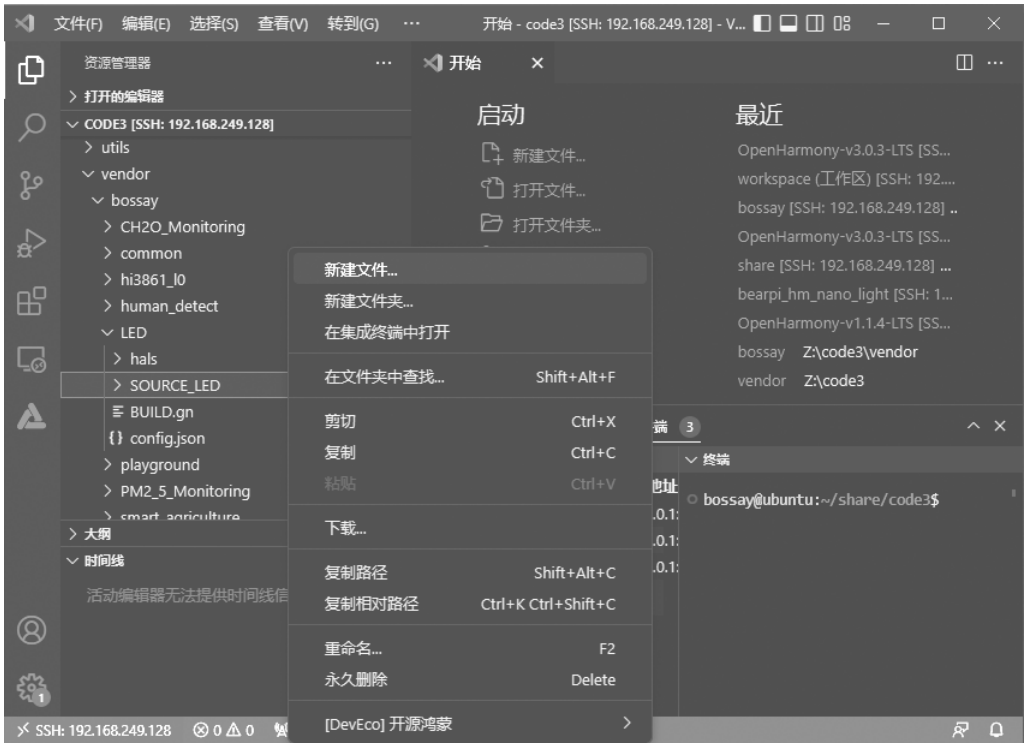

图 3-24　使用菜单在文件夹 SOURCE_LED 下新建文件

（2）如图 3-25 所示，首先单击打开 SOURCE_LED 文件夹，然后单击“新建文件”按钮，接着在弹出的文件名称输入框中输入“LED.c”，输入完成后按回车键就在 SOURCE_LED 文件夹下建立了 LED.c 文件，如图 3-26 所示。

一定要注意文件名的扩展名是小写的字符“c”。

2）编辑 LED.c 文件

在如图 3-26 所示的窗口中，保持文件 LED.c 处于打开编辑状态，然后用键盘在窗口右

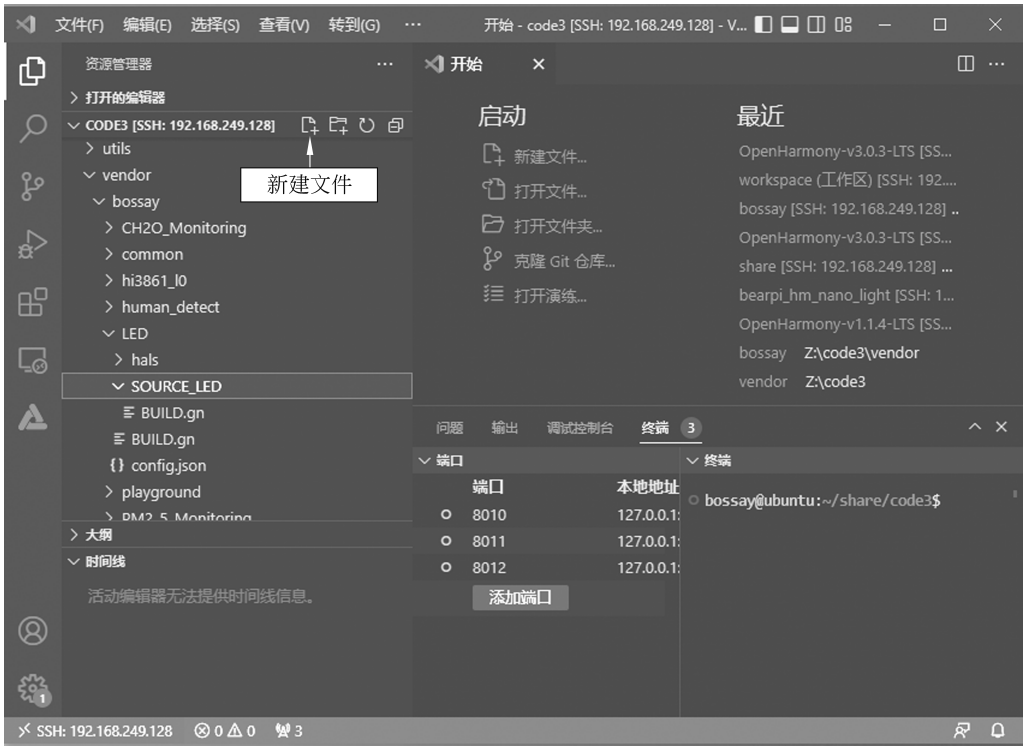

图 3-25　使用图标按钮在 SOURCE_LED 文件夹新建文件

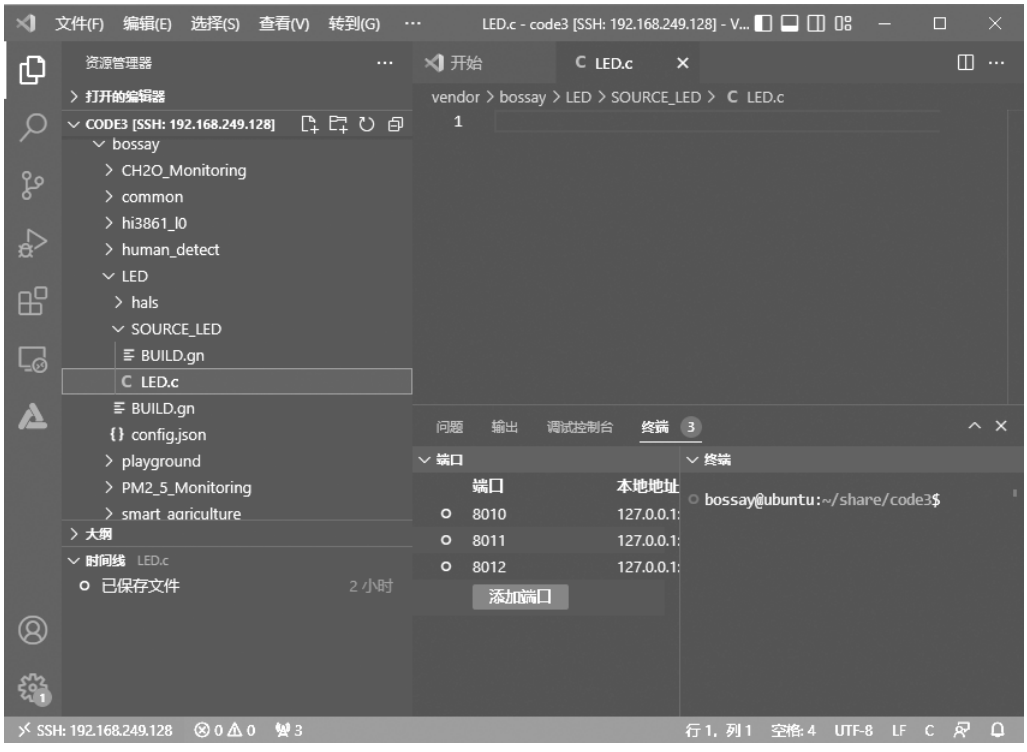

图 3-26　在 SOURCE_LED 文件夹新建文件 LED.c