

LiteOS Studio

使用指南

文档版本

01

发布日期

2019-03-15



版权所有 © 华为技术有限公司 2019。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<http://www.huawei.com>

客户服务邮箱：support@huawei.com

客户服务电话：4008302118

前言

概述

本文档详细的描述了LiteOS Studio的安装及使用方法。

读者对象

本文档主要适用于LiteOS Studio的使用人员。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	用于警示紧急的危险情形，若不避免，将会导致人员死亡或严重的人身伤害。
 警告	用于警示潜在的危险情形，若不避免，可能会导致人员死亡或严重的人身伤害。
 小心	用于警示潜在的危险情形，若不避免，可能导致中度或轻微的人身伤害。
 注意	用于传递设备或环境安全警示信息，若不避免，可能导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “注意”不涉及人身伤害。
 说明	用于突出重要/关键信息、最佳实践和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

修改记录

文档版本	发布日期	修改说明
01	2019-03-15	第一次正式发布。

目 录

前言.....	ii
1 简介.....	1
2 LiteOS Studio 支持的硬件.....	2
3 LiteOS 工程建立.....	3
3.1 新建工程.....	3
3.2 打开工程.....	6
3.3 导入工程.....	6
4 环境准备.....	8
5 快速上手.....	12
5.1 STM32F429IG 工程示例.....	12
5.2 华为海思 3516CV300 工程示例.....	18
5.2.1 环境准备.....	18
5.2.2 编译前注意事项.....	19
5.2.3 工程示例.....	19
5.3 STM32L431xx 工程示例.....	25
6 LiteOS Studio 使用.....	31
6.1 了解工程界面.....	31
6.1.1 界面.....	31
6.1.2 工具栏.....	32
6.2 设置 Studio.....	33
6.2.1 Studio 设置入口.....	33
6.2.2 SDK&依赖项.....	33
6.2.3 网络配置.....	34
6.2.4 版本更新.....	35
6.2.5 代码编辑.....	37
6.3 代码编辑.....	37
6.3.1 语法高亮.....	37
6.3.2 智能提示/自动补齐.....	38
6.3.3 源码符号扫描.....	39
6.3.4 符号定义跳转.....	39
6.3.5 符号引用跳转.....	40

6.3.6 符号列表查看.....	42
6.3.7 函数调用树查看.....	42
6.3.8 文本编码切换.....	43
6.3.9 格式自动对齐.....	44
6.3.10 视图单步前进/后退.....	45
6.3.11 全局搜索.....	46
6.3.12 当前文件代码查找/替换.....	47
6.3.13 文件搜索.....	48
6.4 断点控制.....	49
6.4.1 新增断点.....	49
6.4.2 新增条件断点.....	50
6.4.3 编辑断点.....	51
6.4.4 删除断点.....	52
6.4.5 禁用/启用断点.....	52
6.5 控制台输出.....	53
6.6 编译, 烧录及调试 (非华为海思芯片)	54
6.6.1 编译 (Windows)	54
6.6.2 烧录.....	55
6.6.3 调试.....	56
6.7 编译, 烧录及调试 (华为海思芯片)	57
6.7.1 环境准备.....	57
6.7.2 编译 (Linux)	57
6.7.2.1 编译前注意事项.....	57
6.7.2.2 在 Linux 系统下搭建开发环境.....	57
6.7.2.3 在 Linux 系统下进行编译.....	60
6.7.3 烧录.....	67
6.7.4 调试.....	70
6.8 调试中的各面板作用.....	71
6.8.1 反汇编.....	71
6.8.2 调试面板.....	72
6.8.2.1 监视.....	73
6.8.2.2 CPU 寄存器.....	74
6.8.2.3 地址空间.....	76
6.8.2.4 异常堆栈.....	78
6.8.3 断点信息.....	78
6.8.4 串口终端.....	79
7 附录.....	82
7.1 获取连接目标板的实际串口号.....	82
8 常见问题 (针对华为海思芯片目标板)	84
8.1 弹出 JLink 警告如何解决.....	84
8.2 华为海思芯片烧录/调试失败.....	84
8.3 烧录或调试时, JLink 弹出框未置顶显示.....	87

8.4 通过 LiteOS Studio 修改网驱动器中的文件注意事项.....	89
8.5 华为海思 3516CV300 开发板硬件如何设置.....	90

1 简介

LiteOS Studio是基于LiteOS嵌入式系统软件开发的工具，支持C、C++、汇编等多种开发语言，提供代码编辑、编译、烧录及调试等一站式开发体验。

2 LiteOS Studio 支持的硬件

- LiteOS Studio目前支持ARM Cortex-M0, Cortex-M4, Cortex-M7, Cortex-A7, ARM926EJ-S等芯片架构。
- LiteOS Studio目前已经适配了10种开发板，其中包括ST、HiSilicon、Fudan Microelectronics等主流厂商的开发板。
ST: STM32F429IG, STM32F411RE, STM32L431RB, STM32L431RC, STM32F746ZG
HiSilicon: Hi3516CV300, Hi3516CV500, Hi3516EV200, Hi3516EV300
Fudan Microelectronics: FM33A04xx
- LiteOS Studio支持新增MCU列表，以满足用户其他开发板的业务需求。

3 LiteOS 工程建立

背景信息

通过“创建LiteOS Studio工程”、“打开LiteOS Studio工程”或“导入其他嵌入式工程(gcc)”建立LiteOS工程。

“创建LiteOS Studio工程”支持创建STM32F429IG工程，空白工程，Hello word工程及快速上云STM32L431工程。

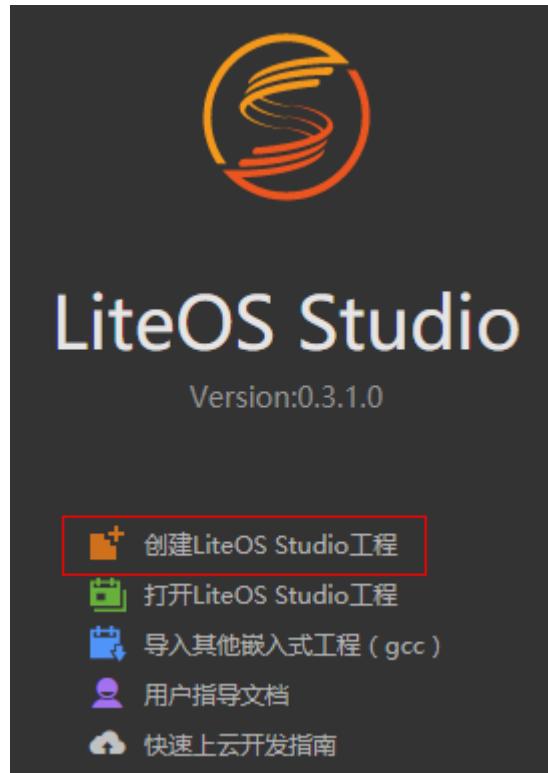
[3.1 新建工程](#)

[3.2 打开工程](#)

[3.3 导入工程](#)

3.1 新建工程

步骤1 单击“创建LiteOS Studio工程”。



步骤2 自定义“工程名称”及选择“工程目录”。

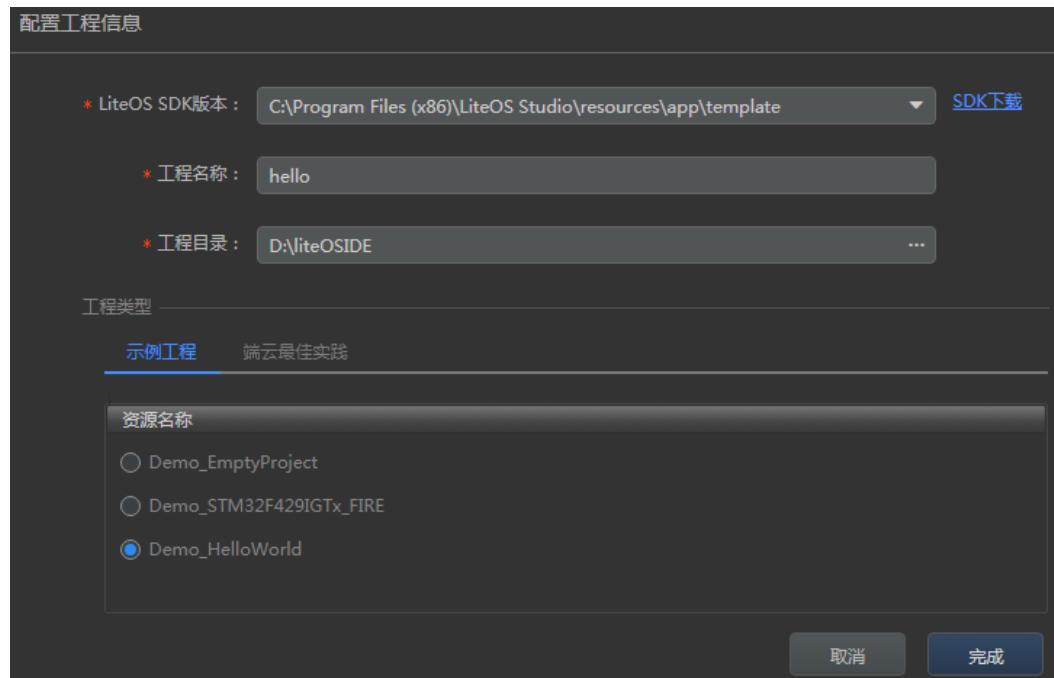
说明

- 内置SDK，自动配置“LiteOS SDK版本”，如需更改路径，单击“SDK下载”，自定义SDK保存路径。
- 工程目录中不能使用标点符号。

步骤3 在“示例工程”界面下，选择需要创建的工程模板。

说明

“端云最佳实践”界面提供STM32L431工程模板，可进行智慧路灯开发体验，具体操作请参见**OC快速上云**。



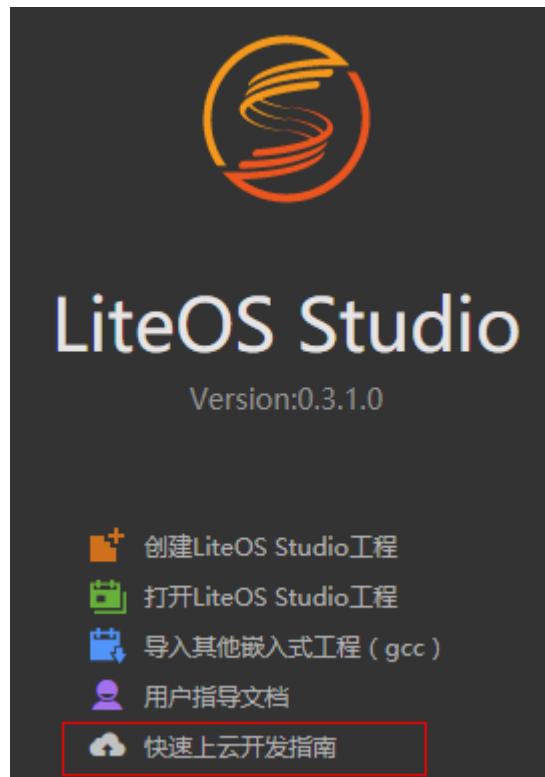
步骤4 单击“完成”，完成新建工程。

----结束

OC 快速上云

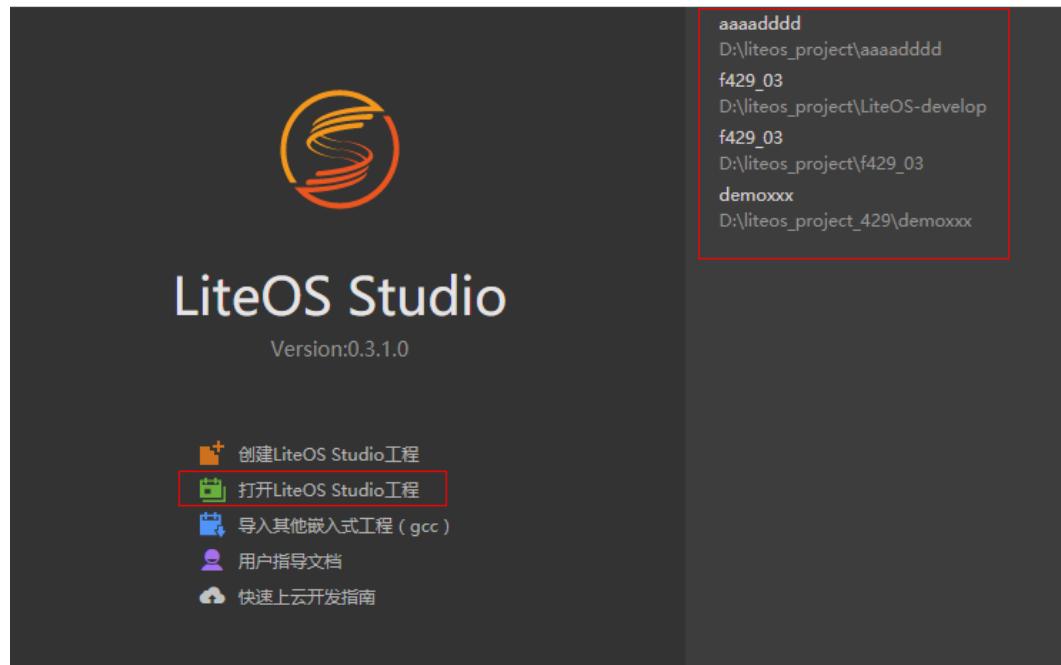
以“智慧路灯”为例，提供快速上云开发体验。

单击“快速上云开发指南”，查看“智慧路灯”案例操作指导。

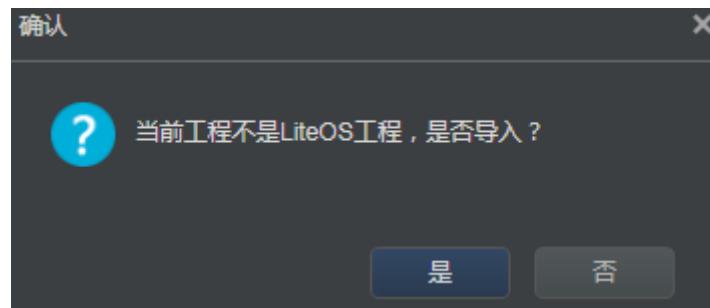


3.2 打开工程

单击“打开LiteOS Studio工程”，或在右边区域单击最近打开的工程名，打开工程。如下所示：



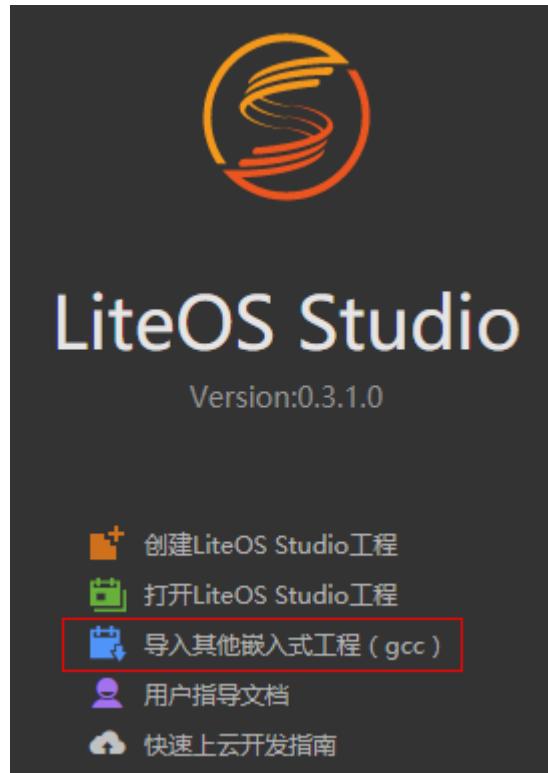
- 若打开的工程为LiteOS Studio工程，进入LiteOS Studio主界面。
- 若打开的工程为非LiteOS Studio工程，弹出如下提示弹框。



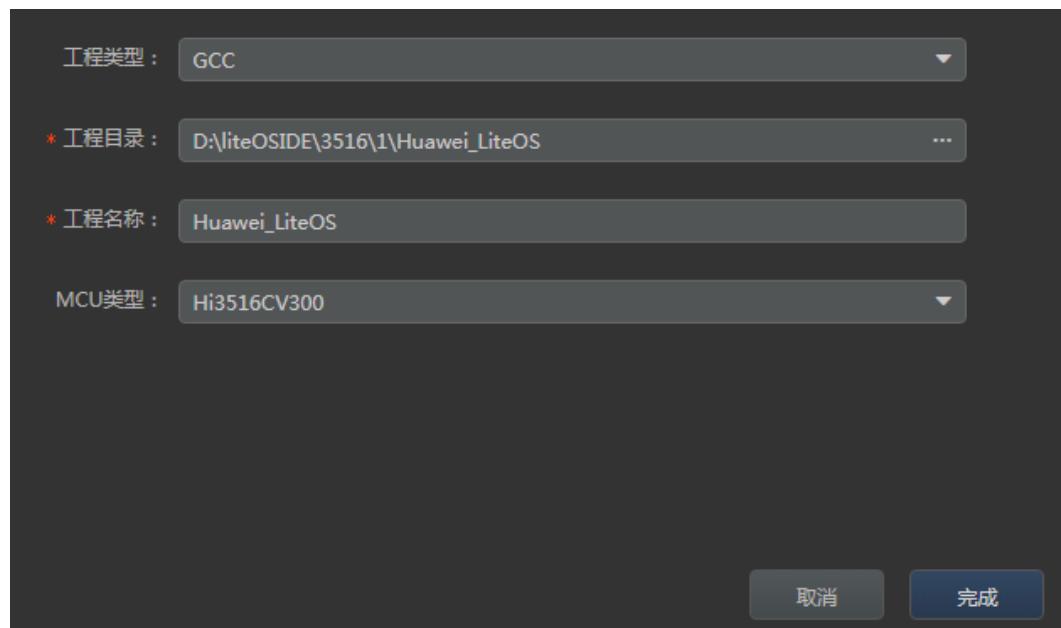
- 单击“是”，进入“导入”界面。导入操作具体请参见[3.3 导入工程](#)章节。
- 单击“否”，取消打开工程。

3.3 导入工程

步骤1 单击“导入其他嵌入式工程（gcc）”。



步骤2 弹出导入界面，选择需要导入的工程目录及对应的MCU类型。



步骤3 单击“完成”，完成工程导入。

----结束

4 环境准备

在进行工程的编译，烧录及调试前，请先下载LiteOS Studio依赖的JLink工具。

操作步骤

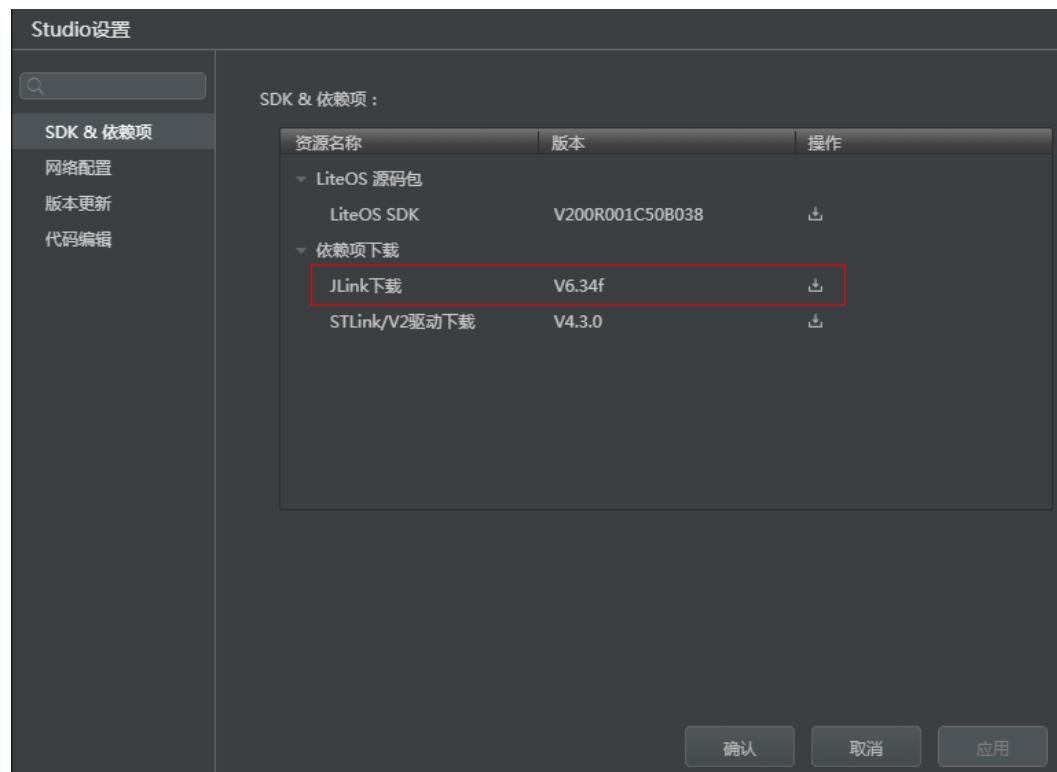
步骤1 进入任一LiteOS工程。具体操作请参见[3 LiteOS工程建立](#)章节。

步骤2 安装JLink工具。如已安装，请忽略此步骤。

- 单击菜单栏中的“文件>首选项”。



- 在“SDK&依赖项”界面，单击“JLink下载”所在行的 Δ ，获取JLink安装包“Link_Windows_V634f.exe”。



3. 运行“JLink_Windows_V634f.exe”，依照屏幕提示，安装JLink。

步骤3 (可选) 配置JLink系统环境变量。

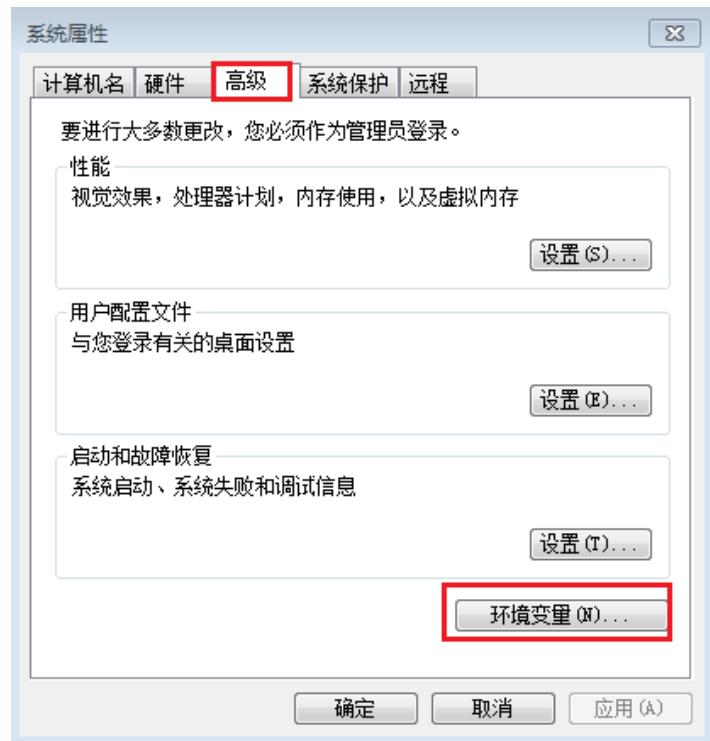
说明

- 配置JLink系统环境变量后，Studio将对打开的任一工程，自动读取系统配置的JLink参数，无需再逐一对每个工程进行手动配置。
- 如未自动读取系统配置的JLink参数，请重启电脑即可。

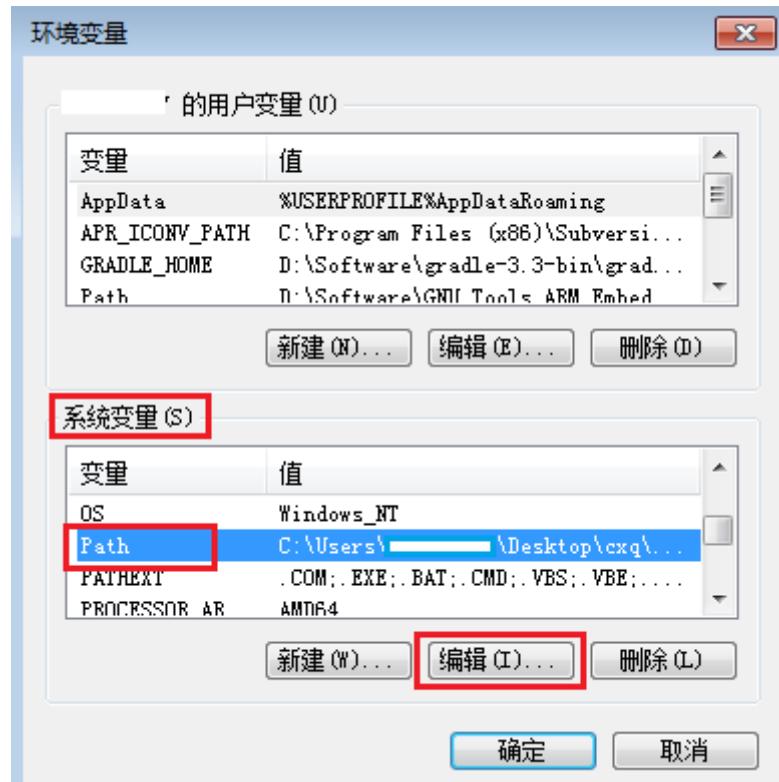
1. 单击“控制面板>系统和安全>系统>高级系统设置”。



2. 在“高级”页签，单击“环境变量”。



3. 在“系统变量”区域下，单击“Path”变量所在行，再单击“编辑”。



4. 在“变量值”文本框的最后添加JLink的环境变量值“<JLink安装目录>\SEGGER\\JLink_V634f”，不同值之间以英文分号（;）隔开。

说明

请根据实际情况修改JLink安装目录，如：C:\Program Files (x86)\SEGGER\JLink_V634f。



5. 依次单击“确定”，完成环境变量配置。

----结束

5 快速上手

本章以STM32F429IG、华为海思3516Cv300及STM32L431xx工程示例，进行编译、烧录及调试功能演示，快速掌握LiteOS Studio工具使用方法，详细的LiteOS Studio工具介绍请参见[6 LiteOS Studio使用](#)章节。

5.1 STM32F429IG 工程示例

第三方芯片（非华为海思芯片）工程演示以STM32F429IG开发板为例，并在Windows系统中进行编译，烧录及调试。其中STM32L431xx工程的编译，烧录及调试请参见[5.3 STM32L431xx工程示例](#)章节。

环境准备

- Windows 7以上 64位操作系统。
- 准备STM32F429IG开发板及USB连接线。
- 确保电脑的USB端口正常且可用。

工程示例

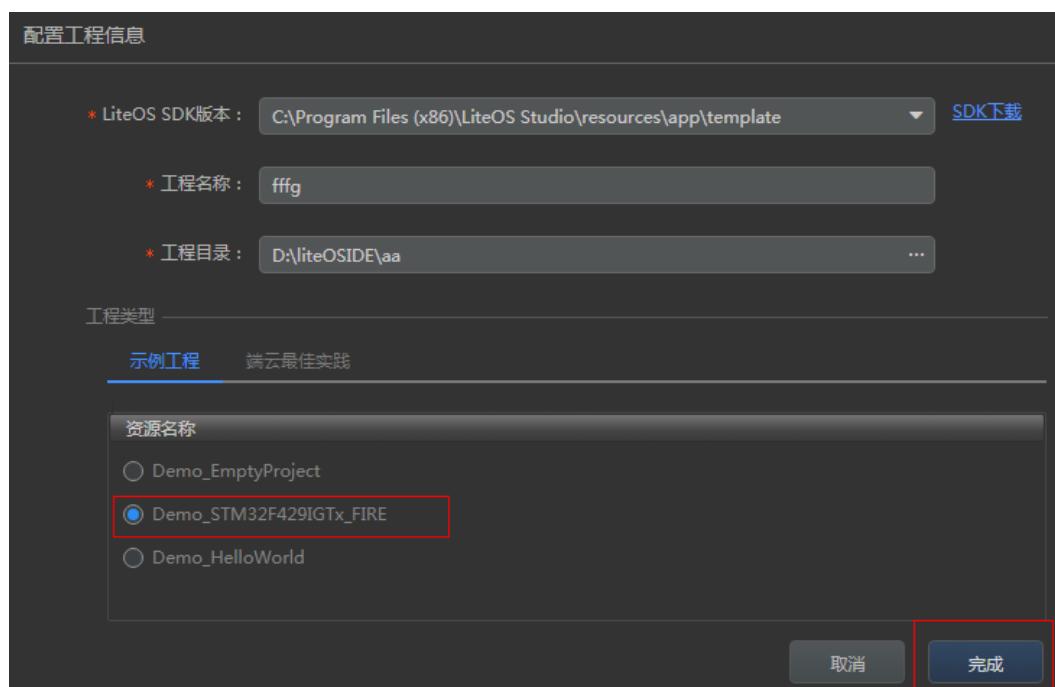
步骤1 将STM32F429IG开发板与电脑连接。

步骤2 新建STM32F429IG工程。

1. 单击“创建LiteOS Studio工程”。



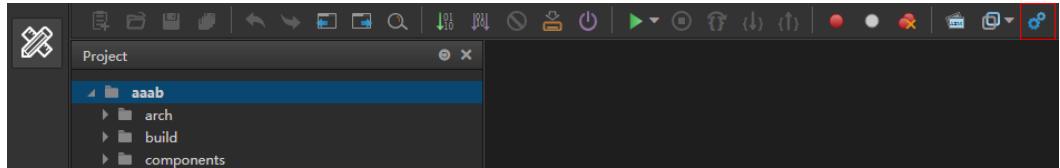
2. 自定义“工程名称”及选择“工程目录”。
- 说明
 - 内置SDK，自动配置“LiteOS SDK版本”，如需更改路径，单击“SDK下载”，自定义SDK保存路径。
 - 工程目录中不能使用标点符号。
3. 在“示例工程”页签下，选择STM32F429IG工程。



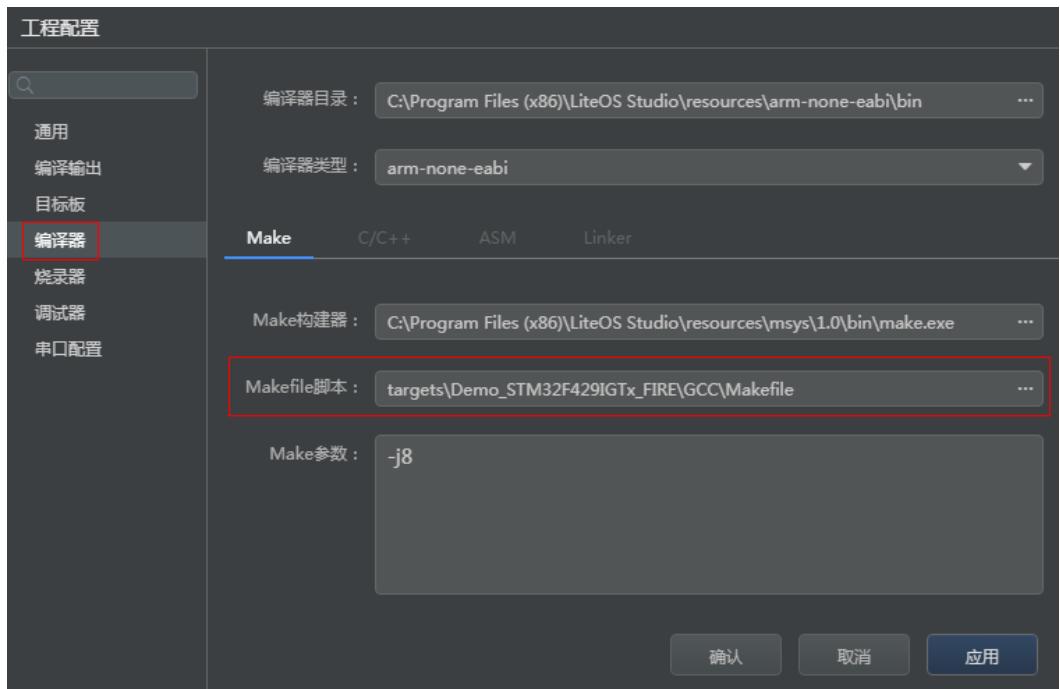
4. 单击“完成”，完成新建工程。

步骤3 编译前，请进行工程配置。

- 单击工具栏中的，进行工程配置。



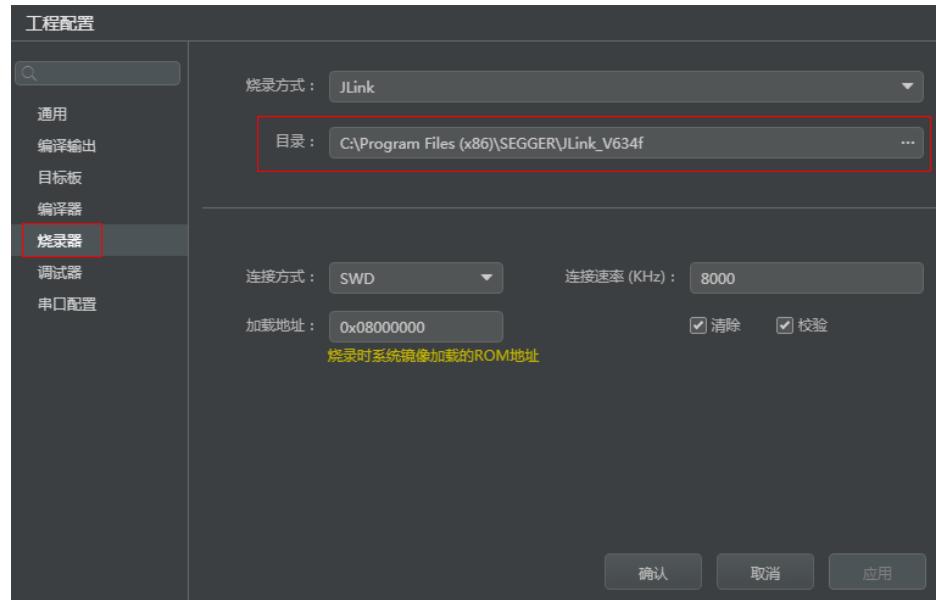
- 单击“编译器”，配置“Makefile脚本”，选择当前工程下的主makefile，如下图所示，具体路径请根据实际情况修改。“Make参数”可配置为“-j8”。



- 配置JLink目录为“<JLink安装目录>\SEGGER\JLink_V634f”。如已完成**4 环境准备**章节中的**步骤3**，已配置JLink系统环境变量，请忽略此步骤。

 **说明**

- 请根据实际情况修改JLink安装目录，如：C:\Program Files (x86)\SEGGER\JLink_V634f。
- 配置烧录器中的JLink目录。



- 配置调试器中的“JLink目录”。



4. 单击“确认”，完成工程配置。

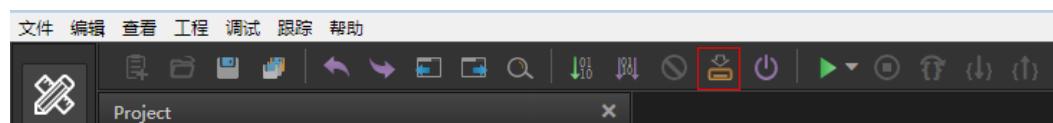
步骤4 单击工具栏中的，对当前工程进行编译。



编译成功后，在控制台输出面板中显示“编译成功”。

The screenshot shows the 'Control Console' window of the LiteOS Studio. The log displays the compilation process of various source files: usart.c, dwt.c, eth.c, sys_init.c, user_task.c, at_api.c, at_main.c, at_hal.c, esp8266.c, sim900a.c, bg36.c, bc95.c, los_nb_api.c, and los_dispatch_gcc.S. It also shows the creation of the arm-none-eabi-size build/Huawei_LiteOS.elf file, the generation of hex and binary files, and the final message indicating 0 Error(s) and 0 Warning(s). A red box highlights the success message: [2019-03-06 18:41:14] 编译成功。编译耗时: 181117ms.

步骤5 单击工具栏中的 ，将已编译的程序烧录至开发板。



控制台输出面板中显示“烧录成功”后，完成烧录。

```
控制台
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x08009000 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x13 Size:0x800
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x08009800 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x14 Size:0x800
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x0800a000 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x15 Size:0x800
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x0800a800 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x16 Size:0x800
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x0800b000 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x17 Size:0x800
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x0800b800 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x18 Size:0x800
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x0800c000 erased
2019-03-05T16:49:05 INFO common.c: EraseFlash - Page:0x19 Size:0x800
2019-03-05T16:49:05 INFO common.c: Finished erasing 26 pages of 2048 (0x800) bytes
2019-03-05T16:49:05 INFO common.c: Startin
2019-03-05T16:49:05 INFO common.c: Flash page at addr: 0x0800c800 erased
2019-03-05T16:49:05 INFO common.c: g Flash write for F2/F4/L4
2019-03-05T16:49:05 INFO flash_loader.c: Successfully loaded flash loader in sram
2019-03-05T16:49:06 INFO common.c: Start
2019-03-05T16:49:06 INFO common.c: ing verification of write complete
2019-03-05T16:49:06 INFO common.c: Flash written
and verified! jolly good!

size: 32768
size: 19560
[2019-03-05 16:49:06] 烧录成功!
```

步骤6 调试。

烧录成功后，单击 开始调试。如需更改调试方式，单击 下拉框选择调试方式，调试方式介绍请参见表5-1。

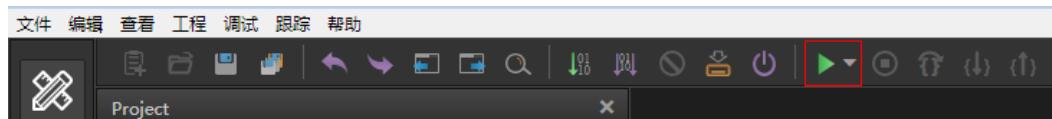


表 5-1 调试方式介绍

调试方式	说明
复位调试	每次启动复位调试时都会重启目标板，目标板停在复位状态，再进行调试（目标板停止位置固定，停在main函数入口）。
附加调试	在目标板运行状态中介入调试，不影响目标板原运行轨迹和状态，当启动附加调试时，主动命令目标板停止，再进行调试（目标板停止位置随机，取决于介入调试的时间）。

调试方式	说明
自定义	通过修改gdbinit文件，自定义调试方式。

----结束

5.2 华为海思 3516CV300 工程示例

华为海思芯片工程演示以华为海思3516CV300开发板为例，在Linux系统中进行编译，并对在Linux系统中编译出来的系统镜像二进制文件在Windows系统中进行烧录和调试。（暂不支持在Windows系统中对华为海思芯片工程进行编译）。

5.2.1 环境准备

操作系统

支持windows 7以上 64位操作系统。

硬件接口

- 准备华为海思3516CV300开发板及JLink仿真器（V9.4及以上版本）。
- 确保电脑的USB端口正常且可用。
- 开发板预留JTAG接口，并验证JTAG接口是否就绪。
 - a. 将开发板通过JTAG接口连接至J-Link。
 - b. 确保开发板pin拨码开关为全ON。
 - c. 打开J-Link Commander工具。
 - d. 输入connect，回车。
 - e. 若打印信息中检测到目标芯片内核，说明连接正常。

```
J-Link>connect
Please specify device / core. <Default>: AT91SAM9XE256
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
TIF>
Device position in JTAG chain (IRPre,DRPre) <Default>: -1,-1 => Auto-detect
JTAGConf>
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "AT91SAM9XE256" selected.

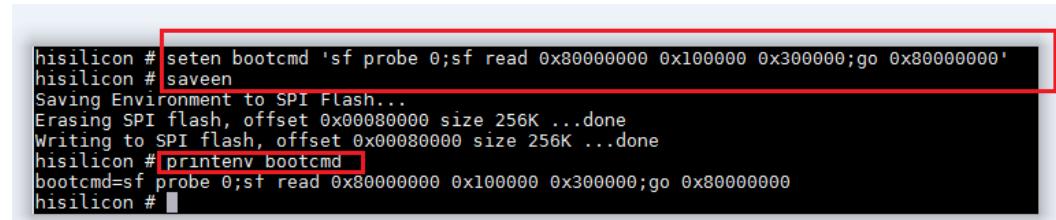
Connecting to target via JTAG
TotalIRLen = 4, IRRprint = 0x01
JTAG chain detection found 1 devices:
#0 Id: 0x07926477, IRLen: 04, ARM926EJ-S Core
Using DBGRQ to halt CPU
CP15.0.0: 0x41069265: ARM, Architecture 5TEJ
CP15.0.1: 0x1D192192: ICache: 32kB (4*256*32), DCache: 32kB (4*256*32)
Cache type: Separate, Write-back, Format C (WT supported)
ARM9 identified. ←
```

uboot 环境变量

3516C开发板复位调试依赖uboot重启，需要确保开发板断电后能正常重启，若不能正常启动，需要配置uboot环境变量。

以华为海思3516CV300开发板为例，依照下图红框所示，配置如下环境变量：

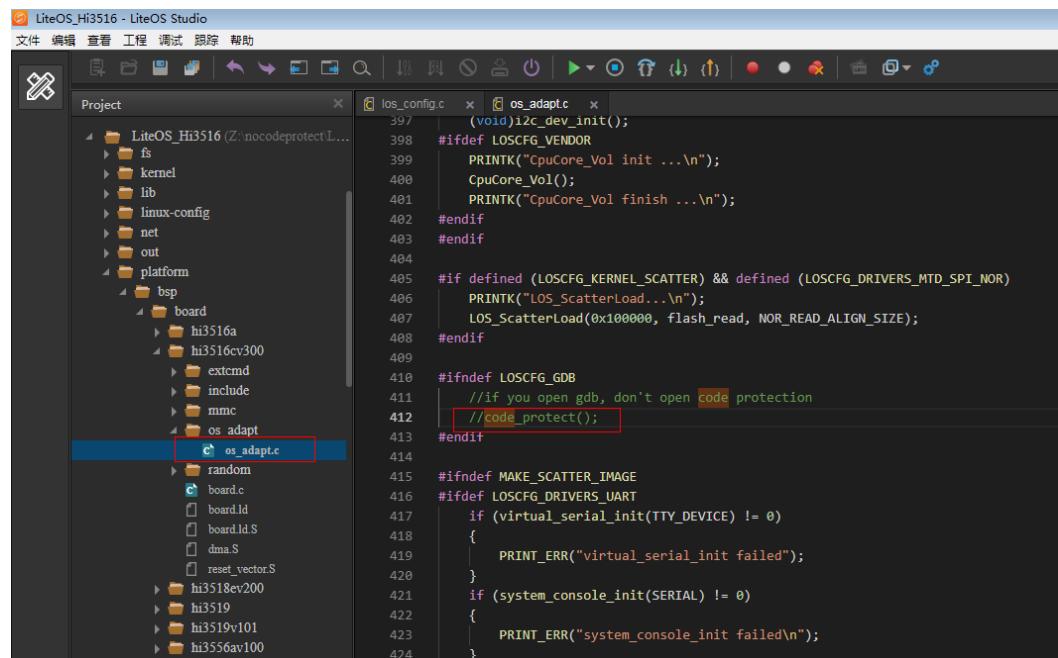
bootcmd=sf probe 0;sf read 0x80000000 0x100000 0x300000;go 0x80000000。



```
hisilicon # setenv bootcmd 'sf probe 0;sf read 0x80000000 0x100000 0x300000;go 0x80000000'
hisilicon # saveenv
Saving Environment to SPI Flash...
Erasing SPI flash, offset 0x00080000 size 256K ...done
Writing to SPI flash, offset 0x00080000 size 256K ...done
hisilicon # printenv bootcmd
bootcmd=sf probe 0;sf read 0x80000000 0x100000 0x300000;go 0x80000000
hisilicon #
```

5.2.2 编译前注意事项

- 修改编译参数。也可参考[6.7.2.3 在Linux系统下进行编译](#)章节中的**步骤6**进行配置。
 - 在内核根目录运行make menuconfig。
 - 在Debug选项中勾选Enable GCC -g。
 - 若使用了单独的makefile，需要在编译选项中添加 -g -O0选项。
- 关闭代码保护：若编译选项中开启了Os_adapt，关闭代码保护，并重新编译。注释platform/bsp/board/hi3516cv300/os_adapt/os_adapt.c文件里的code_protect()函数。



5.2.3 工程示例

前提条件

已在Linux系统下完成编译，具体请参考[6.7.2 编译（Linux）](#)章节。

工程演示

步骤1 通过JLink仿真器将华为海思3516CV300开发板与电脑连接。

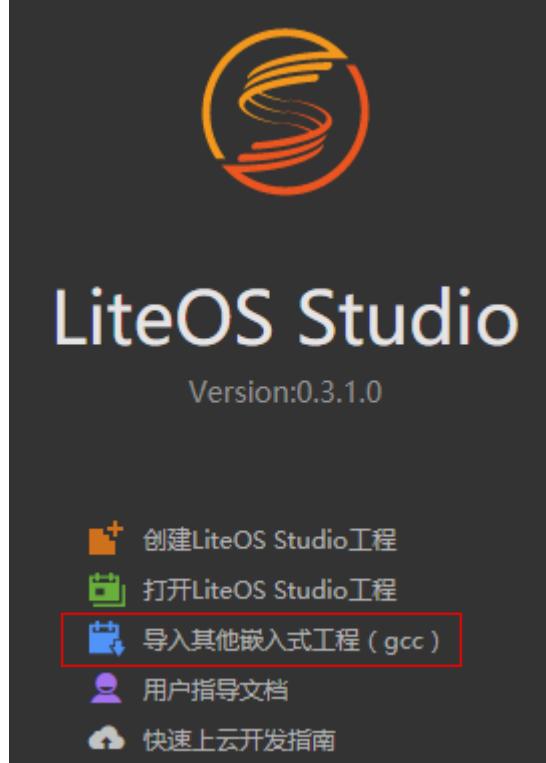


说明

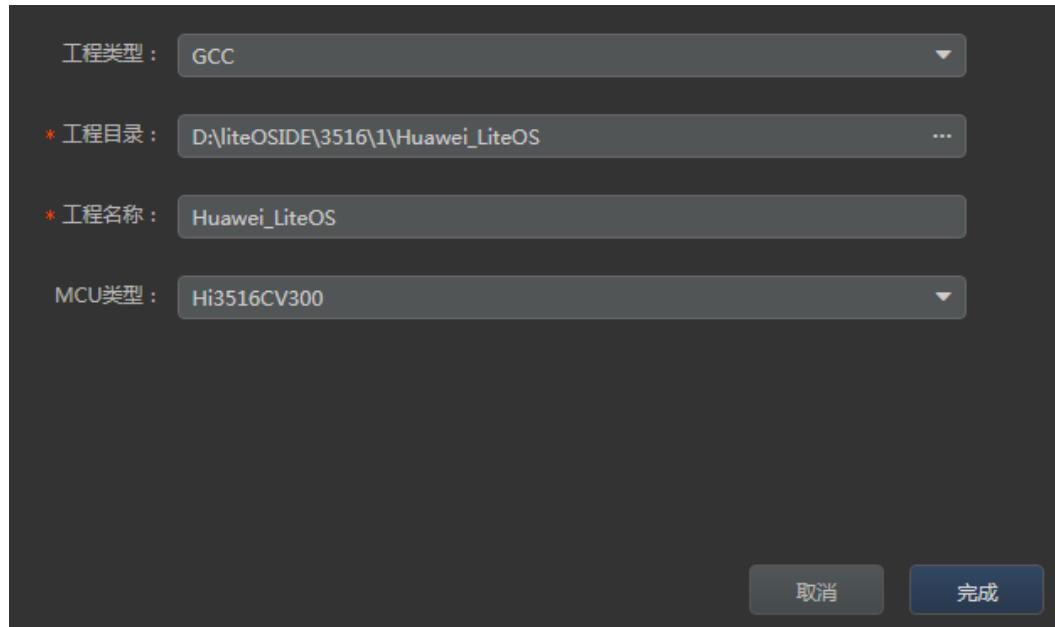
- 请先连接开发板电源，再将开发板与JLink仿真器连接。
- 若初次使用华为海思3516CV300开发板，请先完成开发板的硬件设置，具体请参见[8.5 华为海思3516CV300开发板硬件如何设置](#)章节。

步骤2 打开在Linux系统上编译后的华为海思3516CV300工程。

1. 单击“导入其他嵌入式工程（gcc）”。



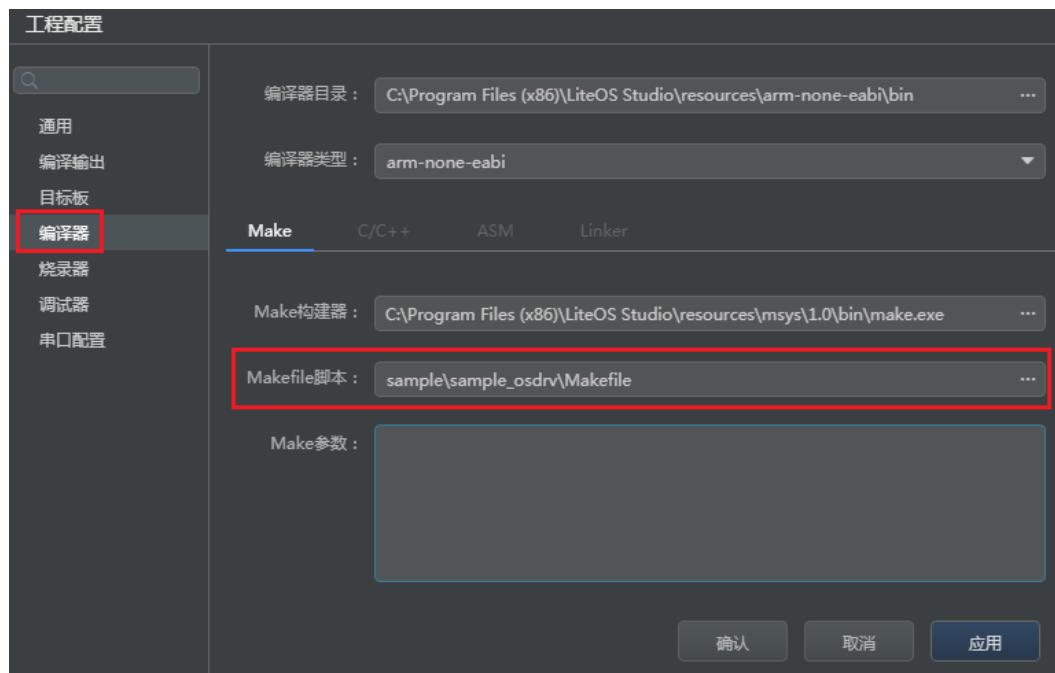
2. 弹出导入界面，选择需要导入的华为海思3516CV300工程目录及对应的MCU类型。



3. 单击“完成”，完成配置。

步骤3 在烧录前请进行如下工程配置。

1. 在打开的华为海思3516CV300工程中，单击工具栏的 ，进行工程配置。
2. 单击“编译器”，“Makefile脚本”配置为编译输出目录中的makefile文件，具体路径请根据实际情况修改。“Make参数”可配置为“-j8”。

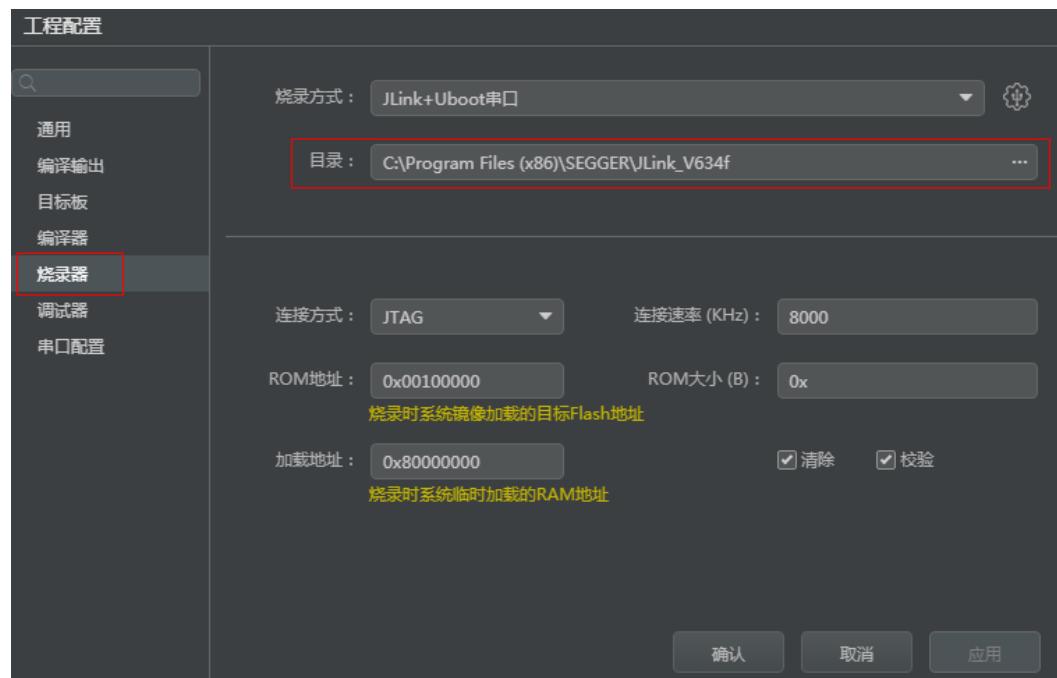


3. 配置JLink目录为“<JLink安装目录>\SEGGER\JLink_V634f”。如已完成[4 环境准备](#)章节中的[步骤3](#)，已配置JLink系统环境变量，请忽略此步骤。

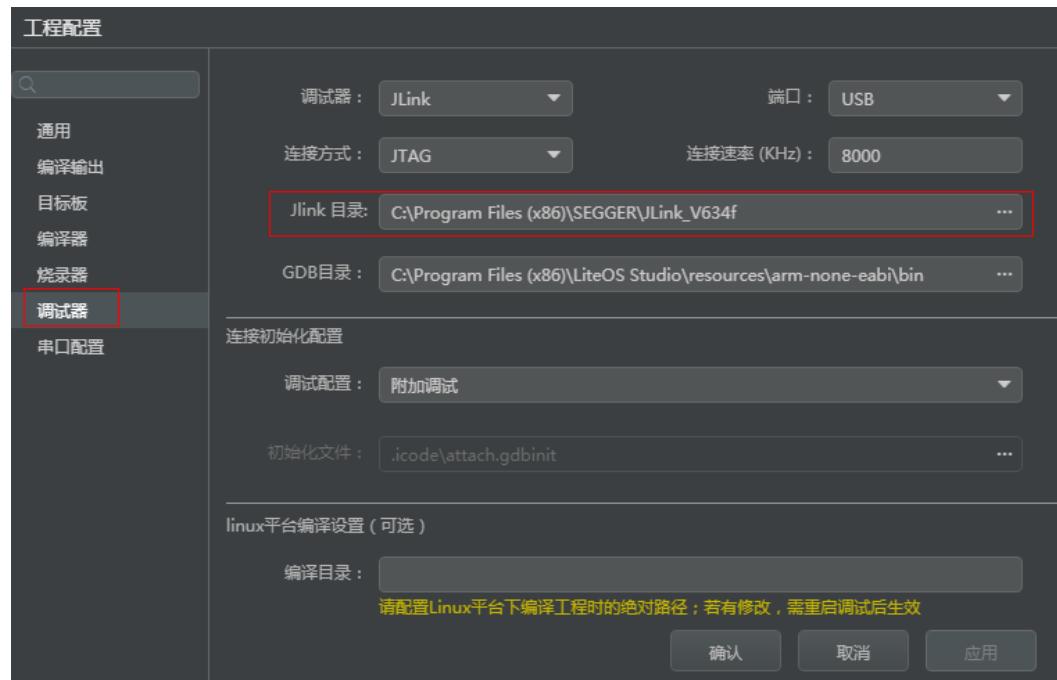
说明

请根据实际情况修改JLink安装目录，如：C:\Program Files (x86)\SEGGER\JLink_V634f。

- 配置烧录器中的JLink目录。

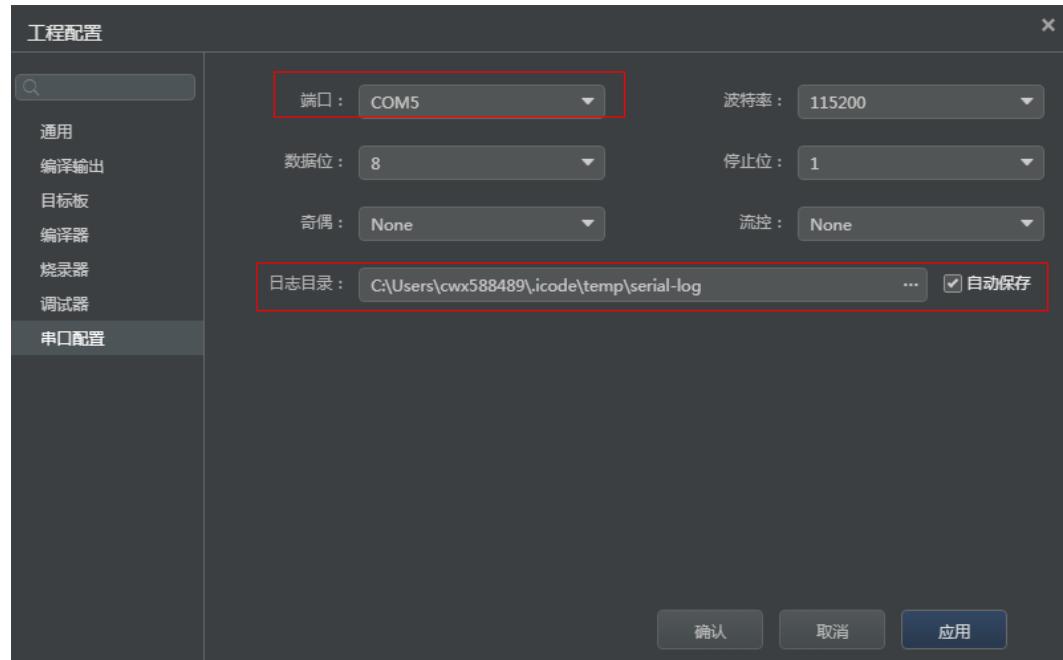


- 配置调试器中的JLink目录。



4. 单击“串口配置”，在“端口”下拉框中选择与开发板实际连接的端口号。获取端口号具体请参见[7.1 获取连接目标板的实际串口号](#)章节。

如需自动保存日志，请勾选“自动保存”，您也可以单击“日志目录”的...按钮，更改日志保存路径。



5. 单击“确认”，保存配置。

步骤4 单击工具栏中的 ，将已编译好的程序烧录至开发板。



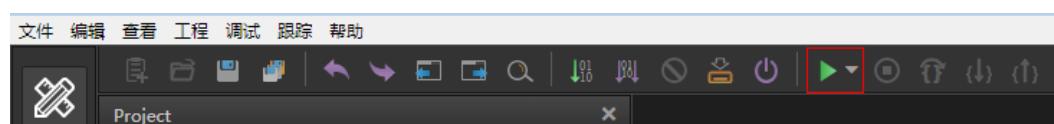
控制台输出面板中显示“烧录成功”后，完成烧录。

控制台

```
Writing at 0x130000 -- 10% complete.  
Writing at 0x140000 -- 13% complete.  
Writing at 0x150000 -- 16% complete.  
Writing at 0x160000 -- 20% complete.  
Writing at 0x170000 -- 23% complete.  
Writing at 0x180000 -- 26% complete.  
Writing at 0x190000 -- 30% complete.  
Writing at 0x1a0000 -- 33% complete.  
Writing at 0x1b0000 -- 36% complete.  
Writing at 0x1c0000 -- 40% complete.  
Writing at 0x1d0000 -- 43% complete.  
Writing at 0x1e0000 -- 46% complete.  
Writing at 0x1f0000 -- 50% complete.  
Writing at 0x200000 -- 53% complete.  
Writing at 0x210000 -- 56% complete.  
Writing at 0x220000 -- 60% complete.  
Writing at 0x230000 -- 63% complete.  
Writing at 0x240000 -- 66% complete.  
Writing at 0x250000 -- 70% complete.  
Writing at 0x260000 -- 73% complete.  
Writing at 0x270000 -- 76% complete.  
Writing at 0x280000 -- 80% complete.  
Writing at 0x290000 -- 83% complete.  
Writing at 0x2a0000 -- 86% complete.  
Writing at 0x2b0000 -- 90% complete.  
Writing at 0x2c0000 -- 93% complete.  
Writing at 0x2d0000 -- 96% complete.  
Writing at 0x2e0000 -- 100% complete.  
Flash 写入成功。  
[2018-11-09 15:20:32] 烧录成功
```

步骤5 调试。

烧录成功后，单击 开始调试。如需更改调试方式，单击 下拉框选择调试方式，调试方式介绍请参见表5-1。



----结束

5.3 STM32L431xx 工程示例

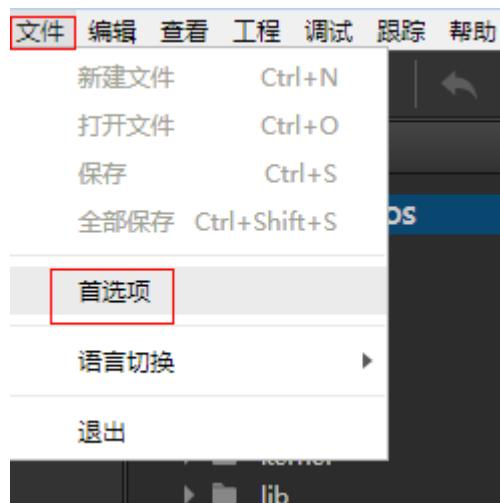
背景信息

针对STM32L431xx开发板，烧录及调试需使用STLink或OpenOCD方式，且烧录及调试方式需保持一致。

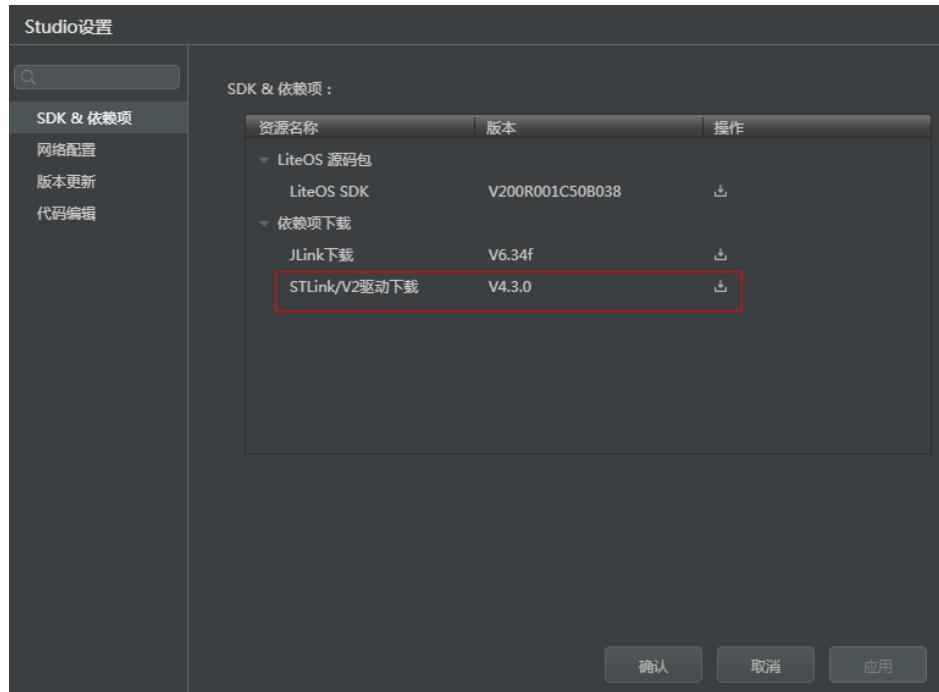
本章节以STLink烧录及调试方式为例，介绍STM32L431xx工程的编译，烧录及调试步骤。

环境准备

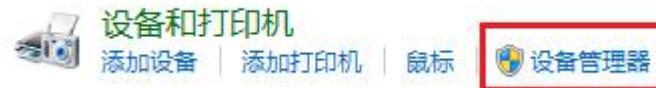
- windows 7以上 64位操作系统。
- 准备STM32L431xx开发板及USB连接线。
- 确保电脑的USB端口正常且可用。
- 安装STLink驱动器。如已安装，请忽略此步骤。
 - a. 单击“文件>首选项”。



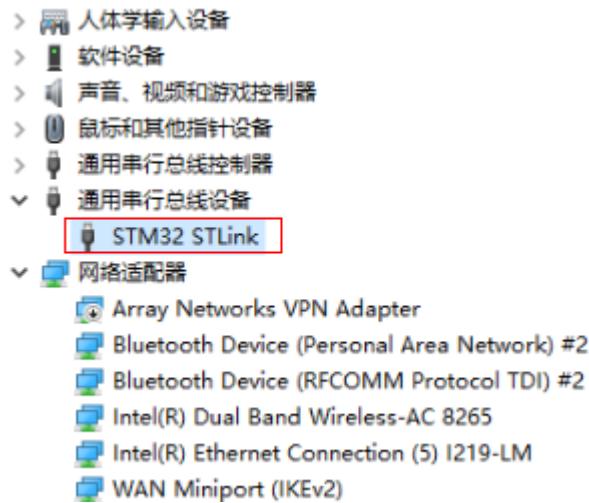
- b. 在“SDK&依赖项”界面下，单击“STLink/V2驱动下载”所在行的⁺，获取STLink/V2驱动安装包“STM32 ST-LINK Utility v4.3.0 setup.exe”。



- c. 运行“STM32 ST-LINK Utility v4.3.0 setup.exe”，安装STLink驱动器。
- d. 安装完成后，打开“控制面板>硬件和声音>设备管理器”



- e. 在“通用串行总线设备”菜单下，找到“STM32 STLink”，则驱动安装成功。

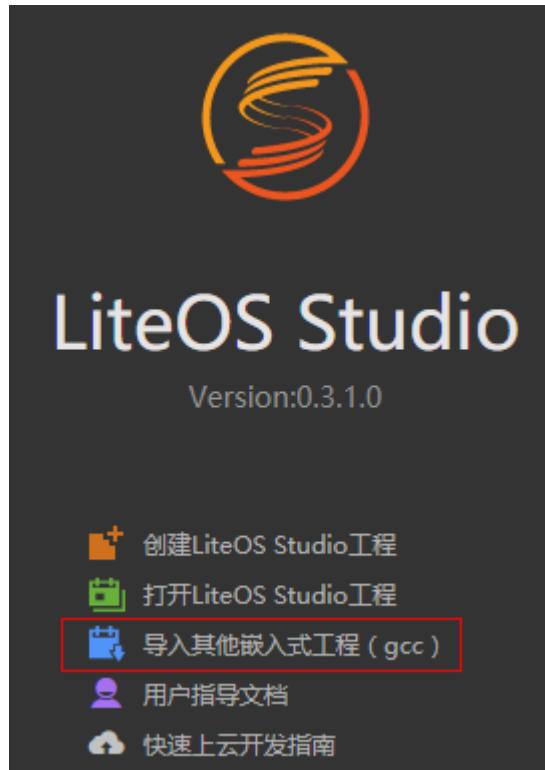


工程示例

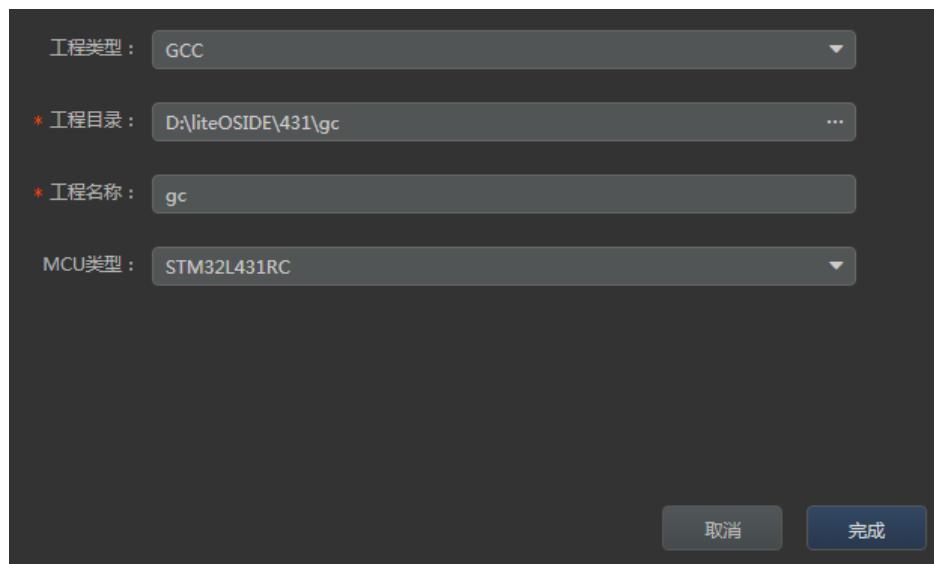
步骤1 将STM32L431xx开发板与电脑连接。

步骤2 打开STM32L431xx工程。

1. 单击“导入其他嵌入式工程（gcc）”，选择STM32L431xx工程。



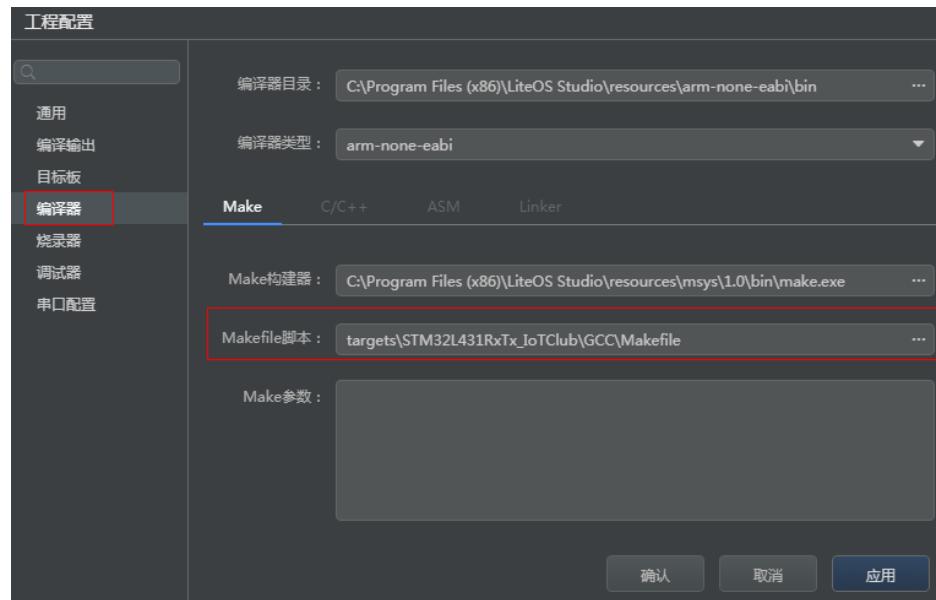
2. 弹出导入界面，选择需要导入的STM32L431xx工程目录及对应的MCU类型。



3. 单击“完成”，完成配置。

步骤3 在编译前请进行如下工程配置。

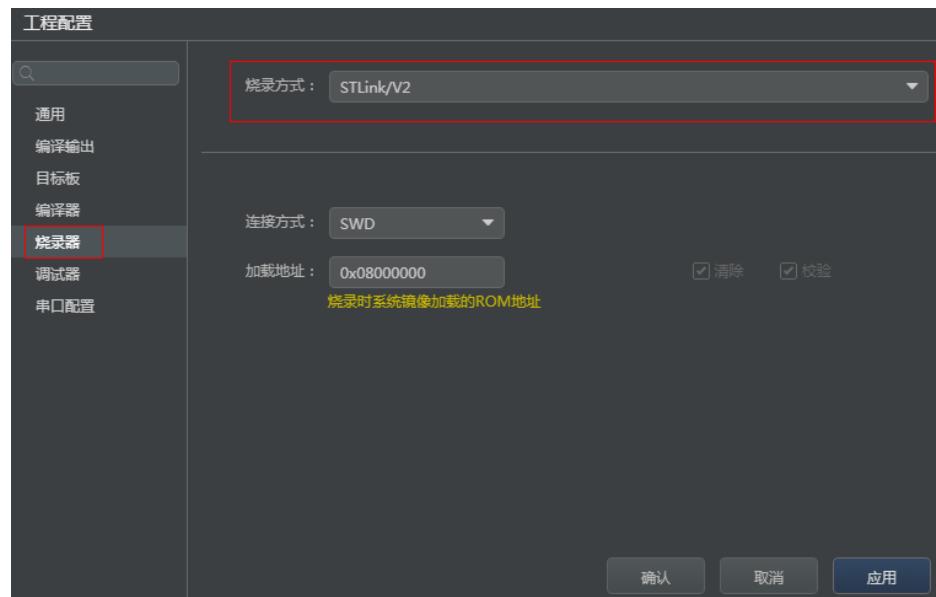
1. 在打开的STM32L431xx工程中，单击工具栏的 ，进行工程配置。
2. 单击“编译器”，“Makefile脚本”选择当前工程下的主makefile，具体路径请根据实际情况修改。Make参数”可配置为“-j8”。



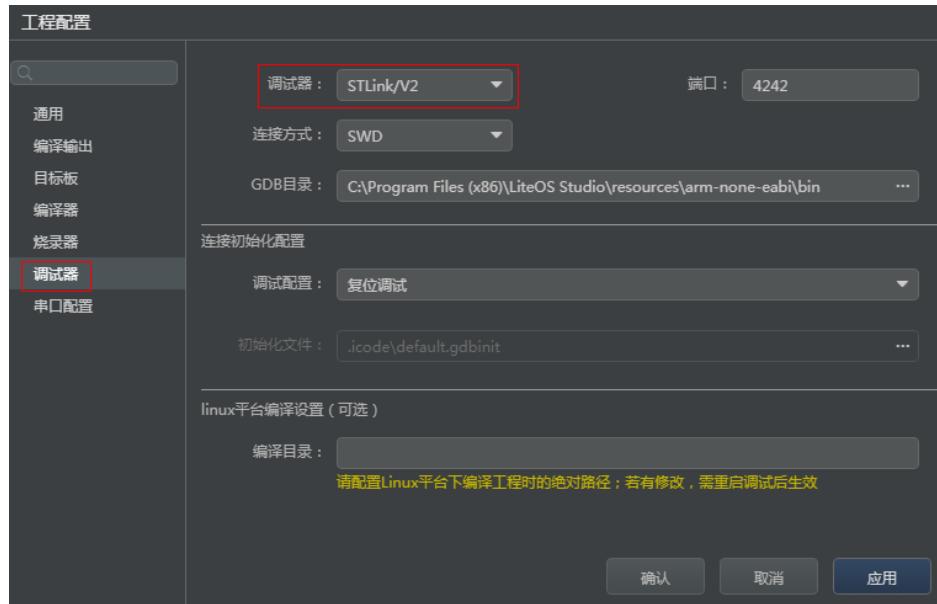
3. 单击“烧录器”，烧录方式选择“STLink/V2”，其他参数默认配置。

说明

烧录方式若选择“OpenOCD”，调试方式需与烧录方式保持一致为“OpenOCD”，其他参数默认配置即可。



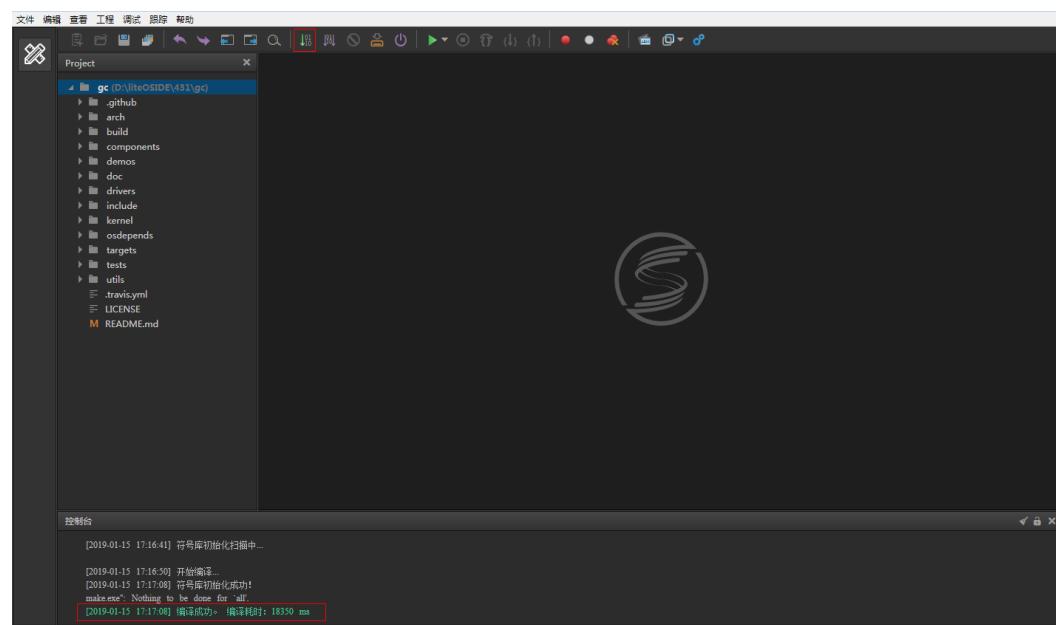
4. 单击“调试器”，调试方式选择“STLink/V2”，其他参数默认配置。



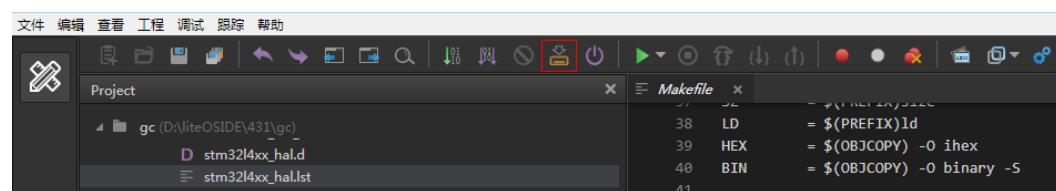
5. 单击“确认”，完成配置。

步骤4 单击工具栏中的，对当前工程进行编译。

编译成功后，在控制台输出面板中显示“编译成功”。



步骤5 单击工具栏中的，将已编译好的程序烧录至开发板。



控制台输出面板中显示“烧录成功”后，完成烧录。

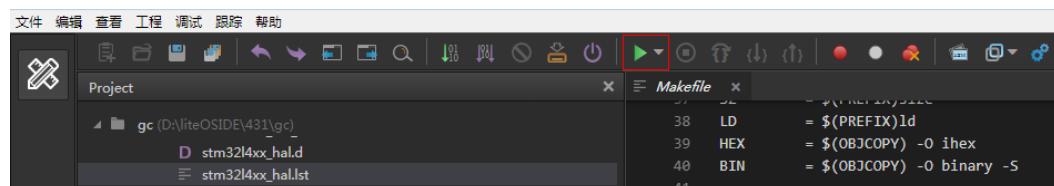
```
控制台
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x08009000 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x13 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x08009800 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x14 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x0800a000 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x15 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x0800a800 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x16 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x0800b000 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x17 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x0800b800 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x18 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x0800c000 erased
[2019-03-05T16:49:05] INFO common.c: EraseFlash - Page:0x19 Size:0x800
[2019-03-05T16:49:05] INFO common.c: Finished erasing 26 pages of 2048 (0x800) bytes
[2019-03-05T16:49:05] INFO common.c: Startin
[2019-03-05T16:49:05] INFO common.c: Flash page at addr: 0x0800c800 erased
[2019-03-05T16:49:05] INFO common.c: g Flash write for F2/F4/L4
[2019-03-05T16:49:05] INFO flash_loader.c: Successfully loaded flash loader in sram
[2019-03-05T16:49:06] INFO common.c: Start
[2019-03-05T16:49:06] INFO common.c: ing verification of write complete
[2019-03-05T16:49:06] INFO common.c: Flash written
[2019-03-05T16:49:06] INFO common.c: and verified! jolly good!

size: 32768
size: 19560
[2019-03-05 16:49:06] 烧录成功!
```

说明

若烧录方式为“OpenOCD”，烧录成功后如需重新烧录，需等待10秒或者重新连接开发板再烧录。

步骤6 烧录成功后，单击 开始调试。如需更改调试方式，单击 下拉框选择调试方式，调试方式介绍请参见[表5-1](#)。



----结束

6 LiteOS Studio 使用

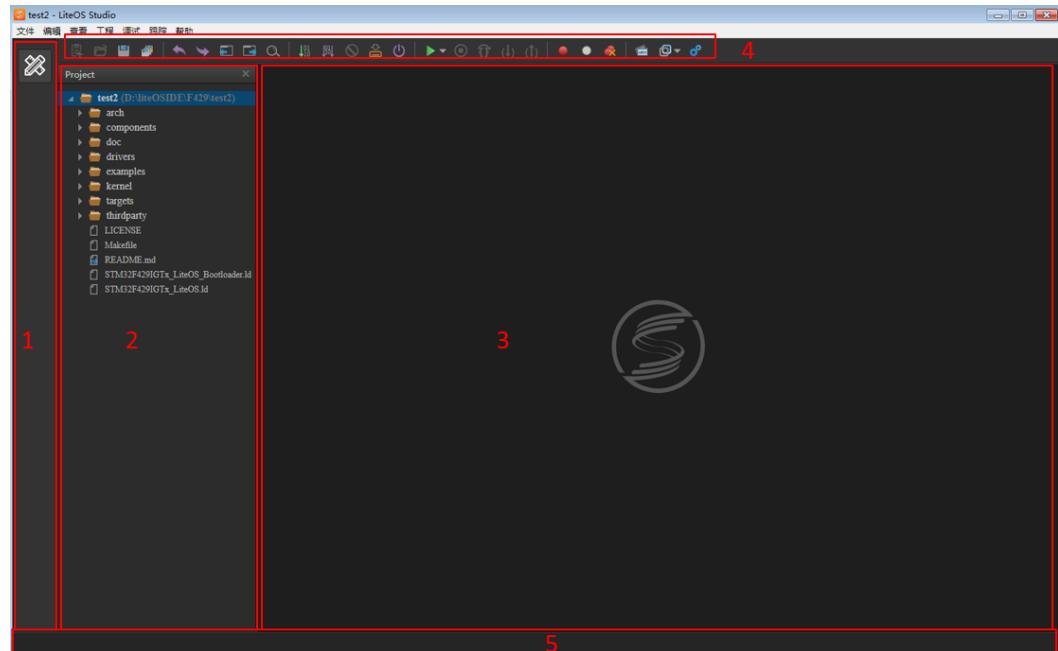
- [6.1 了解工程界面](#)
- [6.2 设置Studio](#)
- [6.3 代码编辑](#)
- [6.4 断点控制](#)
- [6.5 控制台输出](#)
- [6.6 编译，烧录及调试（非华为海思芯片）](#)
- [6.7 编译，烧录及调试（华为海思芯片）](#)
- [6.8 调试中的各面板作用](#)

6.1 了解工程界面

6.1.1 界面

LiteOS Studio工程界面构成如下：

- 区域1：编辑界面。
- 区域2：工程树，由项目工程文件构成，可进行快速新建及打开文件等操作。
- 区域3：代码编辑区。
- 区域4：调试工具栏，可进行编译、烧录、调试等操作。
- 区域5：状态栏，源码符号扫描等的信息显示。



6.1.2 工具栏

工具栏如下图所示：



新建文件



打开文件



编译

单击 (编译) , 对当前打开工程进行编译。

重新编译

单击 (重新编译) , 删除上一次编译生成的文件, 再次执行编译。

停止编译

单击 (停止编译) , 停止正在进行的编译。

烧录

单击 (烧录) , 将程序烧录至目标板。

重启目标板

单击 ，重启开发板。

开始调试

单击  启动调试，可启动或继续运行代码调试，全速运行状态中图标为 。

停止调试

单击  后停止调试。调试终止，调试信息清空（断点信息保留）。

下一步

单击  (下一步) 跳过当前代码行，执行到下一有效代码行。

跳入

单击  (跳入)，单步执行代码。

跳出

单击  (跳出) 跳出当前所在函数内部，跳至函数出口下一行代码。

6.2 设置 Studio

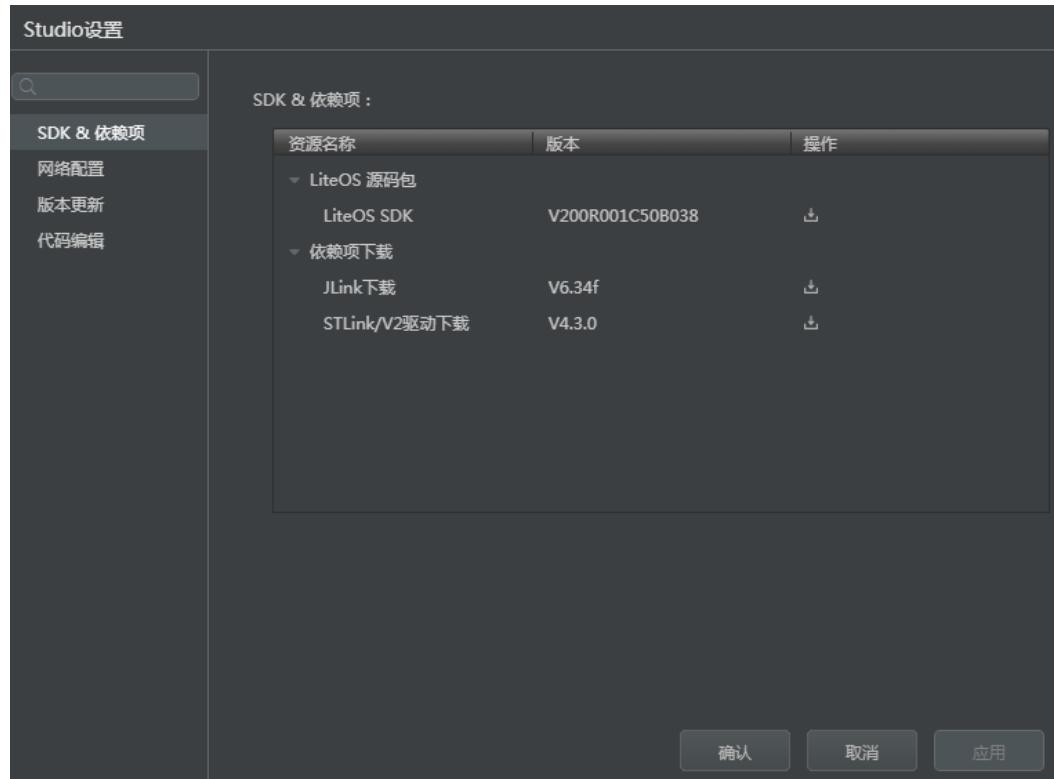
配置Studio后，LiteOS工程将自动读取此Studio的配置，并作为工程的默认配置。

6.2.1 Studio 设置入口

进入工程界面，单击菜单栏中的“文件>首选项”，打开“Studio 设置”界面。

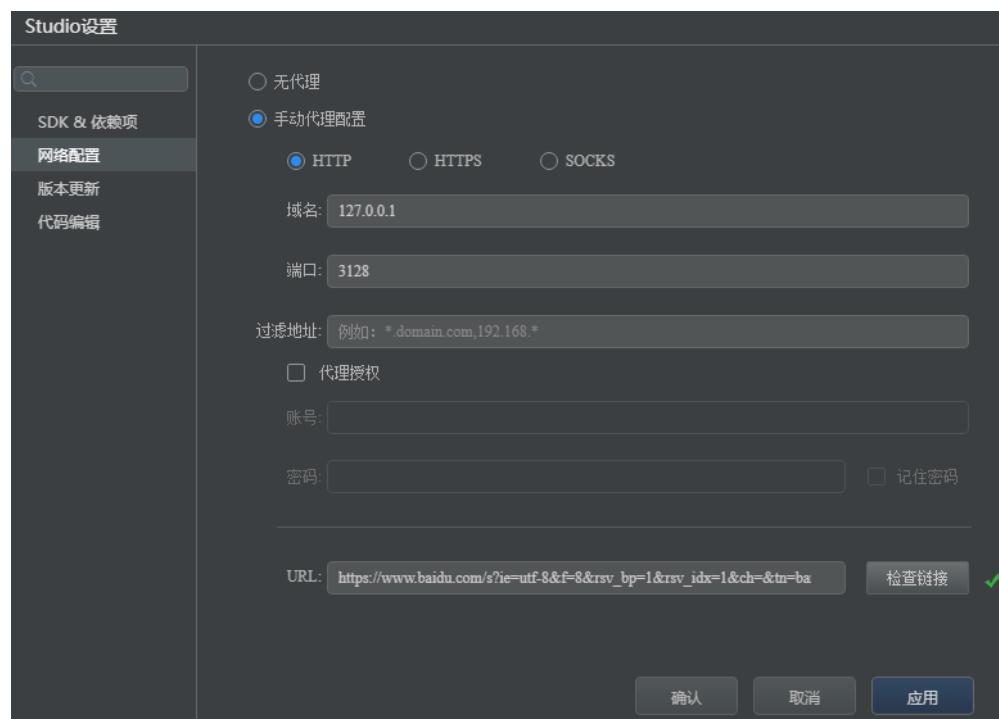
6.2.2 SDK&依赖项

在“SDK&依赖项”界面，您可以下载LiteOS SDK源码包、JLink安装包及STLink/V2驱动器安装包。



6.2.3 网络配置

- 普通上网，无代理，请勾选“无代理”，单击“应用”实时保存配置。
- 通过代理服务器上网，请勾选“手动代理配置”。



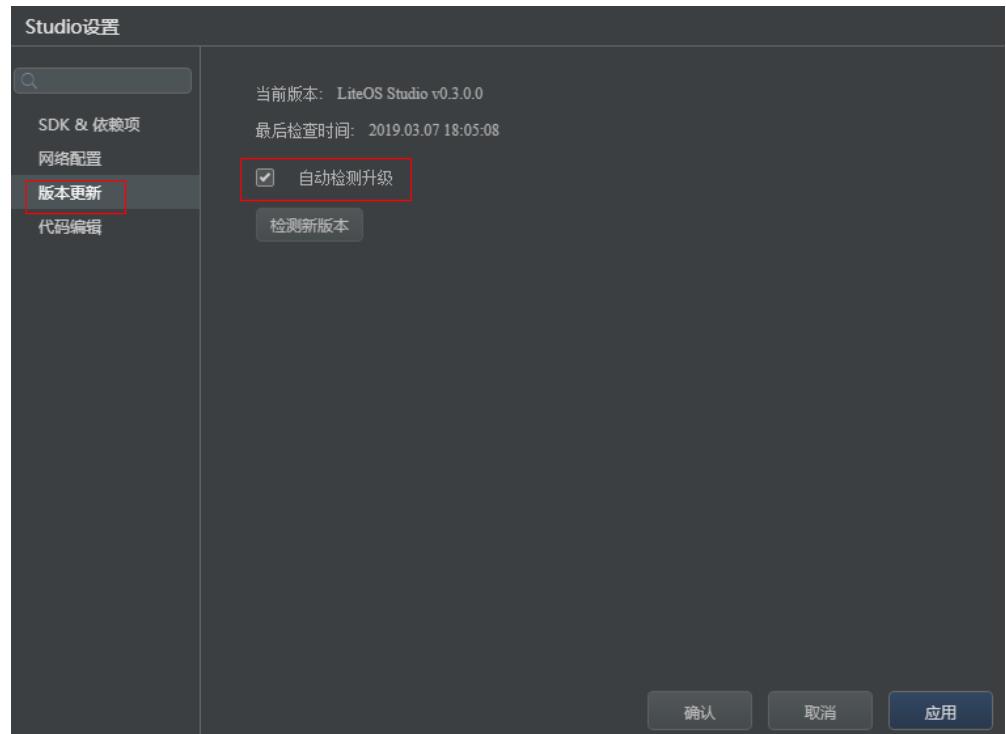
- a. 选择一种服务器代理类型。进入“Internet选项>连接>局域网设置”，在代理服务器区域，点击“高级”，查看配置的代理服务器类型。

- b. 填写域名和端口。如有代理授权，勾选“代理授权”后，填写账号和密码。
- c. 检查网络代理是否配置正确。在“URL”文本框填入网络链接，单击“检查链接”，出现√图标，网络代理配置成功，否则请检查代理配置。
- d. 单击“应用”实时保存配置。

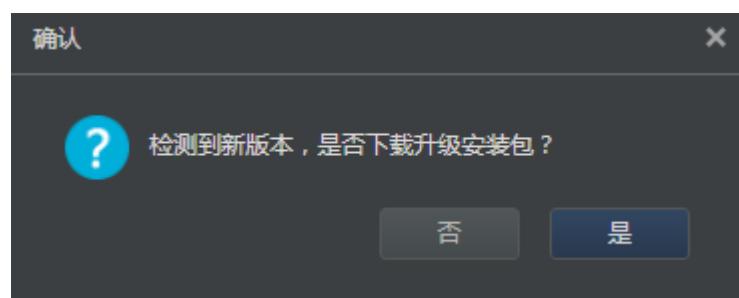
6.2.4 版本更新

检测最新版本有自动检测及手动检测两种方式：

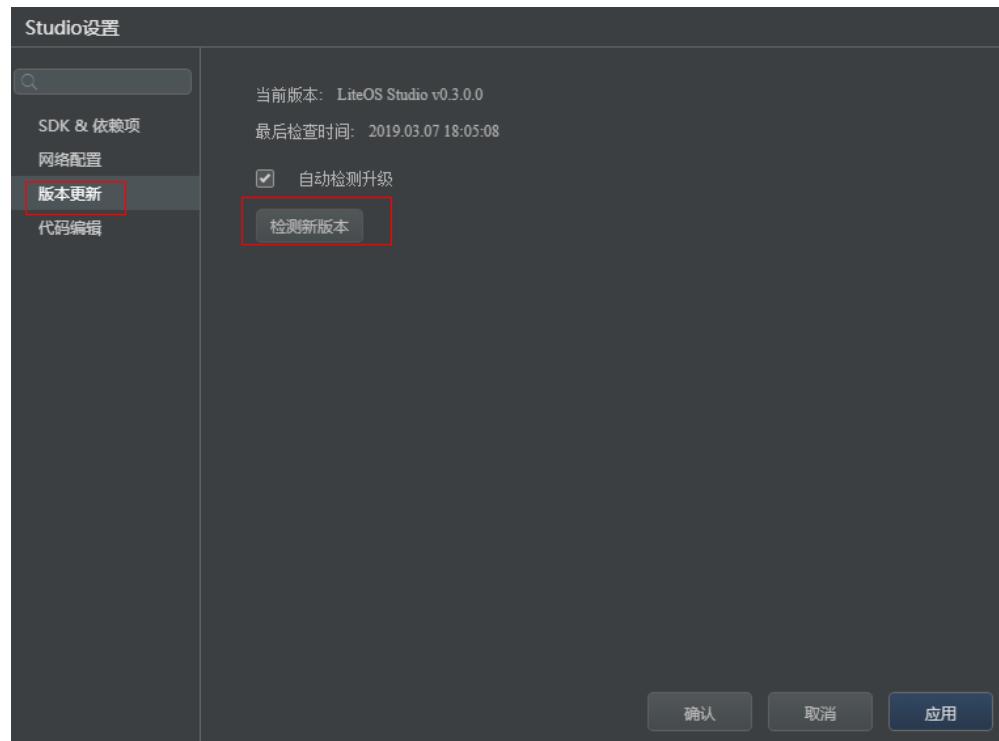
- 勾选“自动检测升级”，单击“应用”，LiteOS Studio自动检测新版本。



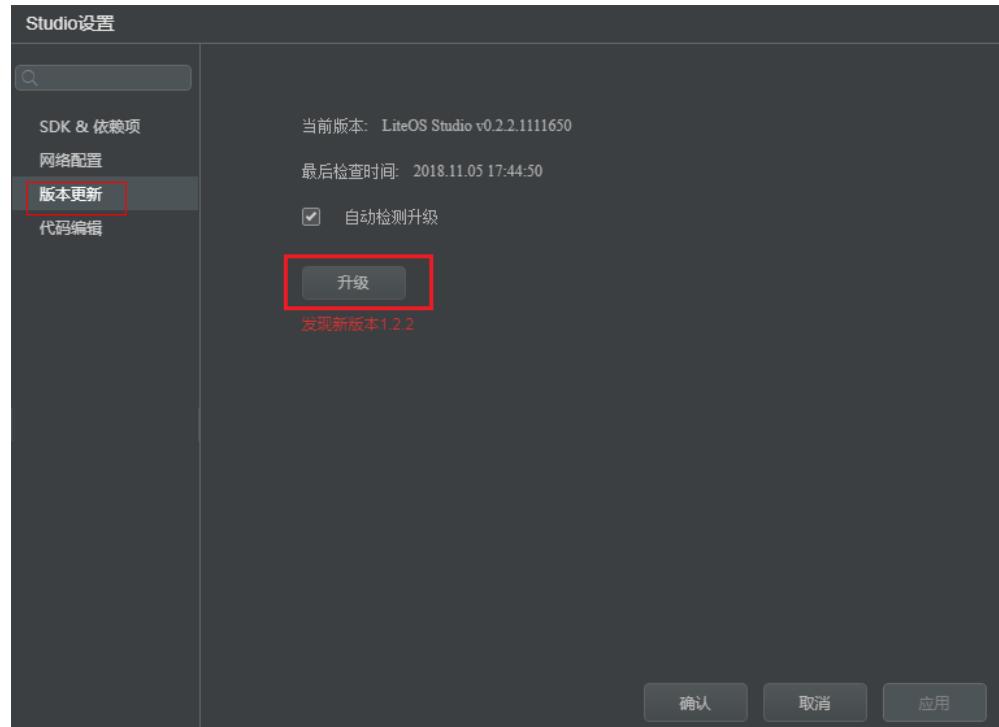
当检测到新版本时，弹出升级提示，按照弹框提示进行升级。



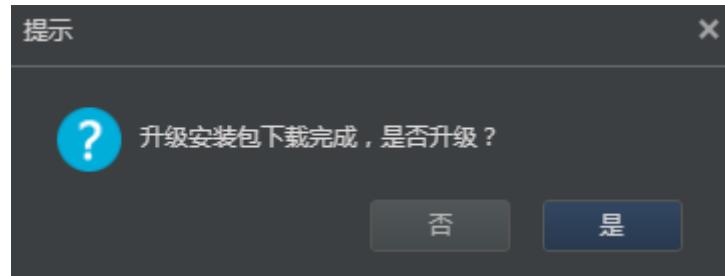
- 单击“检测新版本”，手动检测最新版本。



当检测到新版本时，单击“升级”。

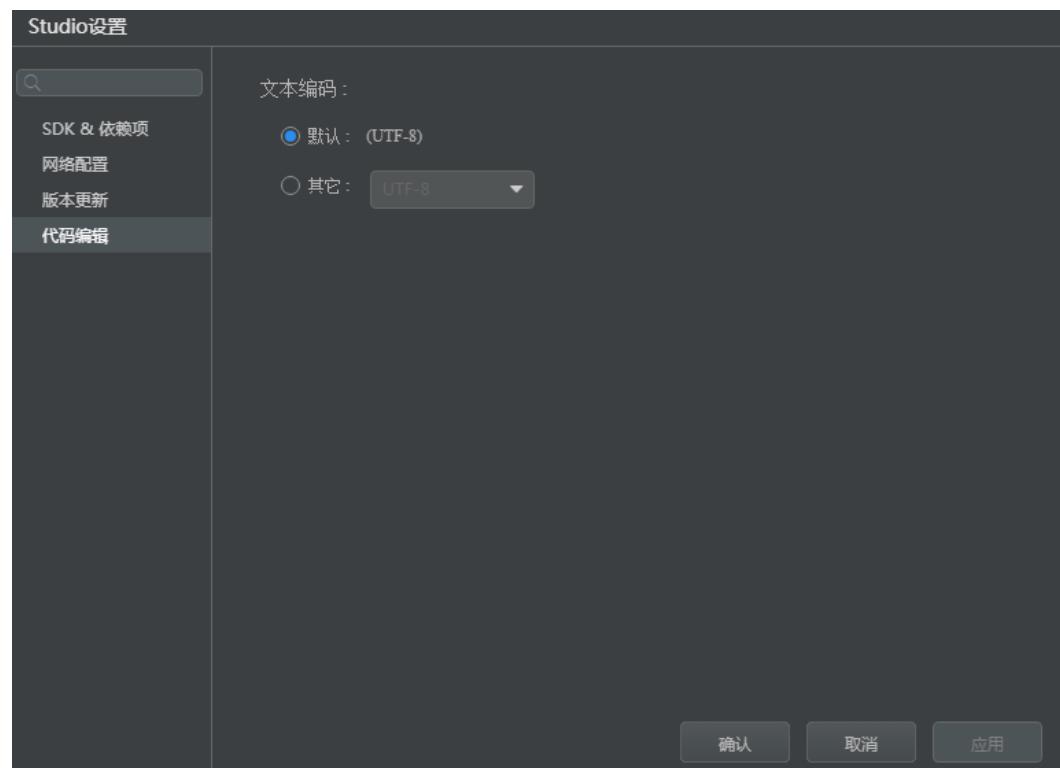


自动下载升级安装包，安装包下载完成后，单击“是”进行升级。



6.2.5 代码编辑

根据需要设置文本编码格式，默认文本编码为“UTF-8”。



6.3 代码编辑

6.3.1 语法高亮

支持语法高亮显示。

```
44 #if defined WITH_AT_FRAMEWORK
45 #include "at_api_interface.h"
46 #if defined USE_NB_NEUL95
47 #include "los_nb_api.h"
48 #endif
49 #endif
50 #ifdef SUPPORT_DTLS_SRV
51 #include "test_dtls_server.h"
52 #endif
53 UINT32 g_TskHandle;
54
55 void USART3_UART_Init(void);
56 VOID HardWare_Init(VOID)
57 {
58     SystemClock_Config();
59     Debug_USART1_UART_Init();
60     hal_rng_config();
61     dwt_delay_init(SystemCoreClock);
62 }
63 int32_t nb_data_rcv_handler(void *arg, int8_t *buf, int32_t len);
64
65 int32_t nb_cmd_match(const char *buf, char* featurestr,int len)
66 {
67     printf("buf:%s feature:%s\n",buf,featurestr);
68     return strncmp((const char * )(buf+2),featurestr,len);
69 }
```

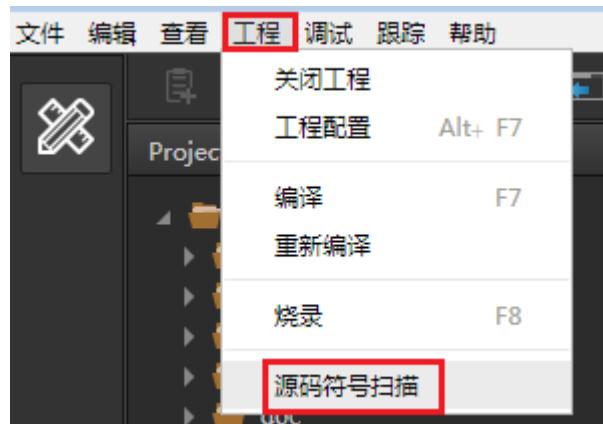
6.3.2 智能提示/自动补齐

输入关键字或作用域里的属性及方法，即可触发智能提示，选择其中一项，补全代码。支持代码模糊匹配。

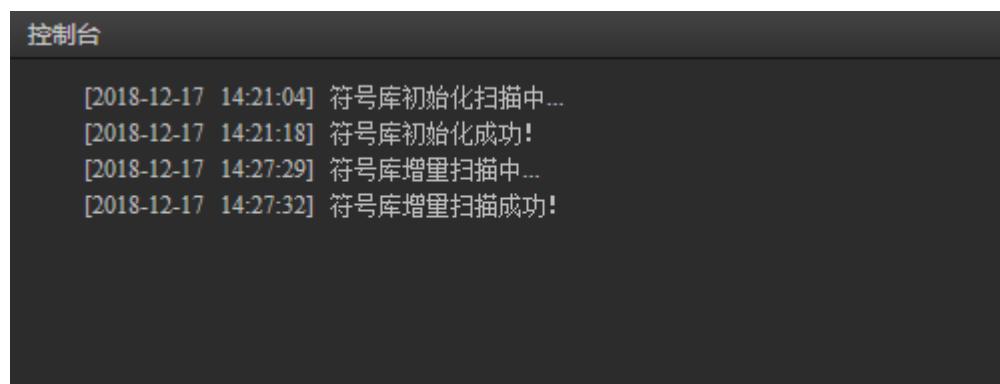
```
65     task_init_param.usTaskPrio = 15;
66     task_init_param.pcName = "helloworld_task";
67     task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)helloworld_task;
68     task_init_param.uwStackSize = 0x1000;
69     task_init_param.uw
70     uwRet = LOS_TaskCr[?] uwRet (main.c 62, targets\Demo_STM32F429IGTx_FI... ⓘ
71     if(LOS_OK != uwRet[?] uart_write
72     {
73         return uwRet;
74     }
75     return uwRet;
76 }
77
78 int main(void)
79 {
80     UINT32 uwRet = LOS[?] udp_teardown
81     HardWare_Init();[?] upper_layers_down
```

6.3.3 源码符号扫描

打开工程时，Studio自动进行符号库扫描，您也可以单击“工程>源码符号扫描”进行符号库增量扫描。

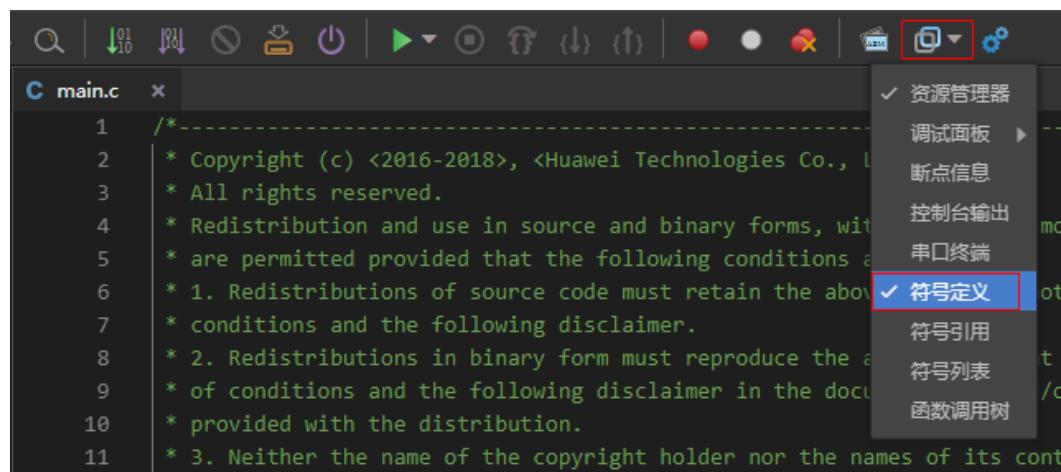


扫描成功如下图所示：



6.3.4 符号定义跳转

单击工具栏中的下拉框，勾选“符号定义”（或单击菜单栏“查看>符号定义”），打开符号定义面板。



- 在代码编辑区，鼠标单击某个符号时，“符号定义”面板上显示该符号定义的所有代码段。或按住Ctrl+鼠标左键单击某个符号时，如为单个定义，可直接打开定义该符号的文件。
- 如有多个定义，双击列表中的结果，打开定义该符号的文件。

The screenshot shows the LiteOS Studio interface. The Project Explorer on the left lists various project components like Cloud_STM32F429ITx_FW, Drivers, EWARM, GCC, etc. The main.c file is currently selected in the code editor. In the editor, the 'SystemClock_Config' symbol is highlighted with a red box. The 'Symbol Definition' panel at the bottom displays the code for 'SystemClock_Config', which includes definitions from multiple files: targets\Cloud_STM32F429ITx_FW\Inc\sys_init.h, targets\Cloud_STM32F429ITx_FW\src\main.c, and targets\Cloud_STM32F429ITx_FW\src\hal\hw.h.

```
#include "at_apl_interface.h"
#ifndef USE_NB_NEUL95
#include "los_nb_api.h"
#endif
UINT32 g_TskHandle;

void USART3_UART_Init(void);
VOID Hardware_Init(VOID)
{
    SystemClock_Config();
    Debug_USART1_UART_Init();
    hal_rng_config();
    dwt_delay_init(SystemCoreClock);
}

extern int32_t nb_data_ioctl(void* arg,int8_t * buf, int32_t len);

VOID main_task(VOID)
{
#if defined(WITH_LINUX) || defined(WITH_LWIP)
    hieth_hw_init();
    net_init();
#elif defined(WITH_AT_FRAMEWORK) && defined(USE_NB_NEUL95)
#define AT_DTLS
#endif
    sec_param_s sec;
    sec.pskid = "868744031131026";
    sec.psk = "d1e1be0c05ac5b8c78ce196412f0cdb0";
#endif
    printf("\r\n=====");
}
```

```
SystemClock_Config targets\Cloud_STM32F429ITx_FW\Inc\sys_init.h (82)
Syst...targets\Cloud_STM32F429ITx_FW\src\main.c (82)
Syst...targets\Cloud_STM32F429ITx_FW\src\hal\hw.h (64)
Syst...targets\Cloud_STM32F429ITx_FW\src\hal\hw.h (134)
76 #ifdef __cplusplus
77 | extern "C" {
78 | #endif
79 void net_init(void);
80 uint32_t HAL_GetTick(void);
81 void SystemClock_Config(void);
82 void _Error_Handler(char *, int);
83 void hieth_hw_init(void);
84 #define Error_Handler() _Error_Handler(__FILE__, __LINE__)
85
86
```

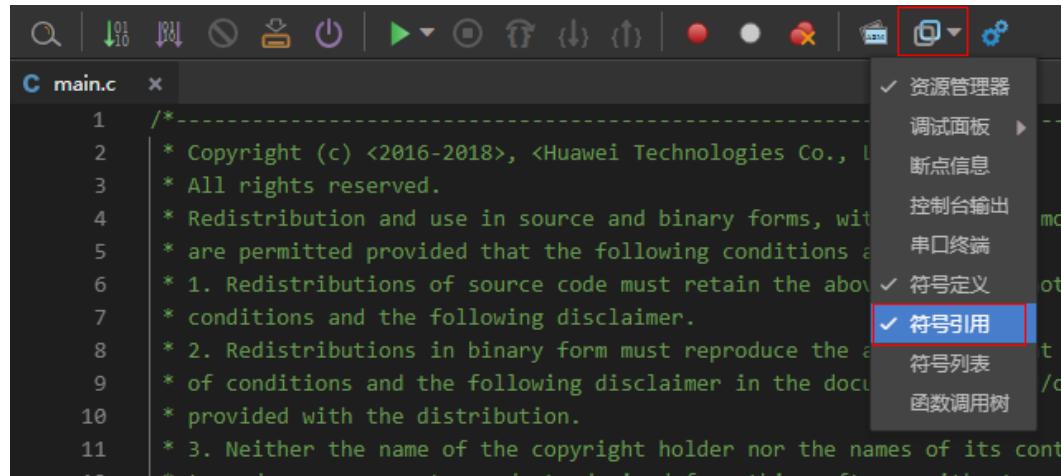
说明

您也可以右键单击符号处，选择“跳转到定义”：

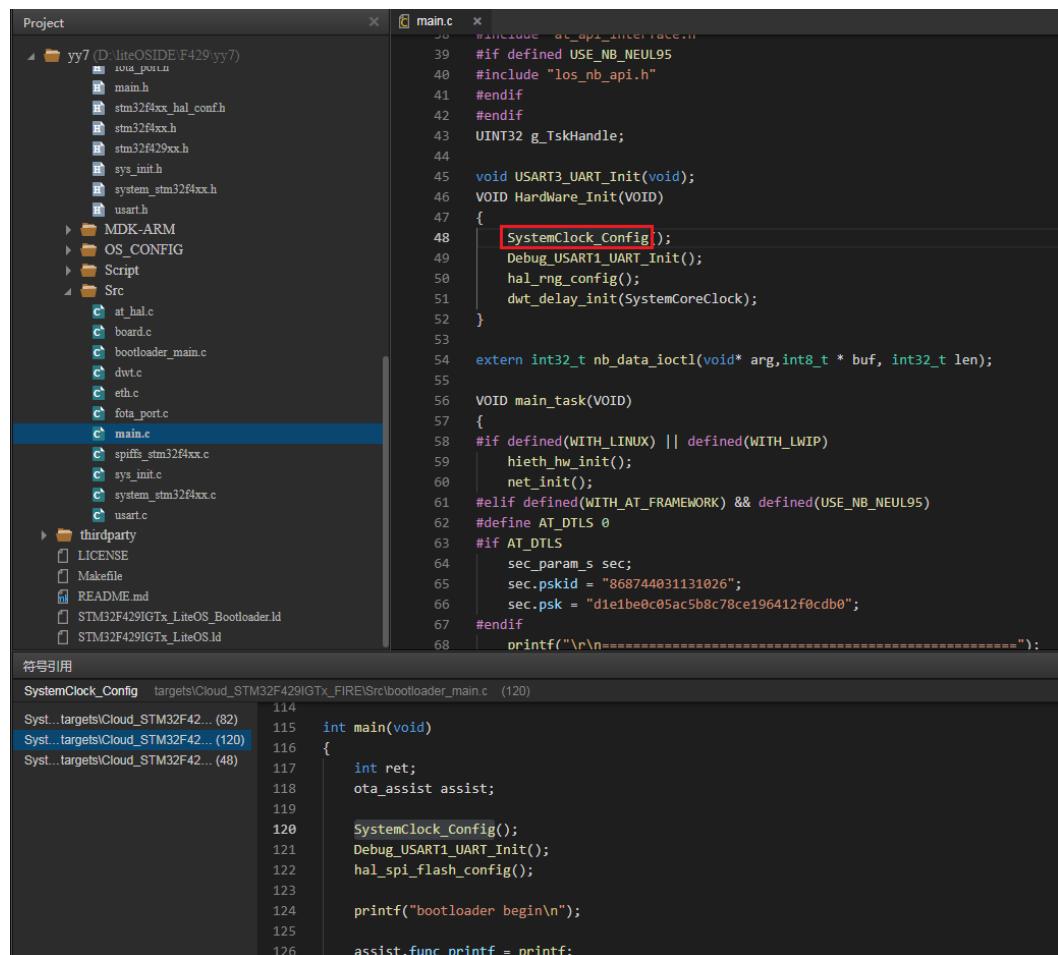
- 如有单个定义，打开对应的文件并跳转到定义该符号的代码行。
- 如有多个定义，打开“符号定义”面板，显示该符号定义的所有代码段。

6.3.5 符号引用跳转

单击工具栏中的下拉框，勾选“符号引用”（或单击菜单栏“查看>符号引用”），打开符号引用面板。



- 在代码编辑区，鼠标单击某个符号时，“符号引用”面板上显示该符号引用的所有代码段。
- 如有多个引用，双击列表中的结果，打开引用该符号的文件。



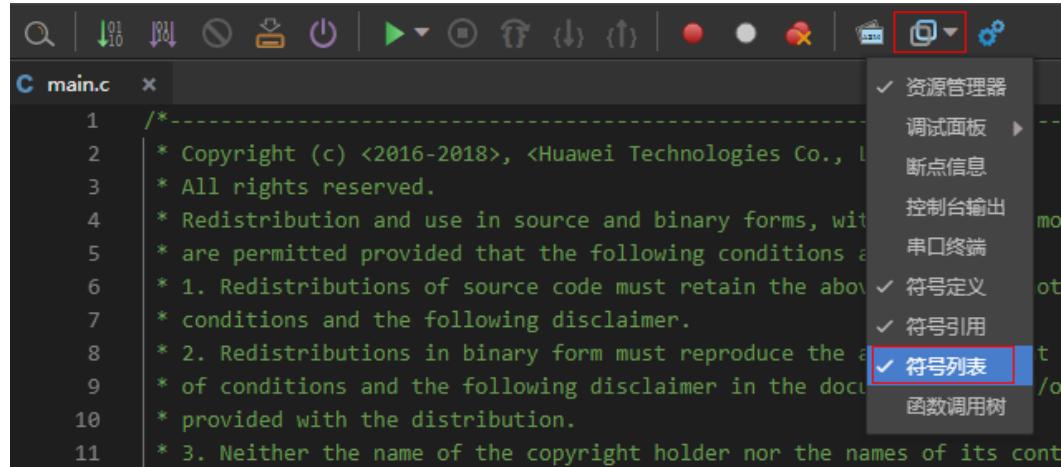
说明

您也可以右键单击符号处，选择“跳转到引用”：

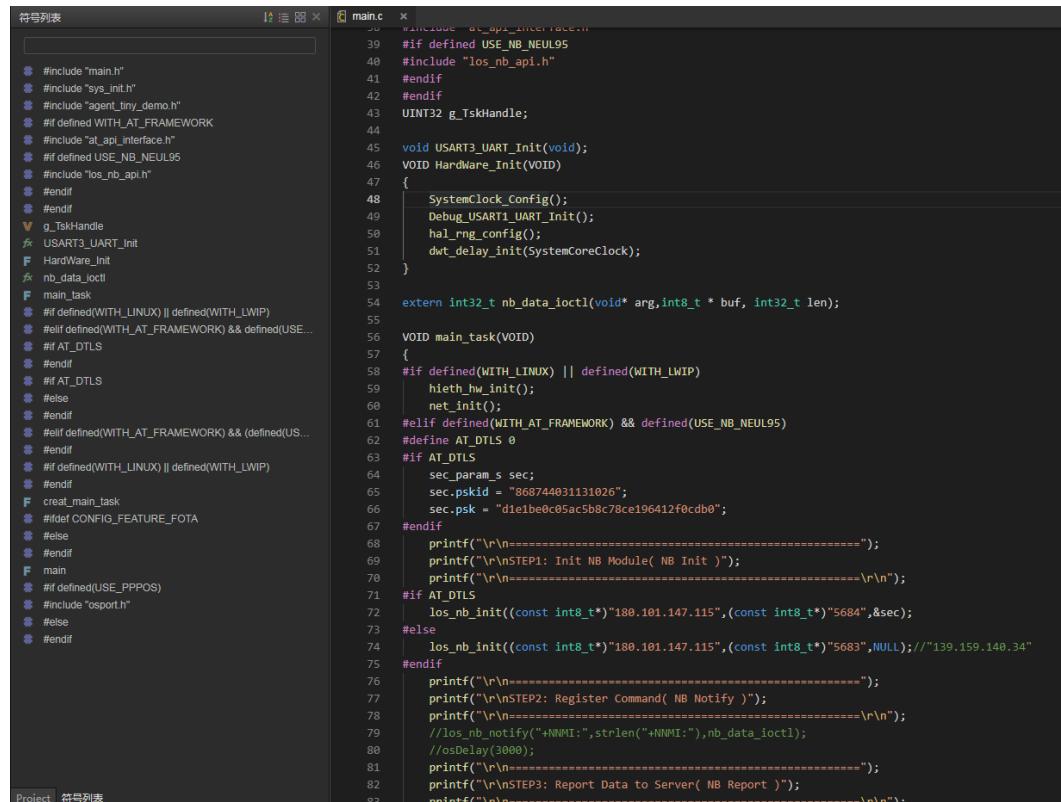
- 如有单个引用，打开对应的文件并跳转到引用该符号的代码行。
- 如有多个引用，打开“符号引用”面板，显示该符号引用的所有代码段。

6.3.6 符号列表查看

单击工具栏中的  下拉框，勾选“符号列表”（或单击菜单栏“查看>符号列表”），打开符号列表面板。

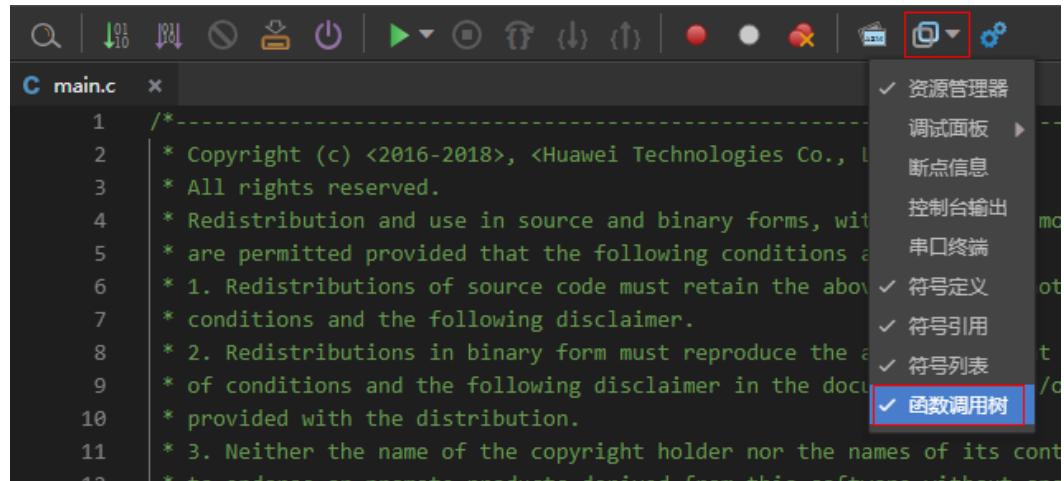


“符号列表”面板中显示当前打开的文件中所有的符号，您可以选择“按字母排序”，“按行号排序”或“按类型排序”，对符号进行排序显示。

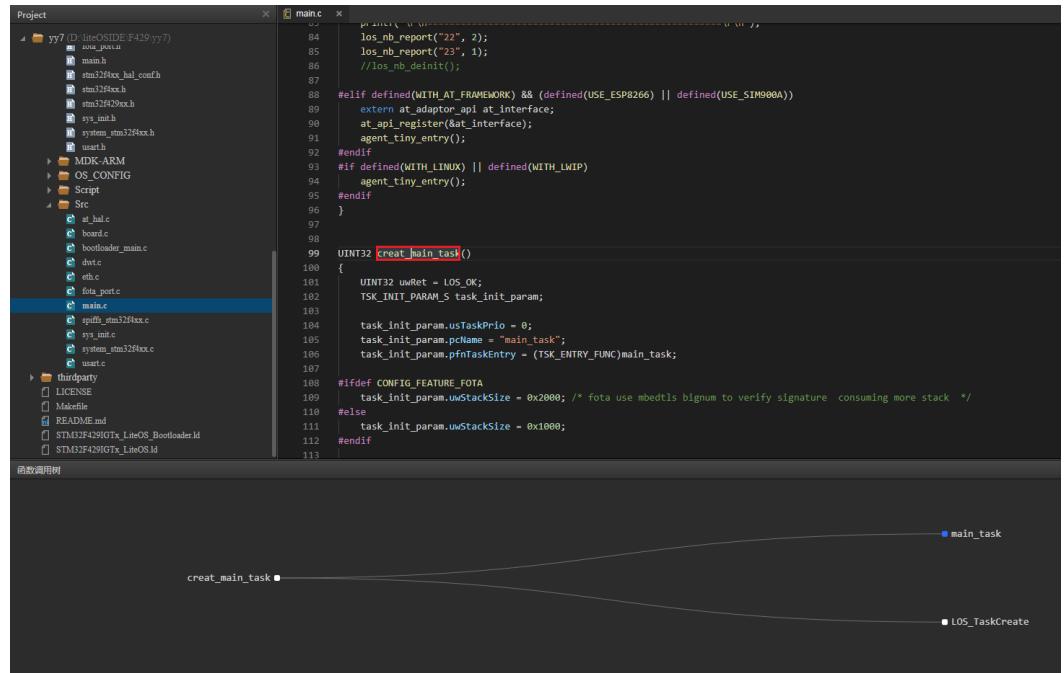


6.3.7 函数调用树查看

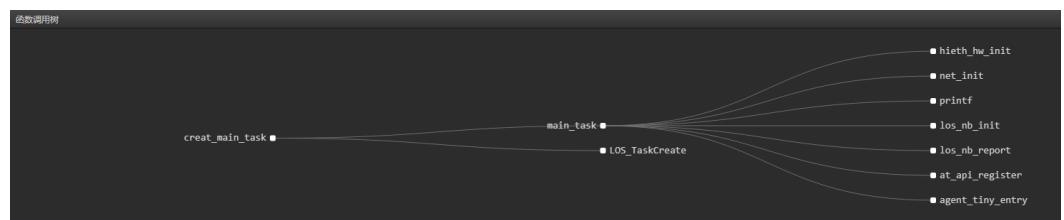
单击工具栏中的  下拉框，勾选“函数调用树”（或单击菜单栏“查看>函数调用树”），打开函数调用树面板。



在代码编辑区，鼠标单击某个函数时，“函数调用树”面板上显示调用该函数的第一层树形结构。



单击树形结构中的蓝色图标可以向下展开，如下所示，最多显示4层树形结构。



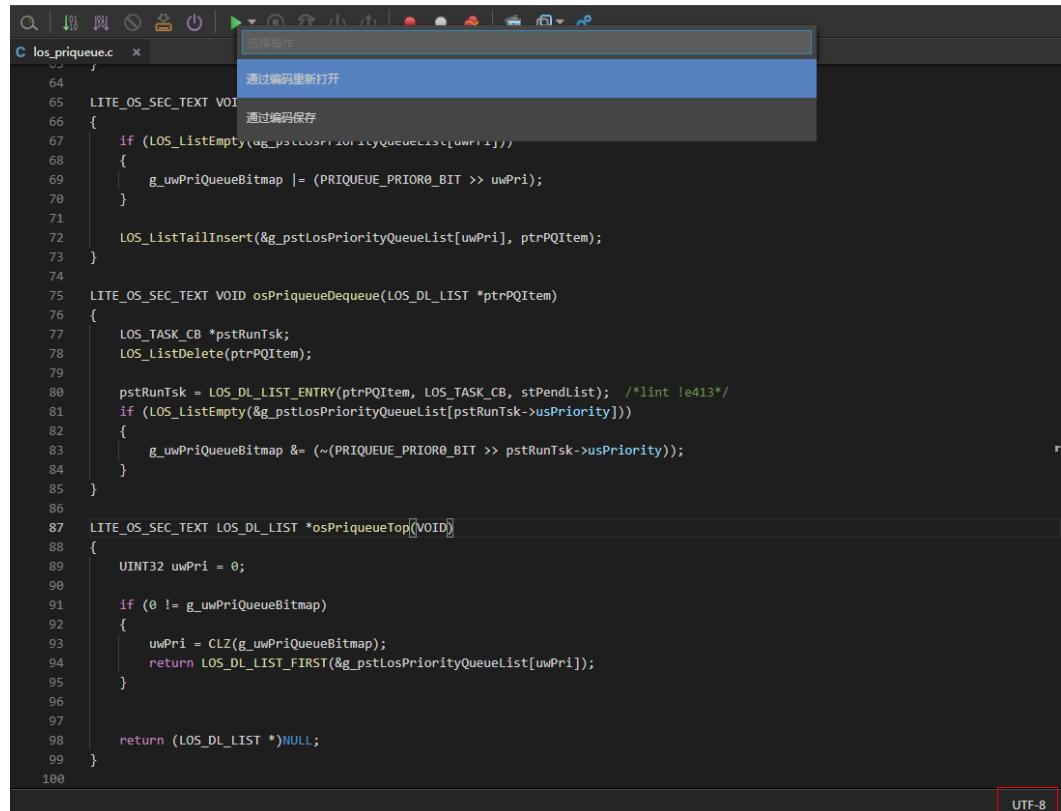
6.3.8 文本编码切换

在当前打开的文件下，单击右下角状态栏的编码名称，如“UTF-8”，您可以进行如下操作：

- 通过编码重新打开：选择一种编码格式打开该文件。
- 通过编码保存：选择一种编码格式保存该文件。

说明

如需更改文件的默认编码格式，单击菜单栏中的“文件>首选项”，在“代码编辑”界面，勾选“其它”，选择其中一种编码格式。

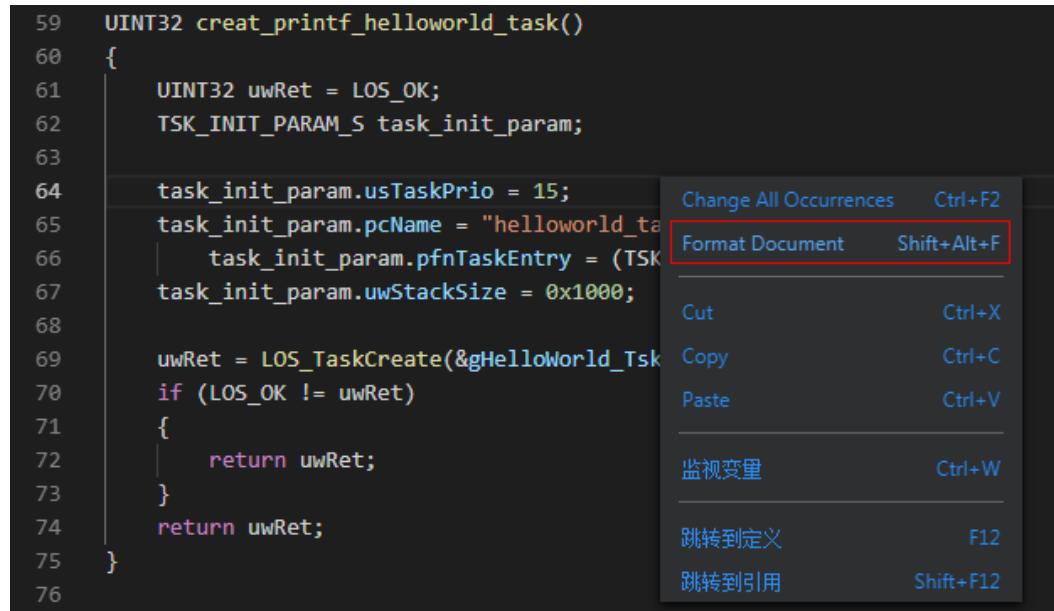


The screenshot shows the LiteOS Studio interface with a code editor window open. The file being edited is 'los_priqueue.c'. A context menu is displayed at the top of the editor, with the 'UTF-8' option highlighted. Other options visible in the menu include '通过编码重新打开' (Open with Encoding) and '通过编码保存' (Save with Encoding).

```
1 // 
2 // 
3 // 
4 // 
5 // 
6 // 
7 // 
8 // 
9 // 
10 // 
11 // 
12 // 
13 // 
14 // 
15 // 
16 // 
17 // 
18 // 
19 // 
20 // 
21 // 
22 // 
23 // 
24 // 
25 // 
26 // 
27 // 
28 // 
29 // 
30 // 
31 // 
32 // 
33 // 
34 // 
35 // 
36 // 
37 // 
38 // 
39 // 
40 // 
41 // 
42 // 
43 // 
44 // 
45 // 
46 // 
47 // 
48 // 
49 // 
50 // 
51 // 
52 // 
53 // 
54 // 
55 // 
56 // 
57 // 
58 // 
59 // 
60 // 
61 // 
62 // 
63 // 
64 // 
65 LITE_OS_SEC_TEXT VOID 
66 { 
67     if ((LOS_ListEmpty(&g_pstLosPriorityQueueList[uwPri]))) 
68     { 
69         g_uwPriQueueBitmap |= (PRIQUEUE_PRIORITY_BIT >> uwPri); 
70     } 
71     LOS_ListTailInsert(&g_pstLosPriorityQueueList[uwPri], ptrPQItem); 
72 } 
73 } 
74 
75 LITE_OS_SEC_TEXT VOID osPriqueueDequeue(LOS_DL_LIST *ptrPQItem) 
76 { 
77     LOS_TASK_CB *pstRunTsk; 
78     LOS_ListDelete(ptrPQItem); 
79 
80     pstRunTsk = LOS_DL_LIST_ENTRY(ptrPQItem, LOS_TASK_CB, stPendList); /*lint le413*/ 
81     if (LOS_ListEmpty(&g_pstLosPriorityQueueList[pstRunTsk->usPriority])) 
82     { 
83         g_uwPriQueueBitmap &= (~(PRIQUEUE_PRIORITY_BIT >> pstRunTsk->usPriority)); 
84     } 
85 } 
86 
87 LITE_OS_SEC_TEXT LOS_DL_LIST *osPriqueueTop[VOID] 
88 { 
89     UINT32 uwPri = 0; 
90 
91     if (0 != g_uwPriQueueBitmap) 
92     { 
93         uwPri = CLZ(g_uwPriQueueBitmap); 
94         return LOS_DL_LIST_FIRST(&g_pstLosPriorityQueueList[uwPri]); 
95     } 
96 
97     return (LOS_DL_LIST *)NULL; 
98 } 
99 } 
100 
```

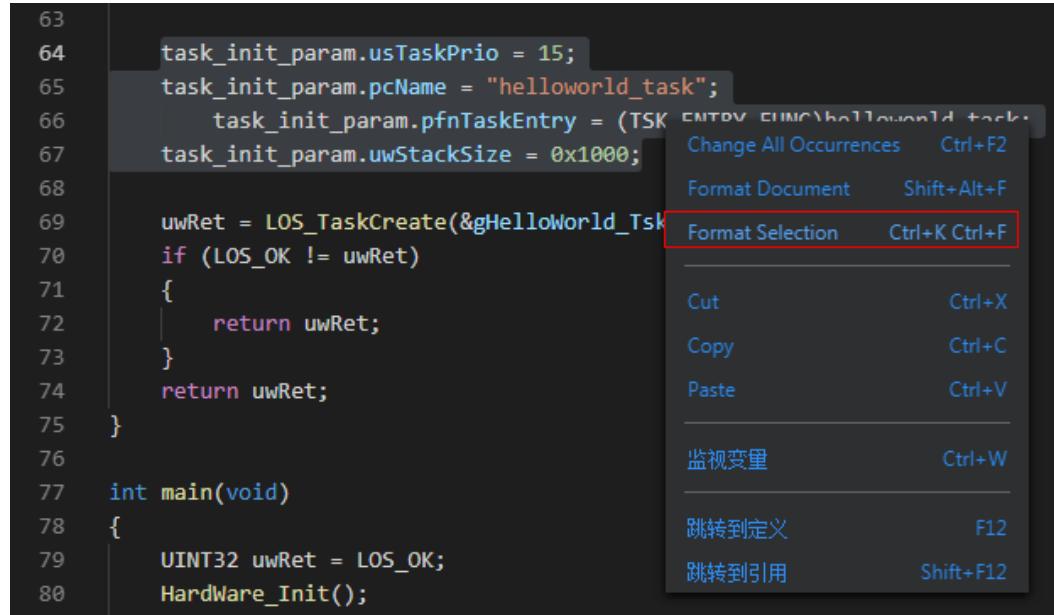
6.3.9 格式自动对齐

在代码编辑区，单击右键选择“Format Document”或按快捷键“Shift+Alt+F”，当前文件的源码格式可自动对齐。



```
59     UINT32 creat_printf_helloworld_task()
60     {
61         UINT32 uwRet = LOS_OK;
62         TSK_INIT_PARAM_S task_init_param;
63
64         task_init_param.usTaskPrio = 15;
65         task_init_param.pcName = "helloworld_task";
66         task_init_param.pfnTaskEntry = (TSK
67         task_init_param.uwStackSize = 0x1000;
68
69         uwRet = LOS_TaskCreate(&gHelloWorld_Tsk
70         if (LOS_OK != uwRet)
71         {
72             return uwRet;
73         }
74         return uwRet;
75     }
76 }
```

选中需要格式化的代码，单击右键选择“Format Selection”或按快捷键“Ctrl+K Ctrl+F”，选中代码自动对齐。



```
63     task_init_param.usTaskPrio = 15;
64     task_init_param.pcName = "helloworld_task";
65     task_init_param.pfnTaskEntry = (TSK
66     task_init_param.uwStackSize = 0x1000;
67
68     uwRet = LOS_TaskCreate(&gHelloWorld_Tsk
69     if (LOS_OK != uwRet)
70     {
71         return uwRet;
72     }
73     return uwRet;
74 }
75
76 int main(void)
77 {
78     UINT32 uwRet = LOS_OK;
79     HardWare_Init();
```

6.3.10 视图单步前进/后退

在当前视图下，单击工具栏的快速回到上一次光标放置处，单击工具栏的快速回到下一次光标放置处。若有多个视图，在多个视图间切换。

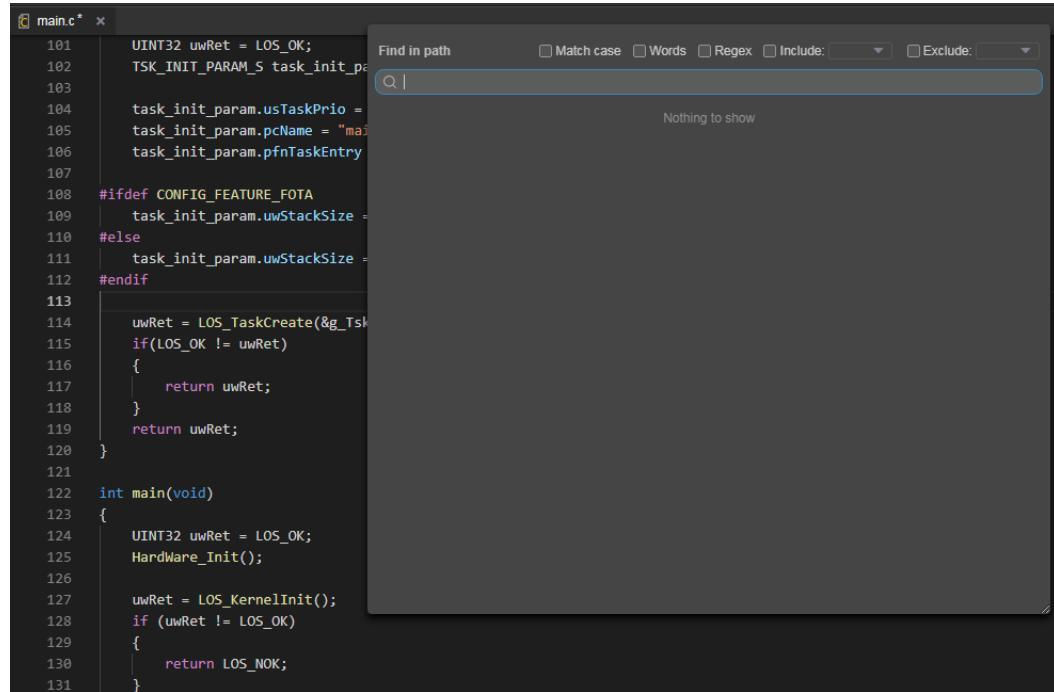


您可以单击菜单栏中的“编辑”，选择“后退”回到上一次光标放置处，选择“前进”回到下一次光标放置处。

6.3.11 全局搜索

操作步骤

步骤1 在当前打开的工程下，单击菜单栏中的“编辑>全局搜索”（或单击工具栏的 或按“Ctrl+Shift+F”快捷键），打开全局搜索框。



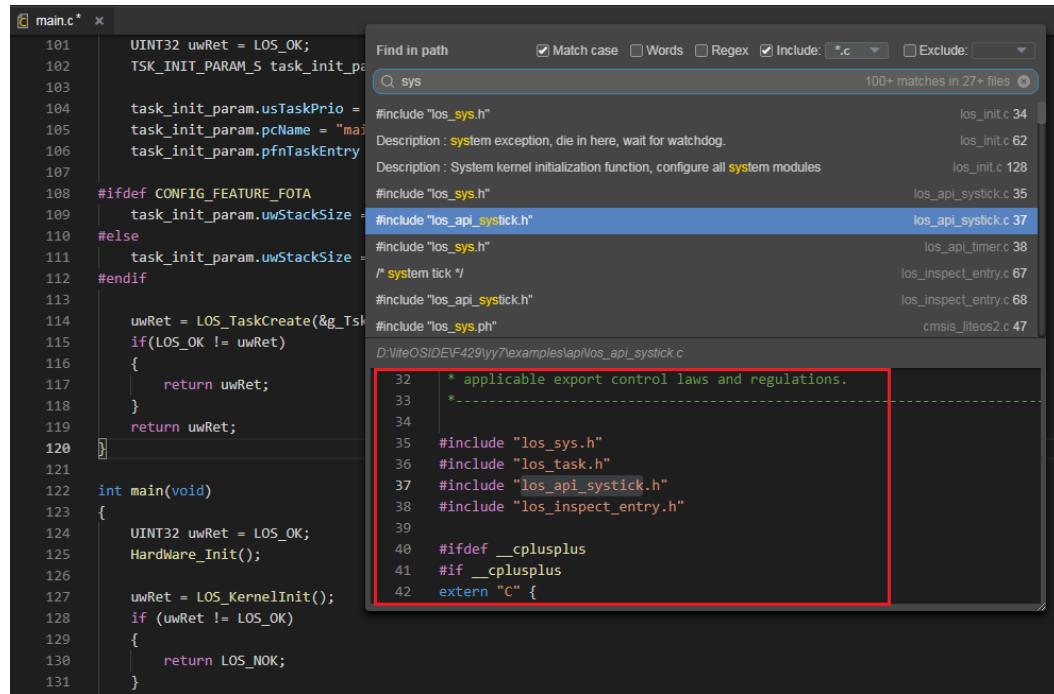
步骤2 在搜索框中输入需要查找的字符串或正则表达式，搜索结果实时显示。

说明

可勾选搜索框上方复选框进行条件搜索：

- 勾选“Match case”后，搜索关键字区分大小写，搜索出的结果按关键字模糊匹配。
- 勾选“Words”后，搜索关键字完全匹配，搜索出的结果按关键字精确匹配。
- 勾选“Regex”后，搜索关键字对应的正则表达式。
- 勾选“Include”后，在下拉框中选择搜索的文件类型。
- 勾选“Exclude”后，在下拉框中选择搜索需去除的文件类型。

步骤3 单击搜索的结果，在下方的方框中显示搜索结果的前后文，双击搜索结果，打开对应的文件并定位到关键字所在行。



----结束

6.3.12 当前文件代码查找/替换

在当前打开的文件下，单击菜单栏中的“编辑>文件内搜索”（或按“Ctrl+F”快捷键）打开搜索框，在右上角搜索框中输入要搜索的内容，相应内容将高亮显示。

说明

可单击搜索框右侧图标进行条件搜索：

- 选中 后，搜索关键字区分大小写，搜索出的结果按关键字模糊匹配。
- 选中 后，搜索关键字完全匹配，搜索出的结果按关键字精确匹配。
- 选中 后，搜索关键字对应的正则表达式。

The screenshot shows the main code editor window with a file named 'main.c'. In the top right corner, there is a search bar with the text 'uw'. A red box highlights the search results list on the right side of the interface, which shows multiple file entries related to 'uw'.

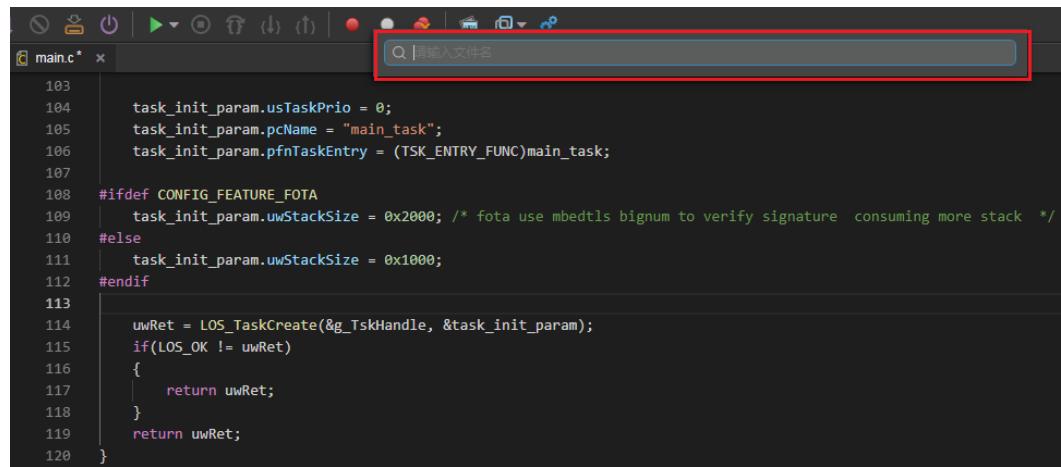
单击搜索框左边的 切换为替换模式，填入替换后的内容，选择替换或全部替换。



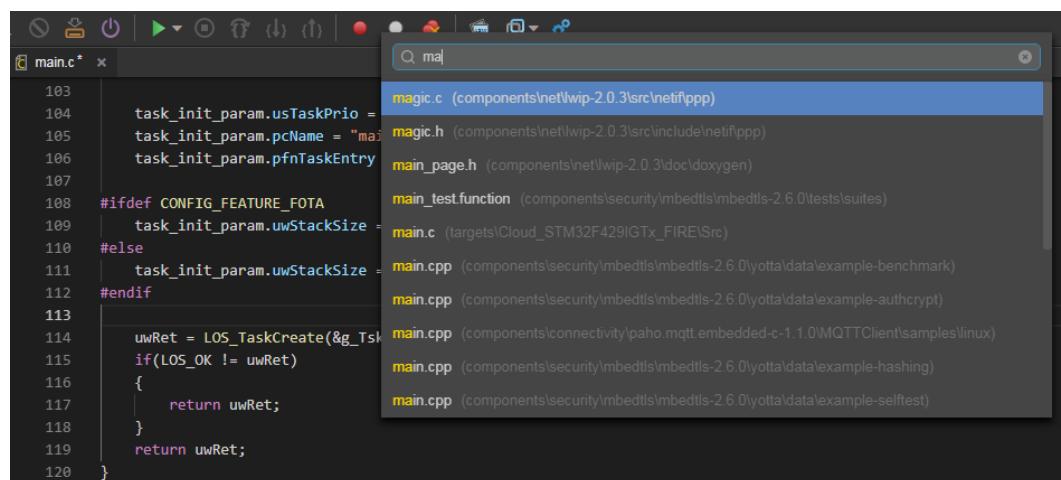
```
main.c * x
103     task_init_param.usTaskPrio = 0;
104     task_init_param.pcName = "main_task";
105     task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)main_task;
106
107 #ifdef CONFIG_FEATURE_FOTA
108     task_init_param.uwStackSize = 0x2000; /* fota use mbedtls bignum to verify signature consuming more stack */
109 #else
110     task_init_param.uwStackSize = 0x1000;
111 #endif
112
113     uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);
114     if(LOS_OK != uwRet)
115     {
116         return uwRet;
117     }
118     return uwRet;
119 }
120
121 int main(void)
122 {
123     UINT32 uwRet = LOS_OK;
124     Hardware_Init();
125 }
```

6.3.13 文件搜索

在当前打开的工程下，单击菜单栏中的“编辑>查找文件”（或单击“Ctrl+P”快捷键），在界面上方弹出文件搜索框。



在搜索框中输入需要搜索的文件名，下拉框中实时显示搜索的结果，搜索结果按关键字模糊匹配。单击文件名，打开对应的文件。



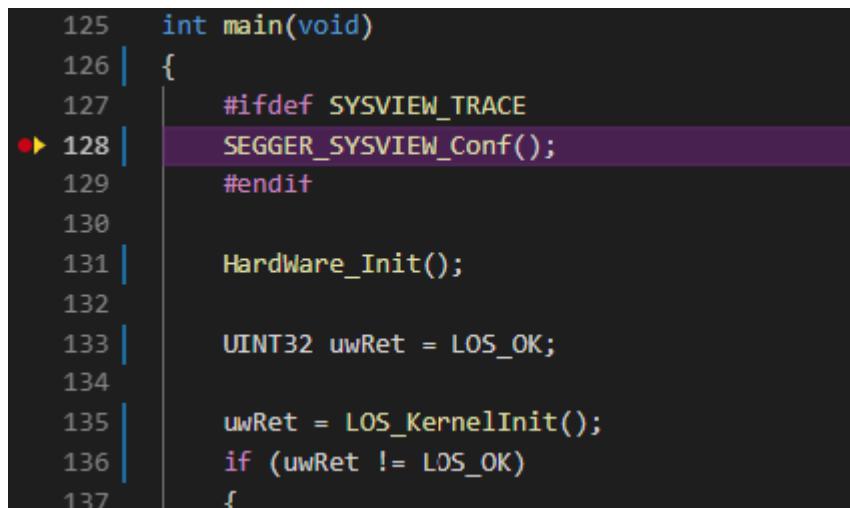
6.4 断点控制

背景信息

- LiteOS Studio会自动保存已有断点，关闭LiteOS Studio后再打开，断点信息依然存在。
- 调试运行时，程序经过有效断点处会停止。
- 中断时对应代码行自动定位，且当前代码行高亮显示。

说明

- 调试运行时，有效代码行有蓝色竖线标示，若对内联函数打断点，可能导致断点插入失败。



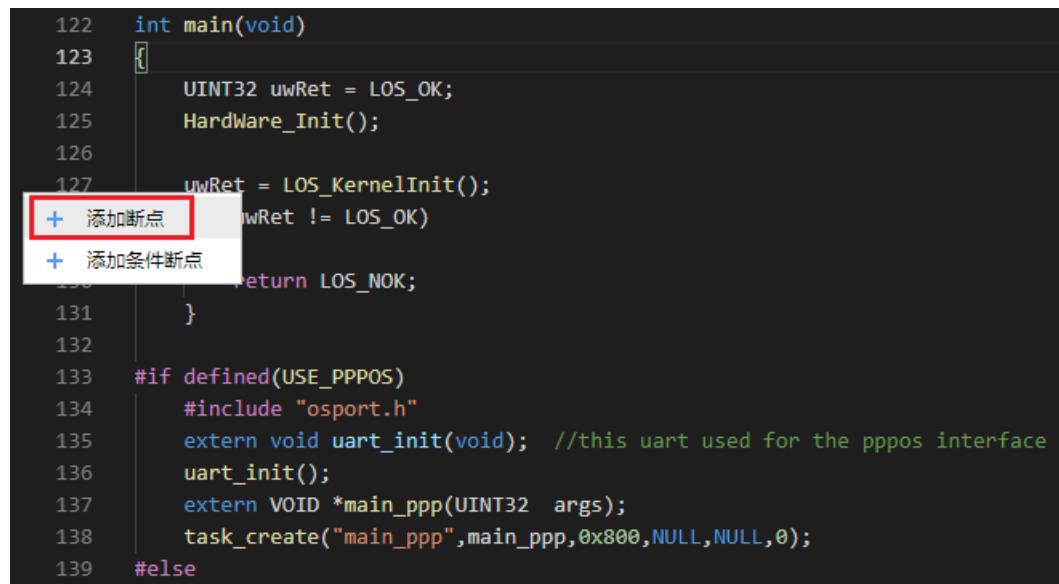
125 int main(void)
126 {
127
128 #ifdef SYSVIEW_TRACE
129 SEGGER_SYSVIEW_Conf();
130 #endif
131 HardWare_Init();
132
133 UINT32 uwRet = LOS_OK;
134
135 uwRet = LOS_KernelInit();
136 if (uwRet != LOS_OK)
137 {

A screenshot of the LiteOS Studio code editor. The code is in a C file named 'main.c'. A red arrow points to line 128, which contains '#ifdef SYSVIEW_TRACE'. This line is highlighted in purple, indicating it is a comment. Line 128 is also marked with a small red dot, indicating it is a valid breakpoint location.

- 程序执行至无效断点时，将跳过无效断点，停止在下一个有效断点处。

6.4.1 新增断点

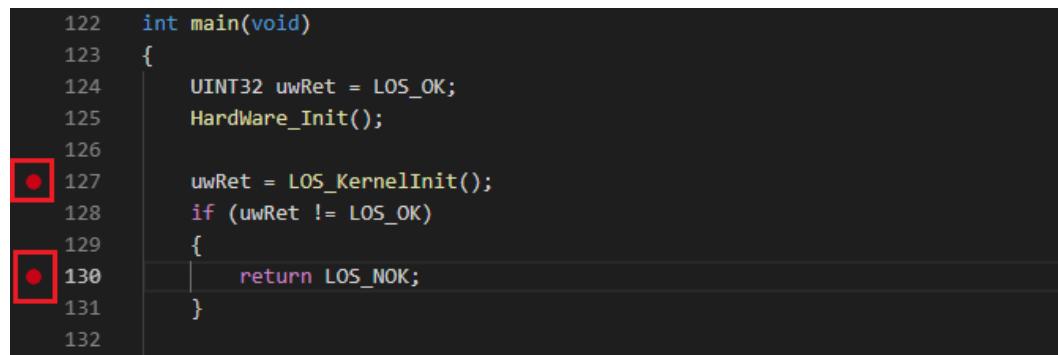
在代码编辑区左侧单击增加断点，或单击右键选择“添加断点”。



或光标放在需要添加断点的所在行，单击工具栏中的添加断点。



新增断点后，图标如下所示：

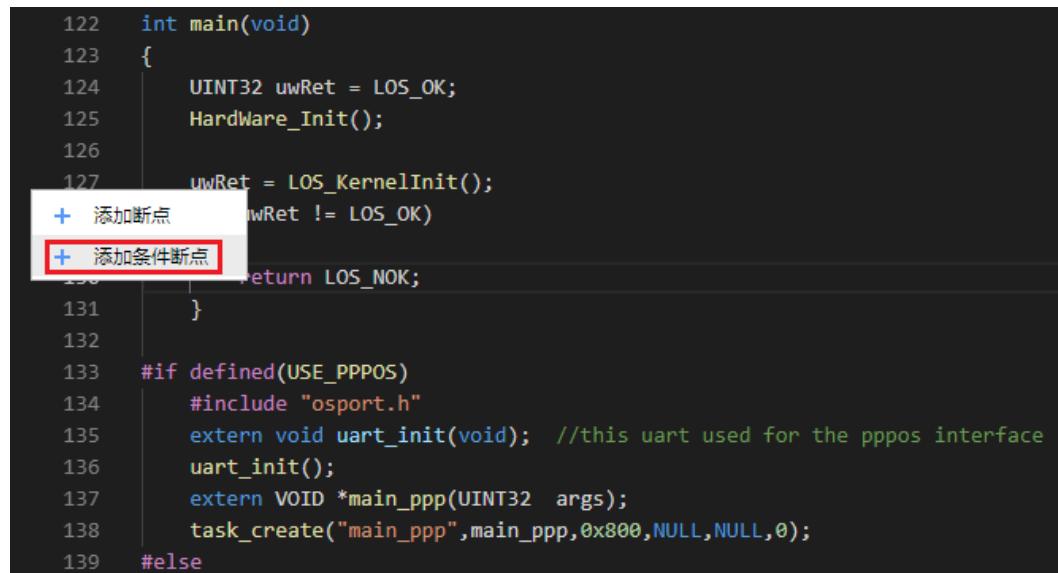


```
122 int main(void)
123 {
124     UINT32 uwRet = LOS_OK;
125     HardWare_Init();
126
127     uwRet = LOS_KernelInit();
128     if (uwRet != LOS_OK)
129     {
130         return LOS_NOK;
131     }
132 }
```

6.4.2 新增条件断点

操作步骤

步骤1 在代码编辑区左侧单击右键选择“添加条件断点”。



```
122 int main(void)
123 {
124     UINT32 uwRet = LOS_OK;
125     HardWare_Init();
126
127     uwRet = LOS_KernelInit();
128     + 添加断点 if (uwRet != LOS_OK)
129     + 添加条件断点 return LOS_NOK;
130 }
131
132
133 #if defined(USE_PPPoS)
134 #include "osport.h"
135 extern void uart_init(void); //this uart used for the pppos interface
136 uart_init();
137 extern VOID *main_ppp(UINT32 args);
138 task_create("main_ppp", main_ppp, 0x800, NULL, NULL, 0);
139 #else
```

步骤2 在弹出的页面中，编辑条件断点。

- 勾选“断点忽略次数”单选框，根据需要设置断点忽略次数。断点忽略次数可以查看某段代码执行特定次数后的堆栈信息。
- 勾选“在表达式计算结果为true时中断”单选框，在文本框中自定义表达式，让断点在满足用户设定条件下停止。
如表达式为 $a > b$ ，当程序中此断点表达式结果为true时，执行中断，结果为false时，忽略此断点。



步骤3 单击“确定”，添加条件断点。鼠标悬停在条件断点处，显示相关条件信息。

新增条件断点后，图标如下所示：

```
122 int main(void)
123 {
124     UINT32 uwRet = LOS_OK;
125     HardWare_Init();
126
127     uwRet = LOS_KernelInit();
128     if (uwRet != LOS_OK)
129     {
130         return LOS_NOK;
131     }
132 }
```

----结束

6.4.3 编辑断点

在代码编辑区，右键单击断点处，选择“编辑断点”。

```
122 int main(void)
123 {
124     UINT32 uwRet = LOS_OK;
125     HardWare_Init();
126
127     uwRet = LOS_KernelInit();
128     if (uwRet != LOS_OK)
129     {
130         return LOS_NOK;
131     }
132
133 #if defined(USE_PPPoS)
134     #include "osport.h"
135     extern void uart_init(void); //this uart used for the pppos interface
136     uart_init();
137     extern VOID *main_ppp(UINT32 args);
138     task_create("main_ppp", main_ppp, 0x800, NULL, NULL, 0);
139 #else
```

在弹出的页面中，进行编辑断点。

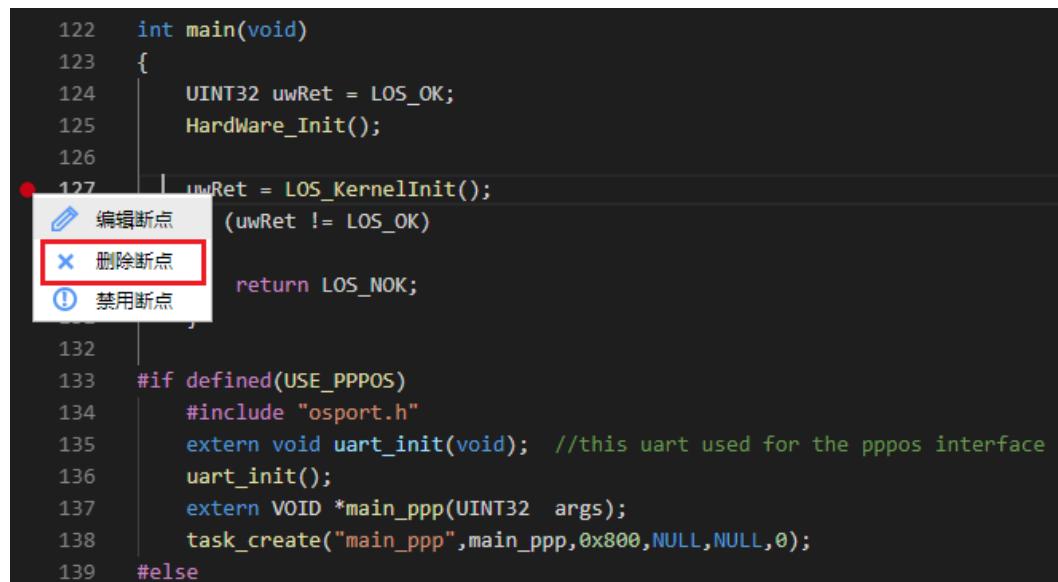
说明

普通断点可通过编辑断点，修改为条件断点。



6.4.4 删除断点

- 代码编辑区：单击断点处可删除断点，或右键单击断点处，选择“删除断点”。

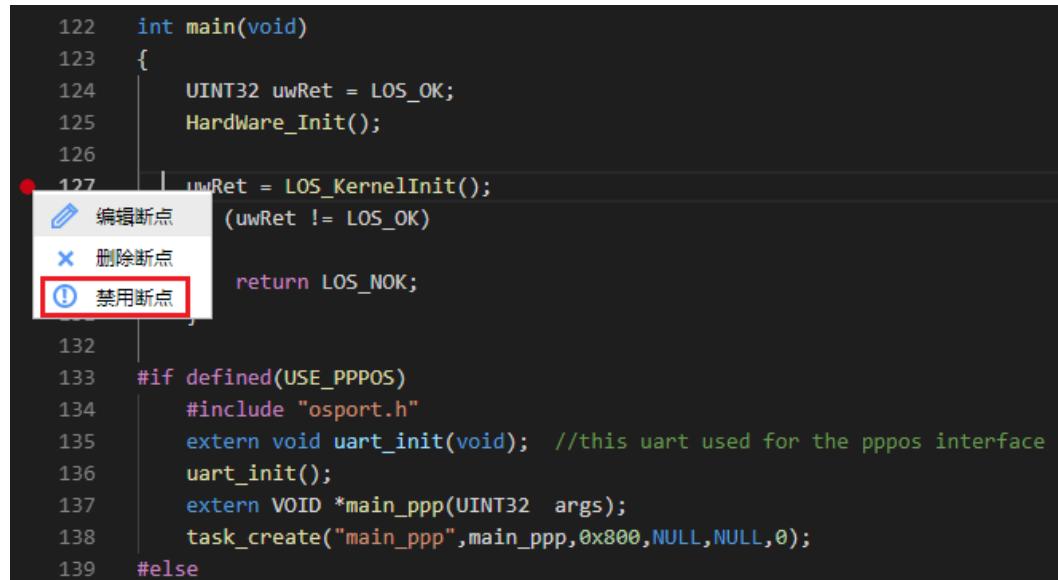


- 单击工具栏中的 删除所有断点。



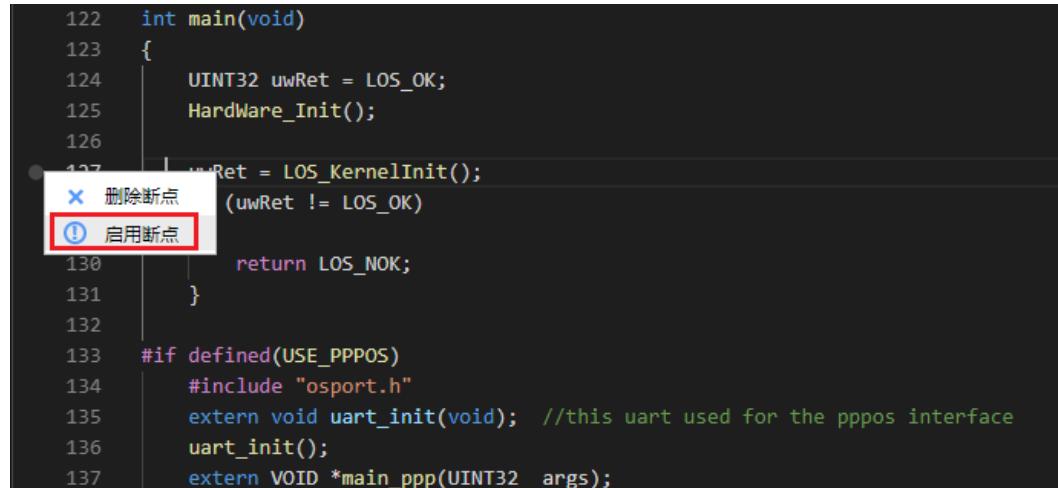
6.4.5 禁用/启用断点

- 代码编辑区：右键单击断点处，选择“禁用断点”，断点置灰并禁用此断点。



```
122 int main(void)
123 {
124     UINT32 uwRet = LOS_OK;
125     HardWare_Init();
126
127     uwRet = LOS_KernelInit();
128     if (uwRet != LOS_OK)
129         return LOS_NOK;
130
131 #if defined(USE_PPPoS)
132     #include "osport.h"
133     extern void uart_init(void); //this uart used for the pppos interface
134     uart_init();
135     extern VOID *main_ppp(UINT32 args);
136     task_create("main_ppp", main_ppp, 0x800, NULL, NULL, 0);
137 #else
```

- 禁用断点后，右键单击此断点处，选择“启用断点”，可重新启用断点。



```
122 int main(void)
123 {
124     UINT32 uwRet = LOS_OK;
125     HardWare_Init();
126
127     uwRet = LOS_KernelInit();
128     if (uwRet != LOS_OK)
129         return LOS_NOK;
130     }
131
132
133 #if defined(USE_PPPoS)
134     #include "osport.h"
135     extern void uart_init(void); //this uart used for the pppos interface
136     uart_init();
137     extern VOID *main_ppp(UINT32 args);
```

- 光标放在断点所在行，单击工具栏中的  禁用/启用断点。



6.5 控制台输出

LiteOS Studio在使用过程中，用户可通过控制台查看程序和系统运行状态，提高开发效率。

控制台输出面板显示如下信息：

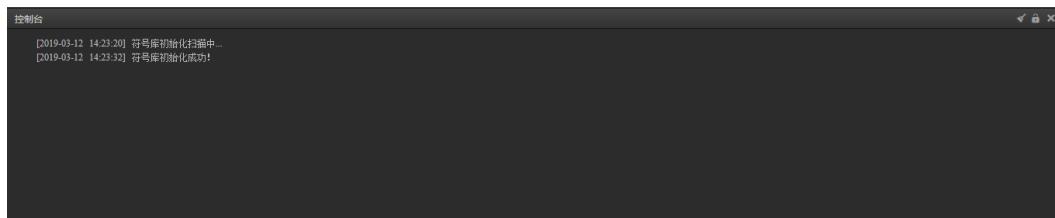
- JLink程序烧录过程中的信息输出至控制台显示。
- 开发板连接或者重启的信息输出至控制台显示。
- 增加或删除断点的信息显示。
- 编译，烧录及调试的信息显示。编译的警告和错误信息若有文件地址及行号可实现跳转，鼠标悬浮在警告或错误信息处出现下划线，按住ctrl+单击左键进行跳转。

- 修改监视的变量值、CPU寄存器值及Memory地址信息显示。
- 导出Memory地址数据信息显示。

操作步骤

在当前打开的工程下，单击工具栏中的下拉框，勾选“控制台输出”，打开控制台输出面板。

如下图所示：



- 单击控制台面板右上角的，可清除控制台输出的日志信息。
- 单击控制台面板右上角的，可锁定当前输出的日志信息位置。
单击可解除锁定状态，有新日志产生时，会自动滚动到最末尾位置。
- 单击控制台面板右上角的，可关闭控制台。

6.6 编译，烧录及调试（非华为海思芯片）

STM32L431xx工程的编译，烧录及调试请参见[5.3 STM32L431xx工程示例](#)章节。

6.6.1 编译（Windows）

在Windows系统中，支持对第三方芯片（非华为海思芯片）工程进行编译，暂不支持对华为海思芯片工程进行编译。

操作步骤

在打开的工程中，单击工具栏中的，对当前工程进行编译。或者单击重新编译，删除上一次编译生成的文件，再次执行编译。



编译成功后，在控制台输出面板中显示“编译成功”。

The screenshot shows the 'Control Console' window of the LiteOS Studio IDE. The log output is as follows:

```
Compiling usart.c...
Compiling dwt.c...
Compiling eth.c...
Compiling sys_init.c...
Compiling user_task.c...
Compiling at_api.c...
Compiling at_main.c...
Compiling at_hal.c...
Compiling esp8266.c...
Compiling sim900a.c...
Compiling bg36.c...
Compiling bc95.c...
Compiling los_nb_api.c...
Compiling los_dispatch_gcc.S...
arm-none-eabi-size build/Huawei_LiteOS.elf
text data bss dec hex filename
362156 1904 8392 372452 5ae4 build/Huawei_LiteOS.elf
arm-none-eabi-objcopy -O ihex build/Huawei_LiteOS.elf build/Huawei_LiteOS.hex
arm-none-eabi-objcopy -O binary -S build/Huawei_LiteOS.elf build/Huawei_LiteOS.bin
0 Error(s), 0 Warning(s).
```

[2019-03-06 18:41:14] 编译成功。 编译耗时: 181117ms

6.6.2 烧录

编译完成后，可将编译生成的二进制文件烧录至开发板。

操作步骤

步骤1 将开发板与电脑连接。

步骤2 单击工具栏中的 ，将已编译的程序烧录至目标板。



控制台输出面板中显示“烧录成功”后，完成烧录。

The screenshot shows the 'Control Panel' window with a log of flash operations. The log includes multiple 'EraseFlash' commands for pages 0x13 to 0x19 at addresses 0x8009000 to 0x800c800, each with a size of 0x800 bytes. It also shows a 'Flash write for F2/F4/L4' operation and a successful completion message: '[2019-03-05 16:49:06] 烧录成功!' (Burn successful!).

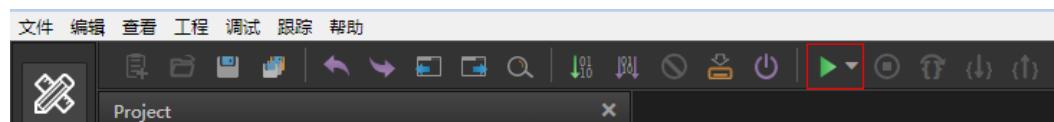
----结束

6.6.3 调试

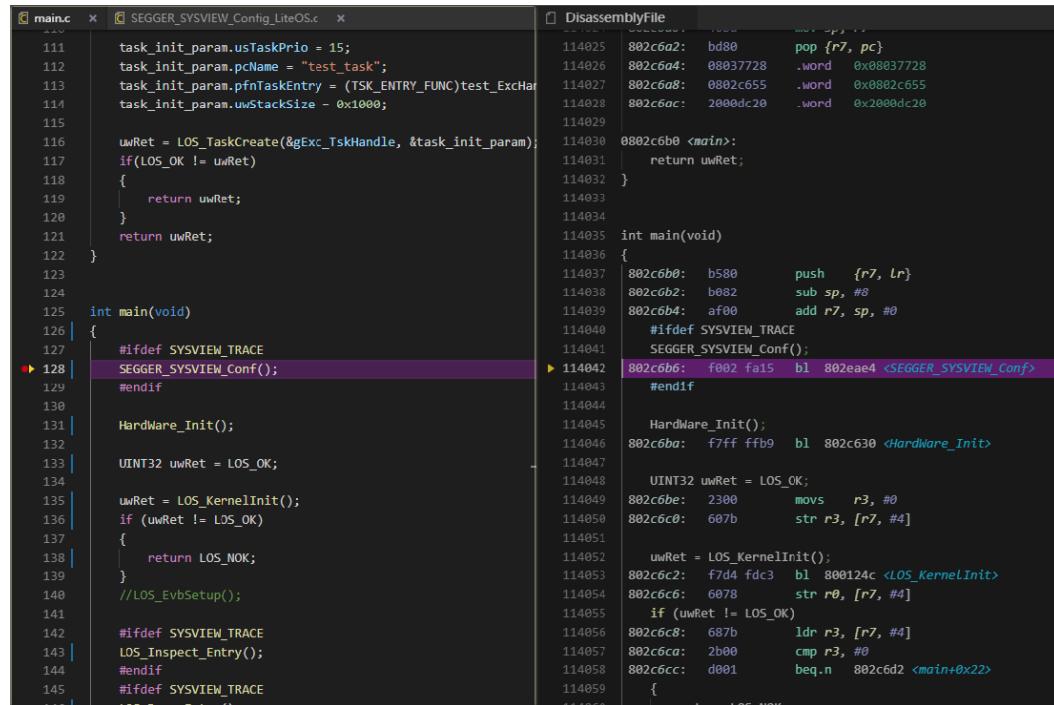
烧录成功后，可对当前工程进行调试。

操作步骤

单击工具栏中的 开始调试。如需更改调试方式，单击 下拉框选择调试方式，调试方式介绍请参见 [表5-1](#)。



调试界面如下图所示：



```
main.c x SEGGER_SYSVIEW_Config_LiteOS.c
111     task_init_param.usTaskPrio = 15;
112     task_init_param.pcName = "test_task";
113     task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)test_ExcHandle;
114     task_init_param.uvStackSize = 0x1000;
115
116     uwRet = LOS_TaskCreate(&gExc_TskHandle, &task_init_param);
117     if(LOS_OK != uwRet)
118     {
119         return uwRet;
120     }
121     return uwRet;
122 }
123
124
125 int main(void)
126 {
127     #ifdef SYSVIEW_TRACE
128     SEGGER_SYSVIEW_Conf();
129     #endif
130
131     HardWare_Init();
132
133     UINT32 uwRet = LOS_OK;
134
135     uwRet = LOS_KernelInit();
136     if (uwRet != LOS_OK)
137     {
138         return LOS_NOK;
139     }
140     //LOS_EvbSetup();
141
142     #ifdef SYSVIEW_TRACE
143     LOS_Inspect_Entry();
144     #endif
145     #ifdef SYSVIEW_TRACE
146     LOS_Report_Inf...
```

DisassemblyFile
114025 802c6a2: bd80 pop {r7, pc} 114026 802c6a4: 08037728 .word 0x08037728 114027 802c6a8: 0802c655 .word 0x0802c655 114028 802c6ac: 2000dc20 .word 0x2000dc20 114029 114030 0802c6b0 <main>: 114031 return uwRet; 114032 } 114033 114034 114035 int main(void) 114036 { 114037 802c6b0: b580 push {r7, lr} 114038 802c6b2: b682 sub sp, #8 114039 802c6b4: af00 add r7, sp, #0 114040 #ifndef SYSVIEW_TRACE 114041 SEGGER_SYSVIEW_Conf(); 114042 ► 802c6b6: f002 fa15 bl 802eae4 <SEGGER_SYSVIEW_Conf> 114043 #endif 114044 114045 HardWare_Init(); 114046 802c6ba: f7ff ffb9 bl 802c630 <Hardware_Init> 114047 114048 UINT32 uwRet = LOS_OK; 114049 802c6be: 2300 movs r3, #0 114050 802c6c0: 607b str r3, [r7, #4] 114051 114052 uwRet = LOS_KernelInit(); 114053 802c6c2: f7d4 fdc3 bl 800124c <LOS_KernelInit> 114054 802c6c6: 6078 str r0, [r7, #4] 114055 if (uwRet != LOS_OK) 114056 802c6c8: 687b ldr r3, [r7, #4] 114057 802c6ca: 2b00 cmp r3, #0 114058 802c6cc: d001 beq.n 802c6d2 <main+0x22> 114059 {

6.7 编译，烧录及调试（华为海思芯片）

6.7.1 环境准备

请参见[5.2.1 环境准备](#)章节。

6.7.2 编译（Linux）

LiteOS Studio支持在Linux系统中对华为海思芯片进行编译，并对编译出来的系统镜像二进制文件在Windows系统中进行烧录和调试。

6.7.2.1 编译前注意事项

请参见[5.2.2 编译前注意事项](#)章节。

6.7.2.2 在 Linux 系统下搭建开发环境

本章节以MobaXterm为例，介绍如何在Linux系统下搭建开发环境。

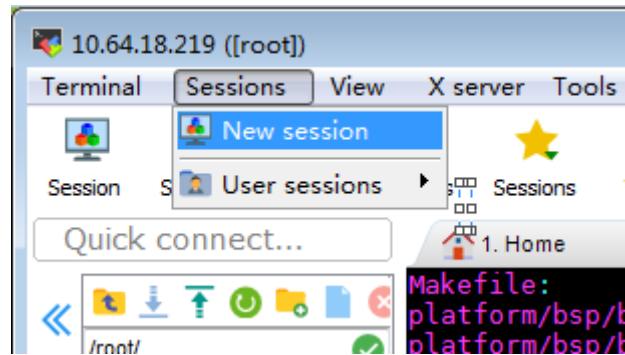
前提条件

- 已安装MobaXterm工具。
- 已下载arm-himix100-linux工具链。

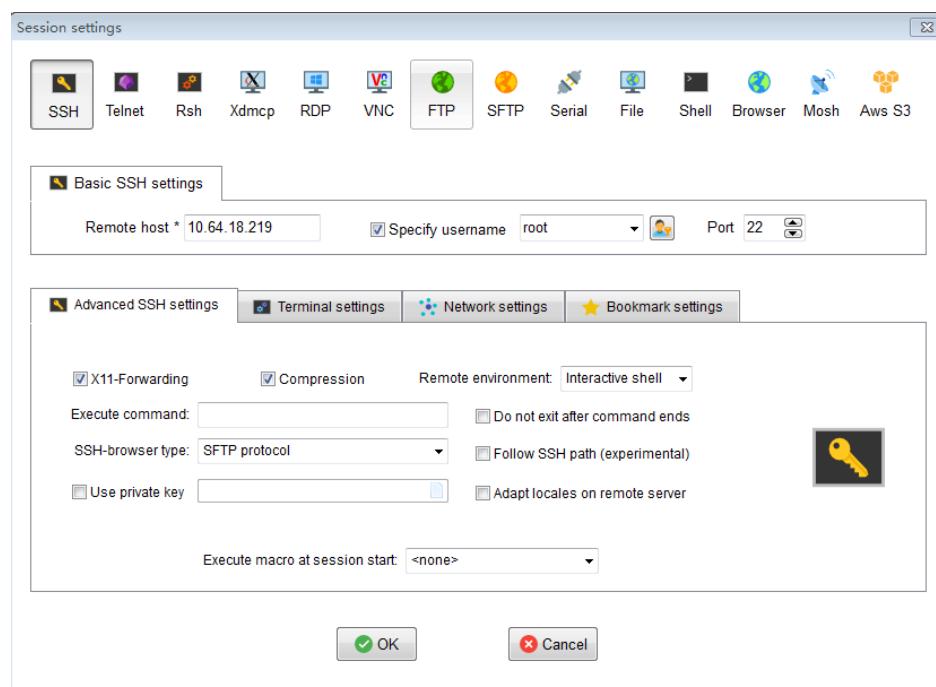
操作步骤

步骤1 通过MobaXterm连接Linux服务器。

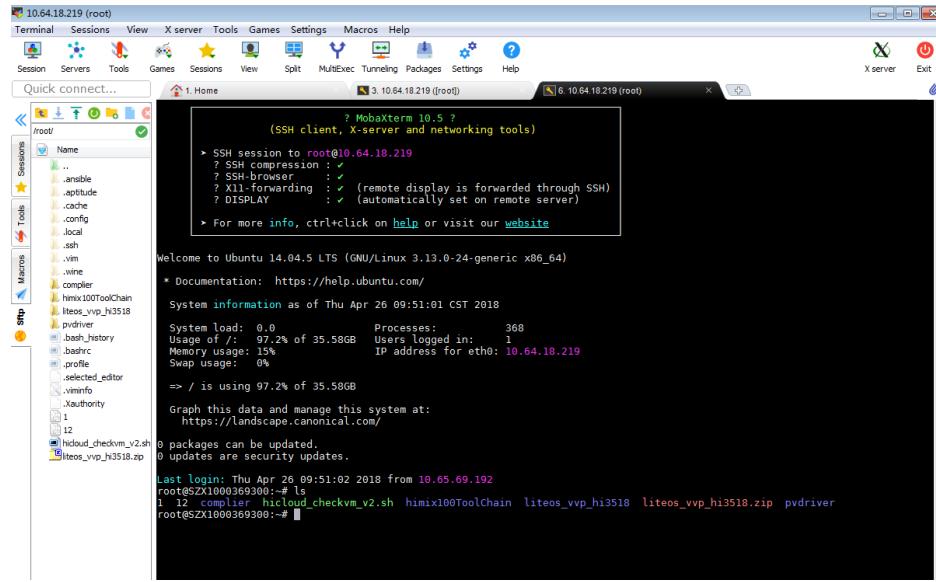
1. 打开MobaXterm软件，新建session（菜单栏Sessions>New session）。



2. 填写Linux主机信息。
 - Remote host: Linux服务器的IP地址。
 - Specify username: root。



3. 填写完成后，单击“OK”，连接Linux服务器。



步骤2 确认Linux系统中有make文件。

在Linux系统中，执行以下命令，查找make文件。

find / -name make

- 如有make文件，执行**步骤3**。
- 如没有找到make文件，请先在其他Linux系统中查找make文件，并将找到的make文件放入到Linux服务器的/usr/bin目录下，再执行**步骤3**。

步骤3 在linux环境下搭建arm-himix100-linux交叉编译器。

1. 压缩arm-himix100-linux为arm-himix100-linux.zip。
2. 通过MobaXterm将arm-himix100-linux.zip包拷贝到云Linux主机上。
3. 执行以下命令，解压压缩包。
unzip arm-himix100-linux.zip
4. 执行以下命令，进入arm-himix100-linux文件夹。
cd arm-himix100-linux
5. 执行以下命令，安装himix100交叉编译器。默认将工具链安装在 /opt/hisi-linux/x86-arm 目录下。
source ./arm-himix100-linux.install

说明

如果想指定工具链的安装位置，执行以下命令：

source ./arm-himix100-linux.install <安装路径>

如：**source ./arm-himix100-linux.install dirname**，指定安装在dirname目录下。

安装后，就可以使用arm-himix100-linux-xxx 工具链了。

```
root@5ZX1000369300:~# arm-himix100-linux-
arm-himix100-linux-addr2line    arm-himix100-linux-g++      arm-himix100-linux-gcov-tool  arm-himix100-linux-ranlib
arm-himix100-linux-ar          arm-himix100-linux-gcc      arm-himix100-linux-gprof    arm-himix100-linux-readelf
arm-himix100-linux-as          arm-himix100-linux-gcc-6.3.0  arm-himix100-linux-ld      arm-himix100-linux-size
arm-himix100-linux-c++         arm-himix100-linux-gcc-ar   arm-himix100-linux-bfd     arm-himix100-linux-strings
arm-himix100-linux-c++filt     arm-himix100-linux-gcc-nm   arm-himix100-linux-nm     arm-himix100-linux-strip
arm-himix100-linux-cpp         arm-himix100-linux-gcc-ranlib arm-himix100-linux-objcopy
arm-himix100-linux-elfedit     arm-himix100-linux-gcov     arm-himix100-linux-objdump
```

步骤4 配置交叉编译器的环境变量。这里以arm-himix100-linux为例。

- 执行以下命令，打开`~/.bashrc`文件。

```
vi ~/.bashrc
```

- 增加如下交叉编译器的环境变量。

```
export PATH=$PATH:/root/himix100ToolChain/arm-himix100-linux/bin
```

```
export PATH=$PATH:/root/himix100ToolChain/arm-himix100-linux/bin
```

按“Esc”，然后输入“:wq”后按“Enter”，保存并退出编辑器。

- 执行以下命令，编译环境准备完成。

```
Source ~/.bashrc
```

----结束

6.7.2.3 在 Linux 系统下进行编译

华为海思芯片工程需在Linux系统下进行编译，在Linux系统下编译完成后，您可通过download方式，将编译后的文件下载到Windows系统中，也可通过Samba服务器搭建及Windows 网络驱动器映射方式，共享在Liunx系统中编译后的文件。

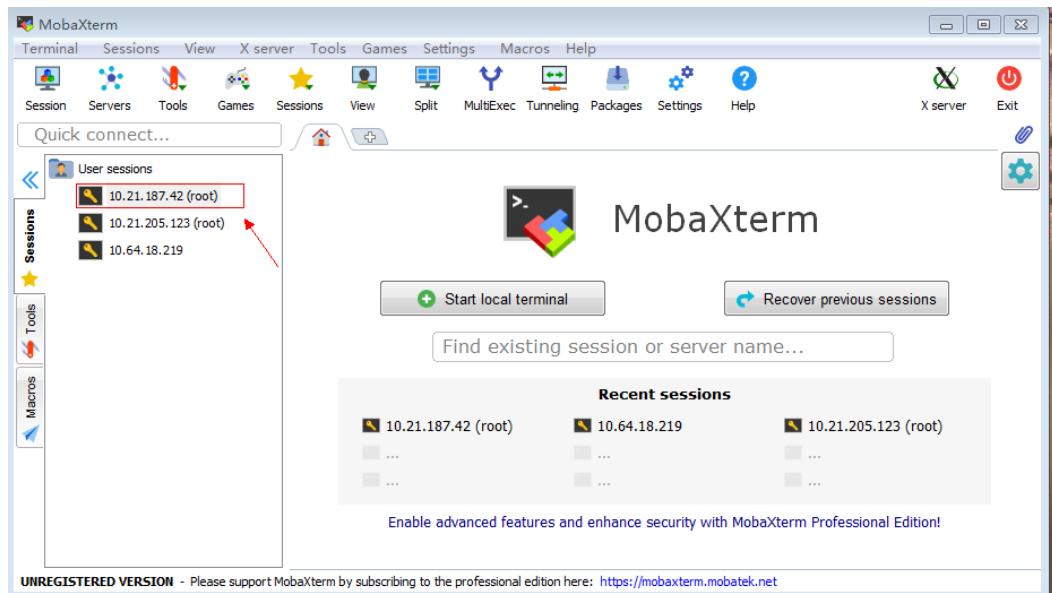
本章以download方式及华为海思3516CV300工程源码为例，介绍如何在Linux系统中进行编译。

前提条件

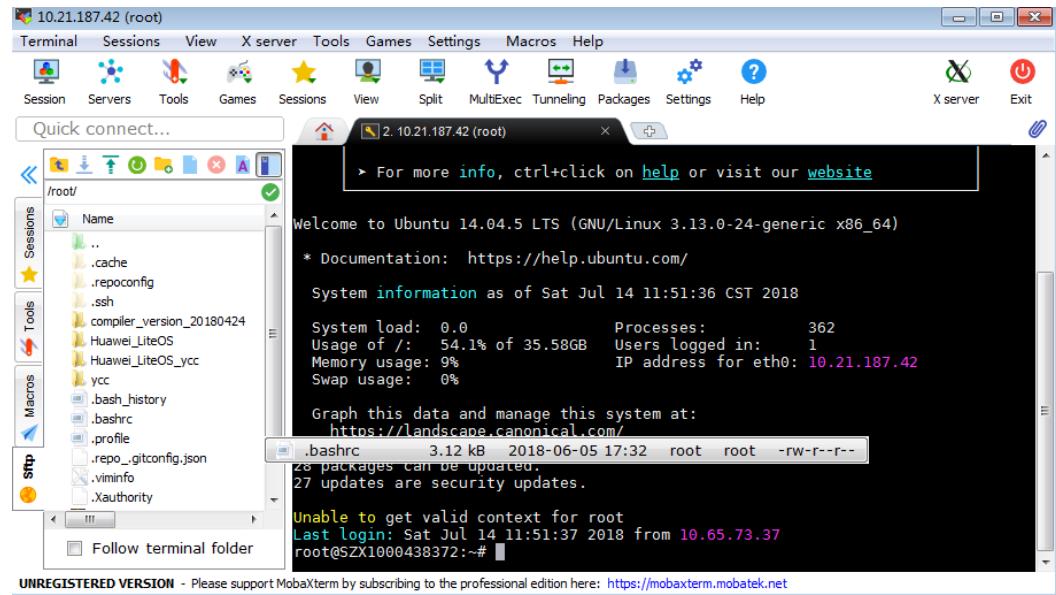
已在Linux环境下搭建开发环境。具体请参见[6.7.2.2 在Linux系统下搭建开发环境](#)章节。

操作步骤

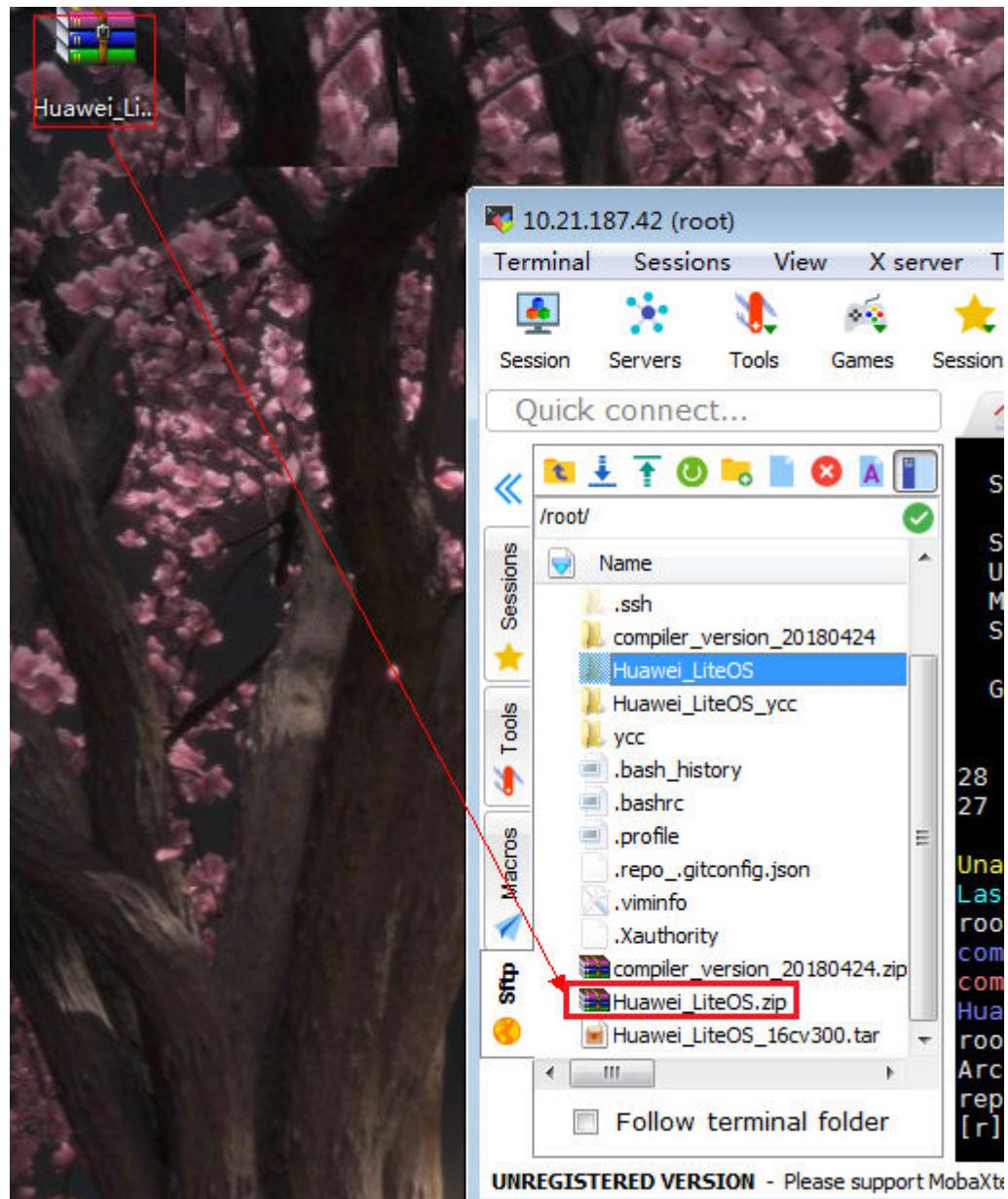
步骤1 打开MobaXterm工具，单击已连接的Linux服务器地址，如下图：



进入Linux服务器后，如下图所示：

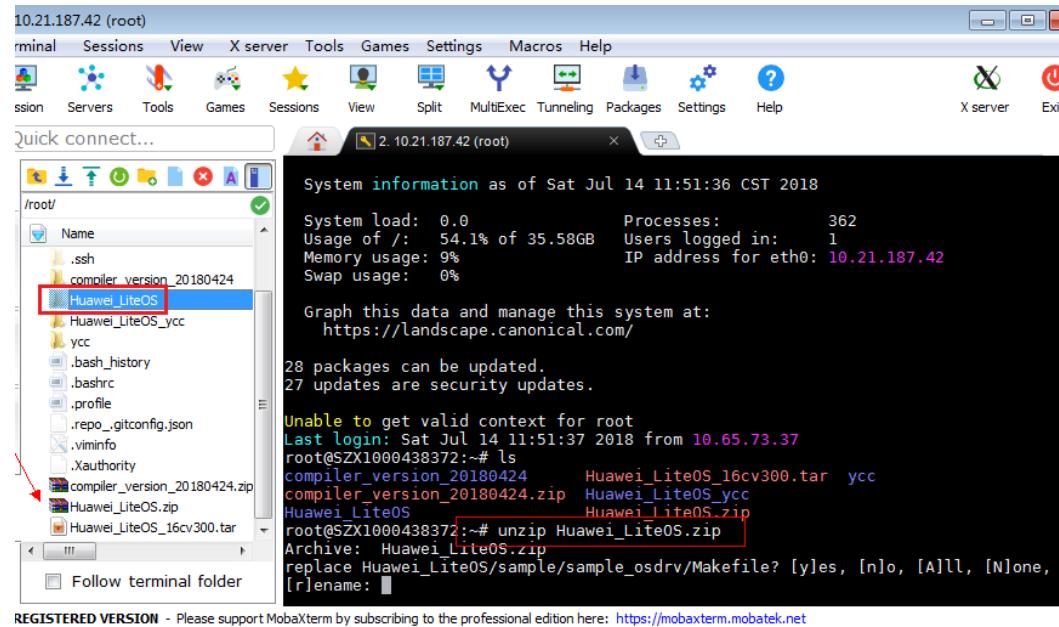


步骤2 将项目源码包拖入root根目录下。



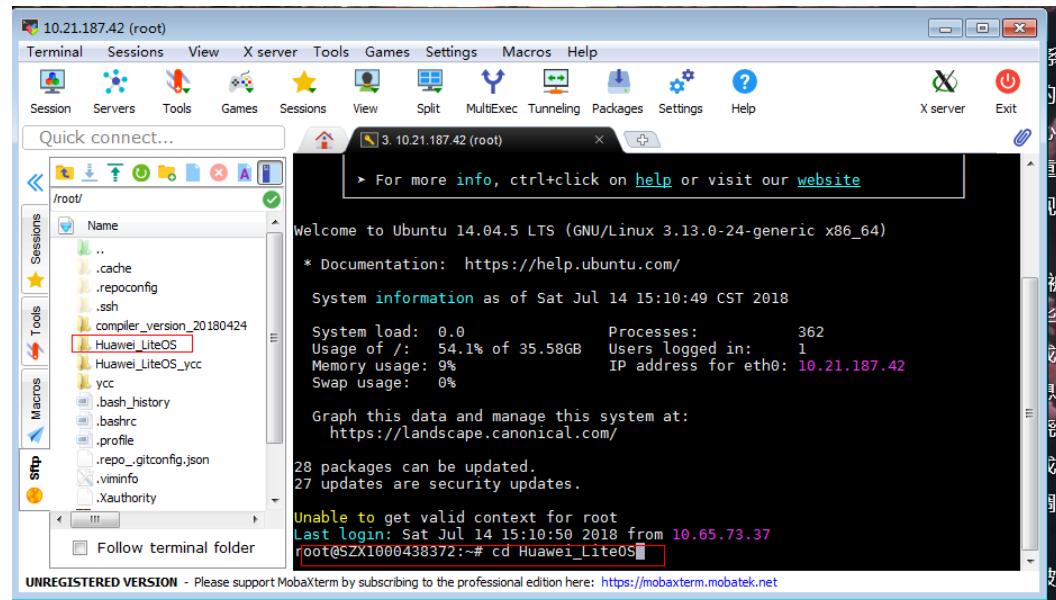
步骤3 执行以下命令，解压项目源码压缩包。

```
unzip Huawei_LiteOS.zip
```



步骤4 执行以下命令，进入Huawei_LiteOS文件夹。

```
cd Huawei_LiteOS
```



步骤5 修改.config文件。

说明

默认生成的ELF文件所附带的调试信息很少，要生成带更多调试信息的ELF文件需要配置工程根目录下的.config文件中的`LOSCFG_COMPILE_DEBUG`。

- 执行以下命令，编辑.config文件。

```
vi .config
```

- 如图中区域所示，添加如下信息。

```
LOSCFG_COMPILE_DEBUG = y
```

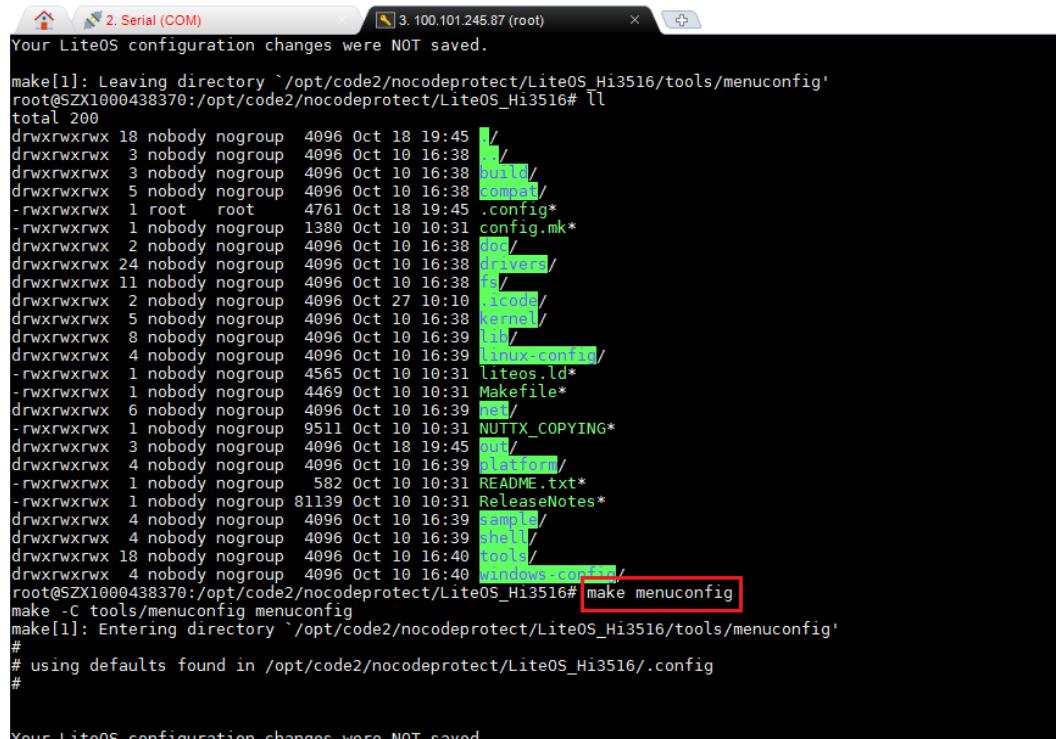
```
#  
# Debug  
#  
# LOSCFG_COMPILE_DEBUG is not set  
LOSCFG_COMPILE_DEBUG=y
```

- 按“Esc”，然后输入“:wq”后按“Enter”，保存并退出编辑器。

步骤6 在menuconfig中，配置“Enable GCC -g Option”选项。如已配置，请忽略此步骤。

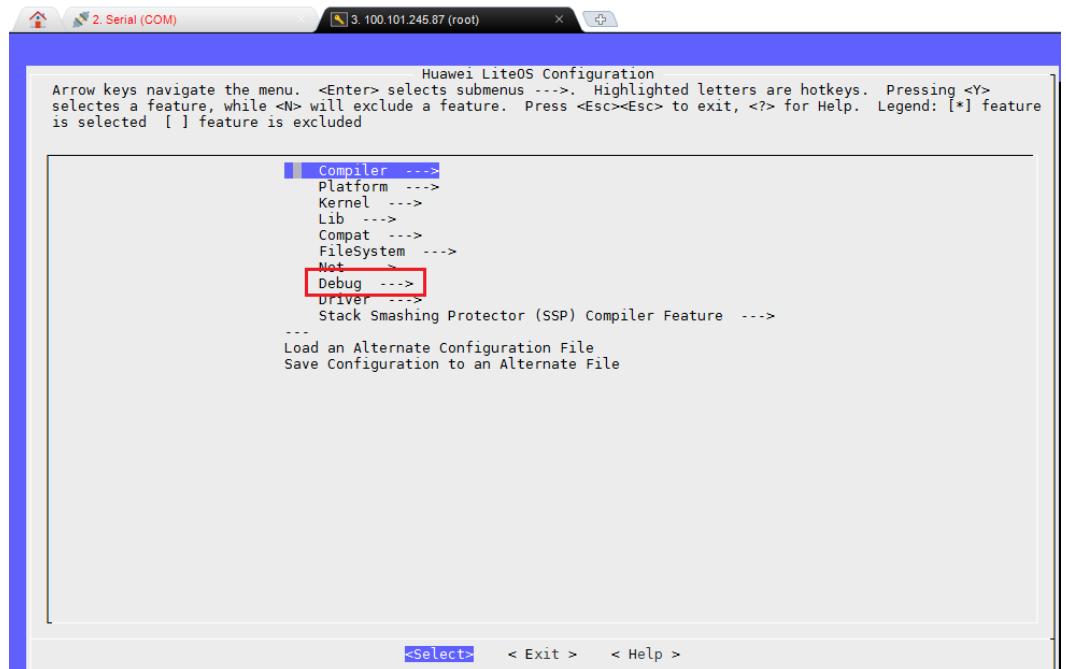
- 在工程根目录下，执行以下命令，进入menuconfig配置界面。

make menuconfig

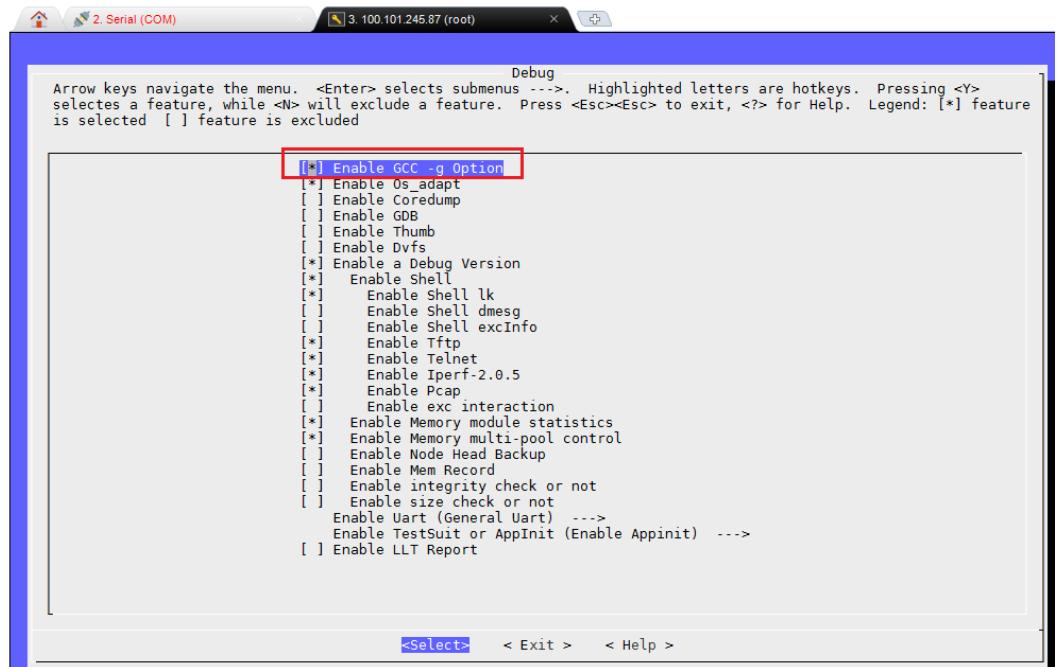


```
Your LiteOS configuration changes were NOT saved.  
make[1]: Leaving directory `/opt/code2/nocodeprotect/LiteOS_Hi3516/tools/menuconfig'  
root@5ZX1000438370:/opt/code2/nocodeprotect/LiteOS_Hi3516# ll  
total 200  
drwxrwxrwx 18 nobody nogroup 4096 Oct 18 19:45 /  
drwxrwxrwx 3 nobody nogroup 4096 Oct 10 16:38 ./  
drwxrwxrwx 3 nobody nogroup 4096 Oct 10 16:38 build/  
drwxrwxrwx 5 nobody nogroup 4096 Oct 10 16:38 compat/  
-rwxrwxrwx 1 root root 4761 Oct 18 19:45 .config*  
-rwxrwxrwx 1 nobody nogroup 1380 Oct 10 10:31 config.mk*  
drwxrwxrwx 2 nobody nogroup 4096 Oct 10 16:38 doc/  
drwxrwxrwx 24 nobody nogroup 4096 Oct 10 16:38 drivers/  
drwxrwxrwx 11 nobody nogroup 4096 Oct 10 16:38 fs/  
drwxrwxrwx 2 nobody nogroup 4096 Oct 27 10:10 icode/  
drwxrwxrwx 5 nobody nogroup 4096 Oct 10 16:38 kernel/  
drwxrwxrwx 8 nobody nogroup 4096 Oct 10 16:39 lib/  
drwxrwxrwx 4 nobody nogroup 4096 Oct 10 16:39 linux-contin/  
-rwxrwxrwx 1 nobody nogroup 4565 Oct 10 10:31 liteos.ld*  
-rwxrwxrwx 1 nobody nogroup 4469 Oct 10 10:31 Makefile*  
drwxrwxrwx 6 nobody nogroup 4096 Oct 10 16:39 net/  
-rwxrwxrwx 1 nobody nogroup 9511 Oct 10 10:31 NUTTX_COPYING*  
drwxrwxrwx 3 nobody nogroup 4096 Oct 18 19:45 out/  
drwxrwxrwx 4 nobody nogroup 4096 Oct 10 16:39 platform/  
-rwxrwxrwx 1 nobody nogroup 582 Oct 10 10:31 README.txt*  
-rwxrwxrwx 1 nobody nogroup 81139 Oct 10 10:31 ReleaseNotes*  
drwxrwxrwx 4 nobody nogroup 4096 Oct 10 16:39 sample/  
drwxrwxrwx 4 nobody nogroup 4096 Oct 10 16:39 shell/  
drwxrwxrwx 18 nobody nogroup 4096 Oct 10 16:40 tools/  
drwxrwxrwx 4 nobody nogroup 4096 Oct 10 16:40 windows-config/  
root@5ZX1000438370:/opt/code2/nocodeprotect/LiteOS_Hi3516# make menuconfig  
make -C tools/menuconfig menuconfig  
make[1]: Entering directory `/opt/code2/nocodeprotect/LiteOS_Hi3516/tools/menuconfig'  
#  
# using defaults found in /opt/code2/nocodeprotect/LiteOS_Hi3516/.config  
#  
  
Your LiteOS configuration changes were NOT saved.
```

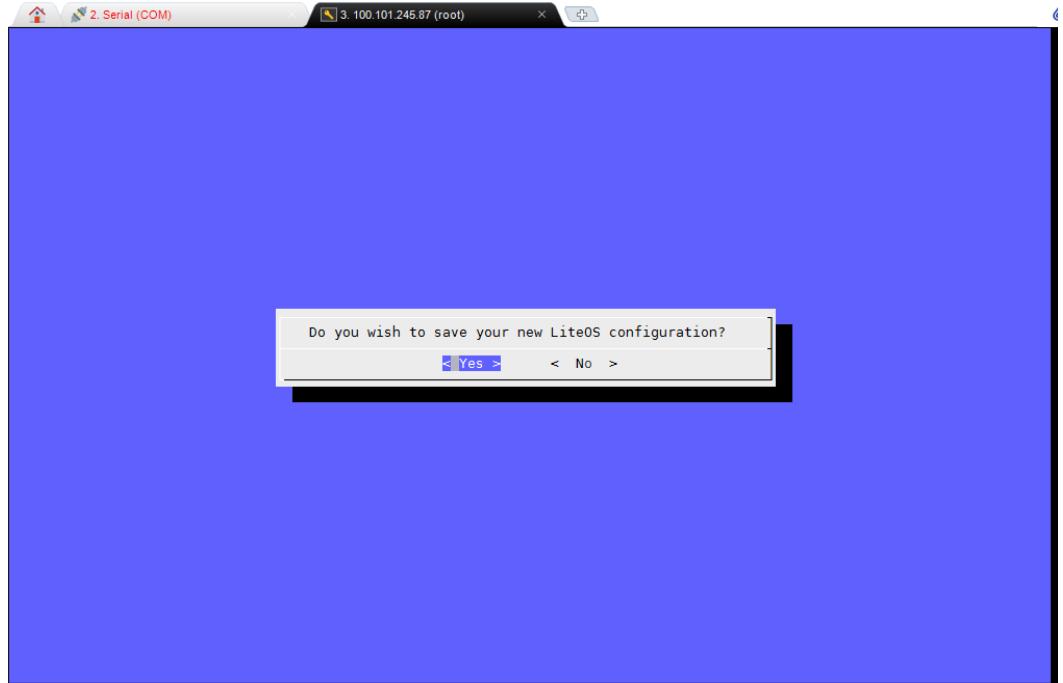
- 进入“Debug”目录。



3. 按空格键选中“Enable GCC -g Option”选项，选择完后单击“Exit”退出。



4. 选中“Yes”保存配置。



步骤7 (可选) 若工程使用了单独的makefile, 可在编译选项中添加 -g -O0选项。

步骤8 在工程根目录下执行make命令, 进行内核编译。

make -j8

```
drwxr-xr-x 3 nobody nogroup 4096 Dec  5 10:25 ./
drwxrwxrwx 8 nobody nogroup 4096 Dec 14 10:04 ./_
drwxr-xr-x 16 nobody nogroup 4096 Dec  5 10:34 liteos/
root@S ZX1000438370:/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020# cd liteos/
root@S ZX1000438370:/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos# ll
total 144
drwxr-xr-x 16 nobody nogroup 4096 Dec  5 10:34 ./
drwxr-xr-x  3 nobody nogroup 4096 Dec  5 10:25 ../_
drwxr-xr-x  3 nobody nogroup 4096 Dec  5 10:25 build/
drwxr-xr-x  5 nobody nogroup 4096 Dec  5 10:25 compat/
-rw-r--r--  1 root   root  3264 Dec  5 10:34 .config
-rwxr--r--  1 nobody nogroup 3275 May  2 2018 config_hi3516cv300*
-rwxr--r--  1 nobody nogroup 2052 May  2 2018 config.mk*
drwxr-xr-x  2 nobody nogroup 4096 Dec  5 10:26 doc/
drwxr-xr-x 17 nobody nogroup 4096 Dec  5 10:25 drivers/
drwxr-xr-x 10 nobody nogroup 4096 Dec  5 10:25 fs/
drwxr-xr-x  2 nobody nogroup 4096 Dec 12 21:40 icode/
drwxr-xr-x  5 nobody nogroup 4096 Dec  5 10:25 kernel/
drwxr-xr-x 12 nobody nogroup 4096 Dec  5 10:25 lib/
-rwxr--r--  1 nobody nogroup 4020 May  2 2018 liteosexc.ld*
-rwxr--r--  1 nobody nogroup 4016 May  2 2018 liteos.ld*
-rwxr--r--  1 nobody nogroup 2069 May  2 2018 Makefile*
drwxr-xr-x  5 nobody nogroup 4096 Dec  5 10:26 net/
-rwxr--r--  1 nobody nogroup 9511 May  2 2018 NUTTX_COPYING*
drwxr-xr-x  3 root   root  4096 Dec 12 11:31 out/
drwxr-xr-x  4 nobody nogroup 4096 Dec  5 10:25 platform/
-rwxr--r--  1 nobody nogroup 44869 May  2 2018 ReleaseNotes*
drwxr-xr-x  5 nobody nogroup 4096 Dec  5 10:25 sample/
drwxr-xr-x  4 nobody nogroup 4096 Dec  5 10:25 shell/
drwxr-xr-x 10 nobody nogroup 4096 Dec  5 10:25 tools/
root@S ZX1000438370:/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos# make clean ;make -j8
clean cxx header finish
clean c library finish
```

步骤9 编译后生成libsbuild.date文件, 进入sample/sample_osdrv文件, 执行make命令, 进行源码工程编译。

make -j8

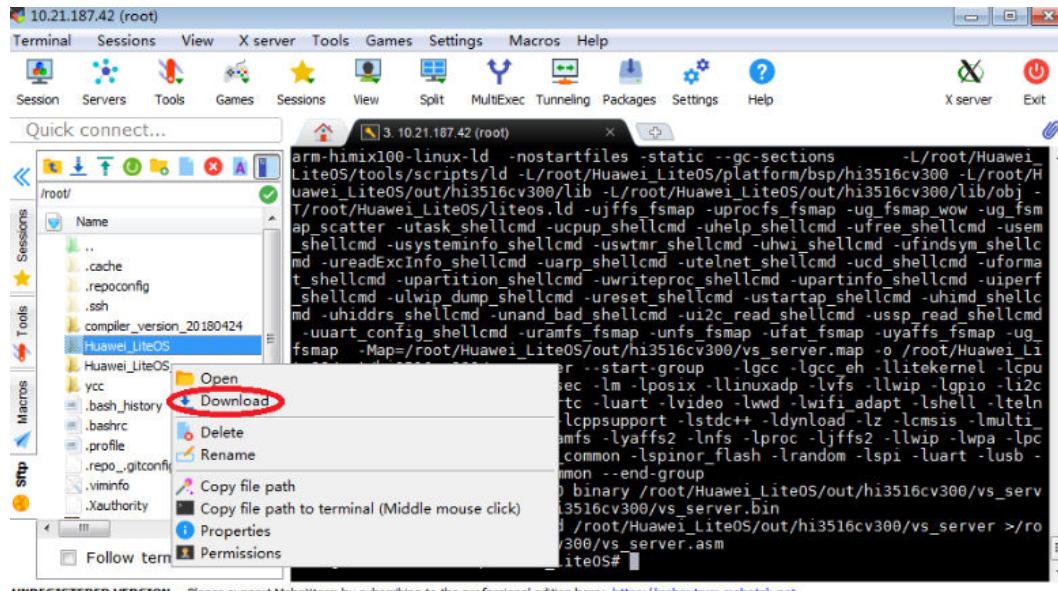
```

make11: Leaving directory '/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos/platform/bsp/common'
date > /opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos/out/hi3516cv300/libsbuild.date
===== make lib done =====
root@SX1000438370:/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos# cd sample/sample_osdrv/
root@SX1000438370:/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos/sample/sample_osdrv# ll
total 14040
drwxr-xr-x 2 nobody nogroup 4096 Dec 12 11:32 .
drwxr-xr-x 5 nobody nogroup 4096 Dec 5 10:25 ../
-rw-r--r-- 1 nobody nogroup 604 May 2 2018 Makefile*
-rwxr-xr-x 1 root  root  2262976 Dec 12 11:32 sample*
-rw-r--r-- 1 root  root  9547367 Dec 12 11:32 sample.asm
-rw-r--r-x 1 root  root  1218984 Dec 12 11:32 sample.bin*
-rw-r--r-- 1 nobody nogroup 7764 May 2 2018 sample.c*
-rw-r--r-- 1 root  root  1272999 Dec 12 11:32 sample.map
-rw-r--r-- 1 root  root  31020 Dec 12 11:32 sample.o
root@SX1000438370:/opt/code2/songwang/LiteOS_Hi3516CV300_BVT_SDK020/liteos/sample/sample_osdrv# make clean ;make -j8
arm-himix100-linux-gcc -g -gdwarf-2 -O0 -mpcu=arm926ej-s -fno-aggressive-loop-optimizations -fno-builtin -nostdinc -nostdlib -mno-tdep99 -Wpointer-arith -Wstrict-prototypes -Wno-write-strings -mthumb-interwork -ffunction-sections -fdata-sections -fno-exceptions -D_LITEOS -DHISI3516CV300 -DLOSCFG_ARM926 -DLOSCFG_MEM_WATERLINE -DLOSCFG_KERNEL_CUPUP -DLOSCFG_KERNEL_CPSUPPORT -DLOSCFG_KERNEL_CONFIG_STERROR -DLOSCFG_LIB_TMR -DLOSCFG_COMPAT_POSIX -DLOSCFG_COMPAT_LINUX -DLOSCFG_FS_VFS -DLOSCFG_FS_FAT -DLOSCFG_FS_FAT_CACHE

```

编译后生成可进行烧录的二进制文件，如“sample.elf”，“sample.bin”及“sample.asm”文件，具体生成文件名请以实际工程为准。

步骤10 单击“Download”下载编译后的文件到Windows系统中。



----结束

6.7.3 烧录

操作步骤

步骤1 将华为海思芯片开发板与电脑连接。

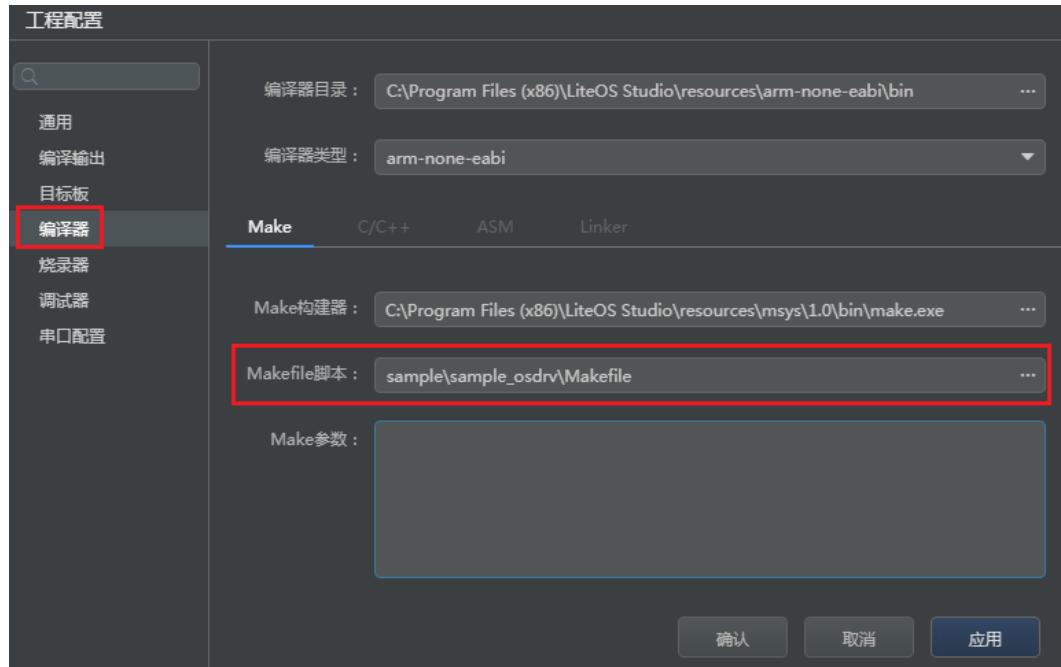


说明

请先连接开发板电源，再将开发板与JLink仿真器连接。

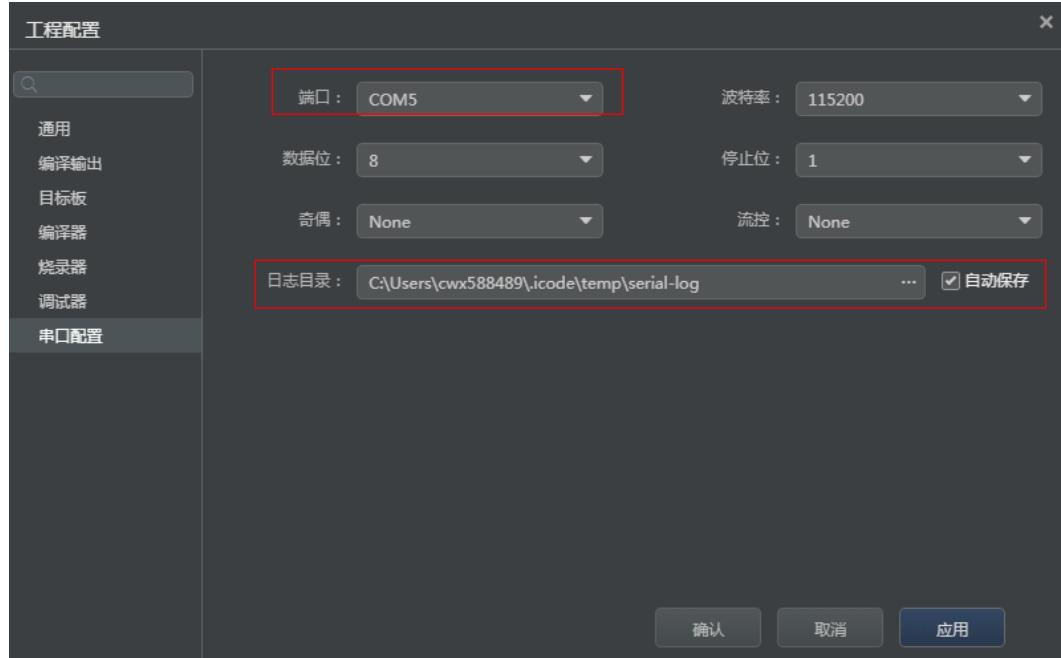
步骤2 在烧录前请进行工程配置。

- 单击工具栏的 ，进行工程配置。
- 单击“编译器”，“Makefile脚本”配置为编译输出目录中的makefile文件，具体路径请根据实际情况修改。“Make参数”可配置为“-j8”。



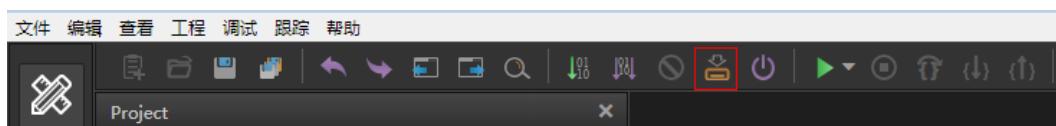
3. 单击“串口配置”，在“端口”下拉框中选择与开发板实际连接的端口号。获取端口号具体请参见[7.1 获取连接目标板的实际串口号](#)章节。

如需自动保存日志，请勾选“自动保存”，您也可以单击“日志目录”的...按钮，更改日志保存路径。



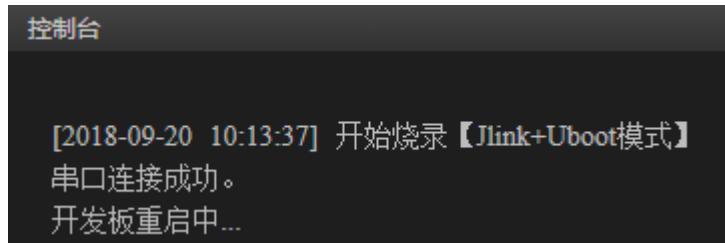
4. 单击“确认”，保存配置。

步骤3 单击工具栏中的，将已编译的程序烧录至开发板。



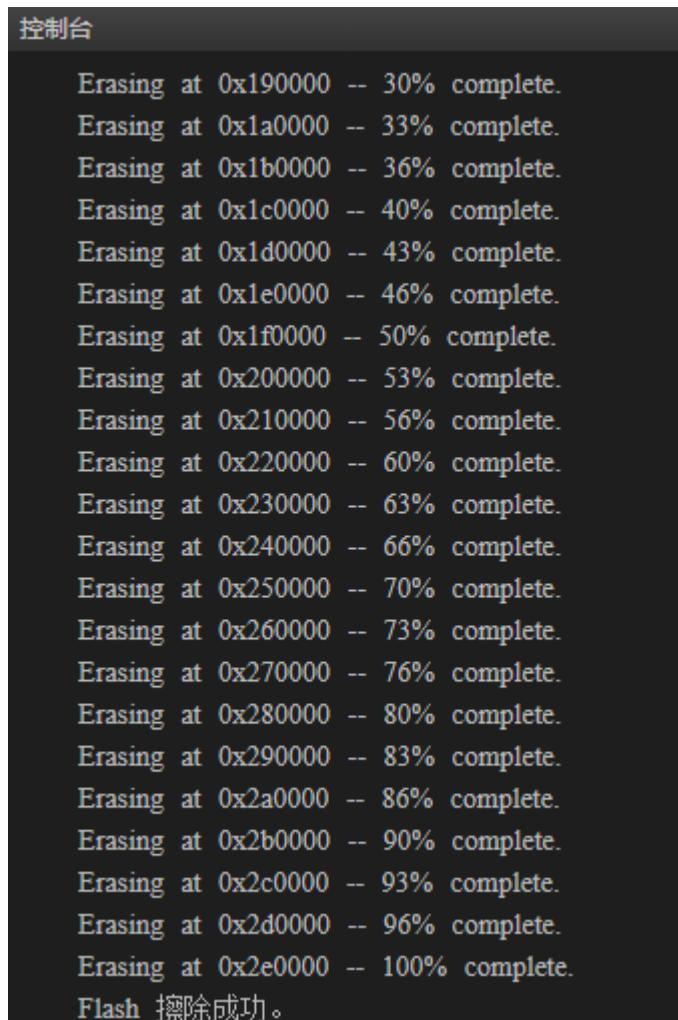
烧录现象如下：

单击烧录后，当显示如下信息时，自动重启开发板。



烧录进程如下：

通过Uboot擦除加载操作系统的对应flash地址的内容，擦除完成后通过jlink将新的系统文件加载至目标板内存。



再通过Uboot将已加载至内存中的系统写入flash，控制台输出面板中显示“烧录成功”后，完成烧录。

控制台

```
Writing at 0x130000 -- 10% complete.  
Writing at 0x140000 -- 13% complete.  
Writing at 0x150000 -- 16% complete.  
Writing at 0x160000 -- 20% complete.  
Writing at 0x170000 -- 23% complete.  
Writing at 0x180000 -- 26% complete.  
Writing at 0x190000 -- 30% complete.  
Writing at 0x1a0000 -- 33% complete.  
Writing at 0x1b0000 -- 36% complete.  
Writing at 0x1c0000 -- 40% complete.  
Writing at 0x1d0000 -- 43% complete.  
Writing at 0x1e0000 -- 46% complete.  
Writing at 0x1f0000 -- 50% complete.  
Writing at 0x200000 -- 53% complete.  
Writing at 0x210000 -- 56% complete.  
Writing at 0x220000 -- 60% complete.  
Writing at 0x230000 -- 63% complete.  
Writing at 0x240000 -- 66% complete.  
Writing at 0x250000 -- 70% complete.  
Writing at 0x260000 -- 73% complete.  
Writing at 0x270000 -- 76% complete.  
Writing at 0x280000 -- 80% complete.  
Writing at 0x290000 -- 83% complete.  
Writing at 0x2a0000 -- 86% complete.  
Writing at 0x2b0000 -- 90% complete.  
Writing at 0x2c0000 -- 93% complete.  
Writing at 0x2d0000 -- 96% complete.  
Writing at 0x2e0000 -- 100% complete.  
Flash 写入成功。  
[2018-11-09 15:20:32] 烧录成功
```

----结束

6.7.4 调试

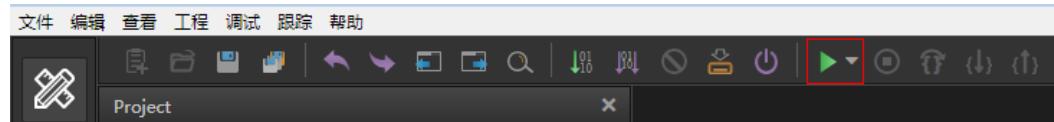
烧录成功后，可对当前工程进行调试。

操作步骤

单击工具栏中的 开始调试。如需更改调试方式，单击 下拉框选择调试方式，调试方式介绍请参见 [表5-1](#)。



启动调试时，自动配置“ > 调试器 > 编译目录”中的Linux平台编译路径。



调试界面如下图所示：

6.8 调试中的各面板作用

6.8.1 反汇编

前提条件

当前工程处于调试模式下，才能进行反汇编调试。

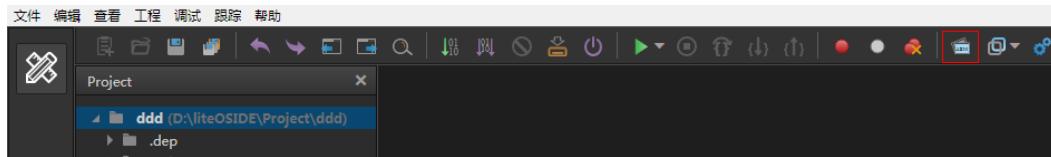
背景信息

反汇编调试，可以查看源代码对应的汇编代码，从更底层，更本质的角度去分析代码的漏洞和性能瓶颈。

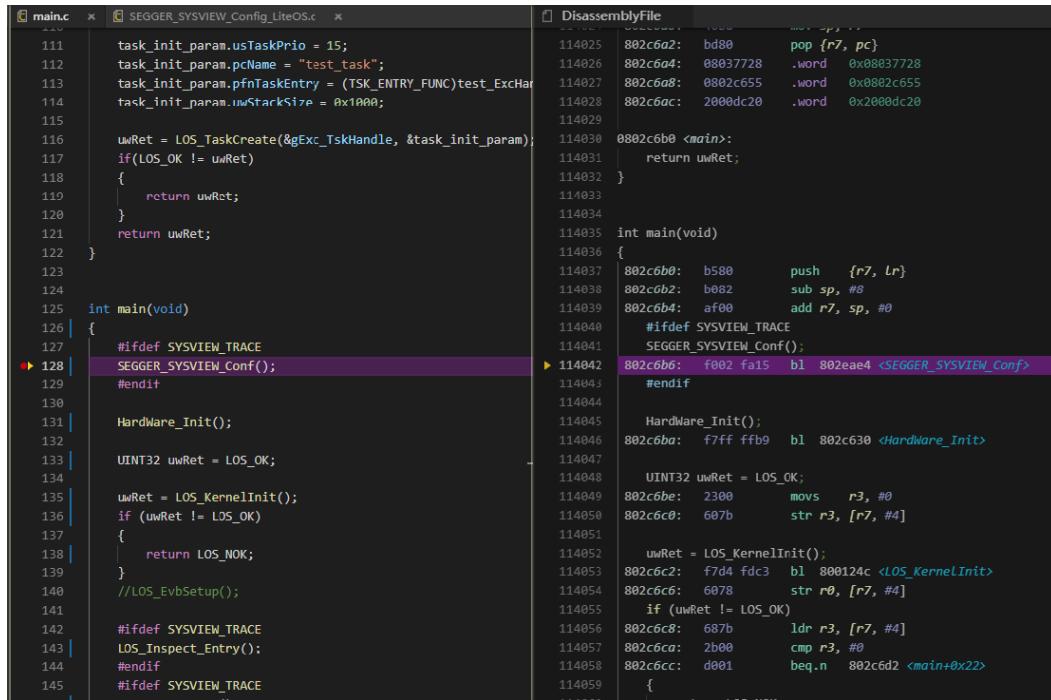
操作步骤



打开工具栏中的 反汇编开关，启动反汇编。



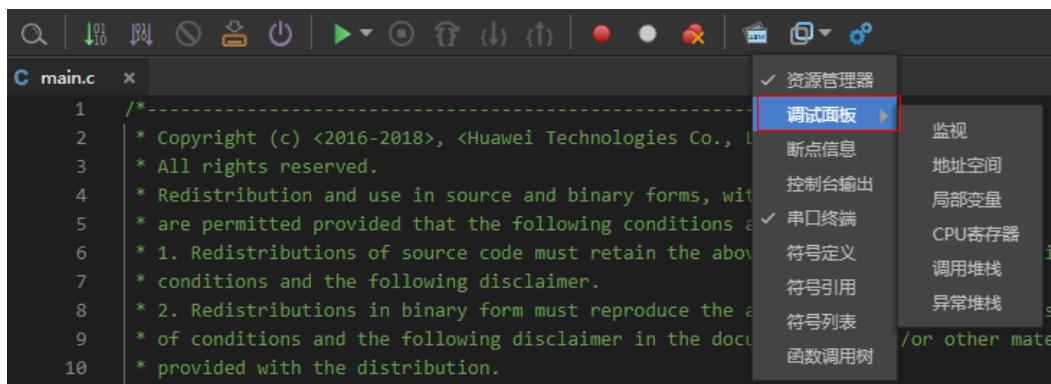
在调试模式下，打开的反汇编文件如下图所示：



- 调试源代码的断点时，会跳转到汇编代码对应的行数。
- 调试模式下查看源代码时，会跳转到汇编代码对应的行数。
- 调试模式下查看汇编代码时，会跳转到源代码对应的行数。
- 调试模式下，可对反汇编文件进行添加、禁用断点（不支持添加条件断点），执行下一步、跳入、跳出、继续调试等操作。

6.8.2 调试面板

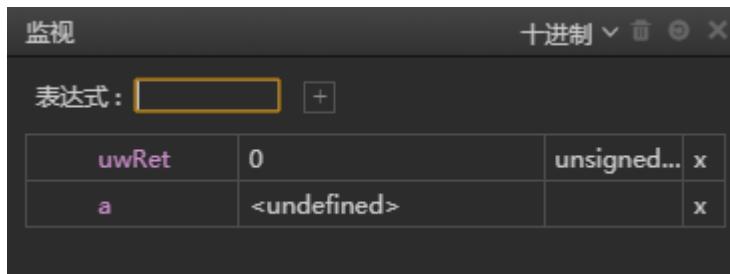
启动调试后，单击工具栏中的 下拉框，勾选“调试面板”，根据需要打开“监视”、“地址空间”、“局部变量”、“CPU寄存器”、“调用堆栈”及“异常堆栈”面板。



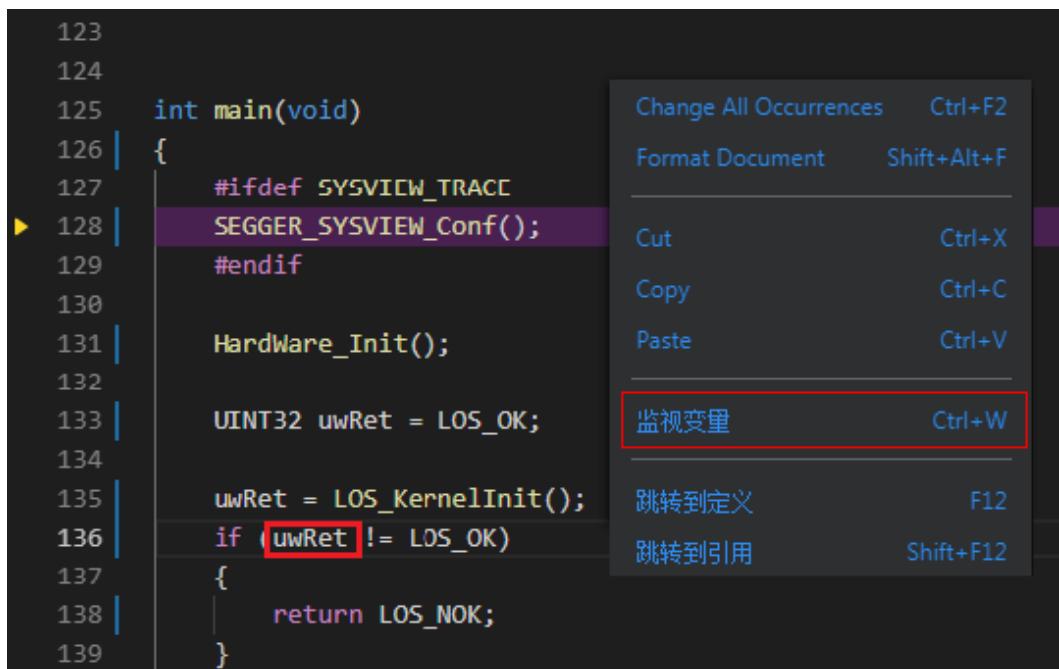
6.8.2.1 监视

添加变量值

- 启动调试后，在“表达式”文本框中填入需要查找的变量名称，单击“Enter”或文本框右边的“+”，在下方显示变量值。若值为“undefined”，表示此变量未定义。

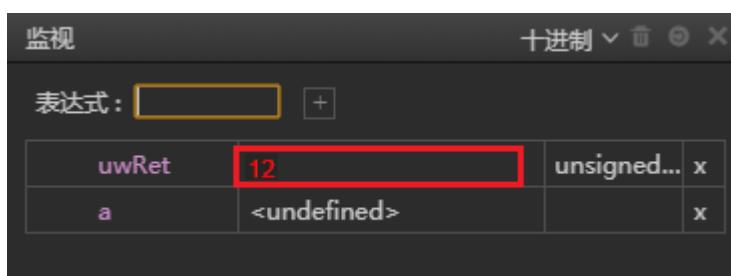


- 启动调试后，在打开的文件中，右键单击需要监视的变量，选择“监视变量”，将此变量添加到“监视”中。

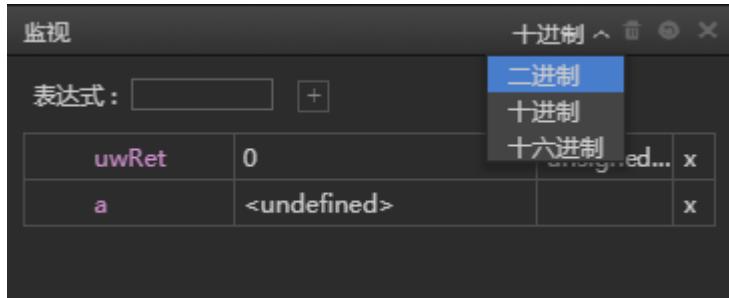


修改变量值

在“监视”面板，双击文本框修改对应变量的值（输入的数值默认为十进制），单击“Enter”完成修改。修改后，值为红色。



- 单击“十进制”下拉框，选择显示的数据类型为“二进制”或“十六进制”，修改对应面板中数据显示方式。



- 右键单击某个变量值文本框，选择数据显示类型，修改对应变量数据显示方式。



6.8.2.2 CPU 寄存器

在“CPU寄存器”区域，修改“Core”及“FPU”对应寄存器的值，双击文本框输入修改后的值，单击“Enter”完成修改。修改后，值为红色。

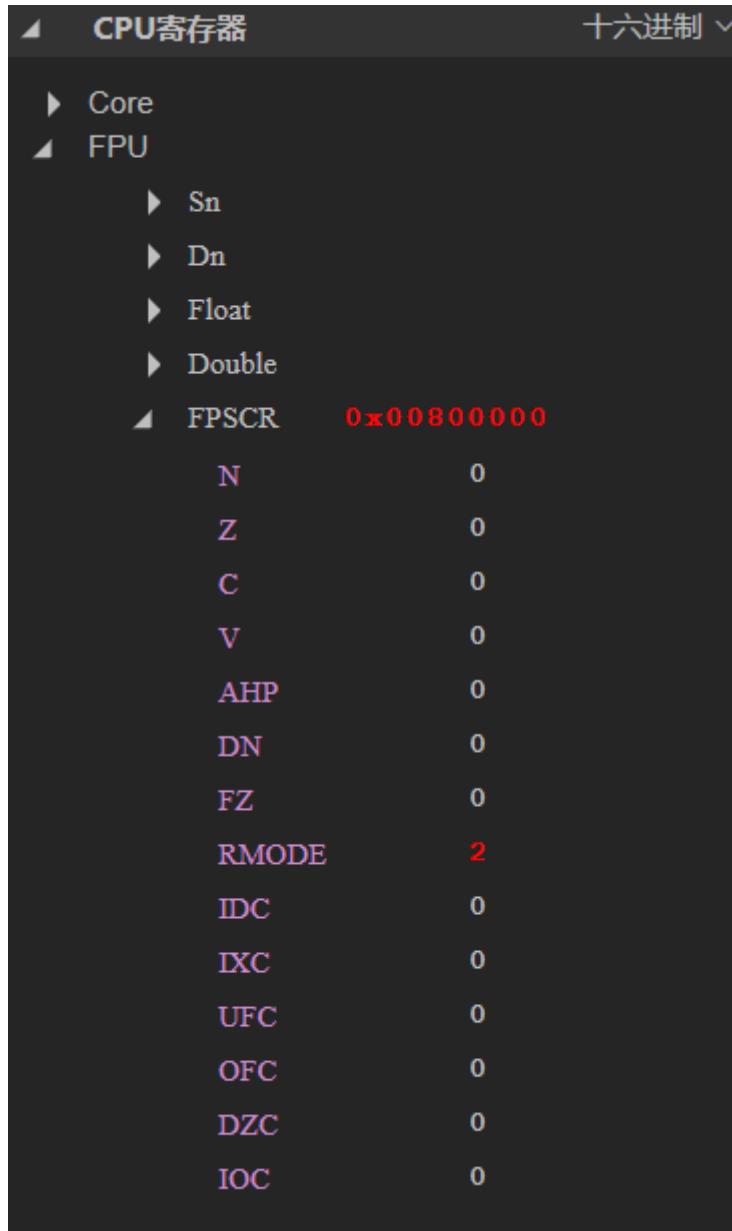
- 寄存器值显示方式可选择“十六进制”、“十进制”或“二进制”。
- 若输入错误的值，单击“Enter”显示为修改前的值，修改无效。
- 若输入的值超过0xFFFFFFFF，显示为0xFFFFFFFF。

CPU寄存器	
	十六进制
Core	
R0	0x20000400
R1	0x00000308
R2	0x08000055
R3	0x08000065
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x2002FFE8
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
SP	0x2002FFE8
LR	0x08030129
PC	0x0802C6A6
XPSR	0x61000000
MSP	0x2002FFE8
PSP	0x00000000
PRIMASK	0x00000000
BASEPRI	0x00000000
FAULTMASK	0x00000000
CONTROL	0x00000000

修改FPU寄存器值规则如下：

- Sn、Dn及Double的值不可修改。
- Float的S0~S22的值可修改， S23~S31的值不可修改。
- FPSCR的值修改规则如下图：

FPSCR格式																																							
舍入																句量长度-1								异常陷阱启动位								累加异常位							
31	30	29	28	27	26	25	24	23	22	21	20	18	17	16	15	12	11	10	9	8	7	6	5	4	3	2	1	0											
N	Z	C	V	QC	AHP	DN	FZ	RMODE	STRIDE	LEN	IDE	IXE	UFE	OFE	DZE	I0E	IDC	IXC	UFC	OFC	CDZ	CIO																	
舍入： 0=舍入到最近， 1=向+～舍入， 2=向-～舍入， 3=向零舍入。																																							

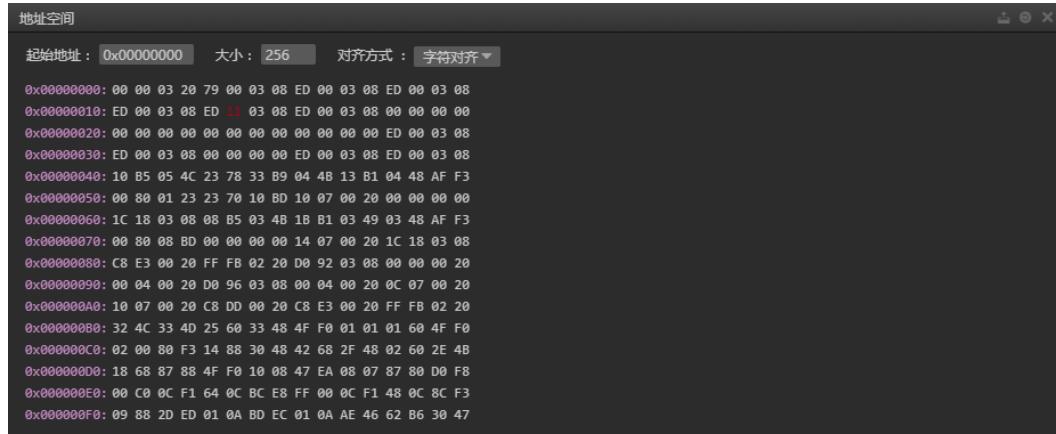


6.8.2.3 地址空间

修改 Memory 地址数据

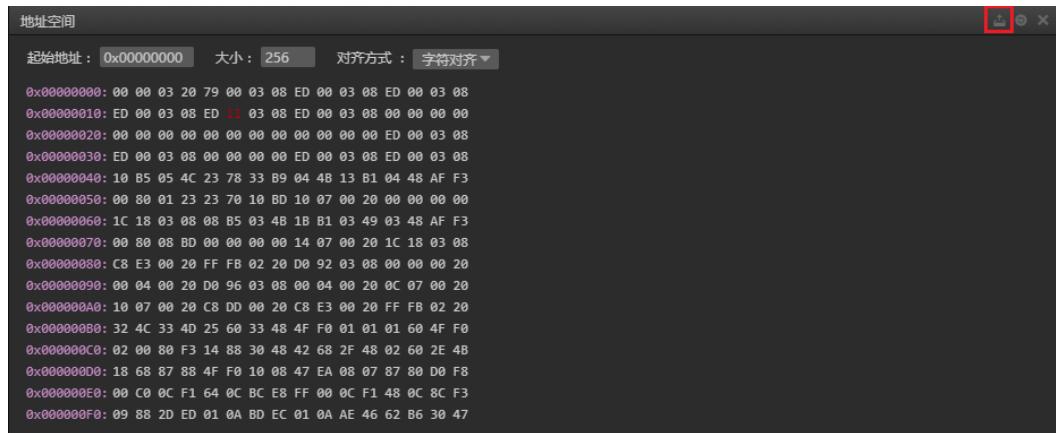
在“地址空间”面板，双击文本框修改对应Memory地址数据，单击“Enter”完成修改。修改后，值为红色。

“大小”最大可输入值为4096，“对齐方式”下拉框中选择数据地址对齐方式。

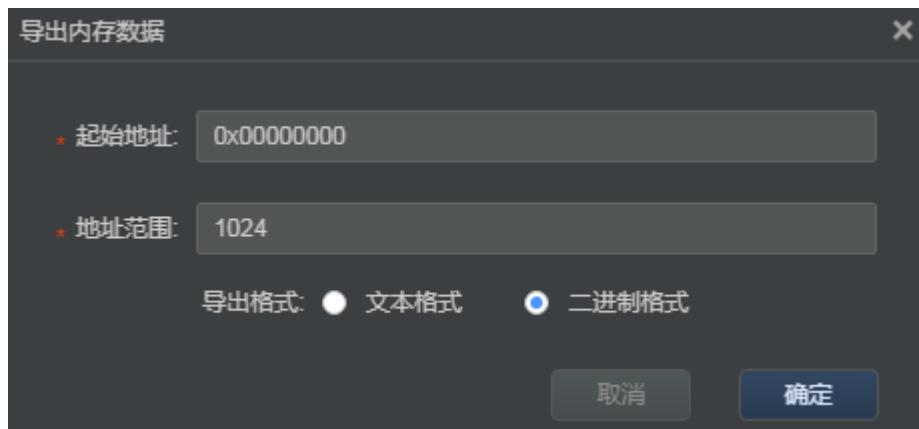


导出 Memory 地址数据

步骤1 调试状态下，在“地址空间”面板，单击 ，导出Memory地址数据。



步骤2 在弹出的“导出内存数据”界面中填入“起始地址”及“地址范围”。地址范围最大值为10240。



步骤3 根据需要选择导出格式，勾选“文本格式”或“二进制格式”单选框。

步骤4 单击“确定”。

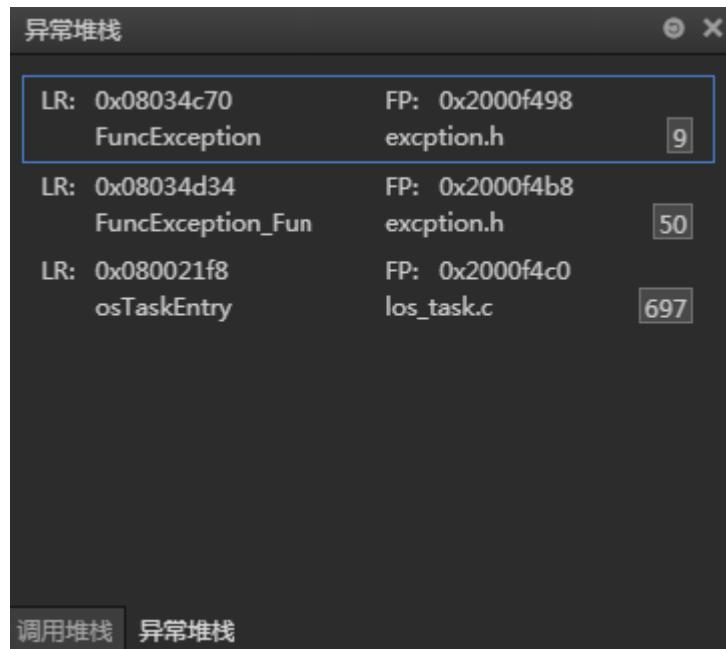
步骤5 选择导出的文件保存路径。

步骤6 单击“保存”，保存导出的Memory地址数据。

----结束

6.8.2.4 异常堆栈

系统异常中断时，在“异常堆栈”中显示发生异常所在的代码行及异常堆栈信息，单击异常信息可跳转到对应的代码行。

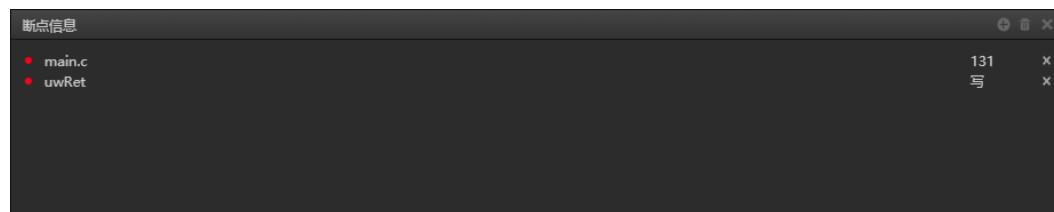


6.8.3 断点信息

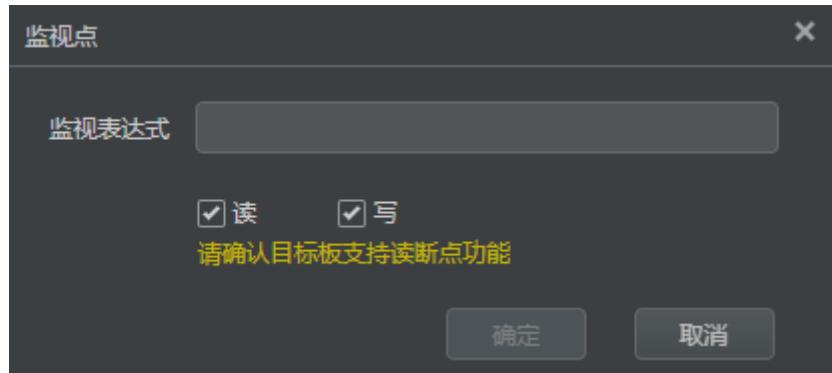
注意

- 全速运行时，对内联函数打断点，可能导致断点插入失败。
- 不建议对结构体进行监视，可能导致GDB异常，调试无法继续。

在“断点信息”面板，显示增加的断点信息及监视点。



- 单击 ，在弹出的界面中新增监视点，当监视点发生“读”或“写”变量修改时，可触发断点。



- 单击 可删除全部断点或监视点。
- 单击面板中的断点，跳转到源码和反汇编文件对应的代码行并高亮显示。不支持监视点单击跳转功能。

6.8.4 串口终端

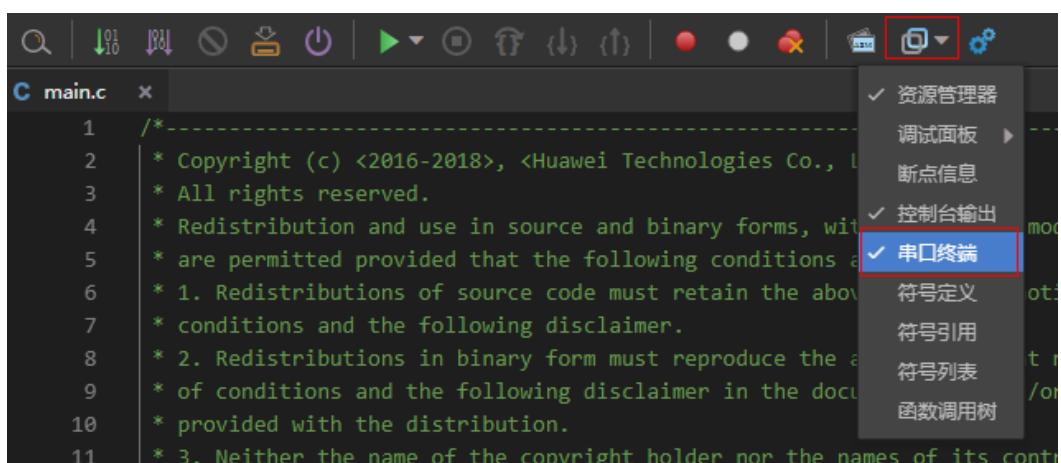
启动调试后，可进行串口调试，快速定位并解决LiteOS系统问题。

背景信息

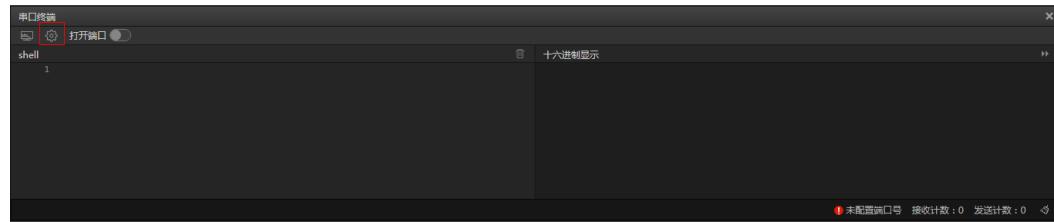
- Shell调试模式：左侧shell区域发送字符调试信息，右侧十六进制区域根据左侧调试信息转换为十六进制数据显示（根据ASCII编码规则转换）。
- 非Shell调试模式：底部发送区输入十六进制数据或字符串进行调试，左侧接收区展示接收的字符信息，右侧接收区根据左侧接收的字符信息转换成十六进制显示（根据ASCII编码规则转换）。

操作步骤

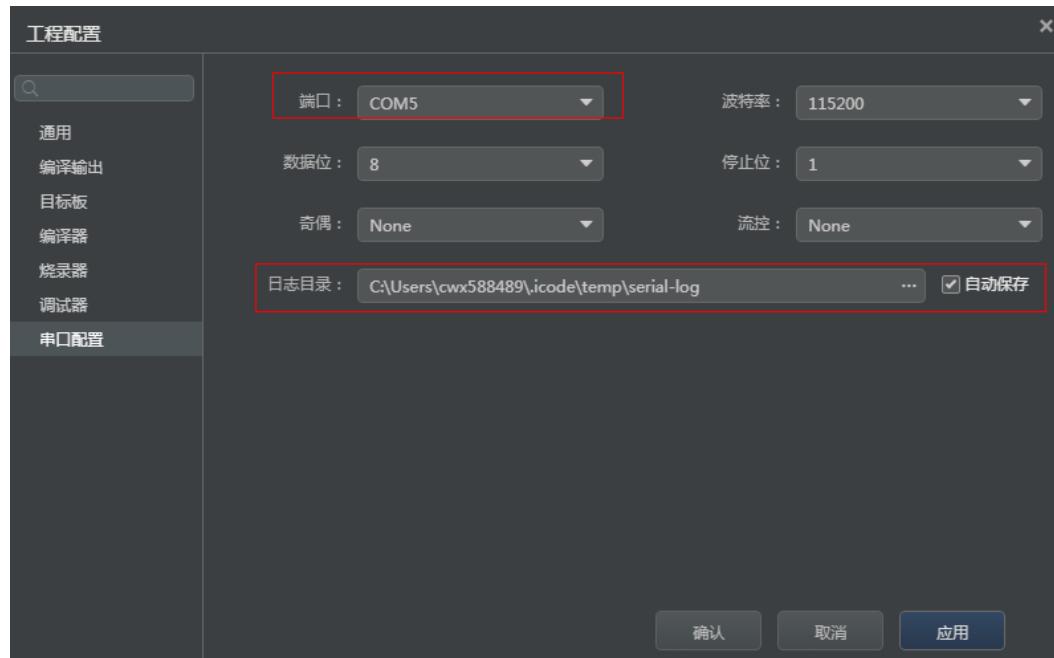
步骤1 单击工具栏 下拉框，勾选“串口终端”（或单击菜单栏的“查看>串口终端”），打开串口终端面板。



步骤2 单击 ，进行串口配置。



步骤3 在“串口配置”界面，配置串口参数。



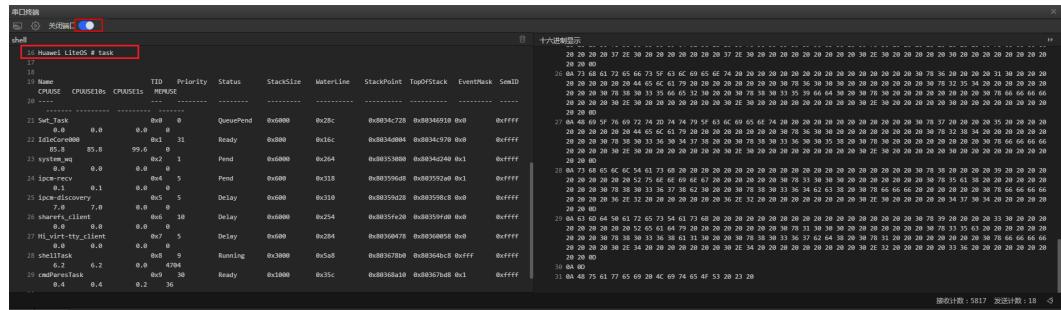
- 端口：在下拉框中选择与目标板连接的实际串口号，获取实际串口号请参见[7.1 获取连接目标板的实际串口号](#)章节。
- 波特率：115200（默认），请根据实际情况修改。
- 数据位：8（默认），请根据实际情况修改。
- 停止位：1（默认），请根据实际情况修改。
- 奇偶：None（默认），请根据实际情况修改。
- 流控：None（默认），请根据实际情况修改。
- 如需自动保存日志，请勾选“自动保存”，您也可以单击“日志目录”的...按钮，更改日志保存路径。

步骤4 打开串口终端开关 ，进行串口调试。打开串口开关后，串口相关参数不能修改。

步骤5 （可选）单击 可在shell调试模式及非shell调试模式间进行切换。

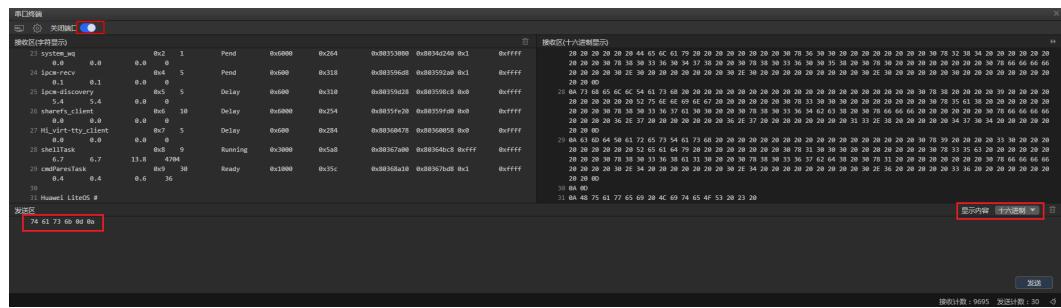
步骤6 串口调试。

- 若为shell调试模式，在左侧“shell”区域，输入字符调试信息进行调试。
如：左侧“shell”区域，输入字符“task”，右侧十六进制区域根据左侧调试信息转换为十六进制数据显示，并打开带有“task”字符的相关文件。

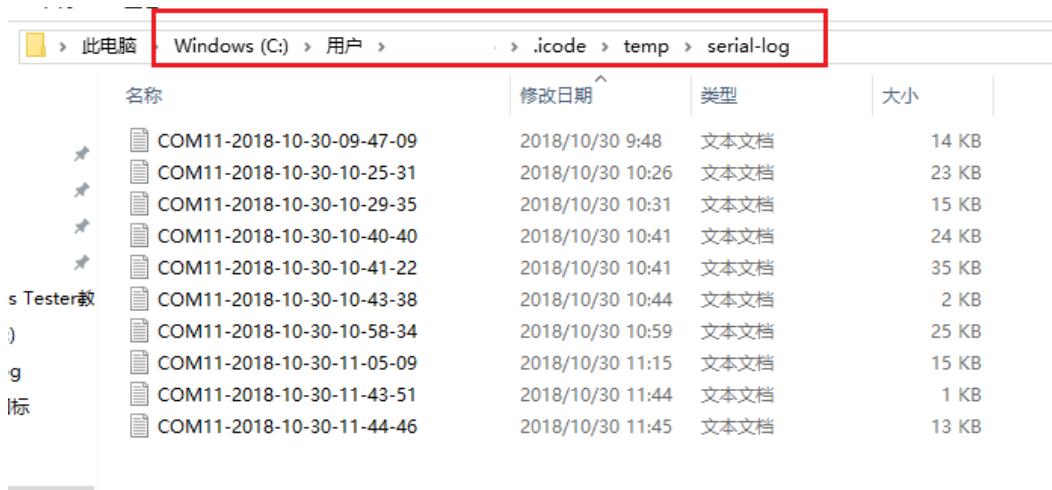


- 若为非shell调试模式，在“显示内容”下拉框中选择“字符串”或“十六进制”，在底部发送区输入字符串或十六进制数据进行调试。

如：输入字符十六进制数据“74 61 73 6B 0d”（即字符“task”），单击“发送”，左侧接收区展示接收的字符信息，右侧接收区域显示左侧字符对应的十六进制数据，并打开带有“task”字符的相关文件。



步骤7 查看串口调试数据。串口调试数据自动保存且默认保存在C盘用户目录下的“.icode\temp\serial-log”文件夹中。



存储规则：

- 最多只能存储10个文件，每个文件最大100M。
- 以“串口号+打开串口终端的时间戳”命名文件。
- 当文件大于100M时，创建新的文件存储数据。
- 当已存储10个文件时，依次删除时间戳最早的文件，保存最新生成的文件。

----结束

7 附录

7.1 获取连接目标板的实际串口号

操作步骤

- 步骤1** 将目标板与电脑连接。
步骤2 连接成功后，打开“控制面板>硬件和声音>设备管理器”



- 步骤3** 在“端口”子菜单下，找到连接设备的串口号，如下图所示：



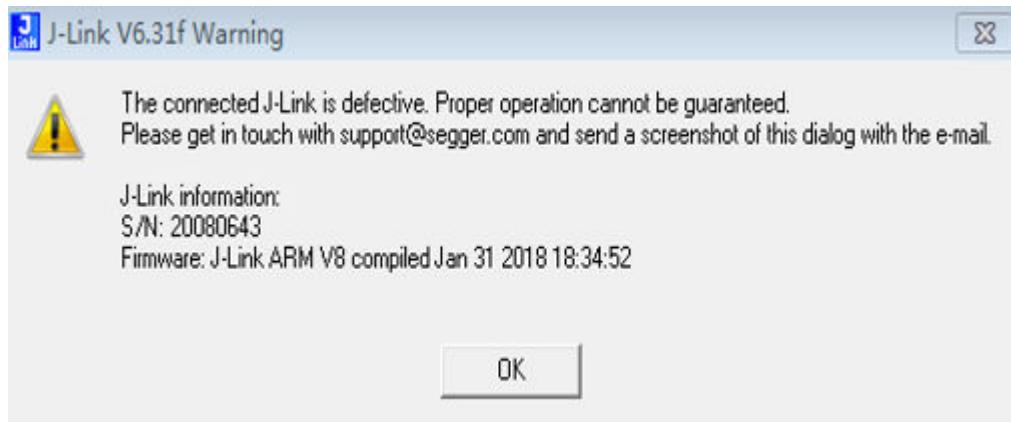
----结束

8 常见问题（针对华为海思芯片目标板）

- [8.1 弹出JLink警告如何解决](#)
- [8.2 华为海思芯片烧录/调试失败](#)
- [8.3 烧录或调试时，JLink弹出框未置顶显示](#)
- [8.4 通过LiteOS Studio修改网驱动器中的文件注意事项](#)
- [8.5 华为海思3516CV300开发板硬件如何设置](#)

8.1 弹出 JLink 警告如何解决

调试时，若弹出如下JLink警告框，请更换为V9.4及以上版本的JLink仿真器。



8.2 华为海思芯片烧录/调试失败

可能原因及解决办法

- 可能原因：目标板与JLink仿真器与电源的连接顺序错误。
解决办法：重新连接：先连接目标板的电源，再将目标板与JLink仿真器连接。
- 可能原因：JLink端口被占用。
解决办法：请检查JLink端口是否被占用，如被占用，请断开连接JLink的进程。

- 可能原因：uboot未默认启动LiteOS，导致无法进入调试。

解决办法：配置uboot环境变量。以华为海思3516CV300目标板为例，依照下图红框所示，配置如下环境变量：bootcmd=sf probe 0;sf read 0x80000000 0x100000 0x300000;go 0x80000000。

```
hisilicon # setenv bootcmd 'sf probe 0;sf read 0x80000000 0x100000 0x300000;go 0x80000000'
hisilicon # saveenv
Saving Environment to SPI Flash...
Erasing SPI flash, offset 0x00080000 size 256K ...done
Writing to SPI flash, offset 0x00080000 size 256K ...done
hisilicon # printenv bootcmd
bootcmd=sf probe 0;sf read 0x80000000 0x100000 0x300000;go 0x80000000
hisilicon #
```

- 可能原因：menuconfig未配置Gcc -g，导致编译后无法调试。

解决办法：在menuconfig中配置Gcc -g。

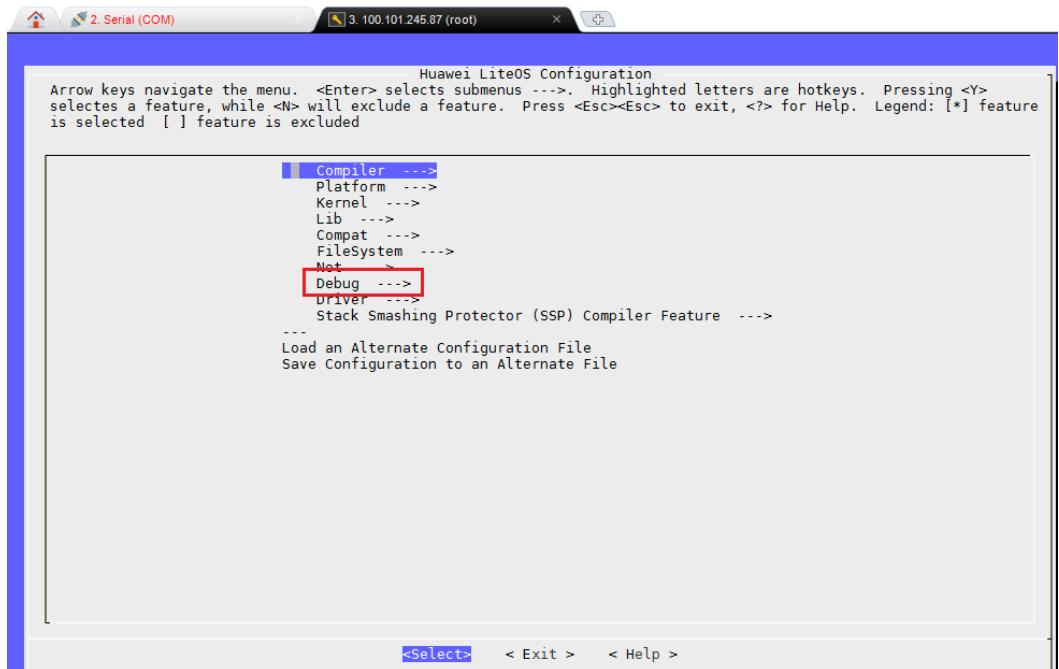
- 在工程根目录下，执行以下命令，进入menuconfig配置界面。

make menuconfig

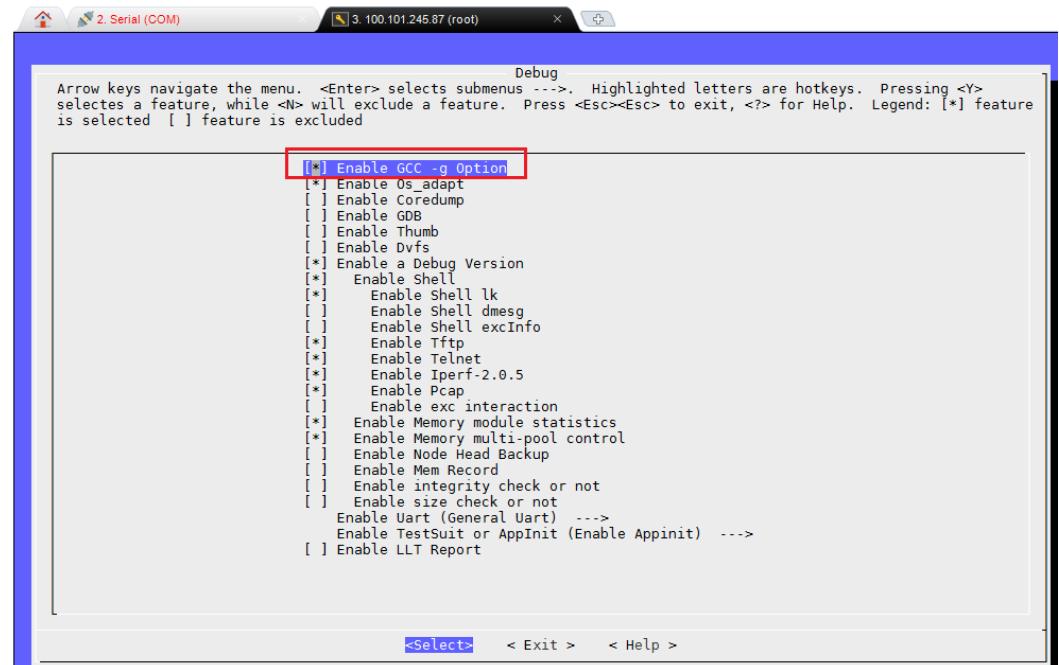
```
2. Serial (COM) 3. 100.101.245.87 (root)
Your LiteOS configuration changes were NOT saved.

make[1]: Leaving directory `/opt/code2/nocodeprotect/LiteOS_Hi3516/tools/menuconfig'
root@SX1000438370:/opt/code2/nocodeprotect/LiteOS_Hi3516# ll
total 200
drwxrwxrwx 18 nobody nogroup 4096 Oct 18 19:45 /
drwxrwxrwx 3 nobody nogroup 4096 Oct 19 16:38 .
drwxrwxrwx 3 nobody nogroup 4096 Oct 19 16:38 built/
drwxrwxrwx 5 nobody nogroup 4096 Oct 19 16:38 compat/
-rwxrwxrwx 1 root root 4761 Oct 18 19:45 .config*
-rwxrwxrwx 1 nobody nogroup 1380 Oct 19 10:31 config.mk*
drwxrwxrwx 2 nobody nogroup 4096 Oct 19 16:38 dev/
drwxrwxrwx 24 nobody nogroup 4096 Oct 19 16:38 drivers/
drwxrwxrwx 11 nobody nogroup 4096 Oct 19 16:38 fs/
drwxrwxrwx 2 nobody nogroup 4096 Oct 27 10:10 icode/
drwxrwxrwx 5 nobody nogroup 4096 Oct 19 16:38 kernel/
drwxrwxrwx 8 nobody nogroup 4096 Oct 19 16:39 lib/
drwxrwxrwx 4 nobody nogroup 4096 Oct 19 16:39 linux-config/
-rwxrwxrwx 1 nobody nogroup 4565 Oct 19 10:31 liteos.ld*
-rwxrwxrwx 1 nobody nogroup 4469 Oct 19 10:31 Makefile*
drwxrwxrwx 6 nobody nogroup 4096 Oct 19 16:39 os/
-rwxrwxrwx 1 nobody nogroup 9511 Oct 19 10:31 NUTTX_COPYING*
drwxrwxrwx 3 nobody nogroup 4096 Oct 18 19:45 out/
drwxrwxrwx 4 nobody nogroup 4096 Oct 19 16:39 platform/
-rwxrwxrwx 1 nobody nogroup 582 Oct 19 10:31 README.txt*
-rwxrwxrwx 1 nobody nogroup 81139 Oct 19 10:31 ReleaseNotes*
drwxrwxrwx 4 nobody nogroup 4096 Oct 19 16:39 sample/
drwxrwxrwx 4 nobody nogroup 4096 Oct 19 16:39 shell/
drwxrwxrwx 18 nobody nogroup 4096 Oct 19 16:40 tools/
drwxrwxrwx 4 nobody nogroup 4096 Oct 19 16:40 windows-config/
root@SX1000438370:/opt/code2/nocodeprotect/LiteOS_Hi3516# make menuconfig
make -C tools/menuconfig menuconfig
make[1]: Entering directory `/opt/code2/nocodeprotect/LiteOS_Hi3516/tools/menuconfig'
#
# using defaults found in /opt/code2/nocodeprotect/LiteOS_Hi3516/.config
#
Your LiteOS configuration changes were NOT saved.
```

- 进入“Debug”目录。



- c. 按空格键选中“Enable GCC -g Option”选项，并选中“Exit”退出。



- d. 选中“Yes”保存配置。



8.3 烧录或调试时，JLink 弹出框未置顶显示

现象

烧录或调试过程中，JLink弹出框未置顶显示（被LiteOS Studio窗体遮挡），如下图所示：

The screenshot shows the LiteOS Studio interface. On the left, there is a code editor window titled "STM32F429IGTx_LiteOS.Id" displaying C code for memory initialization. On the right, there is a "Disassembly/File" window showing assembly code. Below these windows, there is a "输出" (Output) section containing build logs. The logs show the compilation process, including linking and generating hex files. At the bottom, there is a taskbar with various icons, one of which is highlighted with a red box.

```
STM32F429IGTx_LiteOS.Id
114 /* used by the startup to initialize liteos vector */
115 _si_liteos_vector_data = LOADADDR(.vector_ram);
116
117 /* Initialized liteos vector sections goes into RAM, load LMA copy
118 .vector_ram :
119 {
120     . = ORIGIN(RAM);
121     _s_liteos_vector = .;
122     *(.data.vector) /* liteos vector in ram */
123     _e_liteos_vector = .;
124 } > RAM AT> FLASH
125
126 /* used by the startup to initialize data */
127 _sidata = LOADADDR(.idata);
128
129 /* Initialized data sections goes into RAM, load LMA copy after
130 .data :
131 {
132     . = ALIGN(4);
133     _sdata = .; /* create a global symbol at data start */
134     *(.data) /* .data sections */
135     *(.data*) /* .data* sections */
136
137     . = ALIGN(4);
138     _edata = .; /* define a global symbol at data end */
139
140 \_SEAM AT> FLASH
```

```
DisassemblyFile
114618 }
114619     return uwRet;
114620 802c69a: 69fb      ldr r3, [r7, #28]
114621 }
114622 802c69c: 4618      mov r0, r3
114623 802c69e: 3720      adds r7, #32
114624 802c6a0: 46bd      mov sp, r7
114625 802c6a2: bd88      pop {r7, pc}
114626 802c6a4: 00037728 .word 0x00037728
114627 802c6a8: 0002c655 .word 0x0002c655
114628 802c6ac: 2000dc20 .word 0x2000dc20
114629
114630 0002c6b0 <main>;
114631         return uwRet;
114632 }
114633
114634 int main(void)
114635 {
114636     802c6b0: b500      push {r7, lr}
114637     802c6b2: b082      sub sp, #8
114638     802c6b4: af00      add r7, sp, #0
114639
114640     #ifdef SYSVIEW_TRACE
114641         SEGGER_SYSVIEW_Conf();
114642     802c6b6: f002 fa15    bl 0002ea4 <SEGGER_SYSVIEW_Conf>
114643     #endif
```

输出

```
[2018-09-25 21:12:18] 编译耗时: 285 / ms
arm-none-eabi-size build/Huawei_LiteOS.elf
text data bss dec hex filename
234184 1812 56504 292500 47694 build/Huawei_LiteOS.elf
arm-none-eabi-objcopy -O ihex build/Huawei_LiteOS.elf build/Huawei_LiteOS.hex
arm-none-eabi-objcopy -O binary -S build/Huawei_LiteOS.elf build/Huawei_LiteOS.bin

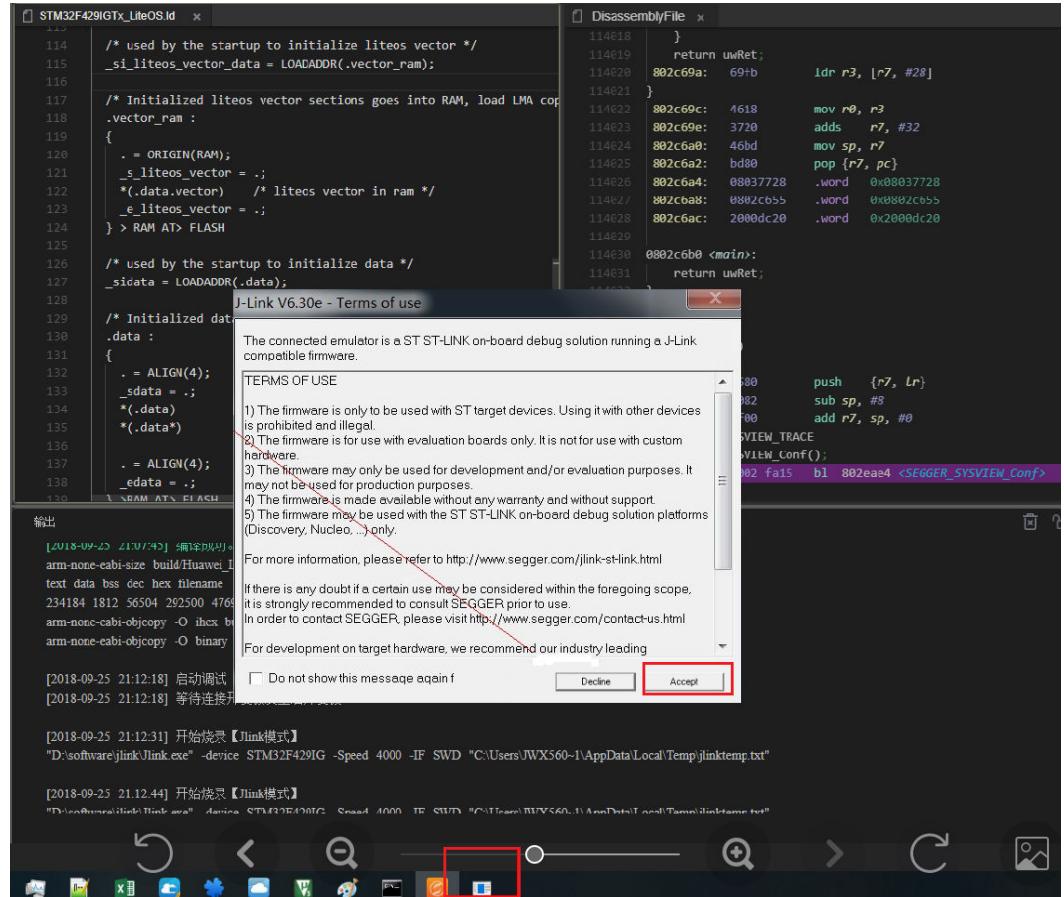
[2018-09-25 21:12:18] 启动调试
[2018-09-25 21:12:18] 等待连接开发板及重启开发板

[2018-09-25 21:12:31] 开始烧录【Jlink模式】
"D:\software\jlink\Jlink.exe" -device STM32F429IG -Speed 4000 -IF SWD "C:\Users\JWX560-1\AppData\Local\Temp\jlinktemp.txt"

[2018-09-25 21:12:44] 开始烧录【Jlink模式】
"Jlink.exe" -device STM32F429IG -Speed 4000 -IF SWD "C:\Users\JWX560-1\AppData\Local\Temp\jlinktemp.txt"
```

原因及解决办法

该问题为Windows系统跨应用消息窗体置顶显示兼容性问题，Windows任务仍会出现相应消息窗体的任务图标，可通过点击任务栏对应的图标，让消息窗体显示出来，即可进行下一步操作，如下图所示：



8.4 通过 LiteOS Studio 修改网驱动器中的文件注意事项

修改网络驱动器中的文件内容，请先在源码工程根目录中执行以下命令开通修改权限。

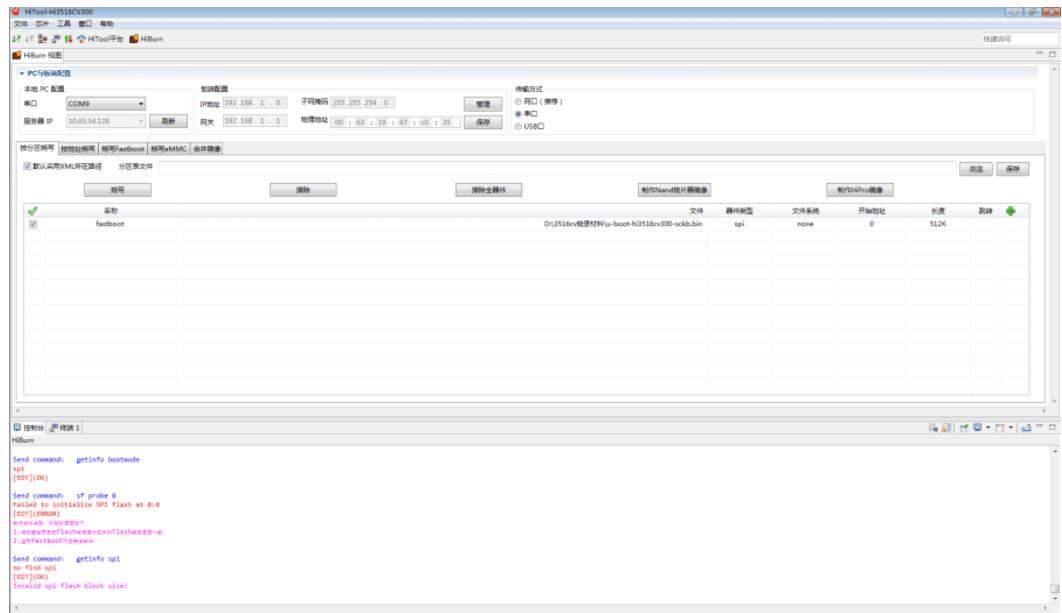
chmod 777 -R .

```
root@SX1000438370:/opt/code2/Huawei_LiteOS-bak# chmod 777 -R .
root@SX1000438370:/opt/code2/Huawei_LiteOS-bak# ll
total 140
drwxrwxrwx 16 1009 users 4096 Sep 26 16:46 /
drwxrwxrwx 14 root root 4096 Sep 28 18:03 ../
drwxrwxrwx 3 1009 users 4096 May 3 19:04 build/
drwxrwxrwx 5 1009 users 4096 May 3 19:04 compat/
drwxrwxrwx 1 1009 users 3333 May 3 19:08 .config*
drwxrwxrwx 1 1009 users 1807 May 3 19:04 config.mk*
drwxrwxrwx 2 1009 users 4096 May 3 19:04 doc/
drwxrwxrwx 18 1009 users 4096 May 3 19:04 drivers/
drwxrwxrwx 10 1009 users 4096 May 3 19:04 fs/
drwxrwxrwx 2 nobody nogroup 4096 Sep 26 17:25 .icode/
drwxrwxrwx 5 1009 users 4096 May 3 19:04 kernel/
drwxrwxrwx 12 1009 users 4096 May 3 19:04 lib/
drwxrwxrwx 1 1009 users 4020 May 3 19:04 liteosexc.ld*
drwxrwxrwx 1 1009 users 4016 May 3 19:04 liteos.ld*
drwxrwxrwx 1 1009 users 3188 May 3 19:04 Makefile*
drwxrwxrwx 6 1009 users 4096 May 3 19:04 net/
drwxrwxrwx 1 1009 users 9511 May 3 19:04 NUTTX_COPYING*
drwxrwxrwx 4 1009 users 4096 May 3 19:08 out/
drwxrwxrwx 4 1009 users 4096 May 3 19:04 platform/
drwxrwxrwx 1 1009 users 44869 May 3 19:04 ReleaseNotes*
drwxrwxrwx 4 1009 users 4096 May 3 19:04 sample/
drwxrwxrwx 5 1009 users 4096 May 3 19:06 shell/
drwxrwxrwx 10 1009 users 4096 May 3 19:04 tools/
```

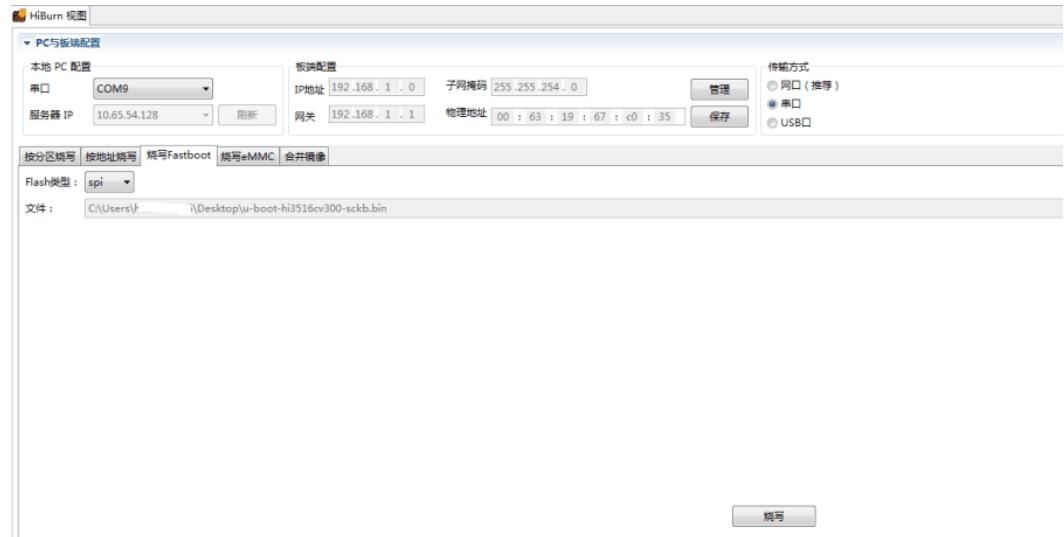
8.5 华为海思 3516CV300 开发板硬件如何设置

用hitool工具，在华为海思3516CV300开发板中烧录u-boot。

1. 打开hitool工具，选择芯片为hi3516cv300。



2. 选择开发板与PC连接的串口号，传输方式选择“串口”，在“烧写Fastboot”区域下，Flash类型为“spi”，“文件”选择相应的u-boot文件，单击“烧写”烧写u-boot。



3. 设置华为海思3516CV300开发板拨码规则。

sw1从上往下是[4:0]0000

sw2从上往下是[4:0]0101

sw3从上往下是[4:0]0000