物联网开发平台

设备开发指南

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2020 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

设备开发指南
版本变更

最近更新时间：2020-03-20 16:44:25

版本 V3.1.0

- 发布日期：2020/3/17
- 开发语言：Java
- 开发环境：Android
- 内容如下：
  - 实现 iot_explorer 模块提供 Android 设备的物联网开发平台接入能力。
  - 支持包括数据模板的基本功能（属性、事件、行为等）、网关代理功能等。
  - 提供相应的入门文档以及使用说明。

版本 V3.1.0

- 发布日期：2019/11/14
- 开发语言：C 语言
- 开发环境：Linux，GNU Make
- 内容如下：
  - 数据模板通信协议更新为 rpc 模式。
  - 事件合为数据模板 client 的成员。
  - 支持 Action。
  - 数据模板相关示例更新。
  - 编译方式支持 cmake，新增 Windows、FreeRTOS、nonOS 的平台支持。
  - 增加支持通用 TCP 模组的网络接入的 AT 适配框架。
  - 删减非 IoT explorer 平台相关示例及代码。
  - 更新相关文档。
  - 物联网开发平台（IoT Explorer）设备端 C-SDK 剥离单独的代码 Git。
  - SDK 版本号更新为3.1.0。

版本 V3.0.3

- 发布日期：2019/08/26
- 开发语言：C 语言
- 开发环境：Linux，GNU Make
- 内容如下：
  - 支持 OTA 断点续传：ota_mqtt_sample.c 示例增加本地固件版本信息管理（版本、断点、MD5），固件下载建立 HTTPS 连接时支持 range 参数。

版权所有：腾讯云计算（北京）有限责任公司

- 设备影子去除设备侧 version 管理。
- SDK 版本号更新为3.0.3。

版本 V3.0.2

- 发布日期：2019/07/18
- 开发语言：C 语言
- 开发环境：Linux，GNU Make
- 内容如下：
  - 数据模板字符串类型支持转义字符处理。
  - 设备影子去除设备侧 version 管理。
  - 优化数据模板相关示例。

版本 V3.0.1

- 发布日期：2019/06/11
- 开发语言：C 语言
- 开发环境：Linux，GNU Make
- 内容如下：
  - 日志上报功能优化，动态分配缓冲区内存，支持较大日志分段上报，适合多种使用场景。
  - MQTT 增加 subscribe 的 event handler 回调，及时通知订阅 topic 的状态变化。
  - 修复一些代码问题，例如对 MQTT API 的返回值判断不当问题。

版本 V3.0.0

- 发布日期：2019/05/13
- 开发语言：C 语言
- 开发环境：Linux，GNU Make
- 腾讯 IoT Explorer 设备接入 SDK 基于 IoT Hub 设备 SDK 开发，新增特性：
  - 新增数据模板功能及对应示例、数据模板代码生成脚本。
  - 新增事件上报功能及对应示例。
  - 完善 JSON 数据处理。
  - 该版本及以后的版本对应支持腾讯 Explorer 开发平台。

版权所有：腾讯云计算（北京）有限责任公司

SDK 说明及下载

最近更新时间：2020-06-30 13:08:59

概述

腾讯云物联网开发平台针对不同的设备开发场景，提供了多版本语言的设备 SDK 供客户使用：

- `qcloud-iot-explorer-sdk-embedded-c` 是针对嵌入式设备接入使用腾讯 IoT Explorer 平台提供的 C 语言版本 SDK。
- `qcloud-iot-explorer-sdk-android` 是针对智能设备接入使用腾讯 IoT Explorer 平台的 Android 版本 SDK。

C SDK 代码托管

- 自 V3.1.0 版本开始，使用独立的 Github 托管 C 设备 SDK 代码。
- 请下载最新版 C-SDK。
- SDK 3.1.0之前的 C SDK 版本 访问此处。

> 注意：
> V3.1.0之前的版本相较于3.1.0及以后的版本代码 Git 路径不一样，同时与平台交互协议有较大区别。

Android SDK 代码托管

自 V3.1.0 版本开始，使用独立的 Github 托管 Android 设备 SDK 代码。

5G SDK 代码托管

- 腾讯云物联开发平台（IoT Explorer）5G SDK 是腾讯5G物联开发套件的设备端组件，通过与 IoT Explorer 配合，为宽带物联应用提供5G模组远程运维，接入腾讯边缘接入和网络加速平台（TSEC）实现 VPN 组网，空口加速等功能，降低用户使用5G网络及边缘计算的门槛。
- 自 V0.1.0 版本开始，使用独立的 Github 托管 5G SDK 代码。

C SDK 使用说明

版权所有：腾讯云计算（北京）有限责任公司

SDK 分四层设计，从上至下分别为平台服务层、核心协议层、网络层、硬件抽象层。

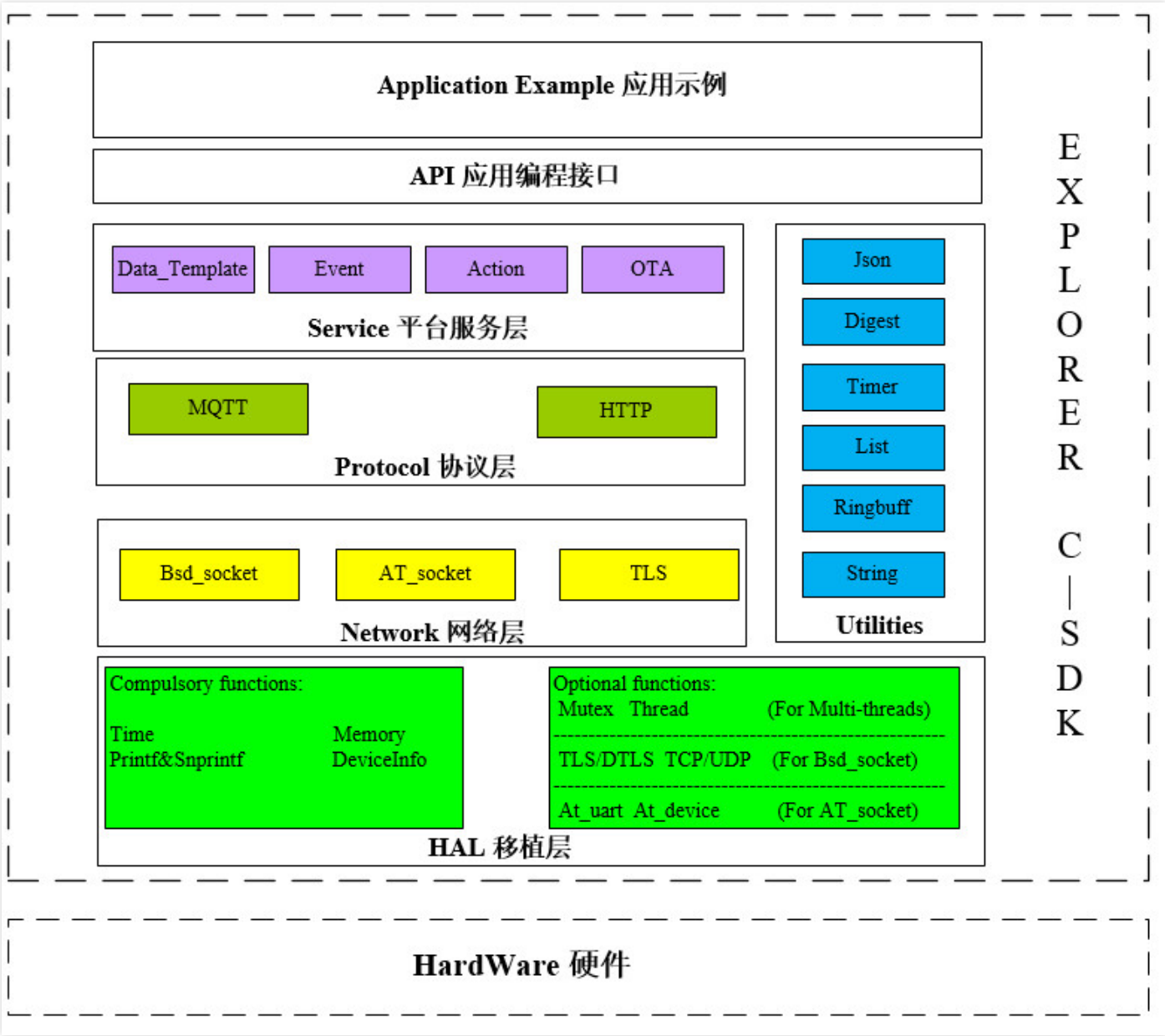
- 设备侧和 IoT Explorer 平台交互的核心协议为 MQTT，基于此核心协议，实现了数据模板和 OTA 功能。平台通过对 IoT 设备的通用抽象定义了 数据模板协议，云端和设备基于 MQTT 的 payload 承载的数据实现数据模板协议数据流交互。OTA 功能的升级命令、版本及固件信息通过 MQTT 协议通道交互，固件下载通过 HTTPS 协议通道交互。

- 网络层的实现支持基于 bsd_socket 方式和 AT_socke t方式,对于资源丰富系统本身集成 TCP/IP 或 LwIP 网络协议栈的，可以选择 bsd_socket 的网络接口。对于部分资源受限设备，通过通信模组（蜂窝模组/WIFI模组等）和 MCU 交互而实现网络接入的可以选择 SDK 提供 at_socket 框架，SDK 未支持的通信模组参照 SDK 支持的通信模组实现 at_device 结构体 at_device_op_t 里的驱动接口即可。

- 硬件抽象层需要针对具体的软硬件平台开展移植，分为必须实现和可选实现两部分 HAL 层接口。其中必选实现的接口为时间（获取毫秒数）、打印、格式化打印、内存操作、设备信息读写。可选实现接口，使用 RTOS 的，需要实现锁、信号量、线程创建及销毁、延时睡眠。使用 AT_Socket 接入网络的，需要实现 AT 串口驱动及模组驱动。SDK 已经支持 Linux、Windows、FreeRTOS、nonOS 四种典型环境的 HAL 层示例移植实现，在 Linux 和

版权所有：腾讯云计算（北京）有限责任公司

Windows 环境可以直接编译运行相关示例。



C-SDK 跨平台移植
设备接入类型说明

最近更新时间：2020-08-21 14:20:13

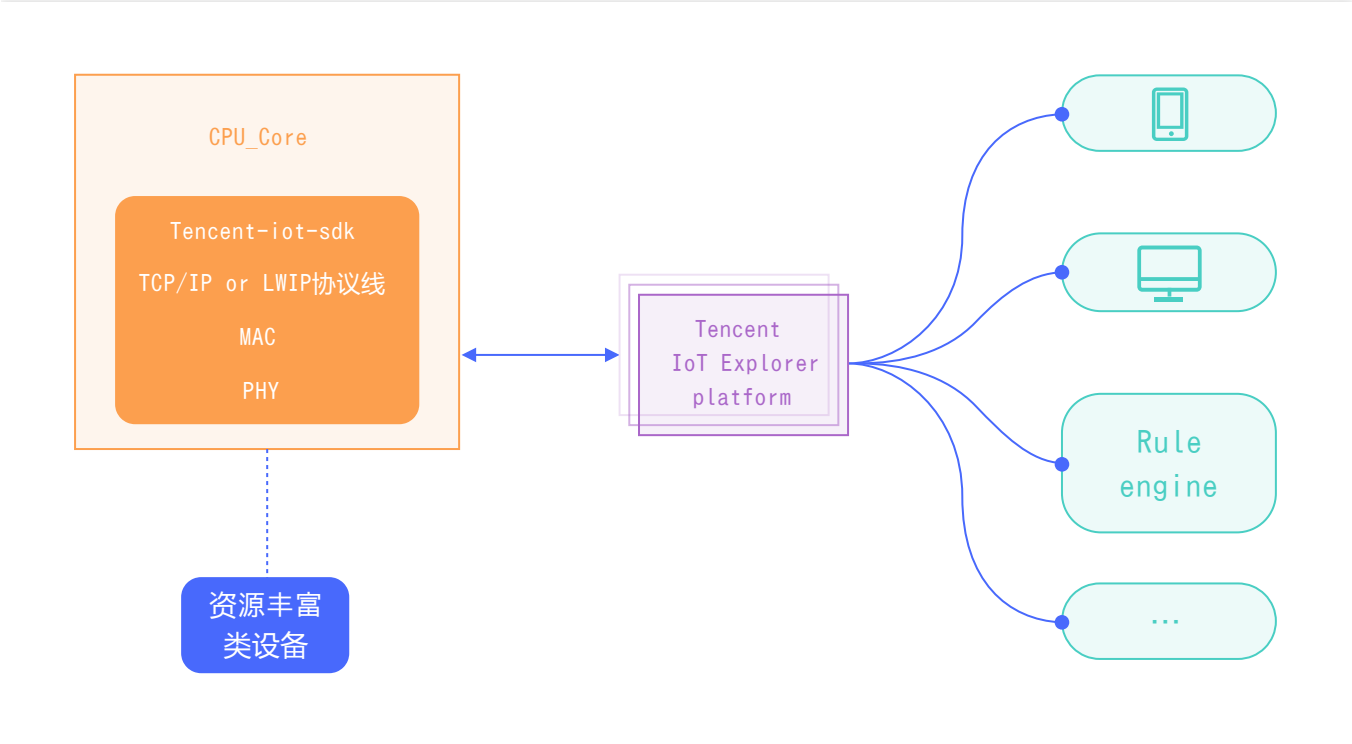
本文对接入腾讯物联网开发平台（下文称 IoT Explorer）的设备类型进行说明，并介绍各设备类型如何移植 C-SDK。

IoT 设备联网类型

IoT 设备首先得具备接入网络的能力，接入网络必须要有 TCP/IP 协议栈，从 TCP/IP 协议栈承载的载体区分，IoT 设备分为如下三类：

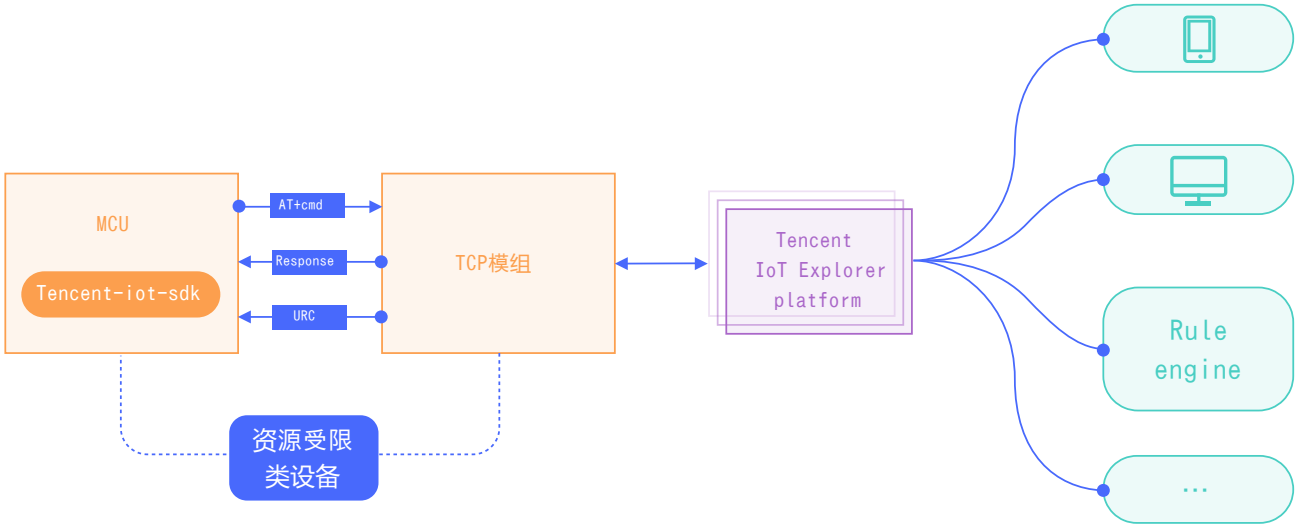
类型一

TCP/IP 协议栈运行在主芯片上，主芯片的处理能力和资源较丰富，譬如路由器，本文称作**资源丰富类设备。**



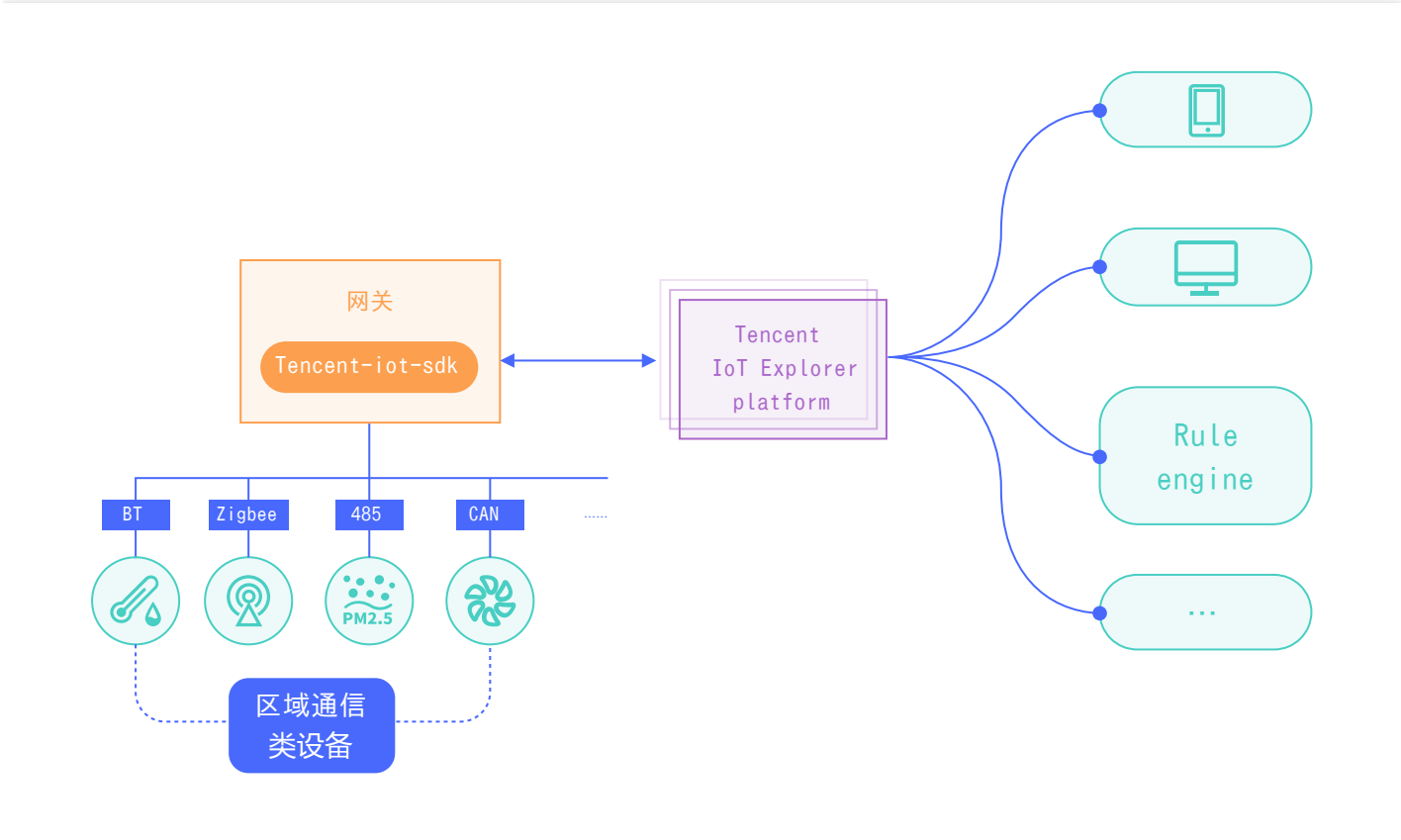
类型二

TCP/IP 协议栈运行在通信模组上，主芯片处理能力和资源特别有限，例如 STM32F103 系列，本文称作**资源受限类设备。**



类型三

TCP/IP 协议栈运行在网关上，设备本身并不能直接接入网络，通过有线（485/CAN 等）或无线（Ble/ZigBee 等）的方式和网关交互，网关（设备类型一或设备类型二），再将数据转发到服务端，这类设备本文称作**区域通信类设备**。



资源丰富类设备接入 IoT Explorer

C-SDK 可以在 Windows 和 Linux 环境直接编译并运行示例，SDK 示例了在 Windows 和 Linux 两个平台 HAL 层移植实现。常见的 FreeRTOS+lwip 的场景和 HAL 层的示例实现，请参见 C-SDK_Porting 跨平台移植概述。

资源受限类设备接入 IoT Explorer

资源受限类设备，借助于通信模组实现网络访问，即 MCU+ 模组方式。模组一般是蜂窝模组（2/3/4/5G）或者 Wi-Fi 模组，市面可选的模组较多，各家的 AT 指令也各不相同，为此我们提供两种方式实现平台接入。

- 第一种：基于 SDK 提供的 AT_Socket 框架和模组的通用 TCP 指令，参照 at_device 目录下已支持的模组，实现 AT_Device 驱动的结构体 at_device_op_t 对应的驱动接口即可，具体请参见 MCU+通用 TCP_AT 模组移植。
- 第二种：我们与主流的模组厂商进行深度合作，将 SDK 的核心协议移植到模组中，模组对外封装统一的腾讯云 AT 指令，同时我们为 MCU 提供实现和定制模组交互的 SDK，具体请参见 MCU+定制MQTT AT模组移植。

区域通信类设备接入 IoT Explorer

对于区域通信类设备，网关需要实现 SDK 的移植，SDK 针对网关提供对应的协议逻辑代理子设备上线、下线及协议数据交互，网关实现和子设备通信方式和数据格式可根据场景定义，网关将子设备数据及平台下行数据做相应转换后实现上传下达。网关设备归属于类型一或类型二设备，可以对应参照实现 SDK 移植。

版权所有：腾讯云计算（北京）有限责任公司

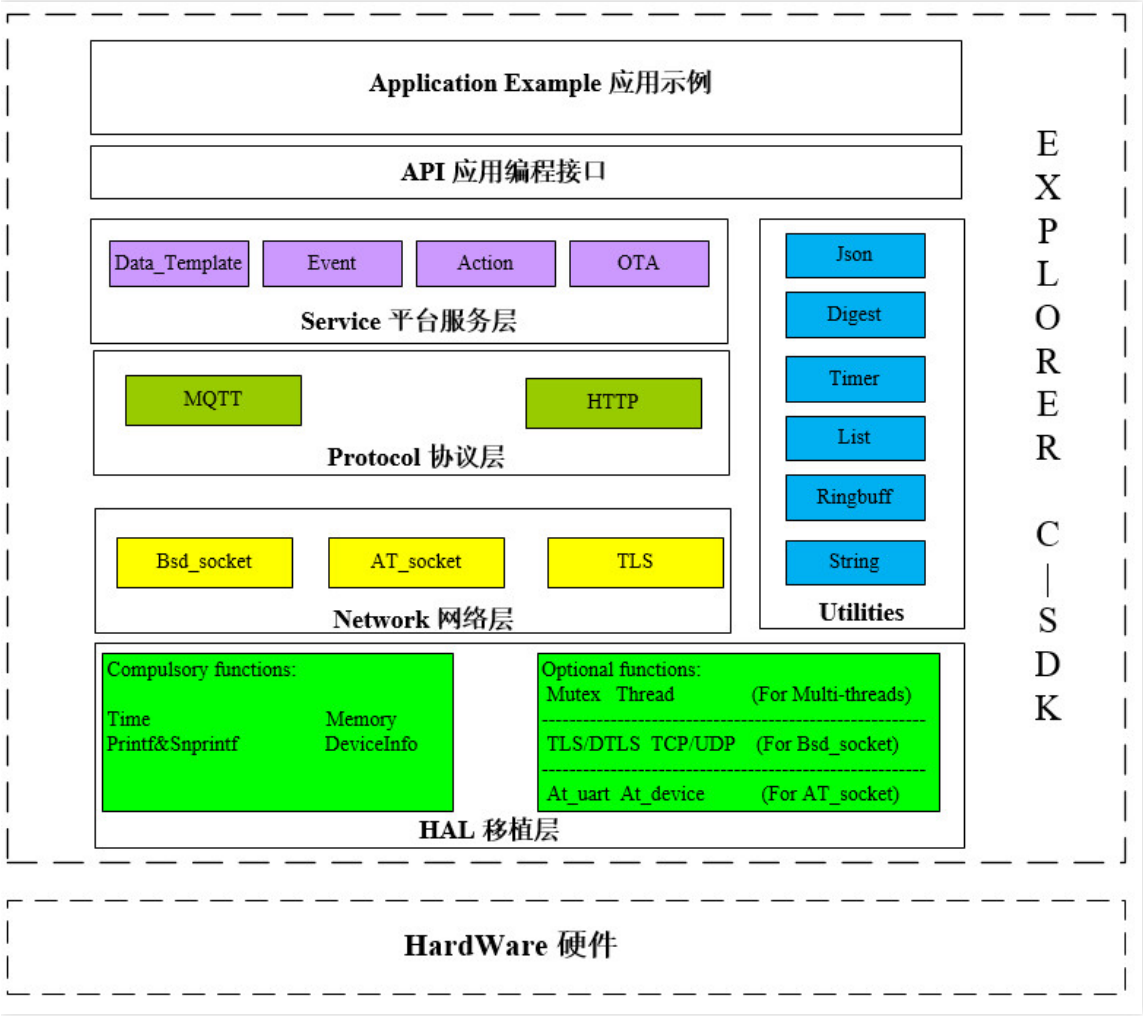
C-SDK_Porting 跨平台移植概述

最近更新时间：2020-06-29 16:18:50

本文档介绍如何将设备端 C-SDK 移植到目标硬件平台。C-SDK 采用模块化设计，分离核心协议服务与硬件抽象层，在进行跨平台移植时，一般只需对硬件抽象层进行修改适配即可。

C-SDK 架构

架构图



架构说明

SDK 分四层设计，从上至下分别为平台服务层、核心协议层、网络层、硬件抽象层。

- **服务层**
  在网络协议层之上，实现了包括设备接入鉴权，设备影子，网关，动态注册，日志上报和 OTA 等功能。

- **协议层**
  设备端和 IoT 平台交互的网络协议包括 MQTT/COAP/HTTP。
- **网络层**
  实现基于 TLS/SSL（TLS/DTLS），POSIX_socket（TCP/UDP）和 AT_socket 方式的网络协议栈，不同服务可根据需要使用不同的协议栈接口函数。
- **硬件抽象层**
  实现对不同硬件平台的底层操作抽象封装，需要针对具体的软硬件平台开展移植，分为必须实现和可选实现两部分 HAL 层接口。

硬件抽象层移植

HAL 层主要有几大块的移植，分别是 OS 相关、网络及 TLS 相关、时间及打印相关、设备信息相关。
SDK 在 **platform/os** 目录示例了 Linux、Windows、FreeRTOS 及 nonOS 四个场景的硬件抽象层实现，可以参考最相近的目录展开目标平台的移植。

OS 相关接口

序号	函数名	说明
1	HAL_Malloc	动态申请内存块
2	HAL_Free	释放内存块
3	HAL_ThreadCreate	线程创建
4	HAL_ThreadDestroy	线程销毁
5	HAL_MutexCreate	创建互斥锁
6	HAL_MutexDestroy	销毁互斥锁
7	HAL_MutexLock	mutex 加锁
8	HAL_MutexUnlock	mutex 解锁
9	HAL_SemaphoreCreate	创建信号量
10	HAL_SemaphoreDestroy	销毁信号量
11	HAL_SemaphoreWait	等待信号量
12	HAL_SemaphorePost	释放信号量
13	HAL_SleepMs	休眠

版权所有：腾讯云计算（北京）有限责任公司

网络及 TLS 相关的 HAL 接口

网络相关接口提供二选一的适配移植。对于具备网络通讯能力并且本身集成 TCP/IP 网络协议栈的设备，需要实现 POSIX_socket 的网络 HAL 接口，使用 TLS/SSL 加密通讯的还需要实现 TLS 相关的 HAL 接口。而对于 **MCU+ 通用 TCP_AT 模组** 的设备，则可以选择 SDK 提供的 AT_Socket 框架，并实现相关的 AT 模组接口。

基于 POSIX_socket 的 HAL 接口

其中 TCP/UDP 相关接口基于 POSIX socket 函数实现。TLS 相关接口依赖于 **mbedtls** 库，移植之前必须确保系统上有可用的 **mbedtls** 库。如果采用其他 TLS/SSL 库，可参考 **platform/tls/mbedtls** 相关实现进行移植适配。UDP/DTLS 相关的函数仅在使能 **COAP** 通讯的时候才需要移植。

序号	函数名	说明
1	HAL_TCP_Connect	建立 TCP 连接
2	HAL_TCP_Disconnect	断开 TCP 连接
3	HAL_TCP_Write	TCP 写
4	HAL_TCP_Read	TCP 读
5	HAL_TLS_Connect	建立 TLS 连接
6	HAL_TLS_Disconnect	断开 TLS 连接
7	HAL_TLS_Write	TLS 写
8	HAL_TLS_Read	TLS 读
9	HAL_UDP_Connect	建立 TCP 连接
10	HAL_UDP_Disconnect	断开 TCP 连接
11	HAL_UDP_Write	UDP 写
12	HAL_UDP_Read	UPD 读
13	HAL_DTLS_Connect	建立 DTLS 连接
14	HAL_DTLS_Disconnect	断开 DTLS 连接
15	HAL_DTLS_Write	DTLS 写
16	HAL_DTLS_Read	DTLS 读

基于 AT_socket 的 HAL 接口

通过使能编译宏 **AT_TCP_ENABLED** 选择 AT_socket，则 SDK 会调用 `network_at_tcp.c` 的 `at_socket` 接口，

at_socket 层不需要移植，需要实现 AT 串口驱动及 AT 模组驱动，AT 模组驱动只需要实现 AT 框架中 at_device 的驱动结构体 `at_device_op_t` 的驱动接口即可，可以参照 at_device 目录下的已支持的模组。AT 串口驱动需要实现串口的中断接收，然后在中断服务程序中调用回调函数 `at_client_uart_rx_isr_cb` 即可，可以参考 `HAL_AT_UART_freertos.c` 实现目标平台的移植。

序号	函数名	说明
1	HAL_AT_Uart_Init	初始化 AT 串口
2	HAL_AT_Uart_Deinit	去初始化 AT 串口
3	HAL_AT_Uart_Send	AT 串口发送数据
4	HAL_AT_UART_IRQHandler	AT 串口接收中断服务程序

时间及打印相关的 HAL 接口

序号	函数名	说明
1	HAL_Printf	将格式化的数据写入标准输出流中
2	HAL_Snprintf	将格式化的数据写入字符串
3	HAL_UptimeMs	检索自系统启动以来已运行的毫秒数
4	HAL_DelayMs	阻塞延时，单位毫秒

设备信息相关的 HAL 接口

接入 IoT 平台需要在平台创建产品和设备信息，同时需要将产品及设备信息保存在设备侧的非易失存储介质。可以参考 `platform/os/linux/HAL_Device_linux.c` 示例实现。

序号	函数名	说明
1	HAL_GetDevInfo	设备信息读取
2	HAL_SetDevInfo	设备信息保存

版权所有：腾讯云计算（北京）有限责任公司

FreeRTOS+lwIP 平台移植说明

最近更新时间：2020-06-29 16:18:15

本文档介绍如何将腾讯云物联 C-SDK 移植到 **FreeRTOS+lwIP** 平台。

简介

FreeRTOS 作为一个微内核系统，主要提供任务创建及调度和任务间通信等 OS 核心机制，在不同设备平台还需要搭配多个软件组件包括 C 运行库（例如 newlib 或者 ARM CMSIS 库）和 TCP/IP 网络协议栈（如 lwIP）才能形成完整的嵌入式运行平台。同时各个设备平台的编译开发环境也各不相同，因此在移植 C-SDK 时，需要根据不同设备的具体情况进行适配。

> 说明：
> SDK 在 **platform/os/freertos** 里提供了一个基于 **FreeRTOS+lwIP+newlib** 的参考实现，该实现已在乐鑫 ESP8266 平台上验证测试。

抽取代码

因为基于 RTOS 系统的平台编译方式各不相同，一般无法直接使用 SDK 的 cmake 或者 make 编译，因此 SDK 提供了代码抽取功能，可根据需要将相关代码抽取到一个单独的文件夹，文件夹里面的代码层次目录简洁，方便用户拷贝集成到自己的开发环境。

1. 修改 CMakeLists.txt 中配置为 freertos 平台，并开启代码抽取功能：

```
set(BUILD_TYPE "release")
set(PLATFORM "freertos")
set(EXTRACT_SRC ON)
set(FEATURE_AT_TCP_ENABLED OFF)
```

2. 在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

3. 即可在 output/qcloud_iot_c_sdk 中，找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│ ├── config.h
│ ├── exports
├── platform
└── sdk_src
└── internal_inc
```

说明：

- include 目录：SDK 供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译宏。API 具体介绍请参考 接口及可变参数说明。
- platform目录：平台相关的代码，可根据设备的具体情况进行修改适配。具体的函数说明请参考 C-SDK_Porting 跨平台移植概述。
- sdk_src：SDK 的核心逻辑及协议相关代码，一般不需要修改，其中 internal_inc 为 SDK 内部使用的头文件。

4. 用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

移植示例

在 Linux 开发环境基于乐鑫 ESP8266 RTOS 平台搭建一个工程示例。

1. 请参考 ESP8266_RTOS_SDK 获取 RTOS_SDK 和交叉编译器，并创建一个项目工程。
2. 将上面抽取的 qcloud_iot_c_sdk 目录，拷贝到 components/qcloud_iot 下。
3. 在 components/qcloud_iot 下，新建一个编译配置文件 component.mk，内容如下：

```
#
# Component Makefile
#
COMPONENT_ADD_INCLUDEDIRS := \
qcloud_iot_c_sdk/include \
qcloud_iot_c_sdk/include/exports \
qcloud_iot_c_sdk/sdk_src/internal_inc
COMPONENT_SRCDIRS := \
qcloud_iot_c_sdk/sdk_src \
qcloud_iot_c_sdk/platform
```

至此，您可以将 qcloud_iot_c_sdk 作为一个组件进行编译了，之后在用户代码里面就可以调用物联 C-SDK 的接口进行连接和收发消息。

版权所有：腾讯云计算（北京）有限责任公司

MCU+定制MQTT AT模组移植

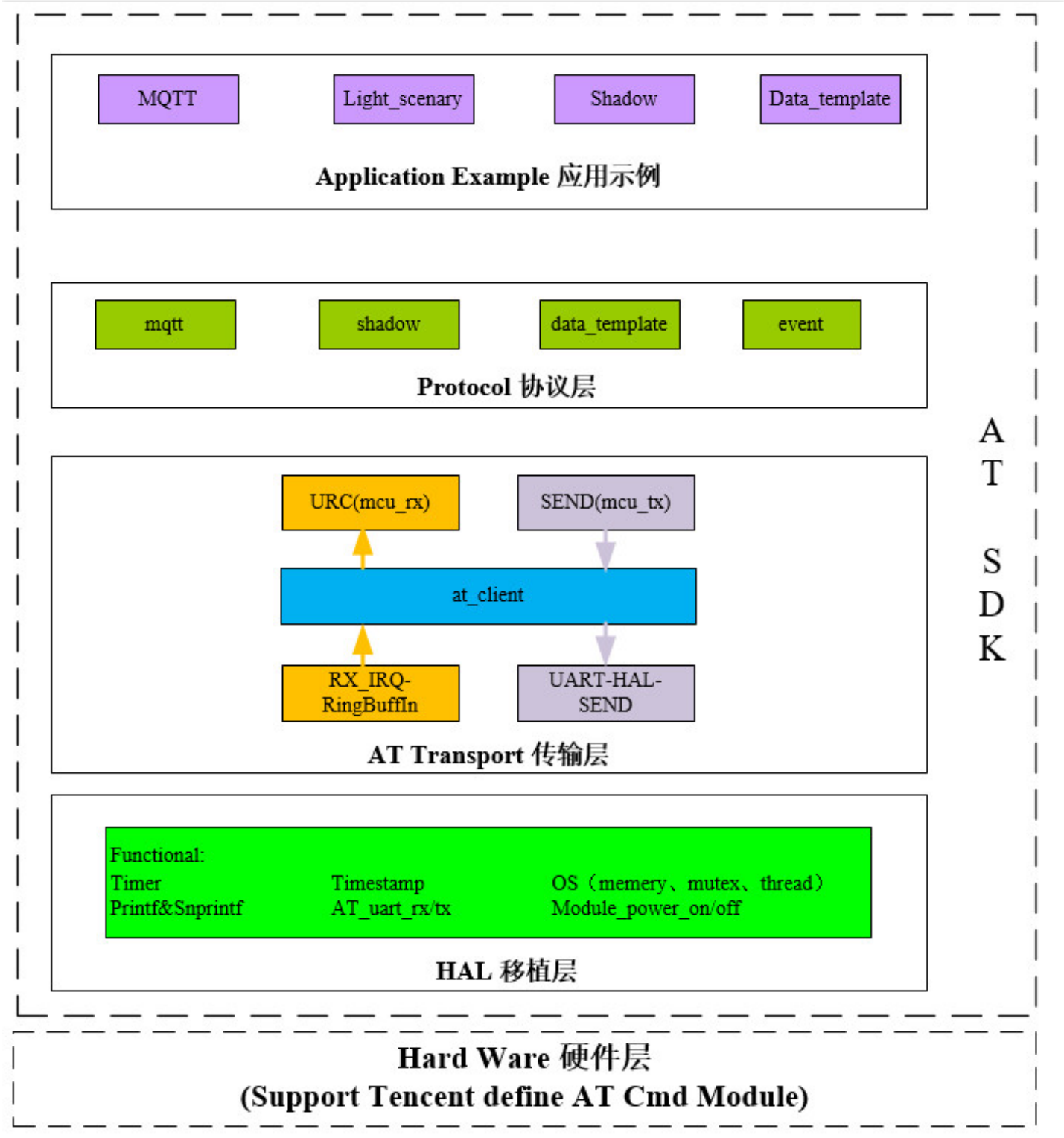
最近更新时间：2020-04-15 10:28:02

简介

在 iot-explorer 平台创建产品和设备后，选择基于 MQTT AT 定制模组开发的方式，会自动生成 MCU 侧的 SDK 代码，代码基于 腾讯 AT_SDK，同时把在平台创建的数据模板和事件生成了对应的配置及初始化代码，SDK 完成实现了 MCU 和模组交互的 AT 框架及上下行数据交互的代码框架，开发者只需要在预留的上下行数据交互函数中实现具体的业务逻辑开发即可。

移植部分需要实现的 HAL 层适配接口见 hal_export.h，需要实现串口的收发接口（中断接收），延时函数，模组上下电及 OS 相关接口适配（互斥锁、动态内存申请释放、线程创建），适配层接口单独剥离在 port 目录。

软件架构


MQTT
Light_scenary
Shadow
Data_template
Application Example 应用示例
mqtt
shadow
data_template
event
Protocol 协议层
URC(mcu_rx)
SEND(mcu_tx)
at_client
RX_IRQ-RingBuffIn
UART-HAL-SEND
AT Transport 传输层
Functional:
Timer
Printf&Snprintf
Timestamp
AT_uart_rx/tx
OS（memery、mutex、thread）
Module_power_on/off
HAL 移植层
A T S D K
Hard Ware 硬件层
(Support Tencent define AT Cmd Module)

目录结构

名称	说明

名称	说明
docs	文档目录，包含腾讯 AT 指令集定义。
port	HAL 层移植目录，需要实现串口的收发接口（中断接收），延时函数，模组上下电及 OS 相关接口。
sample	应用示例，示例使用 MQTT、影子、数据模板的使用方式。
src	AT 框架及协议逻辑实现。
├── event	事件功能协议封装。
├── module_at	at client 抽象，实现 RX 解析，命令下行，urc 匹配，resp 异步匹配。
├── shadow	基于 AT 框架的 shadow 逻辑实现。
├── mqtt	基于AT 框架的 mqtt 协议实现。
├── utils	json、timer、链表等应用。
├── include	SDK 对外头文件及设备信息配置头文件。
usr_logic	自动生成的基于用户产品定义的业务逻辑框架代码。
├── data_config.c	用户定义的数据点。
├── events_config.c	用户定义的事件。
├── data_template_usr_logic.c	用户业务处理逻辑框架，实现预留的上下行业务逻辑处理函数即可。
tools	代码生成脚本。
README.md	SDK 使用说明。

移植指导

根据所选的嵌入式平台，适配 hal_export.h 头文件对应的 HAL 层 API 的移植实现。主要有串口收发（中断接收）、模组开关机、任务/线程创建、动态内存申请/释放、时延、打印等 API。详细操作可参考基于 STM32+FreeRTOS 的 AT-SDK 移植示例。

1. hal_export.h

HAL 层对外的 API 接口及 HAL 层宏开关控制。

序号	宏定义	说明
1	PARSE_THREAD_STACK_SIZE	串口 AT 解析线程栈大小。
2	OS_USED	是否使用 OS，目前的 AT-SDK 是基于多线程框架的，所以 OS 是必须的。
3	AUTH_MODE_KEY	认证方式，证书认证还是密钥认证。
4	DEBUG_DEV_INFO_USED	默认使能该宏，设备信息使用调试信息，正式量产关闭该宏，并实现设备信息存取接口。

2. hal_os.c

该源文件主要实现打印、延时、时间戳、锁、线程创建、设备信息存取等。

序号	HAL_API	说明
1	HAL_Printf	打印函数，log 输出需要，可选实现。
2	HAL_Snprintf	格式化打印，JSON 数据处理需要，必须实现。
3	HAL_Vsnprintf	格式化输出， 可选实现。
4	HAL_DelayMs	毫秒延时，必选实现。
5	HAL_DelayUs	微妙延时，可选实现。
6	HAL_GetTimeMs	获取毫秒数，必选实现。
7	HAL_GetTimeSeconds	获取时间戳，必须实现，时戳不需绝对准确，但不可重复。
8	hal_thread_create	线程创建，必选实现。
9	hal_thread_destroy	线程销毁，必选实现。
10	HAL_SleepMs	放权延时，必选实现。
11	HAL_MutexCreate	互斥锁创建，必选实现。
12	HAL_MutexDestroy	互斥锁销毁，必选实现。
13	HAL_MutexLock	获取互斥锁，必选实现。
14	HAL_MutexUnlock	释放互斥锁，必选实现。
15	HAL_Malloc	动态内存申请，必选实现。

版权所有：腾讯云计算（北京）有限责任公司

序号	HAL_API	说明
16	HAL_Free	动态内存释放，必选实现。
17	HAL_GetProductID	获取产品 ID，必选实现。
18	HAL_SetProductID	设置产品 ID，必须存放在非易失性存储介质，必选实现。
19	HAL_GetDevName	获取设备名，必选实现。
20	HAL_SetDevName	设置设备名，必须存放在非易失性存储介质，必选实现。
21	HAL_GetDevSec	获取设备密钥，密钥认证方式为必选实现。
22	HAL_SetDevSec	设置设备密钥，必须存放在非易失性存储介质，密钥认证方式为必选实现。
23	HAL_GetDevCertName	获取设备证书文件名，证书认证方式为必选实现。
24	HAL_SetDevCertName	设置设备证书文件名，必须存放在非易失性存储介质，证书认证方式为必选实现。
25	HAL_GetDevPrivateKeyName	获取设备证书私钥文件名，证书认证方式为必选实现。
26	HAL_SetDevPrivateKeyName	设置设备证书私钥文件名，必须存放在非易失性存储介质，证书认证方式为必选实现。

3. hal_at.c

该源文件主要实现 AT 串口初始化、串口收发、模组开关机。

序号	HAL_API	说明
1	module_power_on	模组开机，AT 串口初始化，必选实现。
1	module_power_off	模组关机，低功耗需要，可选实现。
2	AT_UART_IRQHandler	串口接收中断 ISR，将收取到的数据放入 ringbuff 中，AT 解析线程会实时解析数据，必选实现。
3	at_send_data	AT 串口发送接口。

4. module_api_inf.c

配网/注网 API 业务适配，该源文件基于腾讯定义的 AT 指令实现了 MQTT 的交互，但有一个关于联网/注网的 API（module_register_network）需要根据模组适配。代码基于 ESP8266 腾讯定制 AT 固件 示例了 Wi-Fi 直连的

方式连接网络，但更常用的场景是根据特定事件（譬如按键）触发配网（softAP/一键配网），这块的逻辑各具体业务逻辑自行实现。

ESP8266 有封装配网指令和示例 App。对于蜂窝模组，则是使用特定的网络注册指令。请参照 module_handshake 应用 AT-SDK 的 AT 框架添加和模组的 AT 指令交互。

```
//模组联网（NB/2/3/4G注册网络）、wifi配网（一键配网/softAP）暂时很难统一,需要用户根据具体模组适配。
//开发者参照 module_handshake API使用AT框架的API和模组交互，实现适配。
eAtResault module_register_network(eModuleType eType)
{
eAtResault result = AT_ERR_SUCCESS;

#ifdef MODULE_TYPE_WIFI
if(eType == eMODULE_ESP8266)
{
#define WIFI_SSID "youga_wifi"
#define WIFI_PW "Iot@2018"

/*此处示例传递热点名字直接联网，通常的做法是特定产品根据特定的事件（譬如按键）触发wifi配网（一键配网/softAP）*/
result = wifi_connect(WIFI_SSID, WIFI_PW);
if(AT_ERR_SUCCESS != result)
{
Log_e("wifi connect fail,ret:%d", result);
}
}
#else
if(eType == eMODULE_N21)
{

/*模组网络注册、或者wifi配网需要用户根据所选模组实现*/
result = N21_net_reg();
if(AT_ERR_SUCCESS != result)
{
Log_e("N21 register network fail,ret:%d", result);
}
}
#endif


return result;
}
```

5. 设备信息修改

调试时，在 hal_export.h 将设备信息调试宏定义打开。量产时需要关闭该宏定义，实现 hal-os 中序列 17-26 的设备信息存取 API。

```
#define DEBUG_DEV_INFO_USED
```

修改下面宏定义的设备信息，则系统将会使用调试信息。

```
#ifdef DEBUG_DEV_INFO_USED

static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1] = "03UKNYBUZG";
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1] = "at_dev";

#ifdef AUTH_MODE_CERT
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME + 1] = "YOUR_DEVICE_NAME_cert.crt";
static char sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_KEY_FILE_NAME + 1] = "YOUR_DEVICE_NAME_private.key";
#else
char sg_device_secret[MAX_SIZE_OF_DEVICE_SERC + 1] = "ttOARy0PjYgzd9OSs4Z3RA==";
#endif

#endif
```

6. 业务逻辑开发

自动生成的代码 data_template_usr_logic.c，已实现数据、事件收发及响应的通用处理逻辑。但是具体的数据处理的业务逻辑需要用户自己根据业务逻辑添加，上下行业务逻辑添加的入口函数分别为 deal_up_stream_user_logic、deal_down_stream_user_logic。

- **下行业务逻辑实现：**

```
/*用户需要实现的下行数据的业务逻辑,pData除字符串变量已实现用户定义的所有其他变量值解析赋值，待用户实现业务逻辑*/
static void deal_down_stream_user_logic(ProductDataDefine * pData)
{
Log_d("someting about your own product logic wait to be done");
}
```

- **上行业务逻辑实现：**

```
/*用户需要实现的上行数据的业务逻辑,此处仅供示例*/
static int deal_up_stream_user_logic(DeviceProperty *pReportDataList[], int *pCount)
```

```
{
int i, j;

for (i = 0, j = 0; i < TOTAL_PROPERTY_COUNT; i++) {
if(eCHANGED == sg_DataTemplate[i].state) {
pReportDataList[j++] = &(sg_DataTemplate[i].data_property);
sg_DataTemplate[i].state = eNOCHANGE;
}
}
*pCount = j;

return (*pCount > 0)?AT_ERR_SUCCESS:AT_ERR_FAILURE;
}
```

7. 示例说明

Smaple 目录一共有3个示例，用户可以参考各示例进行业务逻辑开发，分别是 mqtt_sample.c、shadow_sample.c、light_data_template_sample.c。

各示例说明如下：

序号	示例名称	说明
1	mqtt_sample.c	MQTT 示例，该示例示例基于定制的 AT 指令如何便捷的接入腾讯物联网平台及收发数据。
2	shadow_sample.c	影子示例，基于 AT 实现的 MQTT 协议，进一步封装的影子协议。
3	light_data_template_sample.c	基于智能灯的控制场景，示例具体的产品如何应用数据模板及事件功能。

更多详情请参见 数据模板协议。

SDK 接口说明

关于 SDK 的更多使用方式及接口了解，请参阅 qcloud_iot_api_export.h 文件。

MCU+通用 TCP AT 模组移植（FreeRTOS）

最近更新时间：2020-06-29 16:17:38

对于不具备网络通讯能力的 MCU，一般采用 MCU+ 通讯模组的方式，通讯模组（包括 Wi-Fi/2G/4G/NB-IoT）一般提供了基于串口的 AT 指令协议供 MCU 进行网络通讯。针对这种场景，C-SDK 封装了 AT-socket 网络层，网络层之上的核心协议和服务层无须移植。本文阐述针对 MCU（FreeRTOS）+ 通用 TCP AT 模组的目标环境，如何移植 C-SDK 并接入腾讯云物联网平台。

SDK 下载

下载最新版本设备端 C-SDK。

SDK 功能配置

使用通用 TCP 模组编译配置选项配置如下：

名称	配置	说明
BUILD_TYPE	debug/release	根据需要设置
EXTRACT_SRC	ON	使能代码抽取
COMPILE_TOOLS	gcc/MSVC	根据需要设置，IDE 情况不关注
PLATFORM	Linux/Windows	根据需要设置，IDE 情况不关注
FEATURE_OTA_COMM_ENABLED	ON/OFF	根据需要设置
FEATURE_AUTH_MODE	KEY	资源受限设备认证方式建议选密钥认证
FEATURE_AUTH_WITH_NOTLS	ON/OFF	根据需要是否使能 TLS
FEATURE_EVENT_POST_ENABLED	ON/OFF	根据需要是否使能事件上报
FEATURE_AT_TCP_ENABLED	ON	AT 模组 TCP 功能开关
FEATURE_AT_UART_RECV_IRQ	ON	AT 模组中断接受功能开关
FEATURE_AT_OS_USED	ON	AT 模组多线程功能开关
FEATURE_AT_DEBUG	OFF	默认关闭 AT 模组调试功能，有调试需要再打开

版权所有：腾讯云计算（北京）有限责任公司

代码抽取

1. 在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

2. 即可在 output/qcloud_iot_c_sdk 中，找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│ ├── config.h
│ ├── exports
├── platform
└── sdk_src
└── internal_inc
```

> 说明：
>
> - include 目录：SDK 供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译。
> - platform 目录：平台相关的代码，可根据设备的具体情况进行修改适配。
> - sdk_src ：SDK 的核心逻辑及协议相关代码，一般不需要修改，其中 internal_inc 为 SDK 内部使用的头文件。

3. 用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

HAL 层移植

请先参考 C-SDK_Porting 跨平台移植概述 进行移植。
对于网络相关的 HAL 接口，通过上面的编译选项已选择 SDK 提供的 `AT_Socket` 框架，SDK 会调用 `network_at_tcp.c` 的 `at_socket` 接口， `at_socket` 层不需要移植，需要实现 AT 串口驱动及AT模组驱动，AT模组驱动只需要实现 AT 框架中 `at_device` 的驱动结构体 `at_device_op_t` 的驱动接口即可，可以参照 `at_device` 目录下的已支持的模组。
目前 SDK 针对物联网使用较广的 Wi-Fi 模组 ESP8266 提供了底层接口实现，供移植到其他通讯模组时作为参考。

业务逻辑开发

您可参考 SDK samples 目录下的例程进行开发。

版权所有：腾讯云计算（北京）有限责任公司

MCU+通用 TCP AT 模组移植（nonOS）

最近更新时间：2020-06-29 16:17:02

对于不具备网络通讯能力的 MCU，一般采用 MCU+ 通讯模组的方式，通讯模组（包括 Wi-Fi/2G/4G/NB-IoT）一般提供了基于串口的 AT 指令协议供 MCU 进行网络通讯。针对这种场景，C-SDK 封装了 AT-socket 网络层，网络层之上的核心协议和服务层无须移植。本文阐述针对 MCU（无 OS）+通用 TCP AT 模组的目标环境，如何移植 C-SDK 并接入腾讯云物联网平台。
相较于有 RTOS 场景，at_socket 网络接收数据的处理会有差异，应用层需要周期性的调用 **IOT_MQTT_Yield** 来接收服务端下行数据，错过接收窗口则会存在数据丢失的情况，所以在业务逻辑较为复杂的场景建议使用 RTOS，通过配置 FEATURE_AT_OS_USED = OFF 选择无 OS 方式。

SDK 下载

下载最新版本设备端 C-SDK。

SDK 功能配置

无 RTOS 使用通用 TCP 模组编译配置选项配置如下：

名称	配置	说明
BUILD_TYPE	debug/release	根据需要设置
EXTRACT_SRC	ON	使能代码抽取
COMPILE_TOOLS	gcc/MSVC	根据需要设置，IDE 情况不关注
PLATFORM	Linux/Windows	根据需要设置，IDE 情况不关注
FEATURE_OTA_COMM_ENABLED	ON/OFF	根据需要设置
FEATURE_AUTH_MODE	KEY	资源受限设备认证方式建议选密钥认证
FEATURE_AUTH_WITH_NOTLS	ON/OFF	根据需要是否使能 TLS
FEATURE_EVENT_POST_ENABLED	ON/OFF	根据需要是否使能事件上报
FEATURE_AT_TCP_ENABLED	ON	使能 at_socket 组件
FEATURE_AT_UART_RECV_IRQ	ON	使能 AT 串口中断接收

版权所有：腾讯云计算（北京）有限责任公司

名称	配置	说明
FEATURE_AT_OS_USED	OFF	at_socket 组件无 RTOS 环境使用
FEATURE_AT_DEBUG	OFF	默认关闭 AT 模组调试功能，有调试需要再打开

代码抽取

1. 在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

2. 即可在 output/qcloud_iot_c_sdk 中，找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│  ├── config.h
│  ├── exports
├── platform
└── sdk_src
└── internal_inc
```

> 说明：
>
> - include 目录：SDK 供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译宏。
> - platform 目录：平台相关的代码，可根据设备的具体情况进行修改适配。
> - sdk_src ：SDK 的核心逻辑及协议相关代码，一般不需要修改，其中 internal_inc 为 SDK 内部使用的头文件。

3. 用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

HAL 层移植

请先参考 C-SDK_Porting 跨平台移植概述。

对于网络相关的 HAL 接口，通过上面的编译选项已选择 SDK 提供的 AT_Socket 框架，SDK 会调用 network_at_tcp.c 的 at_socket 接口， at_socket 层不需要移植，需要实现 AT 串口驱动及 AT 模组驱动，AT 模组驱动只需要实现 AT 框架中 at_device 的驱动结构体 at_device_op_t 的驱动接口即可，可以参照 at_device 目录下的已支持的模组。AT 串口驱动需要实现串口的中断接收，然后在中断服务程序中调用回调函数 at_client_uart_rx_isr_cb 即可，可以参考 HAL_OS_nonos.c 实现目标平台的移植。

业务逻辑开发

您可参考 SDK samples 目录下的例程进行开发。

SDK 使用参考
C SDK 使用
使用概述

最近更新时间：2020-06-29 16:15:46

腾讯云物联网设备端 C-SDK 依靠安全且性能强大的数据通道，为物联网领域开发人员提供设备端快速接入云端，并和云端进行双向通信的能力。

C-SDK 适用范围

C-SDK 采用模块化设计，分离核心协议服务与硬件抽象层，并提供灵活的配置选项和多种编译方式，适用于不同设备的开发平台和使用环境。

具备网络通讯能力并使用 Linux/Windows 操作系统的设备

- 对于具备网络通讯能力并使用标准 Linux/Windows 系统的设备。例如 PC/服务器/网关设备，及较高级的嵌入式设备树莓派等，可直接在该设备上编译运行 SDK。
- 对于需要交叉编译的嵌入式 Linux 设备，如果开发环境的 toolchain 具备 glibc 或类似的库，可以提供包括 socket 通讯，select 同步 IO，动态内存分配，获取时间/休眠/随机数/打印函数，以及临界数据保护如 Mutex 机制（仅在需要多线程时）等系统调用，则只要做简单适配（例如，在 CMakeLists.txt 或 make.settings 里修改交叉编译器的设定）即可编译运行 SDK。

具备网络通讯能力并采用 RTOS 系统的设备

- 对于具备网络通讯能力并采用 RTOS 的物联网设备，C-SDK 需要针对不同的 RTOS 做移植适配工作，目前 C-SDK 已经适配了包括 FreeRTOS/RT-Thread/TencentOS tiny 等多个面向物联网的 RTOS 平台。
- 在 RTOS 设备移植 SDK 时，如果平台提供了类似 newlib 的 C 运行库和类似 lwIP 的嵌入式 TCP/IP 协议栈，则移植适配工作也可轻松完成。

MCU+ 通讯模组的设备

- 对于不具备网络通讯能力的 MCU，一般采用 MCU+ 通讯模组的方式，通讯模组（包括 Wi-Fi/2G/4G/NB-IoT）一般提供了基于串口的 AT 指令协议供 MCU 进行网络通讯。针对这种场景，C-SDK 封装了 AT-socket 网络层，网络层之上的核心协议和服务层无须移植。并提供了基于 FreeRTOS 和不带操作系统（nonOS）两种方式的 HAL 实现。
- 除此之外，腾讯云物联网还提供了专用的 AT 指令集，如果通讯模组实现了该指令集，则设备接入和通讯更为简单，所需代码量更少，针对这种场景，请参考面向腾讯云定制 AT 模组专用的 MCU AT SDK。

版权所有：腾讯云计算（北京）有限责任公司

SDK 目录结构简介

目录结构及顶层文件简介如下：

名称	说明
CMakeLists.txt	cmake 编译描述文件
CMakeSettings.json	visual studio下的 cmake 配置文件
cmake_build.sh	Linux 下使用 cmake 的编译脚本
make.settings	Linux 下使用 Makefile 直接编译的配置文件
Makefile	Linux 下使用 Makefile 直接编译
device_info.json	设备信息文件，当 DEBUG_DEV_INFO_USED=OFF 时，将从该文件解析出设备信息
docs	文档目录，SDK 在不同平台下使用说明文档
external_libs	第三方软件包组件，例如 mbedtls
samples	应用示例
include	提供给用户使用的外部头文件
platform	平台相关的源码文件，目前提供了针对不同 OS（Linux/Windows/FreeRTOS/nonOS），TLS（mbedtls）以及 AT 模组下的实现
sdk_src	SDK 核心通信协议及服务代码
tools	SDK 配套的编译及代码生成脚本工具

SDK 编译方式说明

C-SDK 支持三种编译方式：

- cmake 方式。
- Makefile 方式。
- 代码抽取方式。

编译方式以及编译配置选项的详细说明请参考 编译配置说明 和 编译环境说明。

SDK 示例体验

C-SDK 的 samples 目录有使用各个功能的示例，关于运行示例的详细说明，请参考 SDK 文档目录下所有文档。
物联网开发平台快速体验数据模板的数据交互，请参考 智能灯快速入门。

编译配置说明

最近更新时间：2020-06-29 16:15:11

本文档对 C-SDK 的编译方式和编译配置选项进行说明，并介绍了 Linux 和 Windows 开发环境下的编译环境搭建以及编译示例。

C-SDK 编译方式说明

C-SDK 支持以下编译方式。

cmake 方式

- 推荐使用 cmake 作为跨平台的编译工具，支持在 Linux 和 Windows 开发环境下进行编译。
- cmake 方式采用 CMakeLists.txt 作为编译配置选项输入文件。

Makefile 方式

- 对于不支持 cmake 的环境，使用 Makefile 直接编译的方式。
- Makefile 方式采用 make.settings 作为编译配置选项输入文件，修改完成后执行 make 即可。

代码抽取方式

- 该方式可根据需求选择功能，将相关代码抽取到一个单独的文件夹，文件夹里面的代码层次目录简洁，方便用户拷贝集成到自己的开发环境。
- 该方式需要依赖 cmake 工具，在 CMakeLists.txt 中配置相关功能模块的开关，并将 EXTRACT_SRC 设置为 ON，在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

  - 即可在 output/qcloud_iot_c_sdk 中找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
└── internal_inc
```

  - include 目录为 SDK 供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译宏。

- platform 目录为平台相关的代码，可根据设备的具体情况进行修改适配。
- sdk_src 为 SDK 的核心逻辑及协议相关代码，一般无需修改，其中 internal_inc 为 SDK 内部使用的头文件。

说明：
用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

C-SDK 编译选项说明

编译配置选项

以下配置选项大部分都适用于 cmake 和 make.setting。cmake 中的 ON 值对应于 make.setting 的 y，OFF 对应于 n。

名称	cmake 值	说明
BUILD_TYPE	release/debug	• release：不启用 IOT_DEBUG 信息，编译输出到 release 目录下。 • debug：启用 IOT_DEBUG 信息，编译输出到 debug 目录下。
EXTRACT_SRC	ON/OFF	代码抽取功能开关，仅对使用 cmake 有效。
COMPILE_TOOLS	gcc	支持 gcc 和 msvc，也可以是交叉编译器。例如 arm-none-linux-gnueabi-gcc。
PLATFORM	Linux	包括 Linux/Windows/Freertos/Nonos。
FEATURE_OTA_COMM_ENABLED	ON/OFF	OTA 功能使能开关
FEATURE_AUTH_MODE	KEY/CERT	接入认证方式。
FEATURE_AUTH_WITH_NOTLS	ON/OFF	OFF：TLS 使能， ON：TLS 关闭。
FEATURE_EVENT_POST_ENABLED	ON/OFF	事件功能使能开关。
FEATURE_ACTION_ENABLED	ON/OFF	行为功能使能开关。
FEATURE_DEBUG_DEV_INFO_USED	ON/OFF	设备信息获取来源开关。
FEATURE_SYSTEM_COMM_ENABLED	ON/OFF	获取后台时间开关。
FEATURE_DEV_DYN_REG_ENABLED	ON/OFF	设备动态注册开关。

版权所有：腾讯云计算（北京）有限责任公司

名称	cmake 值	说明
FEATURE_LOG_UPLOAD_ENABLED	ON/OFF	日志上报开关。

使用 SDK AT 框架+通用 TCP 模组配置项如下：

名称	cmake 值	说明
FEATURE_AT_TCP_ENABLED	ON/OFF	AT 模组 TCP 功能开关。
FEATURE_AT_UART_RECV_IRQ	ON/OFF	AT 模组中断接受功能开关。
FEATURE_AT_OS_USED	ON/OFF	AT 模组多线程功能开关。
FEATURE_AT_DEBUG	ON/OFF	AT 模组调试功能开关。

配置选项之间存在依赖关系，当依赖选项的值为有效值时，部分配置选项才有效，主要如下：

名称	依赖选项	有效值
FEATURE_AUTH_WITH_NOTLS	FEATURE_AUTH_MODE	KEY
FEATURE_AT_UART_RECV_IRQ	FEATURE_AT_TCP_ENABLED	ON
FEATURE_AT_OS_USED	FEATURE_AT_TCP_ENABLED	ON
FEATURE_AT_DEBUG	FEATURE_AT_TCP_ENABLED	ON

设备信息选项

在腾讯云物联控制台创建设备之后，需要将设备信息（ProductID/DeviceName/DeviceSecret/Cert/Key 文件）配置在 SDK 中才能正确运行。在开发阶段，SDK 提供两种方式存储设备信息：

- 存放在代码中（编译选项 DEBUG_DEV_INFO_USED = ON），则在 `platform/os/xxx/HAL_Device_xxx.c` 中修改设备信息，在无文件系统的平台下可以使用这种方式。
- 存放在配置文件中（编译选项 DEBUG_DEV_INFO_USED = OFF），则在 `device_info.json` 文件修改设备信息，此方式下更改设备信息不需重新编译 SDK，在 Linux/Windows 平台下开发推荐使用这种方式。

编译环境（Linux&Windows）

最近更新时间：2020-06-29 16:14:33

Linux（Ubuntu）环境

说明：
本文演示使用 Ubuntu 的版本为16.04。

1. **必要软件安装**
   SDK 需要 cmake 版本在3.5以上，默认安装的 cmake 版本较低，若编译失败，请单击 下载 并参考 安装说明 进行 cmake 特定版本的下载与安装。

```
$ sudo apt-get install -y build-essential make git gcc cmake
```

2. **配置修改**
   修改 SDK 根目录下的 CMakeLists.txt 文件，并确保以下选项存在（以密钥认证设备为例）：

```
set(BUILD_TYPE "release")
set(COMPILE_TOOLS "gcc")
set(PLATFORM "linux")
set(FEATURE_MQTT_COMM_ENABLED ON)
set(FEATURE_AUTH_MODE "KEY")
set(FEATURE_AUTH_WITH_NOTLS OFF)
set(FEATURE_DEBUG_DEV_INFO_USED OFF)
```

3. **执行脚本编译**
   i. 完整编译库和示例如下：

```
./cmake_build.sh
```

   ii. 输出的库文件，头文件及示例在 output/release 文件夹中。
   在一次完整编译之后，若只需要编译示例，则执行以下代码：

```
./cmake_build.sh samples
```

4. **填写设备信息**
   将在腾讯云物联网平台创建的设备的设备信息（以密钥认证设备为例），填写到 SDK 根目录下 device_info.json 中，示例代码如下：

```
"auth_mode":"KEY",
"productId":"S3EUVBQAZW",
"deviceName":"test_device",
"key_deviceinfo":{
"deviceSecret":"vX6PQqazsGsMyf5SMfs6OA6y"
}
```

5. **运行示例**
   示例输出位于 output/release/bin 文件夹中，例如运行 data_template_sample 示例，输入 ./output/release/bin/data_template_sample 即可。

Windows 环境

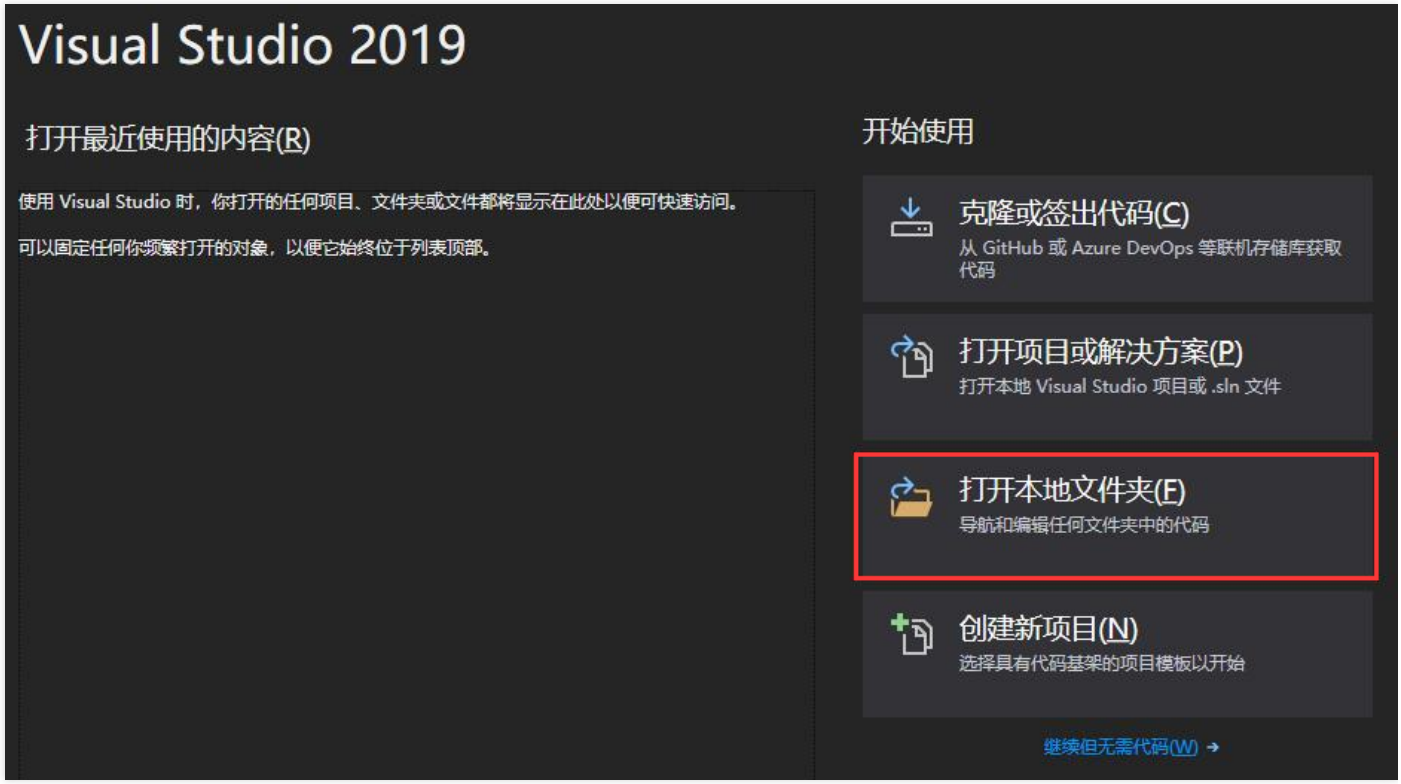
获取和安装 Visio Studio 2019开发环境

1. 请访问 Visual Studio 下载网站，下载并安装 Visio Studio 2019，本文档下载安装的是16.2版本 Community。

2. 选择【使用 C++ 的桌面开发】，并确保勾选【用于 Windows 的 C++ CMAKE 工具】。

编译并运行

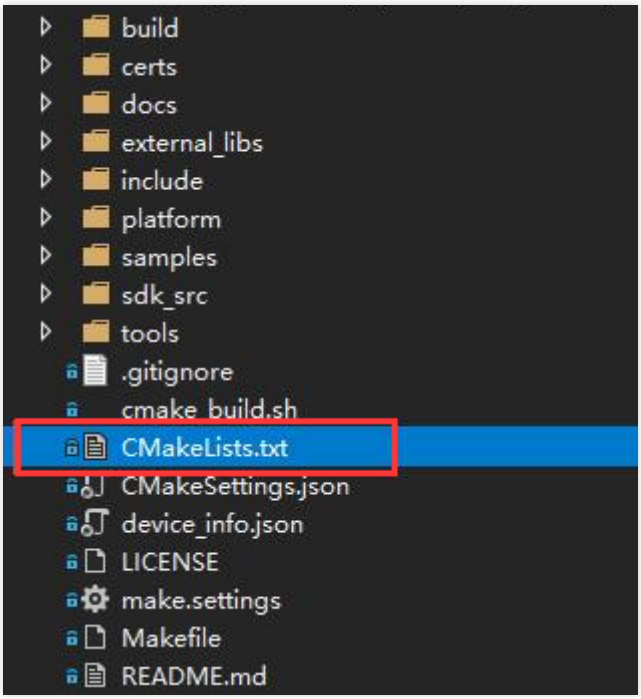
1. 运行 Visual Studio，选择【打开本地文件夹】，并选择下载的 C SDK 目录。



2. 将在腾讯云物联网通信控制台创建的设备的设备信息（以密钥认证设备为例），填写到 device_info.json 中，示例代码如下：

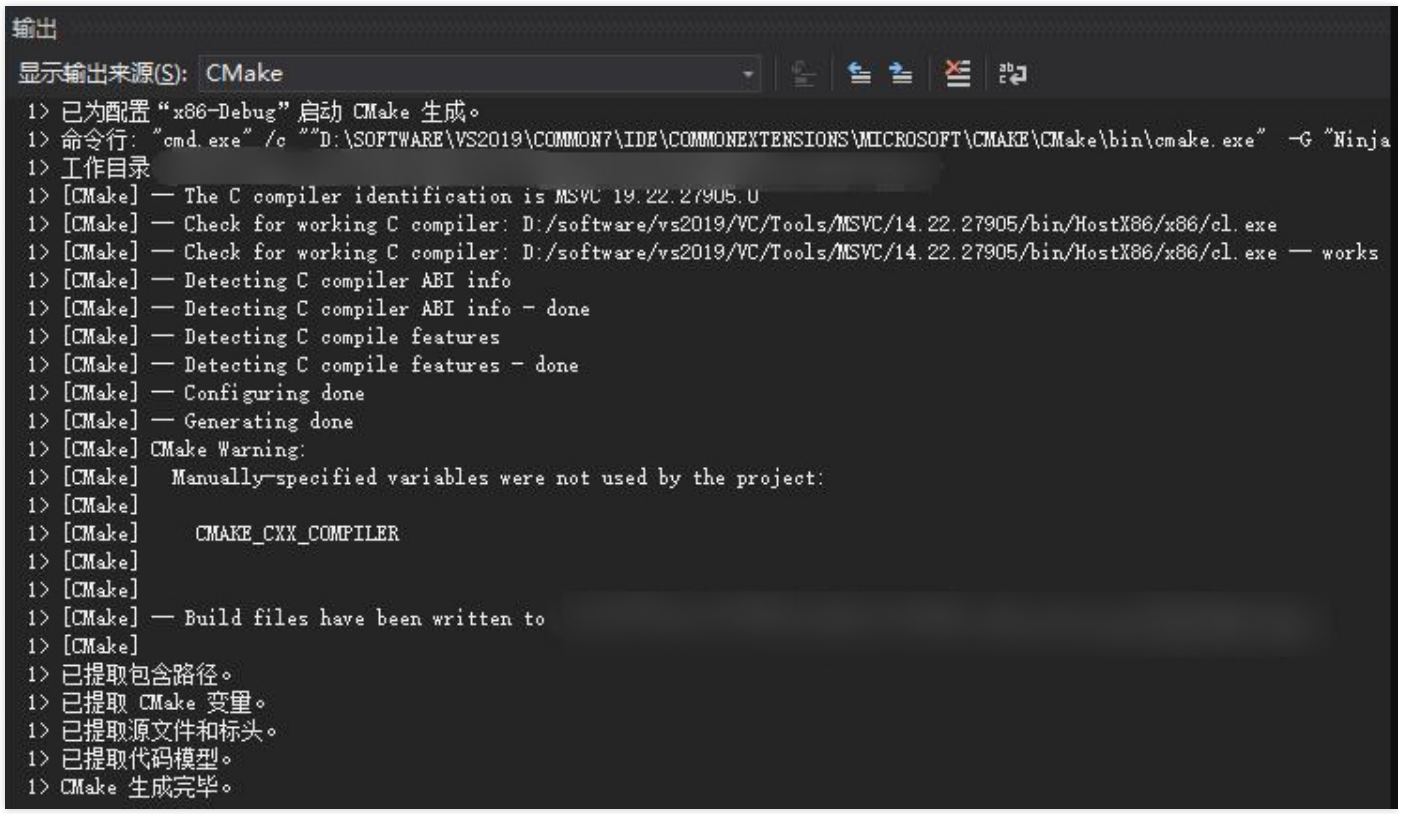
```
"auth_mode":"KEY",
"productId":"S3EUVBQAZW",
"deviceName":"test_device",
"key_deviceinfo":{
"deviceSecret":"vX6PQqazsGsMyf5SMfs6OA6y"
}
```

3. 双击打开根目录的 CMakeLists.txt，并确认编译工具链中设置的平台为 **Windows** 和编译工具为 **MSVC**。



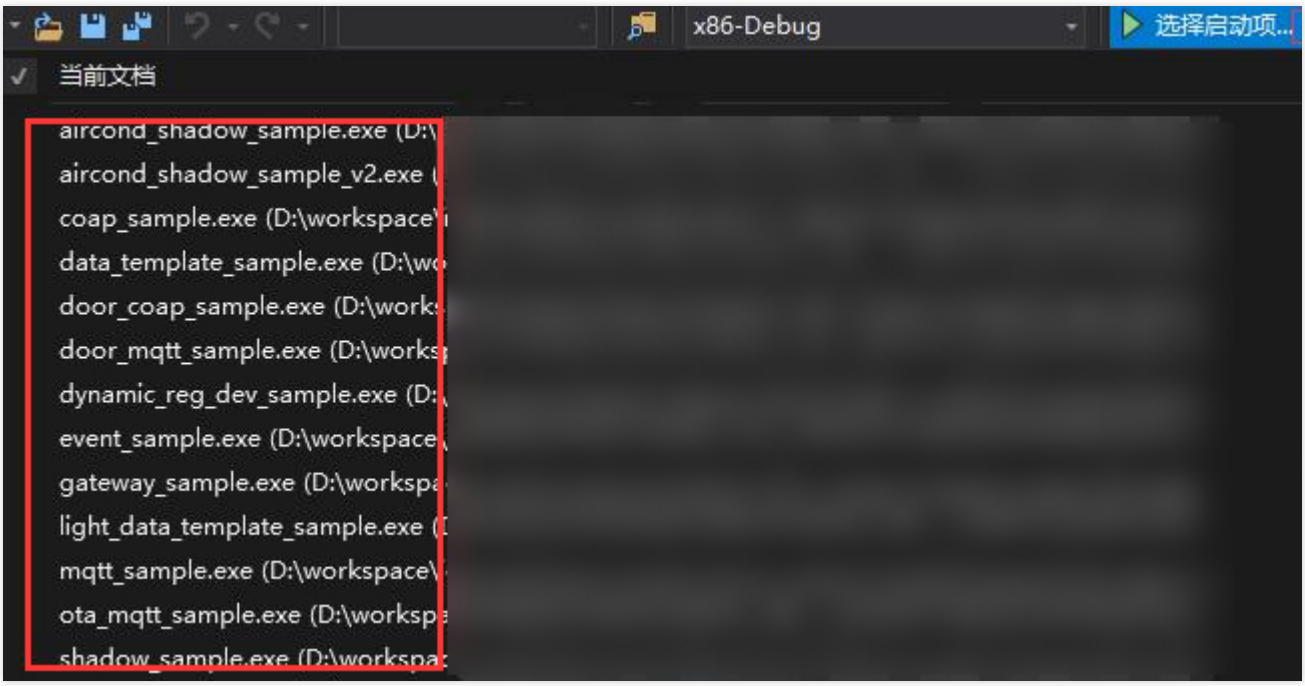

```
# 编译工具链
#set(COMPILE_TOOLS "gcc")
#set(PLATFORM "linux")

set(COMPILE_TOOLS "MSVC")
set(PLATFORM "windows")
```

4. Visual Studio 会自动生成 cmake 缓存，请等待 cmake 缓存生成完毕。



5. 缓存生成完毕后，选择【生成】>【全部生成】。

6. 选择相应的示例运行，示例应与用户信息相对应。



接口及可变参数说明

最近更新时间：2020-06-29 16:13:58

设备端 C-SDK 供用户调用的 API 函数声明，常量以及可变参数定义等头文件位于 include 目录下面，本文档主要对该目录下面的可变参数以及 API 函数进行说明。

可变参数配置

C-SDK 是基于MQTT协议，使用可以根据具体场景需求，配置相应的参数，满足实际业务的运行。可变接入参数包括：

1. MQTT 阻塞调用（包括连接, 订阅, 发布等）的超时时间，单位：毫秒。 建议5000毫秒。
2. MQTT 协议发送消息和接受消息的 buffer 大小默认为2048字节，最大支持16KB。
3. MQTT 心跳消息发送周期，最大值为690秒，单位：毫秒。
4. 重连最大等待时间，单位：毫秒。设备断线重连时，若失败则等待时间会翻倍，当超过该最大等待时间则退出重连。

修改 include/qcloud_iot_export_variables.h 文件如下宏定义可以改变对应接入参数的配置。
修改完需要重新编译 SDK ，示例代码如下：

```
/* default MQTT/CoAP timeout value when connect/pub/sub (unit: ms) */
#define QCLOUD_IOT_MQTT_COMMAND_TIMEOUT (5 * 1000)

/* default MQTT keep alive interval (unit: ms) */
#define QCLOUD_IOT_MQTT_KEEP_ALIVE_INTERNAL (240 * 1000)

/* default MQTT Tx buffer size, MAX: 16*1024 */
#define QCLOUD_IOT_MQTT_TX_BUF_LEN (2048)

/* default MQTT Rx buffer size, MAX: 16*1024 */
#define QCLOUD_IOT_MQTT_RX_BUF_LEN (2048)

/* default COAP Tx buffer size, MAX: 1*1024 */
#define COAP_SENDMSG_MAX_BUFLEN (512)

/* default COAP Rx buffer size, MAX: 1*1024 */
#define COAP_RECVMSG_MAX_BUFLEN (512)

/* MAX MQTT reconnect interval (unit: ms) */
#define MAX_RECONNECT_WAIT_INTERVAL (60 * 1000)
```

版权所有：腾讯云计算（北京）有限责任公司

API 函数说明

以下是 C-SDK v3.1.0版本提供的主要功能和对应 API 接口说明，用于客户编写业务逻辑，更加详细的说明如接口参数及返回值可查看 SDK 代码 include/exports/qcloud_iot_export_*.h 等头文件中的注释。

数据模板接口

数据模板协议及功能介绍，请参见 数据模板协议。

序号	函数名	说明
1	IOT_Template_Construct	构造数据模板客户端 Data_template_client 并连接 MQTT 云端服务
2	IOT_Template_Destroy	关闭 Data_template MQTT 连接并销毁 Data_template Client
3	IOT_Template_Yield	在当前线程上下文中，进行 MQTT 报文读取，消息处理，超时请求，心跳包及重连状态管理等任务
4	IOT_Template_Publish	数据模板客户端发布 MQTT 消息
5	IOT_Template_Subscribe	数据模板客户端订阅 MQTT 主题
6	IOT_Template_Unsubscribe	数据模板客户端取消订阅已订阅的 MQTT 主题
7	IOT_Template_IsConnected	查看当前数据模板客户端的 MQTT 是否已连接

数据模板属性接口

序号	函数名	说明
1	IOT_Template_Register_Property	注册当前设备的数据模板属性
2	IOT_Template_UnRegister_Property	删除已经注册过的数据模板属性
3	IOT_Template_Report	异步方式上报数据模板属性数据
4	IOT_Template_Report_Sync	同步方式上报数据模板属性数据
5	IOT_Template_GetStatus	异步方式获取数据模板属性数据
6	IOT_Template_GetStatus_sync	同步方式获取数据模板属性数据
7	IOT_Template_Report_SysInfo	异步方式上报系统信息
8	IOT_Template_Report_SysInfo_Sync	同步方式上报系统信息
9	IOT_Template_JSON_ConstructSysInfo	构造待上报的系统信息

版权所有：腾讯云计算（北京）有限责任公司

序号	函数名	说明
10	IOT_Template_ControlReply	回复收到的数据模板属性控制消息
11	IOT_Template_ClearControl	删除数据模板属性控制消息，配合 IOT_Template_GetStatus 获取到 control 消息使用

数据模板事件接口

序号	函数名	说明
1	IOT_Post_Event	上报数据模板事件，传入事件，SDK 完成事件消息构造
2	IOT_Post_Event_Raw	上报数据模板事件，传入符合事件消息格式的数据，SDK 完成上报
3	IOT_Event_setFlag	置位事件标志，SDK 默认支持10个事件，可以扩展增加
4	IOT_Event_clearFlag	清除事件标志
5	IOT_Event_getFlag	获取事件标志

数据模板行为接口

序号	函数名	说明
1	IOT_ACTION_REPLY	回复数据模板行为消息

多线程环境使用说明

SDK 对于在多线程环境下的使用有如下注意事项：

- 不允许多线程调用 IOT_Template_Yield，IOT_Template_Construct 以及IOT_Template_Destroy。
- IOT_Template_Yield 作为从 socket 读取并处理 MQTT 报文的函数，应保证一定的执行时间，避免被长时间挂起或抢占。

OTA 接口

关于 OTA 固件下载功能介绍，请参见 设备固件升级。

序号	函数名	说明
1	IOT_OTA_Init	初始化 OTA 模块，客户端在调用此接口之前需要先进行 MQTT/COAP 的初始化
2	IOT_OTA_Destroy	释放 OTA 模块相关的资源

序号	函数名	说明
3	IOT_OTA_ReportVersion	向 OTA 服务器报告本地固件版本信息
4	IOT_OTA_IsFetching	检查是否处于下载固件的状态
5	IOT_OTA_IsFetchFinish	检查固件是否已经下载完成
6	IOT_OTA_FetchYield	从具有特定超时值的远程服务器获取固件
7	IOT_OTA_Ioctl	获取指定的 OTA 信息
8	IOT_OTA_GetLastError	获取最后一个错误代码
9	IOT_OTA_StartDownload	根据获取到的固件更新地址以及本地固件信息偏移（是否断点续传），与固件服务器建立 HTTP 连接
10	IOT_OTA_UpdateClientMd5	断点续传前，计算本地固件的 MD5
11	IOT_OTA_ReportUpgradeBegin	当进行固件升级前，向服务器上报即将升级的状态
12	IOT_OTA_ReportUpgradeSuccess	当固件升级成功之后，向服务器上报升级成功的状态
13	IOT_OTA_ReportUpgradeFail	当固件升级失败之后，向服务器上报升级失败的状态

日志接口

设备日志上报云端功能的详细说明，请参考 SDK docs 目录下物联网通信平台文档设备日志上报功能部分。

序号	函数名	说明
1	IOT_Log_Set_Level	设置 SDK 日志的打印等级
2	IOT_Log_Get_Level	返回 SDK 日志打印的等级
3	IOT_Log_Set_MessageHandler	设置日志回调函数，重定向 SDK 日志于其它输出方式
4	IOT_Log_Init_Uploader	开启 SDK 日志上报云端的功能并初始化资源
5	IOT_Log_Fini_Uploader	停止 SDK 日志上报云端功能并释放资源
6	IOT_Log_Upload	将 SDK 运行日志上报到云端
7	IOT_Log_Set_Upload_Level	设置 SDK 日志的上报等级
8	IOT_Log_Get_Upload_Level	返回 SDK 日志上报的等级
9	Log_d/i/w/e	按级别打印添加 SDK 日志的接口

版权所有：腾讯云计算（北京）有限责任公司

系统时间接口

序号	函数名	说明
1	IOT_Get_SysTime	获取 IoT Hub 后台系统时间，目前仅支持 MQTT 通道对时功能

设备信息存储

最近更新时间：2020-06-29 16:13:16

概述

腾讯云物联网开发平台为每个创建的产品分配唯一标识 ProductID，用户可以自定义 DeviceName 标识设备，用产品标识 + 设备标识 + 设备证书/密钥来验证设备的合法性。设备端需要存储这些设备身份信息。C-SDK 提供了读写设备信息的接口及参考实现，可根据实际情况进行适配。

设备身份信息

- 证书设备要通过平台的安全认证，必须具备四元组信息：产品 ID（ProductId）、设备名（DeviceName）、设备证书文件（DeviceCert）、设备私钥文件（DevicePrivateKey），其中证书文件和私钥文件由平台生成，且一一对应。
- 密钥设备要通过平台的安全认证，必须具备三元组信息：产品 ID（ProductId）、设备名（DeviceName）、设备密钥（DeviceSecret），其中设备密钥由平台生成。

设备身份信息烧录

设备信息烧录分为预置烧录和动态烧录，两者在应用的便捷性和安全性上有区别。

预置烧录

创建产品后，在 物联网开发平台控制台 或者通过 云 API 逐个创建设备，并获取对应的设备信息，将上述的四元组或者三元组信息，在设备生产的特定环节，烧录到非易失介质中，设备 SDK 运行时读取存放的设备信息，进行设备认证。

动态烧录

- 预置烧录：需要在量产过程执行个性化生产动作，影响生产效率，为增加应用的便捷性，平台支持动态烧录的方式。实现方式：产品创建后使能产品的动态注册功能，则会生成产品密钥（ProductSecret）。同一产品下的所有设备在生产过程可以烧录统一的产品信息，即产品 ID（ProductId）、产品密钥（ProductSecret）。设备出厂后，通过动态注册的方式获取设备身份信息并保存，然后用申请到的三元组或者四元组信息进行设备认证。
- 动态烧录设备名（DeviceName）的生成：若使能动态注册的同时使能自动设备创建，则设备名是可以设备自己生成的，但必须保证同一产品 ID（ProductId）下的设备名不重复，一般取 IMEI 或者 MAC 地址。若使能动态注册的同时不使能自动设备创建，则需要把设备名预先录入平台，设备动态注册时会校验所申请的设备名是否为合法录入的设备名，此种方式能在一定程度降低产品密钥泄露后的安全风险。

版权所有：腾讯云计算（北京）有限责任公司

注意：
动态注册需要确保产品密钥（ProductSecret）的安全，否则会产生较大的安全隐患。

设备信息读写接口

SDK 提供了设备信息读写的 HAL 接口，必须实现。可以参考 Linux 平台 HAL_Device_Linux.c 中设备信息读写的实现。

设备信息 HAL 接口 ：

HAL_API	说明
HAL_SetDevInfo	设备信息写入
HAL_GetDevInfo	设备信息读取

开发阶段设备信息配置

创建设备之后，需要将设备信息（ ProductID/DeviceName/DeviceSecret/Cert/Key 文件）配置在 SDK 中才能正确运行示例。在开发阶段，SDK 提供两种方式存储设备信息：

1. 存放在代码中（编译选项 DEBUG_DEV_INFO_USED = ON），则在 platform/os/xxx/HAL_Device_xxx.c 中修改设备信息，在无文件系统的平台下可以使用这种方式。

```
/* product Id */
static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1] = "PRODUCT_ID";

/* device name */
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1] = "YOUR_DEV_NAME";

#ifdef DEV_DYN_REG_ENABLED
/* product secret for device dynamic Registration */
static char sg_product_secret[MAX_SIZE_OF_PRODUCT_SECRET + 1] = "YOUR_PRODUCT_SECRET";
#endif

#ifdef AUTH_MODE_CERT
/* public cert file name of certificate device */
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME + 1] = "YOUR_DEVICE_NAME_cert.crt";
```

```
/* private key file name of certificate device */
static char sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_SECRET_FILE_NAME + 1] = "YOUR_DEVICE_NAME_private.key";
#else
/* device secret of PSK device */
static char sg_device_secret[MAX_SIZE_OF_DEVICE_SECRET + 1] = "YOUR_IOT_PSK";
#endif
```

2. 存放在配置文件中（编译选项 DEBUG_DEV_INFO_USED = OFF），则在 device_info.json 文件修改设备信息，此方式下更改设备信息不需重新编译 SDK，在 Linux/Windows 平台下开发推荐使用这种方式。

```
{
"auth_mode":"KEY/CERT",

"productId":"PRODUCT_ID",
"productSecret":"YOUR_PRODUCT_SECRET",
"deviceName":"YOUR_DEV_NAME",

"key_deviceinfo":{
"deviceSecret":"YOUR_IOT_PSK"
},

"cert_deviceinfo":{
"devCertFile":"YOUR_DEVICE_CERT_FILE_NAME",
"devPrivateKeyFile":"YOUR_DEVICE_PRIVATE_KEY_FILE_NAME"
},

"subDev":{
"sub_productId":"YOUR_SUBDEV_PRODUCT_ID",
"sub_devName":"YOUR_SUBDEV_DEVICE_NAME"
}
}
```

应用示例

- 初始化连接参数

```
static DeviceInfo sg_devInfo;

static int _setup_connect_init_params(MQTTInitParams* initParams)
{
```

```
int ret;

ret = HAL_GetDevInfo((void *)&sg_devInfo);
if(QCLOUD_ERR_SUCCESS != ret){
return ret;
}

initParams->device_name = sg_devInfo.device_name;
initParams->product_id = sg_devInfo.product_id;
......
}
```

- 密钥设备鉴权参数生成

```
static int _serialize_connect_packet(unsigned char *buf, size_t buf_len, MQTTConnectParams *option
s, uint32_t *serialized_len) {
......
......
int username_len = strlen(options->client_id) + strlen(QCLOUD_IOT_DEVICE_SDK_APPID) + MAX_CO
NN_ID_LEN + cur_timesec_len + 4;
options->username = (char*)HAL_Malloc(username_len);
get_next_conn_id(options->conn_id);
HAL_Snprintf(options->username, username_len, "%s;%s;%s;%ld", options->client_id, QCLOUD_IOT_D
EVICE_SDK_APPID, options->conn_id, cur_timesec);

#if defined(AUTH_WITH_NOTLS) && defined(AUTH_MODE_KEY)
if (options->device_secret != NULL && options->username != NULL) {
char sign[41] = {0};
utils_hmac_sha1(options->username, strlen(options->username), sign, options->device_secret, optio
ns->device_secret_len);
options->password = (char*) HAL_Malloc (51);
if (options->password == NULL) IOT_FUNC_EXIT_RC(QCLOUD_ERR_INVAL);
HAL_Snprintf(options->password, 51, "%s;hmacsha1", sign);
}
#endif
......
}
```

Android SDK 使用说明

最近更新时间：2020-03-26 12:46:08

设备端 SDK 使用步骤如下：

- 将 iot_explorer 模块加入工程构建。
- 基于 API 进行开发。

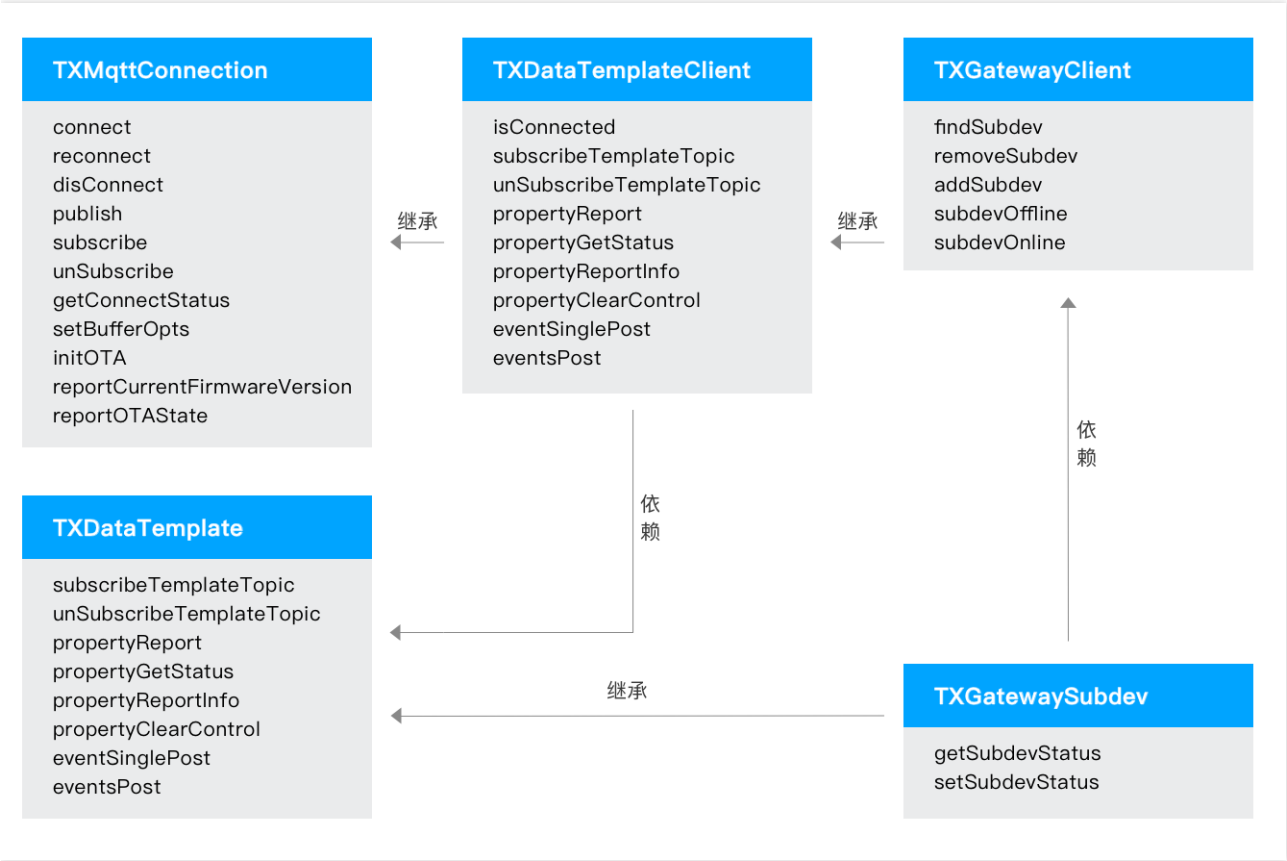
SDK 依赖开源社区实现的 MQTT 协议 **eclipse.paho**，因此需在 App 的 build.gradle 中增加 eclipse.paho 的编译依赖：

```
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
compile "org.eclipse.paho:org.eclipse.paho.client.mqttv3"
}
```

SDK 设计说明

类名	功能
TXMqttConnection	连接物联网开发平台
TXDataTemplate	实现数据模板基本功能
TXDataTemplateClient	实现直连设备根据数据模板连接物联网开发平台
TXGatewayClient	实现网关设备根据数据模板连接物联网开发平台
TXGatewaySubdev	实现网关子设备根据数据模板连接物联网开发平台

版权所有：腾讯云计算（北京）有限责任公司



SDK API 说明

TXMqttConnection

方法名	说明
connect	MQTT 连接
reconnect	MQTT 重连
disConnect	断开 MQTT连接
publish	发布 MQTT 消息
subscribe	订阅 MQTT 主题
unSubscribe	取消订阅 MQTT 主题
getConnectStatus	获取 MQTT 连接状态
setBufferOpts	设置断连状态 buffer 缓冲区

方法名	说明
initOTA	初始化 OTA 功能
reportCurrentFirmwareVersion	上报设备当前版本信息至后台服务器
reportOTAState	上报设备升级状态到后台服务器

TXDataTemplate

方法名	说明
subscribeTemplateTopic	订阅数据模板相关主题
unSubscribeTemplateTopic	取消订阅数据模板相关主题
propertyReport	上报属性
propertyGetStatus	更新状态
propertyReportInfo	上报设备信息
propertyClearControl	清除控制信息
eventSinglePost	上报单个事件
eventsPost	上报多个事件

TXDataTemplateClient

方法名	说明
isConnected	是否已经连接物联网开发平台
subscribeTemplateTopic	订阅数据模板相关主题
unSubscribeTemplateTopic	取消订阅数据模板相关主题
propertyReport	上报属性
propertyGetStatus	更新状态
propertyReportInfo	上报设备信息
propertyClearControl	清除控制信息
eventSinglePost	上报单个事件

版权所有：腾讯云计算（北京）有限责任公司

方法名	说明
eventsPost	上报多个事件

TXGatewayClient

方法名	说明
findSubdev	查找子设备（根据产品 ID 和设备名称）
removeSubdev	删除子设备
addSubdev	添加子设备
subdevOffline	上线子设备
subdevOnline	下线子设备

TXGatewaySubdev

方法名	说明
getSubdevStatus	获取子设备连接状态
setSubdevStatus	设置子设备连接状态

SDK 开发说明

了解数据模板基本功能，可参考 `IoTDataTemplateFragment.java` 和 `DataTemplateSample.java` 。
直连设备开发，可参考 `IoTLightFragment.java` 和 `LightSample.java` 。
网关设备和子设备开发，可参考 `IoTGatewayFragment.java` 、 `GatewaySample.java` 、 `ProductLight.java` 和 `ProductAirconditioner.java` 。

Java SDK 使用说明

最近更新时间：2020-09-09 16:00:09

面向使用 Java 语言的设备，平台提供 Java SDK 实现接入腾讯云 IoT Explorer。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 explorer-device-java。

如果您需要通过 jar 引用方式进行项目开发，可在 module 目录下的 build.gradle 中添加依赖，如下依赖：

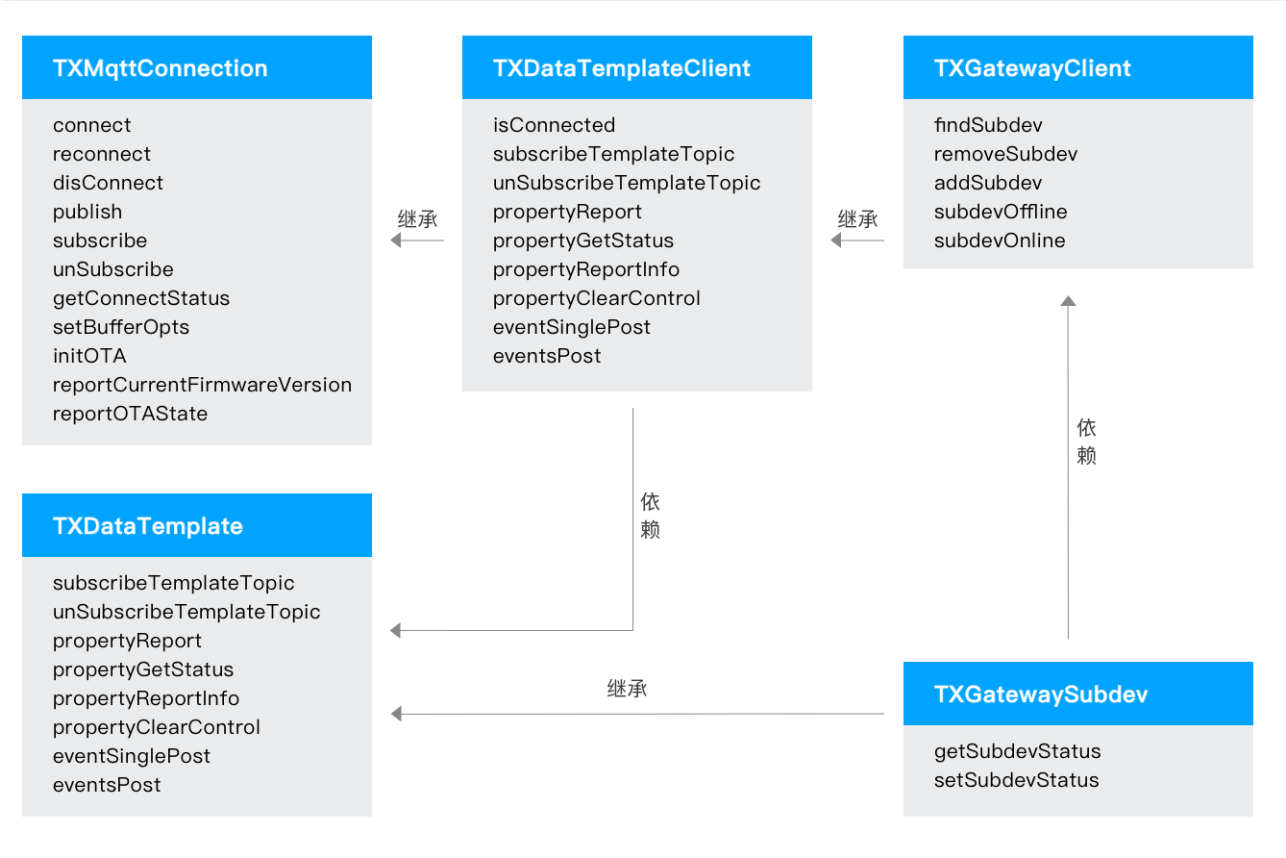
```
dependencies {
...
implementation("com.tencent.iot.explorer:explorer-device-java:1.0.0")
}
```

SDK 设计说明

类名	功能
TXMqttConnection	连接物联网开发平台
TXDataTemplate	实现数据模板基本功能
TXDataTemplateClient	实现直连设备根据数据模板连接物联网开发平台
TXGatewayClient	实现网关设备根据数据模板连接物联网开发平台
TXGatewaySubdev	实现网关子设备根据数据模板连接物联网开发平台

版权所有：腾讯云计算（北京）有限责任公司

腾讯云 IoT Explorer Java SDK 架构图如下：



SDK API 说明

TXMqttConnection

方法名	说明
connect	MQTT 连接
reconnect	MQTT 重连
disConnect	断开 MQTT连接
publish	发布 MQTT 消息
subscribe	订阅 MQTT 主题
unSubscribe	取消订阅 MQTT 主题
getConnectStatus	获取 MQTT 连接状态

方法名	说明
setBufferOpts	设置断连状态 buffer 缓冲区
initOTA	初始化 OTA 功能
reportCurrentFirmwareVersion	上报设备当前版本信息至后台服务器
reportOTAState	上报设备升级状态到后台服务器

TXDataTemplate

方法名	说明
subscribeTemplateTopic	订阅数据模板相关主题
unSubscribeTemplateTopic	取消订阅数据模板相关主题
propertyReport	上报属性
propertyGetStatus	更新状态
propertyReportInfo	上报设备信息
propertyClearControl	清除控制信息
eventSinglePost	上报单个事件
eventsPost	上报多个事件

TXDataTemplateClient

方法名	说明
isConnected	是否已经连接物联网开发平台
subscribeTemplateTopic	订阅数据模板相关主题
unSubscribeTemplateTopic	取消订阅数据模板相关主题
propertyReport	上报属性
propertyGetStatus	更新状态
propertyReportInfo	上报设备信息
propertyClearControl	清除控制信息

版权所有：腾讯云计算（北京）有限责任公司

方法名	说明
eventSinglePost	上报单个事件
eventsPost	上报多个事件

TXGatewayClient

方法名	说明
findSubdev	查找子设备（根据产品 ID 和设备名称）
removeSubdev	删除子设备
addSubdev	添加子设备
subdevOffline	上线子设备
subdevOnline	下线子设备

TXGatewaySubdev

方法名	说明
getSubdevStatus	获取子设备连接状态
setSubdevStatus	设置子设备连接状态

版权所有：腾讯云计算（北京）有限责任公司

基于 TencentOS tiny 开发示例说明

最近更新时间：2020-09-10 14:16:12

本文档为您介绍如何将腾讯物联网终端操作系统（TencentOS tiny）接入腾讯物联网开发平台 IoT Explorer。

简介

TencentOS tiny 是腾讯面向物联网领域开发的实时操作系统，具有低功耗、低资源占用、模块化、安全可靠等特点，可有效提升物联网终端产品开发效率。TencentOS tiny 提供精简的 RTOS 内核，内核组件可裁剪、可配置并且可快速移植到多种主流 MCU 及模组芯片上。而且，基于 RTOS 内核提供了丰富的物联网组件，内部集成主流物联网协议栈（如 CoAP/MQTT/TLS/DTLS/LoRaWAN/NB-IoT 等），可助力物联网终端设备及业务快速接入腾讯云物联网开发平台。

更多详细信息可参见 腾讯物联网终端操作系统。

操作步骤

按照以下步骤，先进行接入准备工作，再分别对内核、框架及 C-SDK 等进行移植，最终通过 TencentOS tiny 接入腾讯云物联网开发平台，实现物联网终端设备及业务快速接入 IoT Explorer。

1. TencentOS tiny 移植环境准备。
2. 内核移植。
3. 移植 AT 框架、SAL 框架、模组驱动。
4. 移植腾讯云 C-SDK。

版权所有：腾讯云计算（北京）有限责任公司

TencentOS tiny 移植环境准备

最近更新时间：2020-09-10 14:16:18

本文为您介绍 TencentOS tiny 接入腾讯物联网开发平台相关的准备工作。

步骤一：准备目标硬件（开发板/芯片/模组）

TencentOS tiny 目前主要支持 ARM Cortex M 核芯片的移植，例如 STM32、NXP 芯片支持 Cortex M 核全系列。本教程将使用 TencentOS tiny 官方开发板 EVB_MX_Plus 演示如何移植，其他 ARM Cortex M 系列开发板和芯片移植方法类似。

调试 ARM Cortex M 核还需要仿真器， 由于 EVB_MX_Plus 开发板没有板载 ST-Link 调试器，需要连接外置的仿真器，例如 J-Link、U-Link 等，本教程中使用 ST-Link。

步骤二：准备编译器环境

TencentOS tiny 支持 Keil MDK、IAR、Gcc 三种开发环境，本文中以 Keil MDK 作为开发环境。

1. 移植内核前需要先安装能编译 ARM Cortex M 核的 Keil 编译器（Keil 编译器别名也称为 MDK），可下载 最新版本5.31 进行使用。

   > 说明：
   > 填写注册信息即可下载，下载完成在 Windows 环境下按照提示安装即可，安装完成后需要自行购买软件 License，避免 32K Flash 下载限制。

2. 由于新版本的 MDK 编译器和芯片支持包分离，所以 MDK 安装完成后，还需要安装对应芯片的器件支持包（PACK 包）。例如本教程示例的 EVB_MX_Plus 开发板的芯片是 STM32L431RCT6，则需要安装 Keil.STM32L0xx_DFP.2.0.1.pack 系列器件支持包。

   > 说明：
   > 您只需根据您的芯片型号，下载对应的 PACK 包即可，同时您也可以在 MDK 集成开发环境中进行在线下载安装。

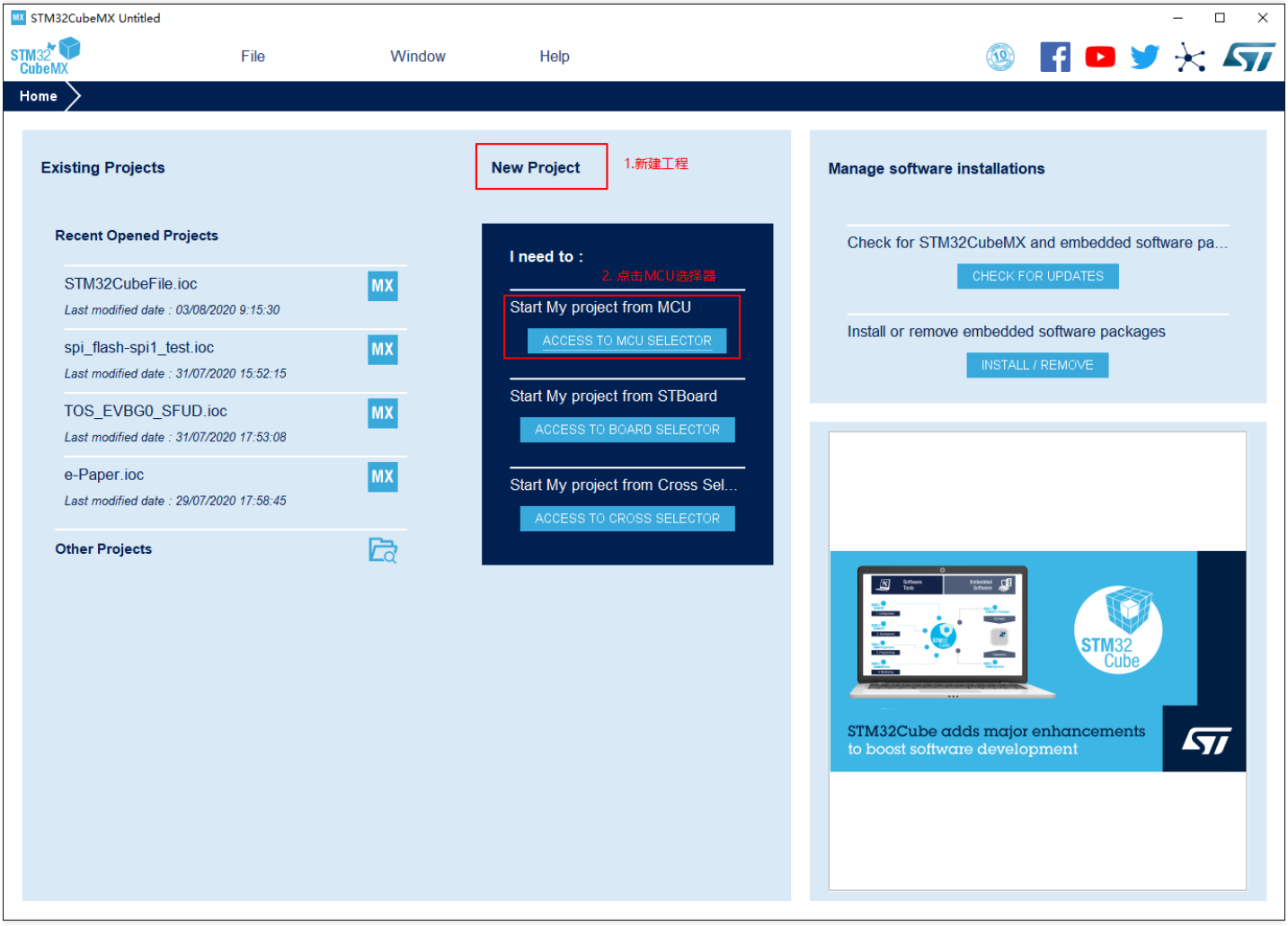
步骤三：准备芯片对应的裸机工程

移植 TencentOS tiny 基础内核需要您提前准备一个芯片对应的裸机工程，裸机工程包含基本的芯片启动文件、基础配置（时钟、主频等）、以及串口和基本 GPIO 驱动，用于进行 RTOS 测试。

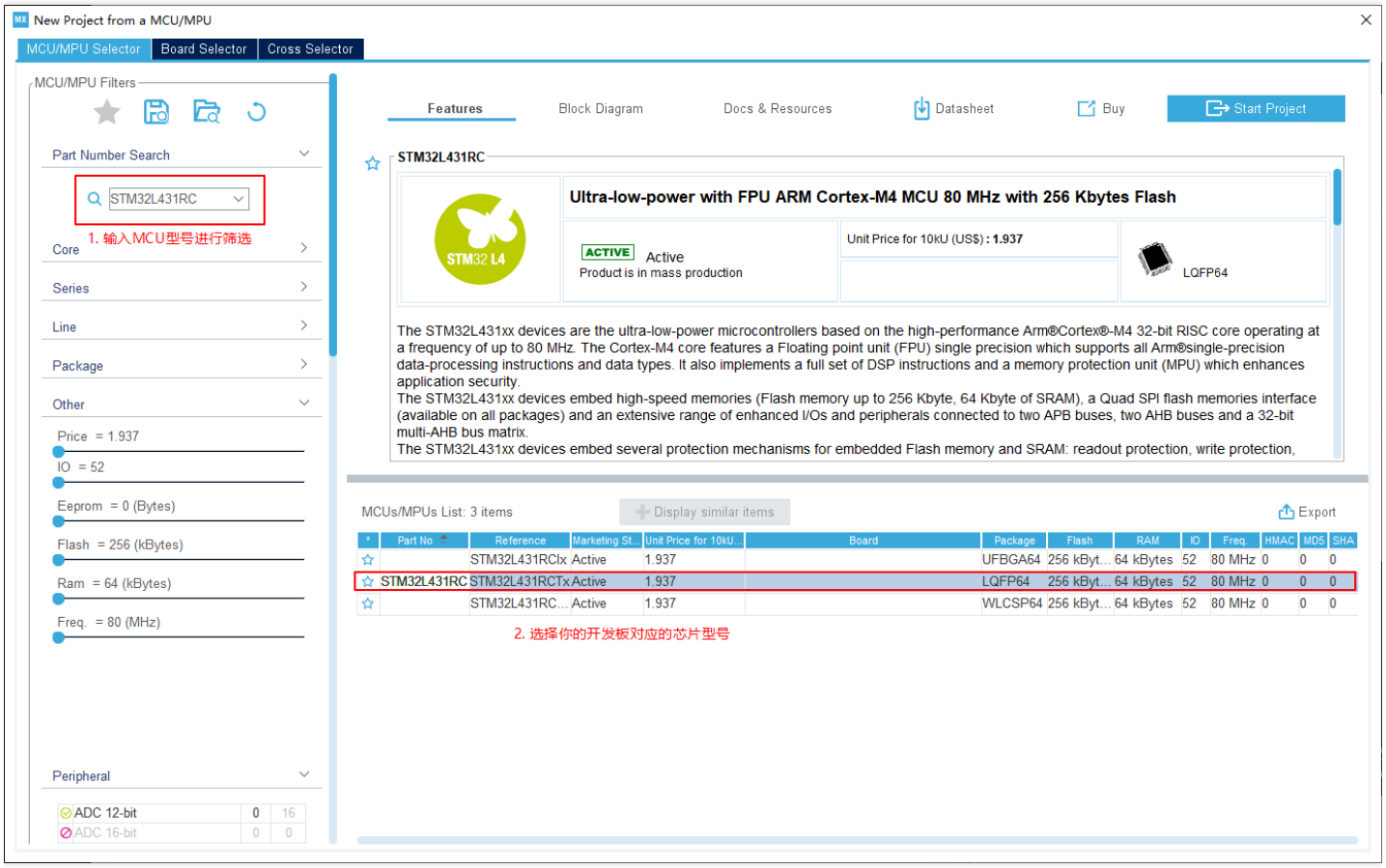
本教程使用 ST 官方的 STM32CubeMX 软件进行自动化生成 MDK 裸机工程，同时，安装 STM32CubeMx 需确保已安装好 JDK 环境。CubeMX 安装完成后，使用 CubeMX 在 EVB_MX_Plus 开发板上生成裸机工程。

> 说明：
> 如果您的芯片不是 STM32，而是其他厂商的 ARM Cortex M 系列，您可以根据产商的指导准备裸机工程，但后续内核移植步骤一致。

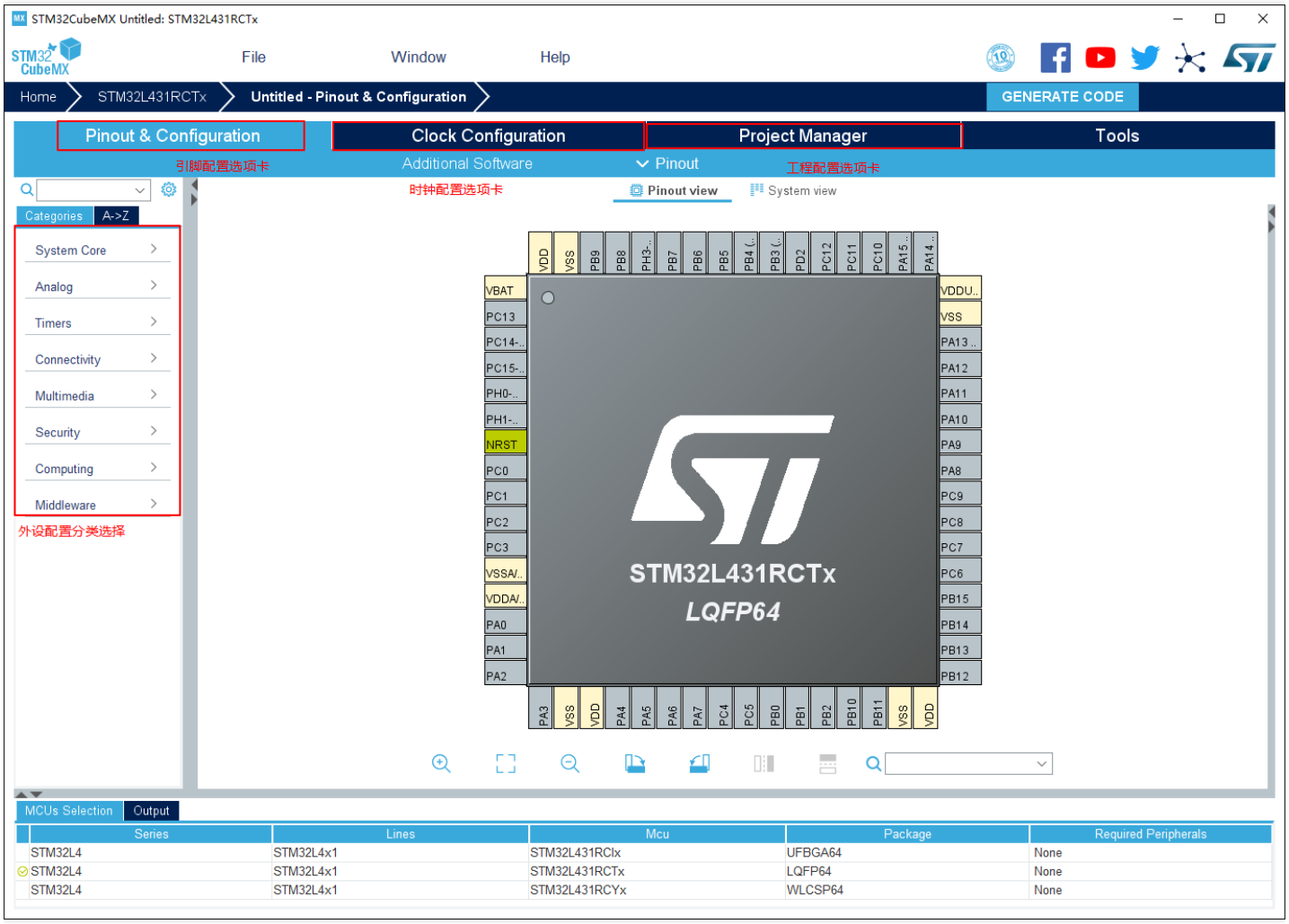
1. 启动 STM32CubeMX，新建工程



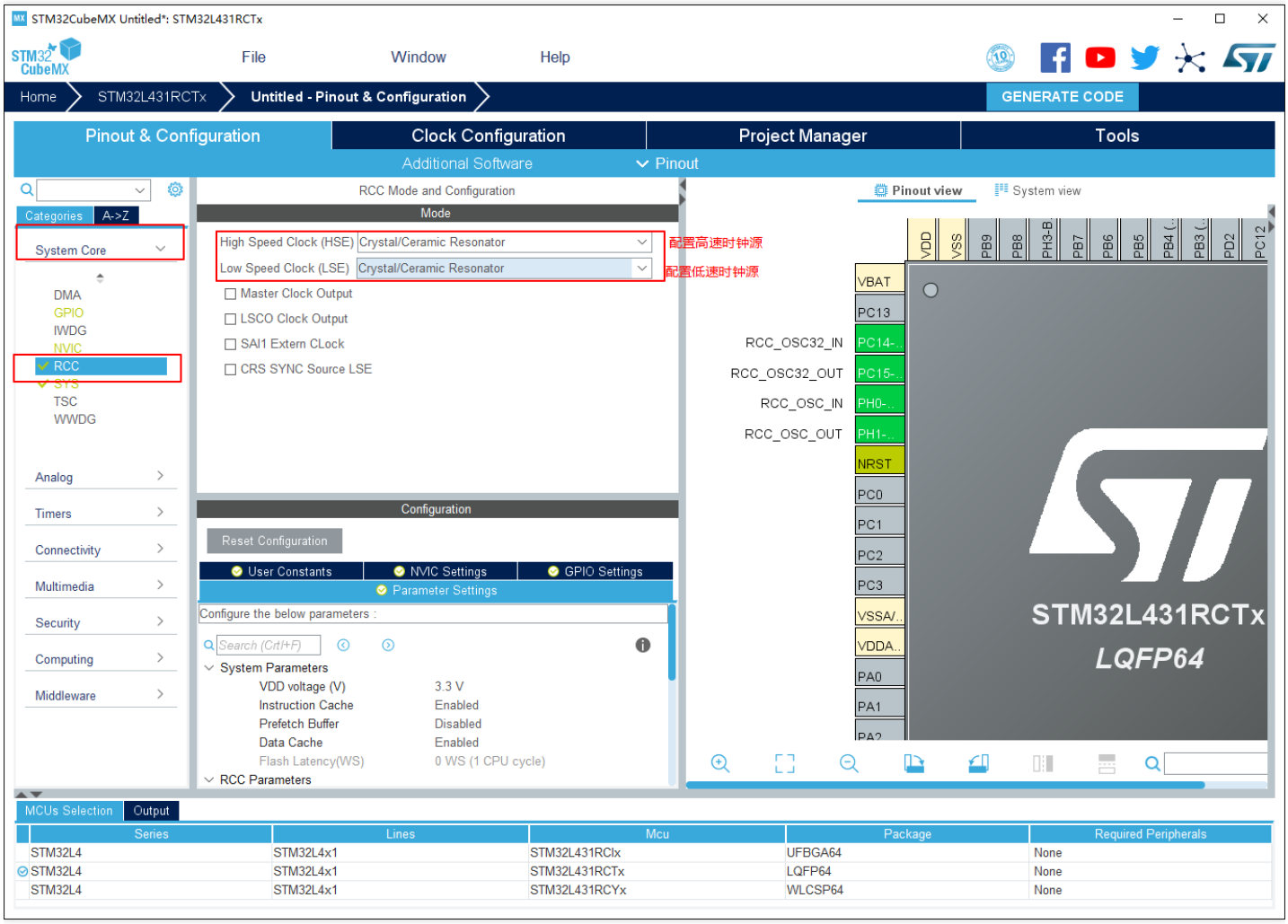
2. 选择 MCU 型号



如上图所示：通过 MCU 筛选来找到自己开发板对应的芯片型号，双击后弹出工程配置界面，如下图：

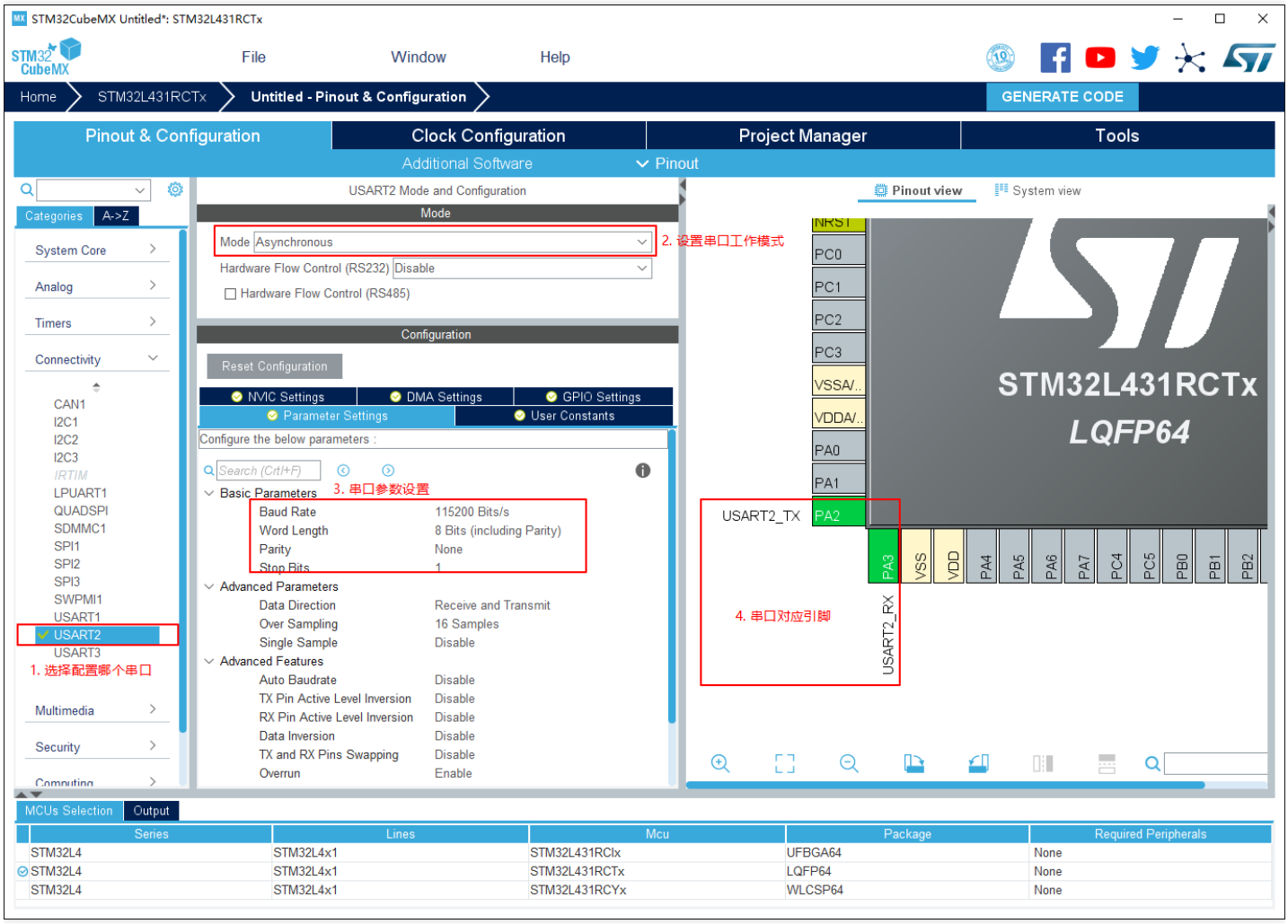


3. Pin 设置界面配置时钟源

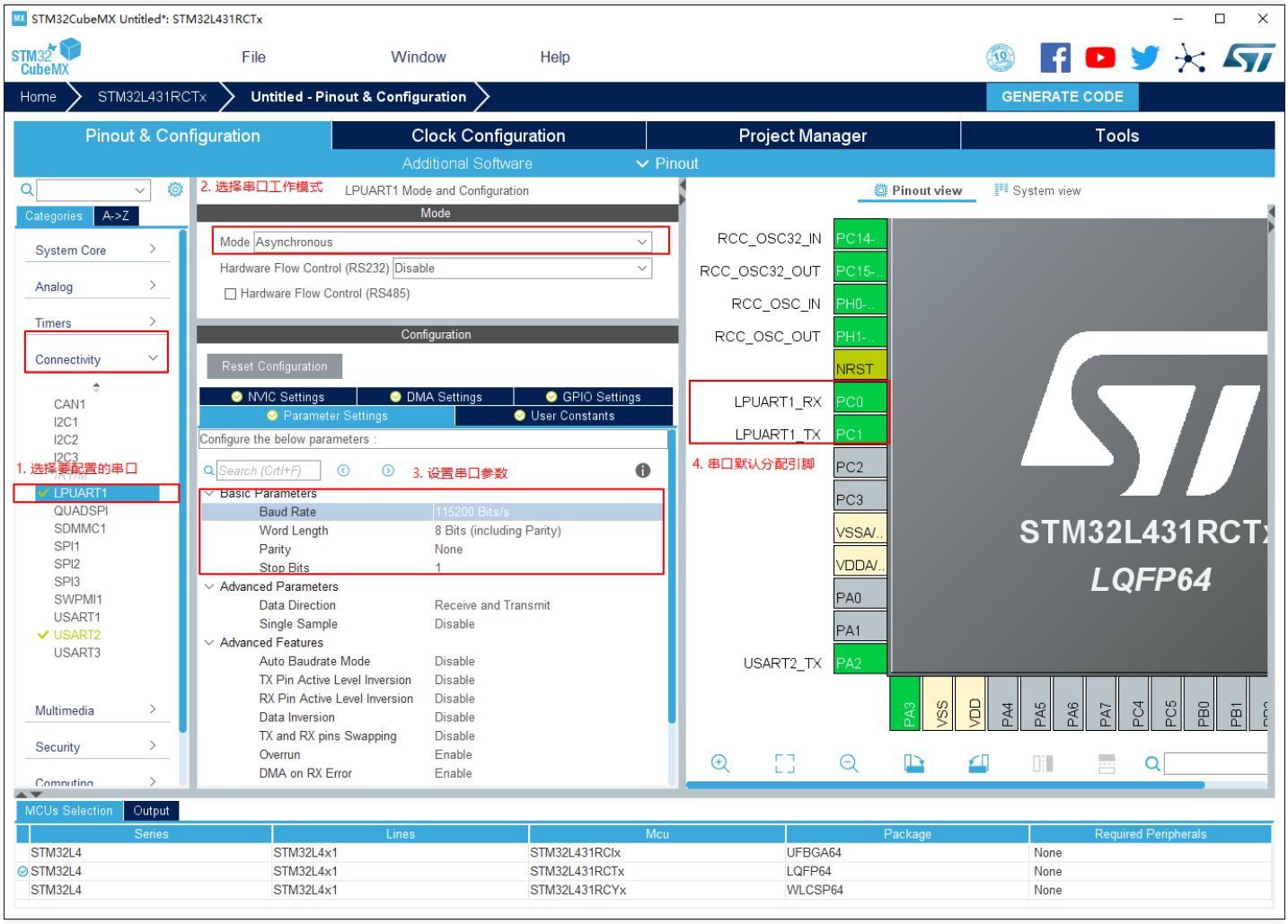


4. Pin 设置界面配置串口

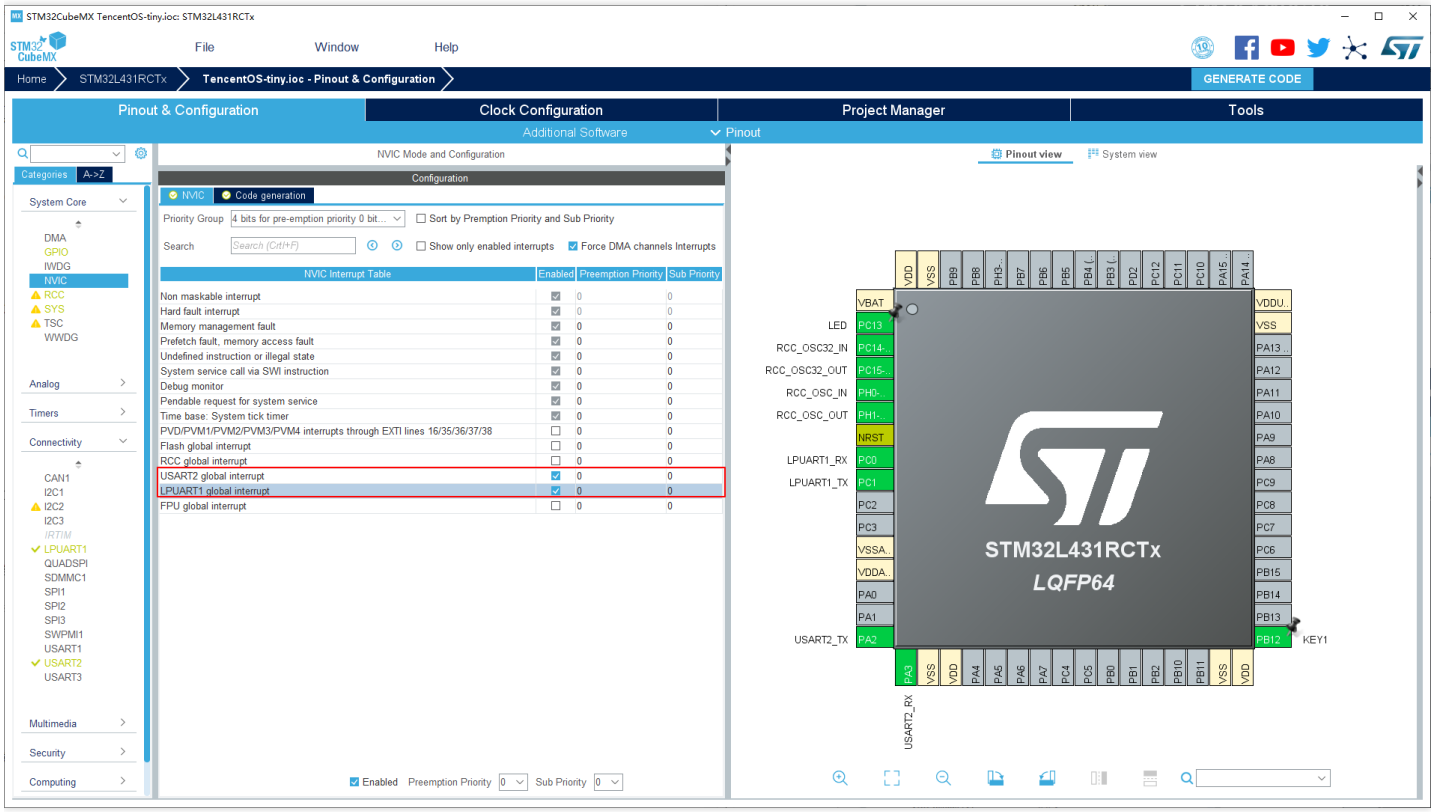
EVB_MX_Plus 开发板用于日志打印 USART2 串口，配置 USART2 如下图：



EVB_MX_Plus 开发板用于与通信模组通信的 LPUART1 串口，配置 LPUART1 如下图：

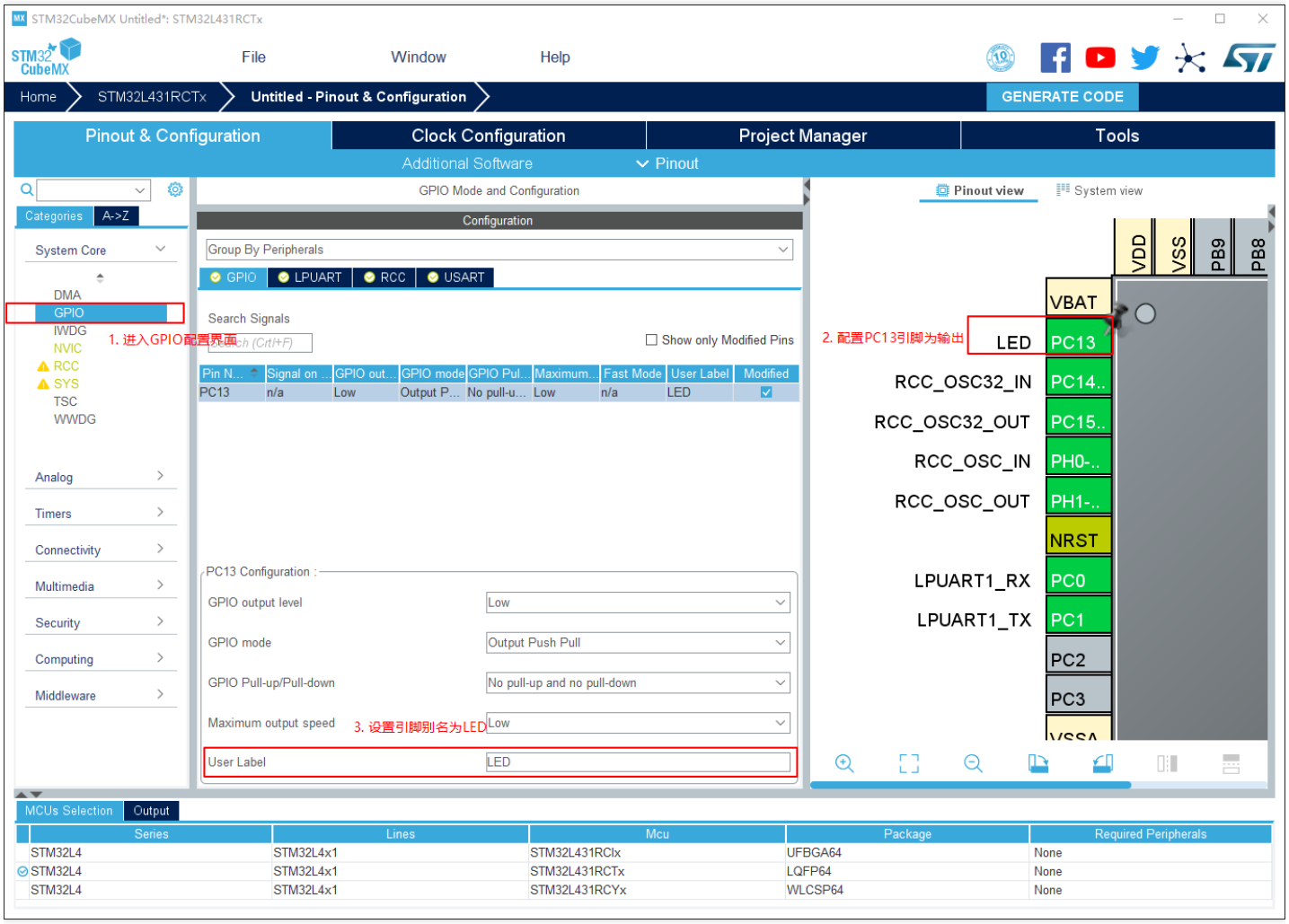


配置使能串口中断，如下图：

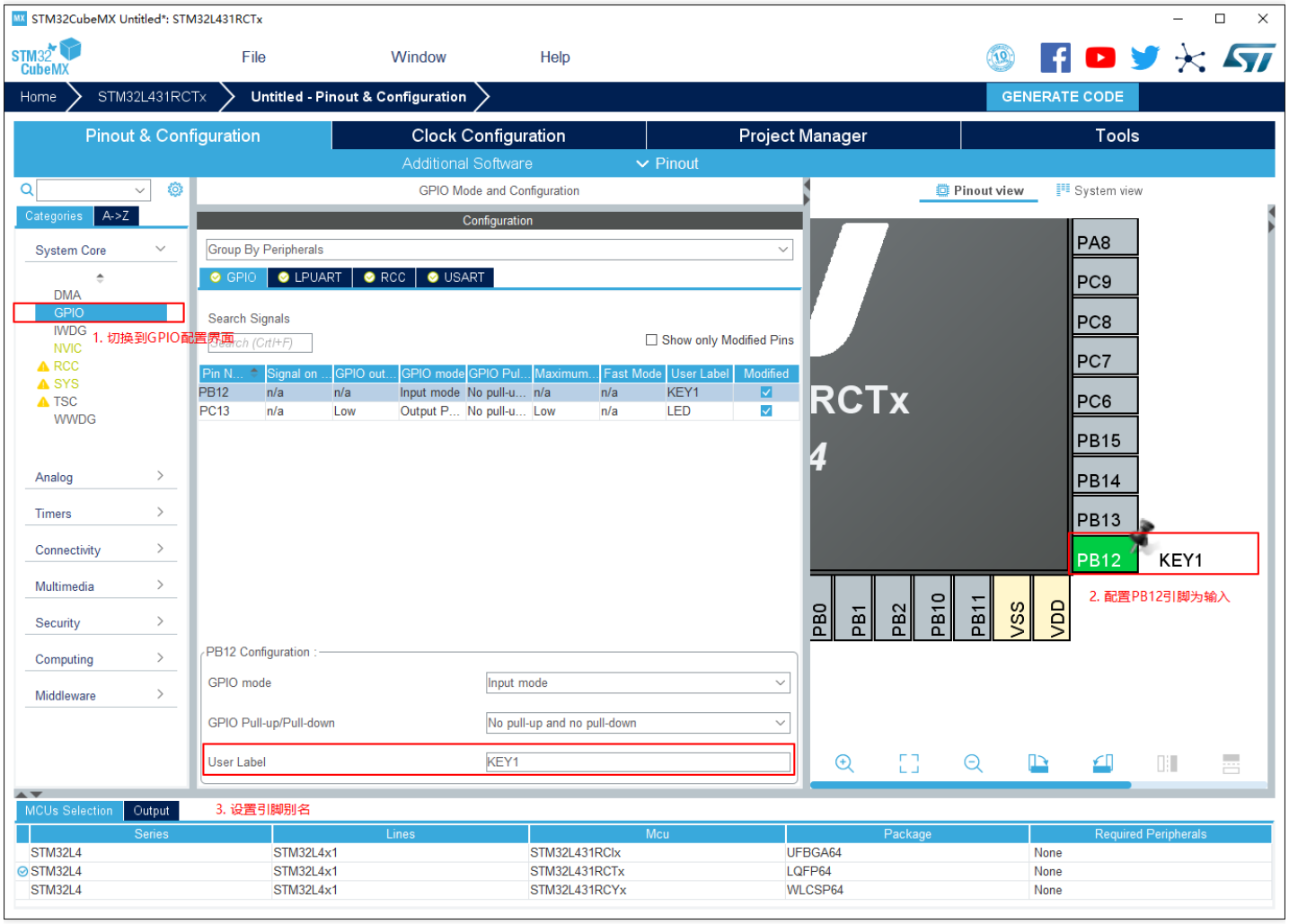


5. Pin 设置界面配置 GPIO

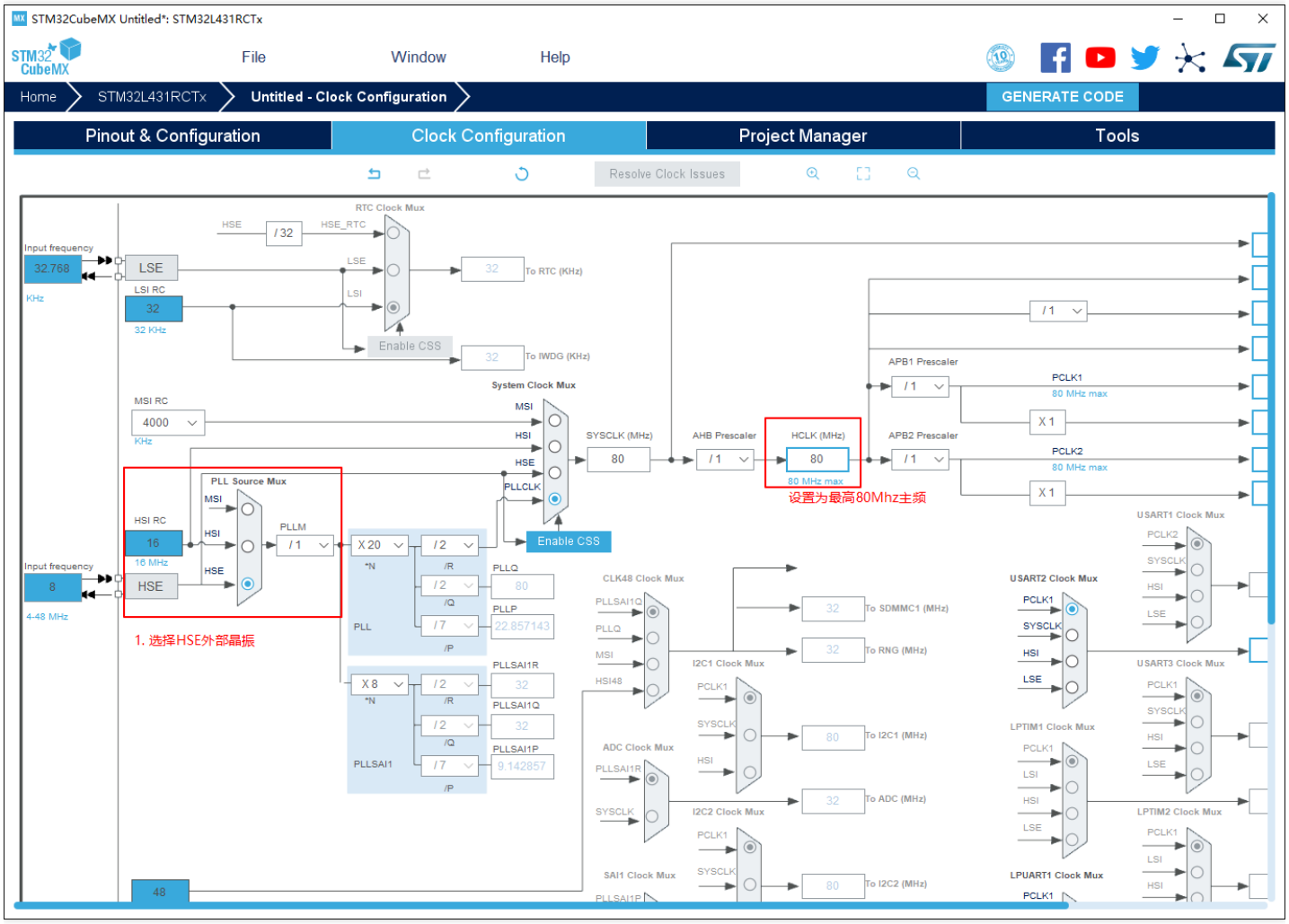
EVB_MX_Plus 开发板板载一颗 LED，连接在 PC13 引脚，相关配置如下图：



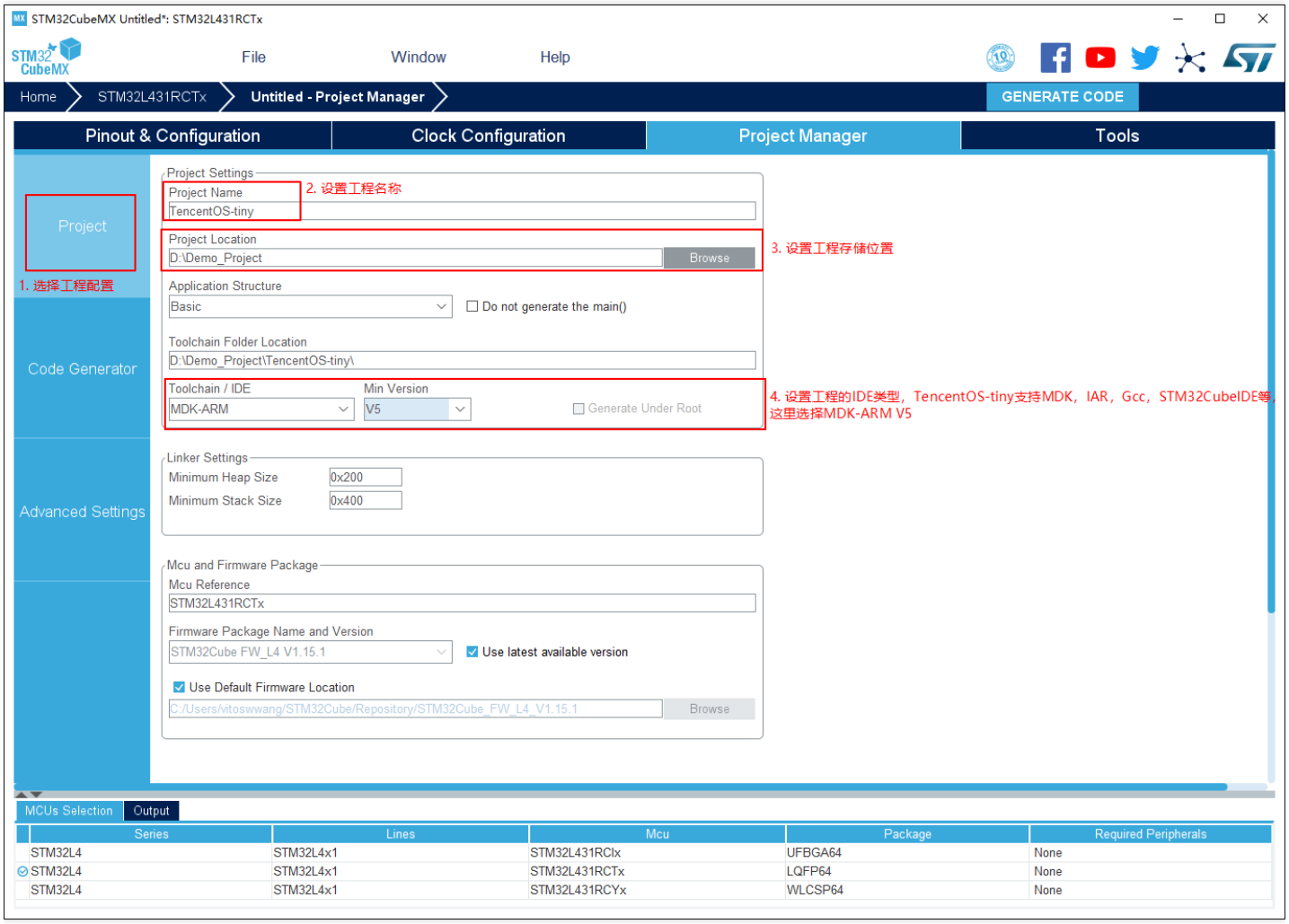
EVB_MX_Plus 开发板板载四个按键，其中 KEY1 连接在 PB12，相关配置如下图：


STM32CubeMX Untitled*: STM32L431RCTx
File
Window
Help
Home
STM32L431RCTx
Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Software
Pinout
GPIO Mode and Configuration
Configuration
Group By Peripherals
GPIO
LPUART
RCC
USART
Search Signals
Show only Modified Pins
Pin N...
Signal on ...
GPIO out...
GPIO mode
GPIO Pul...
Maximum...
Fast Mode
User Label
Modified
PB12
n/a
n/a
Input mode
No pull-u...
n/a
n/a
KEY1
PC13
n/a
Low
Output P...
No pull-u...
Low
n/a
LED
1. 切换到GPIO配置界面
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
TSC
WWDG
Analog
Timers
Connectivity
Multimedia
Security
Computing
Middleware
PB12 Configuration :
GPIO mode
Input mode
GPIO Pull-up/Pull-down
No pull-up and no pull-down
User Label
KEY1
Pinout view
System view
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
PB12
KEY1
2. 配置PB12引脚为输入
PB0
PB1
PB2
PB10
PB11
VSS
VDD
MCUs Selection
Output
3. 设置引脚别名
Series
Lines
Mcu
Package
Required Peripherals
STM32L4
STM32L4x1
STM32L431RCIx
UFBGA64
None
STM32L4
STM32L4x1
STM32L431RCTx
LQFP64
None
STM32L4
STM32L4x1
STM32L431RCYx
WLCSP64
None

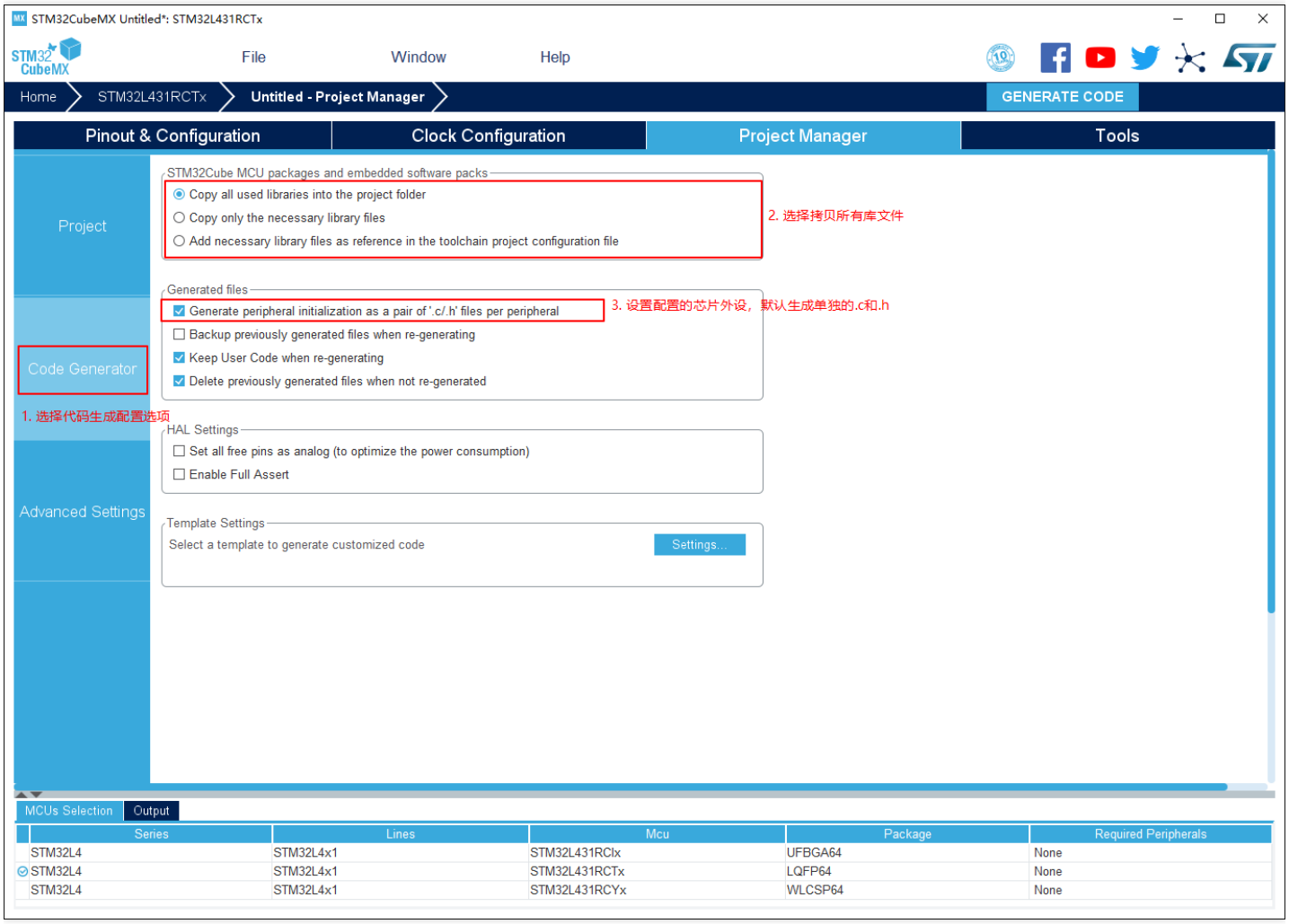
6. 配置总线时钟


STM32CubeMX Untitled*: STM32L431RCTx
File
Window
Help
Home
STM32L431RCTx
Untitled - Clock Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Resolve Clock Issues
RTC Clock Mux
Input frequency
32.768
KHz
LSE
LSI RC
32
32 KHz
HSE
/ 32
HSE_RTC
LSE
LSI
Enable CSS
To RTC (KHz)
To IWDG (KHz)
System Clock Mux
MSI RC
4000
KHz
MSI
HSI
HSE
PLLCLK
SYSCLK (MHz)
AHB Prescaler
HCLK (MHz)
APB1 Prescaler
APB2 Prescaler
PCLK1
PCLK2
80 MHz max
设置为最高80Mhz主频
PLL Source Mux
HSI RC
16
16 MHz
Input frequency
8
4-48 MHz
HSE
PLLM
X 20
/ 2
PLLQ
PLLP
22.857143
X 8
PLLSAI1R
PLLSAI1Q
PLLSAI1P
9.142857
PLL
PLLSAI1
1. 选择HSE外部晶振
CLK48 Clock Mux
ADC Clock Mux
SAI1 Clock Mux
I2C1 Clock Mux
I2C2 Clock Mux
To SDMMC1 (MHz)
To RNG (MHz)
To I2C1 (MHz)
To ADC (MHz)
To I2C2 (MHz)
USART1 Clock Mux
USART2 Clock Mux
USART3 Clock Mux
LPTIM1 Clock Mux
LPTIM2 Clock Mux
LPUART1 Clock Mux
48

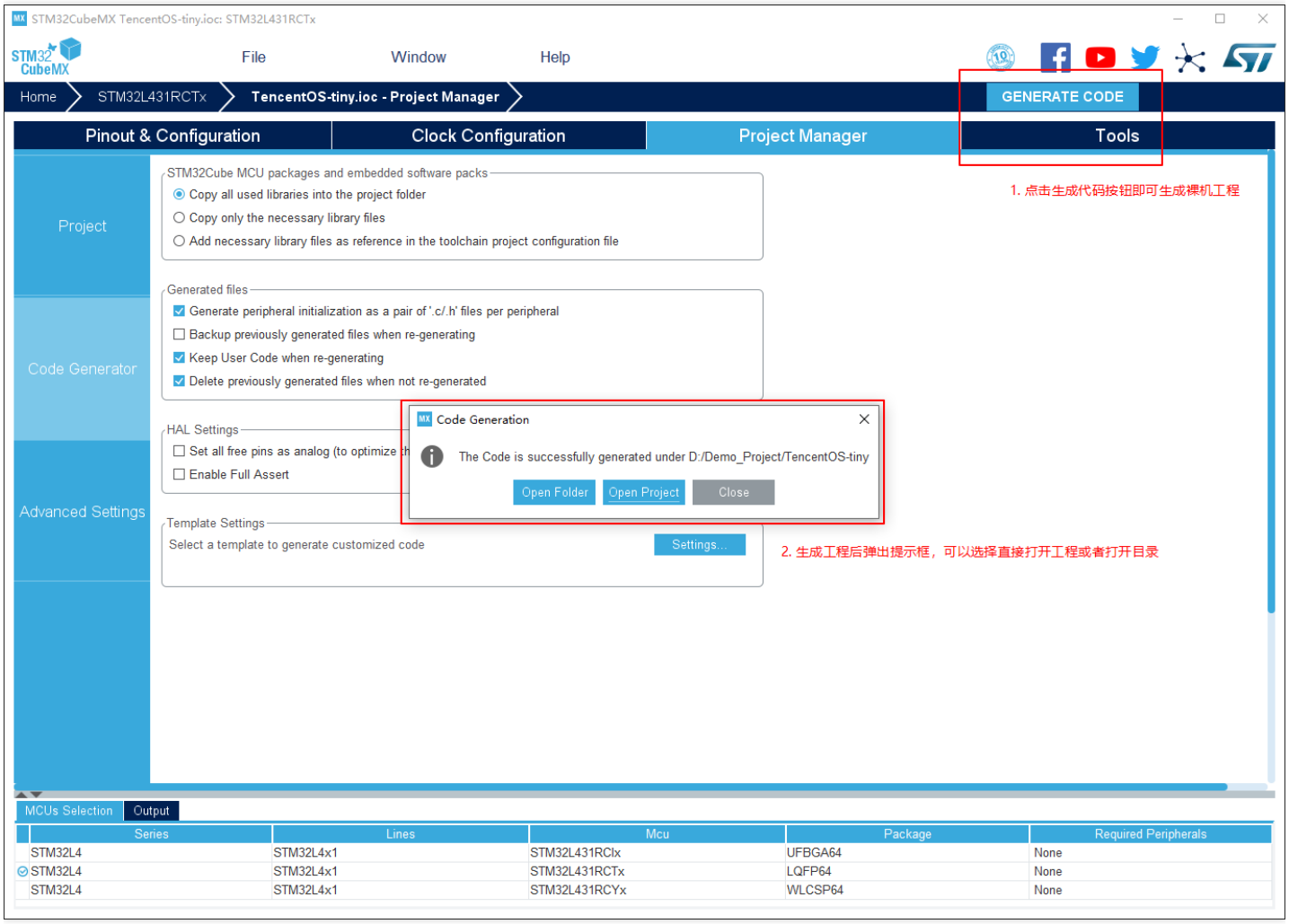
7. 工程生成参数配置


STM32CubeMX Untitled*: STM32L431RCTx
File
Window
Help
Home
STM32L431RCTx
Untitled - Project Manager
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Project
Code Generator
Advanced Settings
1. 选择工程配置
Project Settings
Project Name
TencentOS-tiny
2. 设置工程名称
Project Location
D:\Demo_Project
Browse
3. 设置工程存储位置
Application Structure
Basic
Do not generate the main()
Toolchain Folder Location
D:\Demo_Project\TencentOS-tiny\
Toolchain / IDE
MDK-ARM
Min Version
V5
Generate Under Root
4. 设置工程的IDE类型，TencentOS-tiny支持MDK，IAR，Gcc，STM32CubeIDE等
这里选择MDK-ARM V5
Linker Settings
Minimum Heap Size
0x200
Minimum Stack Size
0x400
Mcu and Firmware Package
Mcu Reference
STM32L431RCTx
Firmware Package Name and Version
STM32Cube FW_L4 V1.15.1
Use latest available version
Use Default Firmware Location
C:/Users/vitoswwang/STM32Cube/Repository/STM32Cube_FW_L4_V1.15.1
Browse
MCUs Selection
Output
Series
Lines
Mcu
Package
Required Peripherals
STM32L4
STM32L4x1
STM32L431RCIx
UFBGA64
None
STM32L4
STM32L4x1
STM32L431RCTx
LQFP64
None
STM32L4
STM32L4x1
STM32L431RCYx
WLCSP64
None

8. 代码生成方式配置

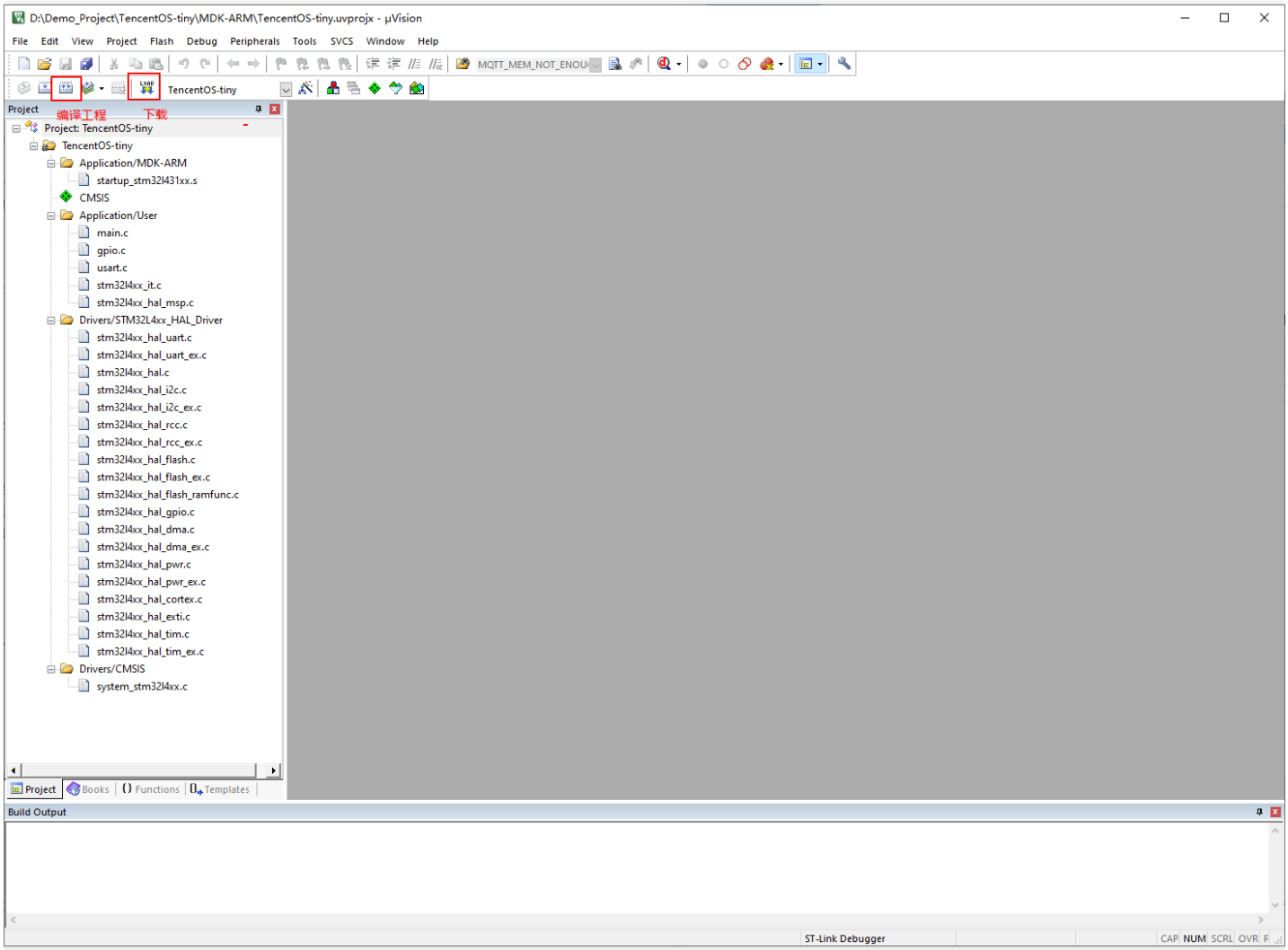

STM32CubeMX Untitled*: STM32L431RCTx
File
Window
Help
Home
STM32L431RCTx
Untitled - Project Manager
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Project
Code Generator
Advanced Settings
STM32Cube MCU packages and embedded software packs
Copy all used libraries into the project folder
Copy only the necessary library files
Add necessary library files as reference in the toolchain project configuration file
2. 选择拷贝所有库文件
Generated files
Generate peripheral initialization as a pair of '.c/.h' files per peripheral
3. 设置配置的芯片外设，默认生成单独的.c和.h
Backup previously generated files when re-generating
Keep User Code when re-generating
Delete previously generated files when not re-generated
1. 选择代码生成配置选项
HAL Settings
Set all free pins as analog (to optimize the power consumption)
Enable Full Assert
Template Settings
Select a template to generate customized code
Settings...
MCUs Selection
Output
Series
Lines
Mcu
Package
Required Peripherals
STM32L4
STM32L4x1
STM32L431RCIx
UFBGA64
None
STM32L4
STM32L4x1
STM32L431RCTx
LQFP64
None
STM32L4
STM32L4x1
STM32L431RCYx
WLCSP64
None

9. 生成工程



10. keil 的裸机工程

打开生成的裸机工程效果如下：



编译工程下载之后，EVB_MX_Plus 开发板的裸机工程生成完成，该工程可直接编译并烧写在开发板上运行。

步骤四：准备 TencentOS tiny 的源码

TencentOS tiny 的源码已经开源，可从 GitHub 文件库中下载使用。

> 说明：
> 由于下一步骤只介绍 TencentOS tiny 的内核移植，所以这里只需要用到 arch、board、kernel、osal 四个目录下的源码。

一级目录	二级目录	说明

一级目录	二级目录	说明
arch	arm	TencentOS tiny 适配的 IP 核架构（含 M 核中断、调度、tick 相关代码）
board	...	移植目标芯片的工程文件
kernel	core	TencentOS tiny 内核源码
	pm	TencentOS tiny 低功耗模块源码
osal	cmsis_os	TencentOS tiny 提供的 cmsis os 适配
net	at	AT 框架源码
	sal_module_wrapper	SAL 框架源码
	socket_wrapper	socket 封装封装层源码
	tencent_firmware_module_wrapper	腾讯云定制固件封装层源码
	lora_module_wrapper	lora 封装层源码
devices	...	各种通信模组的驱动适配层

步骤五：下一步操作

请前往 内核移植 进行内核移植操作。

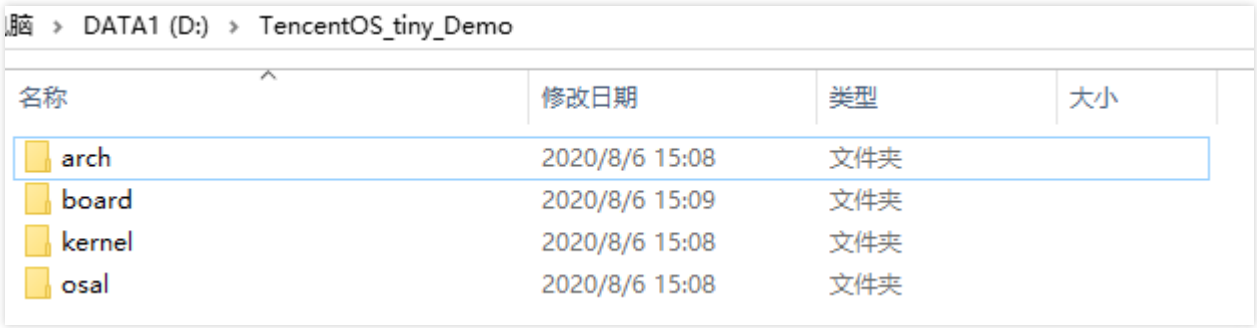
内核移植

最近更新时间：2020-09-11 09:02:47

本文将为您介绍 TencentOS tiny 接入腾讯物联网开发平台内核移植的相关操作。

步骤一：代码目录规划

1. 新建 TencentOS_tiny_Demo 主目录，并在主目录下添加四个子目录：arch、board、kernel、osal，其中 arch、kernel、osal 目录文件可直接从代码仓拷贝。



2. 在 board 目录下放入上个步骤中生成的 裸机工程代码，并将移植的开发板取名为 EVB_MX_Plus_Demo，将裸机代码全部拷贝到该目录下。

3. 进入 `TencentOS_tiny_Demo\board\EVB_MX_Plus_Demo\MDK-ARM` 目录，打开 Keil 工程，开始进行下一步，添加 TencentOS tiny 内核代码。

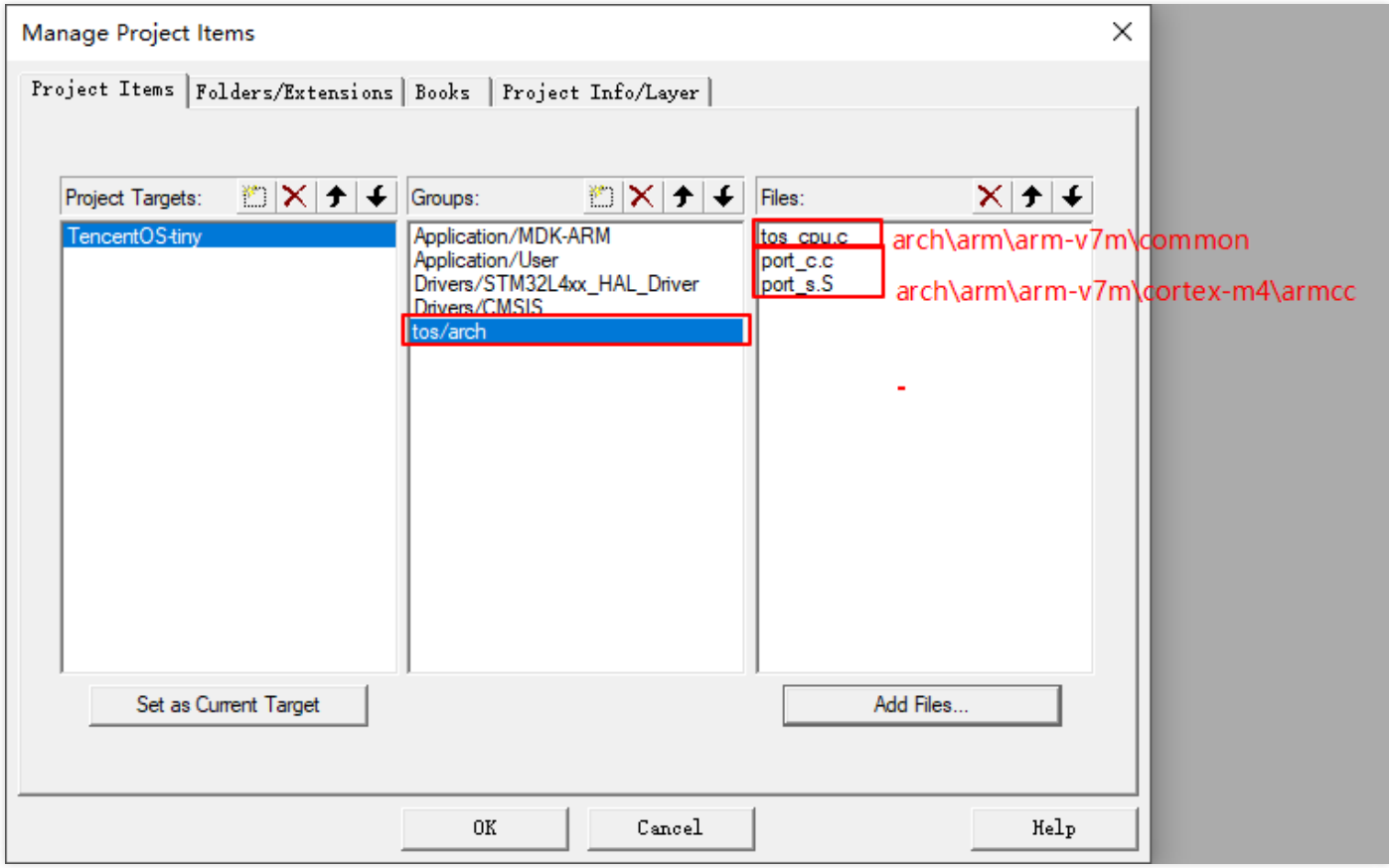
步骤二：添加 TencentOS tiny 内核代码

TencentOS tiny 的内核代码添加分为添加 arch 平台代码、内核源码、 cmsis os 源码及 tos_config.h 头文件等步骤。

1. 添加 arch 平台代码

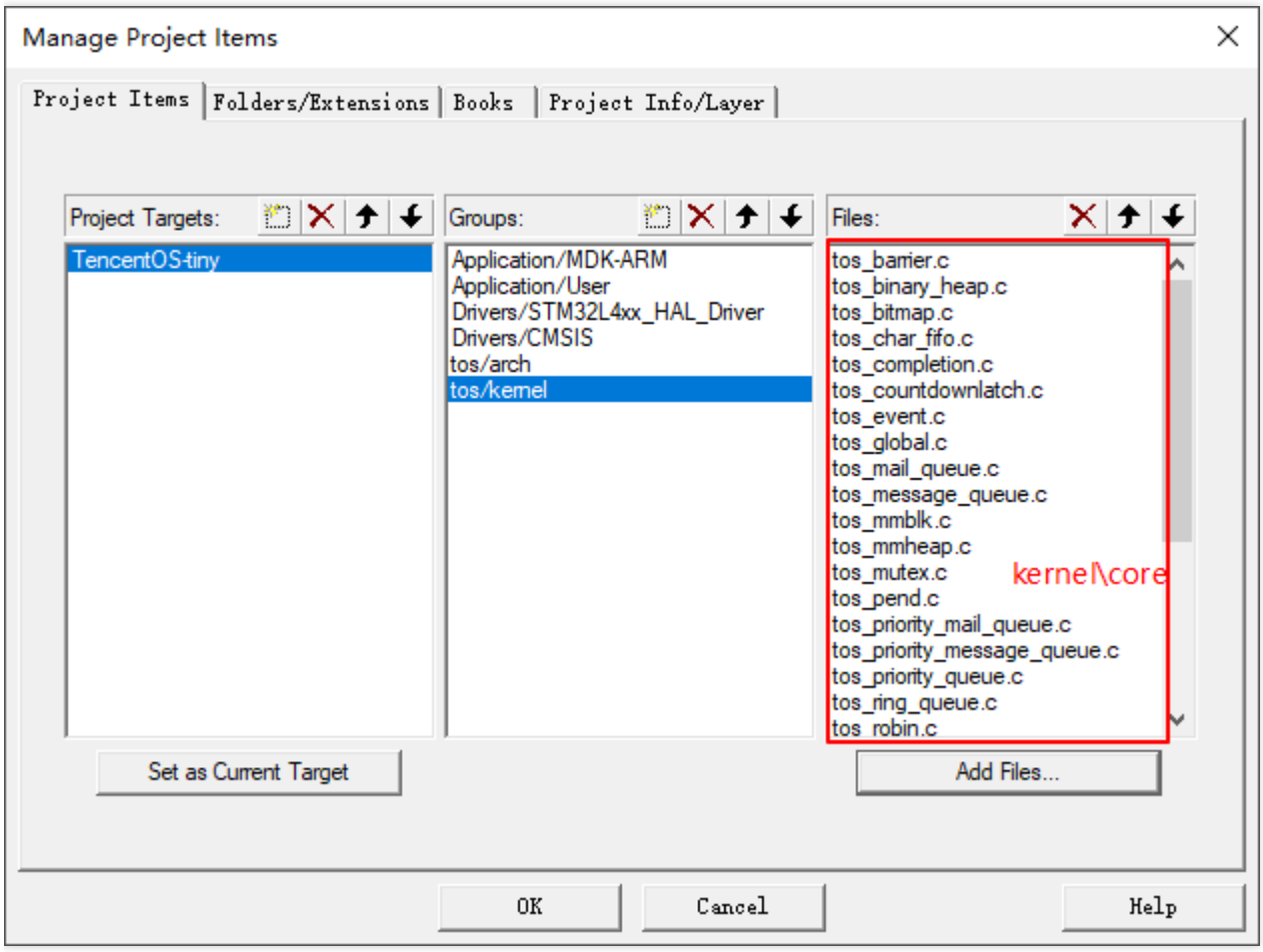
- tos_cpu.c 是 TencentOS tiny 的 CPU 适配文件，包括堆栈初始化、中断适配等。
  - 如果您的芯片是 ARM Cortex M 核，例如：M0、M3 、M4、M7 通用核，则该文件无需改动。
  - 如果您的芯片是其他 IP 核时，则需要重新适配。

- port_s.S 文件是 TencentOS tiny 的任务调度汇编代码，主要用于弹栈压栈等处理工作；port_c.c 文件适配 systick 。这两个文件每个 IP 核和编译器都不相同。
  - 如果您使用的芯片是 ARM Cortex M 核，则无需再进行配置，例如移植的芯片是 STM32L431RCT6，属于 ARM Cortex M4 核，使用的编译器是 Keil，直接选择 `arch\arm\arm-v7m\cortex-m4\armcc` 目录下的适配代码。
  - 如果您的开发板是 STM32F429IG M4 核，使用的编译器是 GCC，则选择 `arch\arm\arm-v7m\cortex-m4\gcc` 目录下的适配文件。



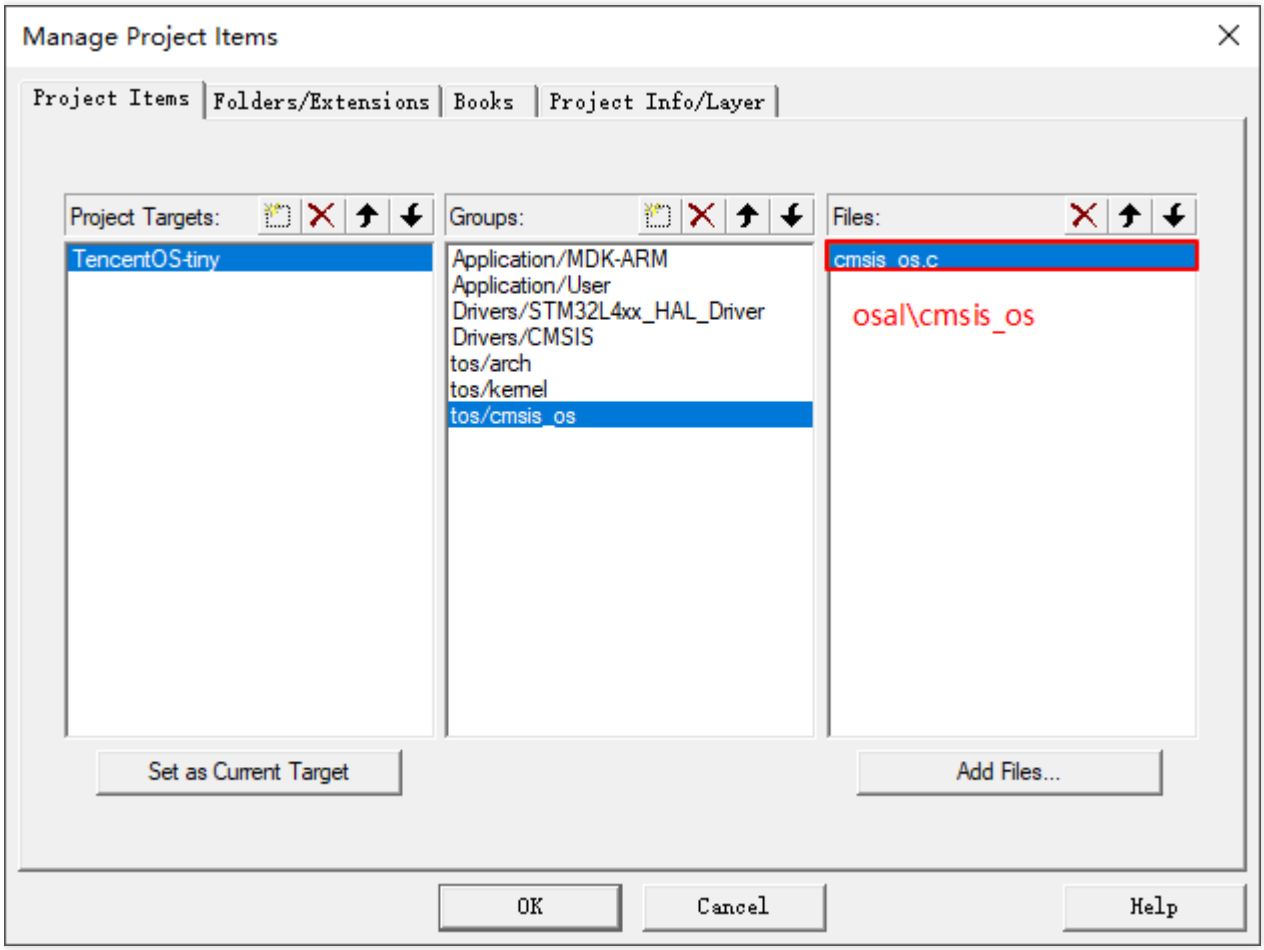
2. 添加内核源码

内核源码 kernel 目录下包含 core 和 pm 两个目录，其中 core 目录下的代码为基础内核，pm 目录下的代码为低功耗组件。进行基础移植时无需添加 pm 目录下的代码，只需添加全部的基本内核源码。



3. 添加 cmsis os 源码

cmsis os 是 TencentOS tiny 为兼容 cmsis 标准而适配的 OS 抽象层，可以简化用户将业务从其他 RTOS 迁移到 TencentOS tiny 的工作量。



4. 新建 TencentOS tiny 系统配置文件 tos_config.h

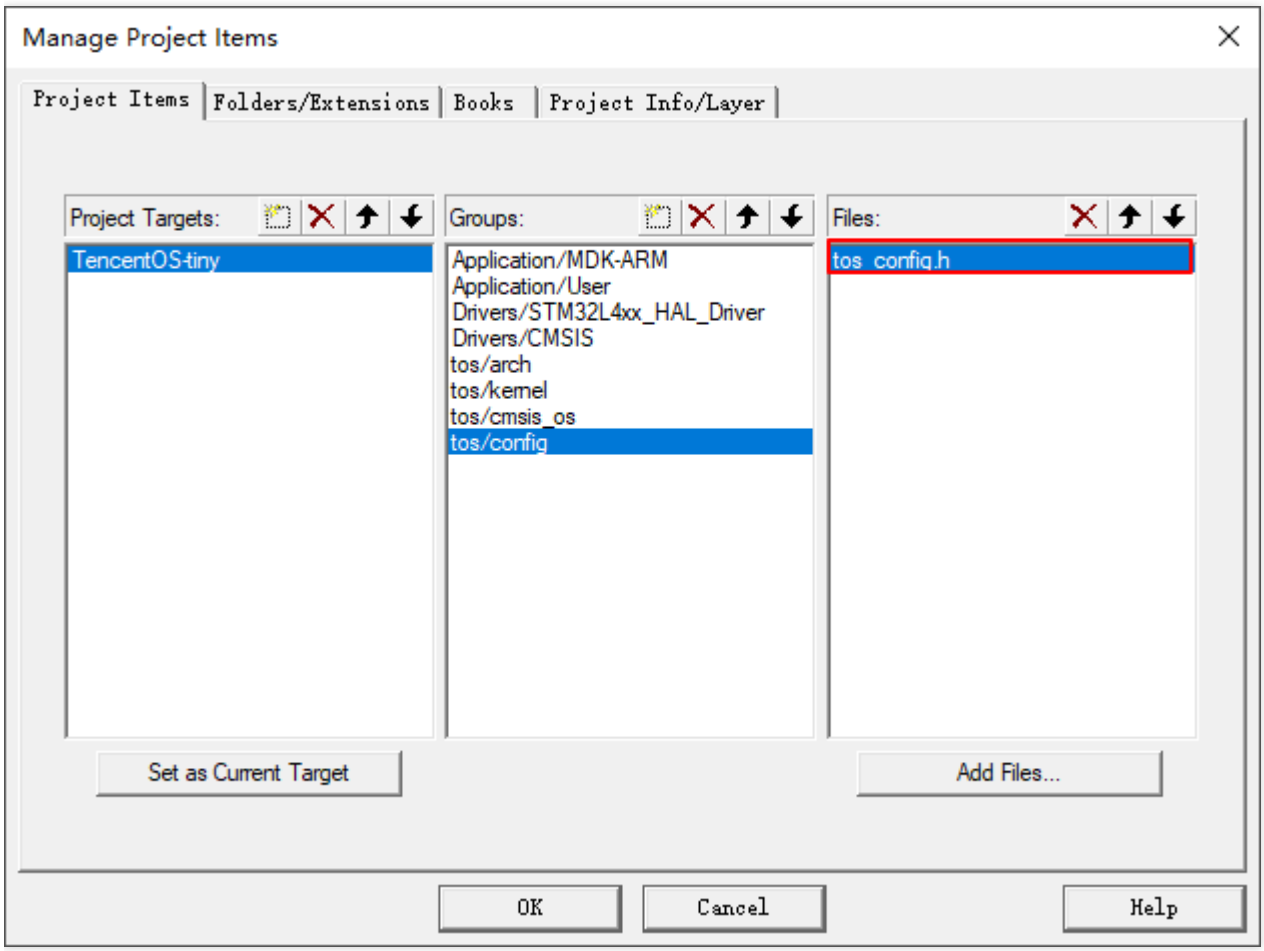
a. 在需要移植的工程目录下新增一个 TOS_CONFIG 文件夹，在该目录下新建一个 tos_config.h 文件，作为当前项目中 TencentOS tiny 的配置头文件，文件内容如下：

```
#ifndef _TOS_CONFIG_H_
#define _TOS_CONFIG_H_

#include "stm32l4xx.h" // 目标芯片头文件，用户需要根据情况更改

#define TOS_CFG_TASK_PRIO_MAX 10u // 配置TencentOS tiny默认支持的最大优先级数量

#define TOS_CFG_ROUND_ROBIN_EN 0u // 配置TencentOS tiny的内核是否开启时间片轮转

#define TOS_CFG_OBJECT_VERIFY_EN 1u // 配置TencentOS tiny是否校验指针合法

#define TOS_CFG_TASK_DYNAMIC_CREATE_EN 1u // TencentOS tiny 动态任务创建功能宏

#define TOS_CFG_EVENT_EN 1u // TencentOS tiny 事件模块功能宏
```

```
#define TOS_CFG_MMBLK_EN 1u //配置TencentOS tiny是否开启内存块管理模块

#define TOS_CFG_MMHEAP_EN 1u //配置TencentOS tiny是否开启动态内存模块

#define TOS_CFG_MMHEAP_DEFAULT_POOL_EN 1u // TencentOS tiny 默认动态内存池功能宏

#define TOS_CFG_MMHEAP_DEFAULT_POOL_SIZE 0x100 // 配置TencentOS tiny默认动态内存池大小

#define TOS_CFG_MUTEX_EN 1u // 配置TencentOS tiny是否开启互斥锁模块

#define TOS_CFG_MESSAGE_QUEUE_EN 1u // 配置TencentOS tiny是否开启消息队列模块

#define TOS_CFG_MAIL_QUEUE_EN 1u // 配置TencentOS tiny是否开启消息邮箱模块

#define TOS_CFG_PRIORITY_MESSAGE_QUEUE_EN 1u // 配置TencentOS tiny是否开启优先级消息队列模块

#define TOS_CFG_PRIORITY_MAIL_QUEUE_EN 1u // 配置TencentOS tiny是否开启优先级消息邮箱模块

#define TOS_CFG_TIMER_EN 1u // 配置TencentOS tiny是否开启软件定时器模块

#define TOS_CFG_PWR_MGR_EN 0u // 配置TencentOS tiny是否开启外设电源管理模块

#define TOS_CFG_TICKLESS_EN 0u // 配置Tickless 低功耗模块开关

#define TOS_CFG_SEM_EN 1u // 配置TencentOS tiny是否开启信号量模块

#define TOS_CFG_TASK_STACK_DRAUGHT_DEPTH_DETACT_EN 1u // 配置TencentOS tiny是否开启任务栈深度检测

#define TOS_CFG_FAULT_BACKTRACE_EN 0u // 配置TencentOS tiny是否开启异常栈回溯功能

#define TOS_CFG_IDLE_TASK_STK_SIZE 128u // 配置TencentOS tiny空闲任务栈大小

#define TOS_CFG_CPU_TICK_PER_SECOND 1000u // 配置TencentOS tiny的tick频率

#define TOS_CFG_CPU_CLOCK (SystemCoreClock) // 配置TencentOS tiny CPU频率

#define TOS_CFG_TIMER_AS_PROC 1u // 配置是否将TIMER配置成函数模式

#endif
```

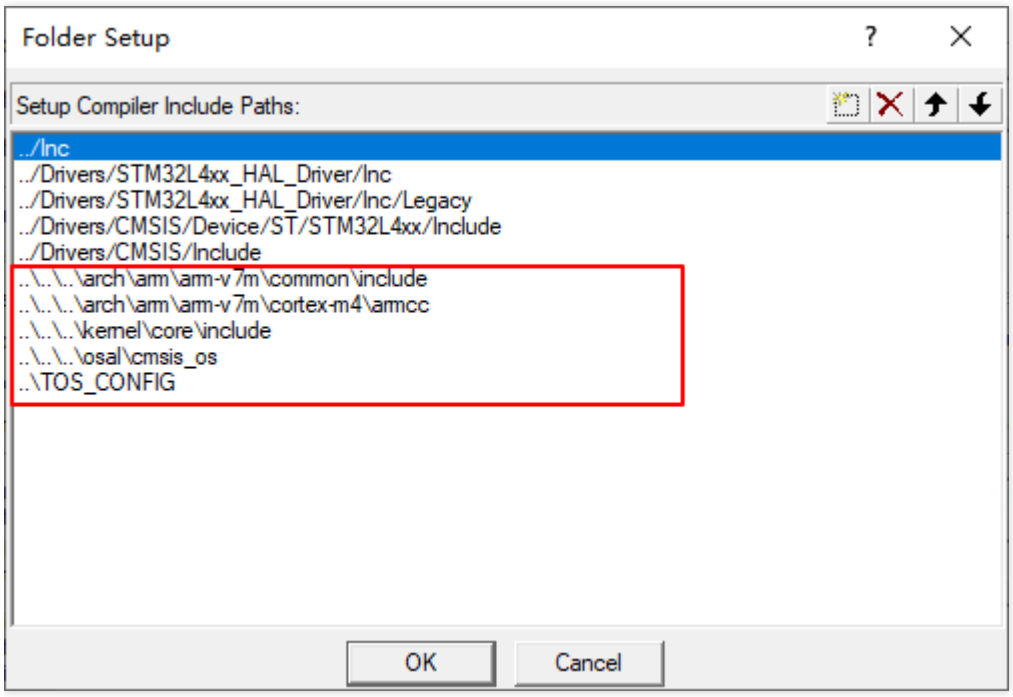
b. tos_config.h 头文件内容添加完成后，将头文件放入需要移植的 board 工程目录下即可，例如本教程放到 `board\EVB_MX_Plus_Demo\TOS_CONFIG` 目录下。再将 tos_config.h 文件加入到 Keil-MDK 工程中，方便修

改。



5. 添加 TencentOS tiny 头文件目录

TencentOS tiny 需要添加的所有头文件目录如下：



完成以上步骤，TencentOS tiny 的内核代码则全部添加完毕。

步骤三：修改冲突代码

修改 stm32l4xx_it.c 的中断函数，在 stm32l4xx_it.c 文件中添加 tos_k.h 头文件。

stm32l4xx_it.c

```
19  /* USER CODE END Header */
20
21  /* Includes ------------------------------------------------------------------*/
22  #include "main.h"
23  #include "stm32l4xx_it.h"
24  /* Private includes ----------------------------------------------------------*/
25  /* USER CODE BEGIN Includes */
26  #include "tos_k.h"
27  /* USER CODE END Includes */
28
29  /* Private typedef -----------------------------------------------------------*/
30  /* USER CODE BEGIN TD */
31
32  /* USER CODE END TD */
```

由于 PendSV_Handler 函数在 TencentOS tiny 的调度汇编中已经重新实现，所以需要在 stm32l4xx_it.c 文件中的 PendSV_Handler 函数前添加 `__weak` 关键字；同时在 SysTick_Handler 函数中添加 TencentOS tiny 的调度处

理函数，如下图：

```
167 /**
168   * @brief This function handles Pendable request for system service.
169   */
170 __weak void PendSV_Handler(void)
171 {
172   /* USER CODE BEGIN PendSV_IRQn 0 */
173
174   /* USER CODE END PendSV_IRQn 0 */
175   /* USER CODE BEGIN PendSV_IRQn 1 */
176
177   /* USER CODE END PendSV_IRQn 1 */
178 }
179
180 /**
181   * @brief This function handles System tick timer.
182   */
183 void SysTick_Handler(void)
184 {
185   /* USER CODE BEGIN SysTick_IRQn 0 */
186
187   /* USER CODE END SysTick_IRQn 0 */
188   HAL_IncTick();
189   /* USER CODE BEGIN SysTick_IRQn 1 */
190   if (tos_knl_is_running()) {
191     tos_knl_irq_enter();
192     tos_tick_handler();
193     tos_knl_irq_leave();
194   }
195
196   /* USER CODE END SysTick_IRQn 1 */
197 }
```

步骤四：编写 TencentOS tiny 测试任务

1. 在 mian.c 中添加 TencentOS tiny 头文件，编写任务函数。

```
#include "cmsis_os.h" //包含 TencentOS tiny 相关头函数
//task1，定义任务栈大小和优先级参数等
#define TASK1_STK_SIZE 256
void task1(void *pdata);
osThreadDef(task1, osPriorityNormal, 1, TASK1_STK_SIZE);
//task2，定义任务栈大小和优先级参数等
#define TASK2_STK_SIZE 256
void task2(void *pdata);
osThreadDef(task2, osPriorityNormal, 1, TASK2_STK_SIZE);
void task1(void *pdata)//任务入口函数，用于测试
{
```

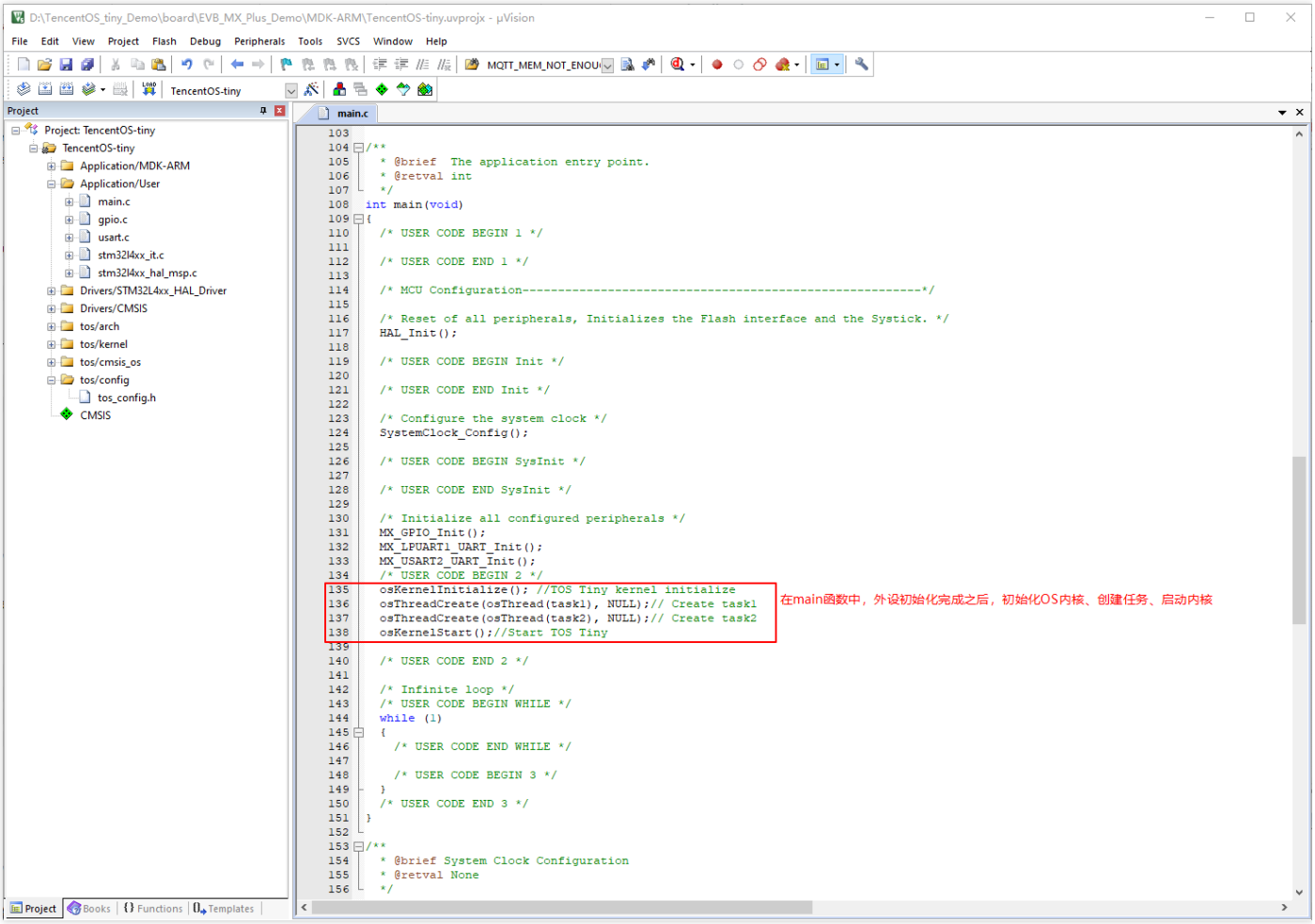
```
int count = 1;
while(1)
{
printf("\r\nHello world!\r\n###This is task1 ,count is %d \r\n", count++);
HAL_GPIO_TogglePin(LED_GPIO_Port,LED_Pin);
osDelay(2000);
}
}
void task2(void *pdata) //任务入口函数，用于测试
{
int count = 1;
while(1)
{
printf("\r\nHello TencentOS !\r\n***This is task2 ,count is %d \r\n", count++);
osDelay(1000);
}
}

int fputc(int ch, FILE *f)//将 printf 函数重定向到串口2
{
if (ch == '\n')
{
HAL_UART_Transmit(&huart2, (void *)"\r", 1,30000);
}
HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
return ch;
}
```

2. 继续在 main.c 的 mian 函数中添加硬件外设初始化代码，添加 TencentOS tiny 的初始化代码：

```
osKernelInitialize(); //TOS Tiny kernel initialize
osThreadCreate(osThread(task1), NULL);// Create task1
osThreadCreate(osThread(task2), NULL);// Create task2
osKernelStart();//Start TOS Tiny
```

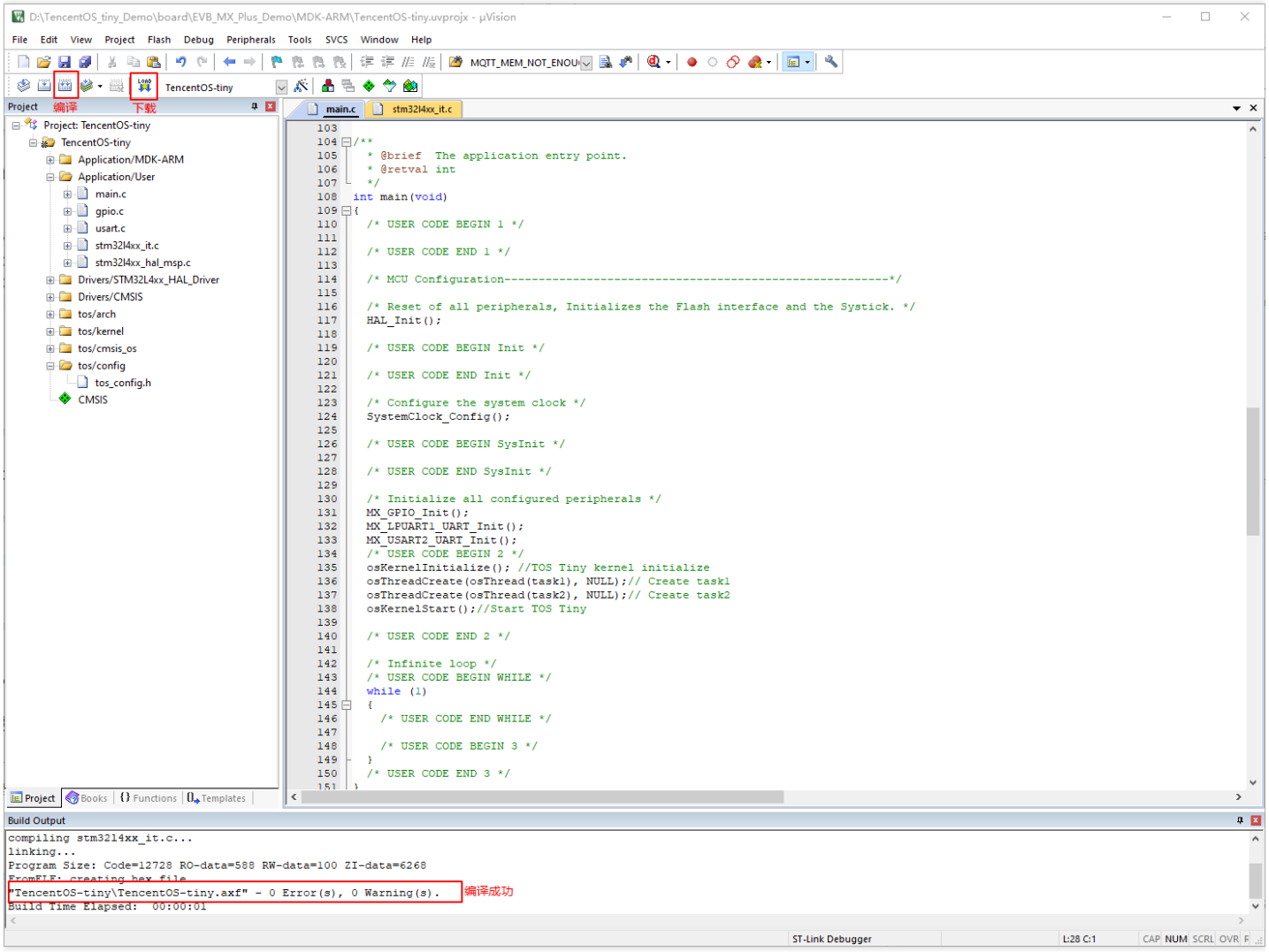
版权所有：腾讯云计算（北京）有限责任公司

添加代码位置如下图：

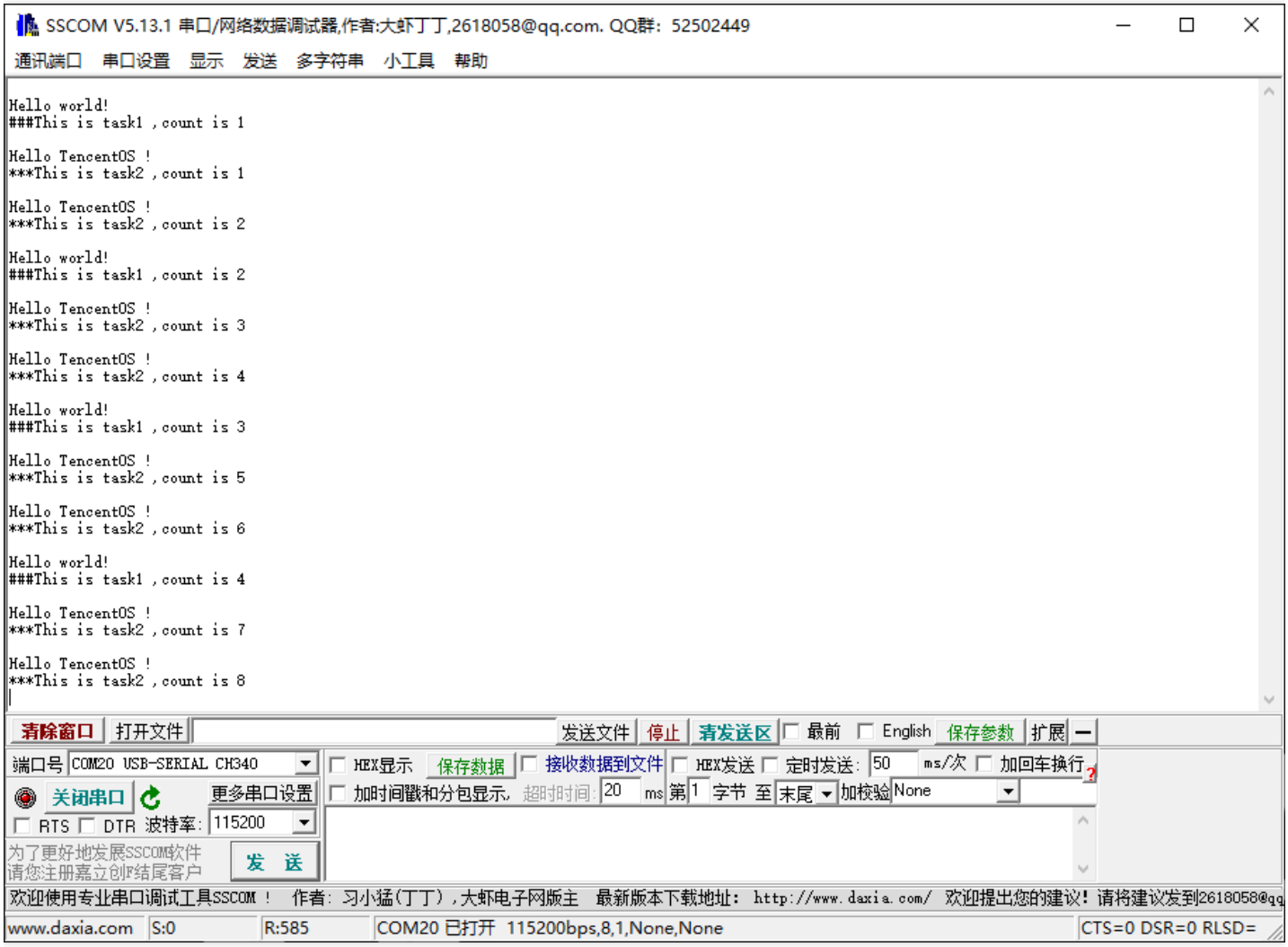


步骤五：编译下载测试 TencentOS tiny 移植结果

按照下图指示，进行编译下载到开发板即可完成 TencentOS tiny 的测试。



如下图所示，可以看到串口交替打印信息，并且板载 LED 闪烁，表示两个任务正在进行调度，切换运行。



步骤六：下一步操作

请前往 移植AT框架、SAL框架、模组驱动 进行框架、模组驱动移植操作。

移植AT框架、SAL框架、模组驱动

最近更新时间：2020-09-10 14:16:27

本文将为您介绍 TencentOS tiny 接入腾讯物联网开发平台的框架、模组驱动移植相关操作。

步骤一：AT 框架、SAL 框架整体架构

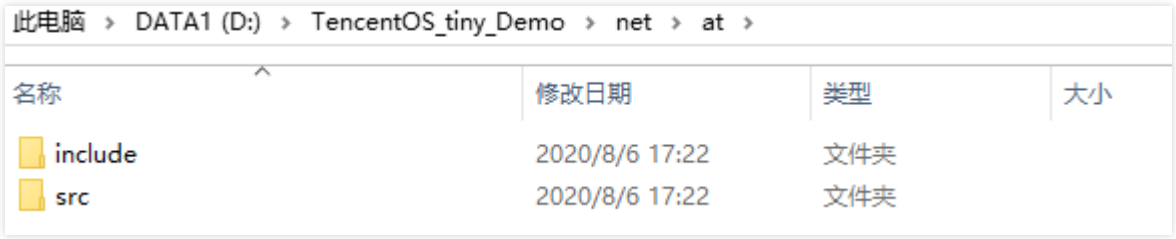
AT 框架是我们编写的一个通用 AT 指令解析任务，使开发者只需调用 AT 框架提供的 API 即可处理与模组的交互数据，SAL 框架全称 Socket Abstract Layer，提供了类似 socket 网络编程的抽象层。基于 AT 框架实现 SAL 的底层函数叫做通信模组的驱动程序。

整体架构如下：

步骤二：移植 AT 框架

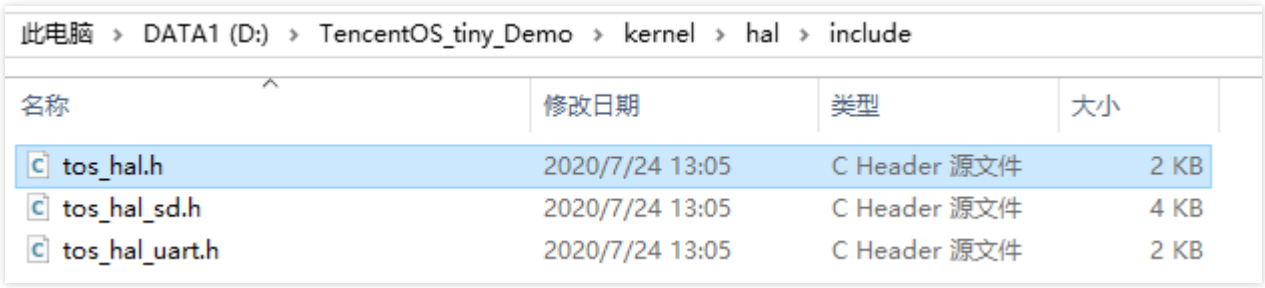
从 TencentOS-tiny 中复制以下五个文件到工程目录中，保持文件架构不变并删除多余文件。

1. 复制 net\at 目录下的 tos_at.h 和 tos_at.c 文件，两个文件实现了 TencentOS tiny AT 的框架。



2. 复制 platform\hal\st\stm32l4xx\src 目录下的 tos_hal_uart.c 文件，该文件为 TencentOS-tiny AT 框架底层使用的串口驱动 HAL 层。

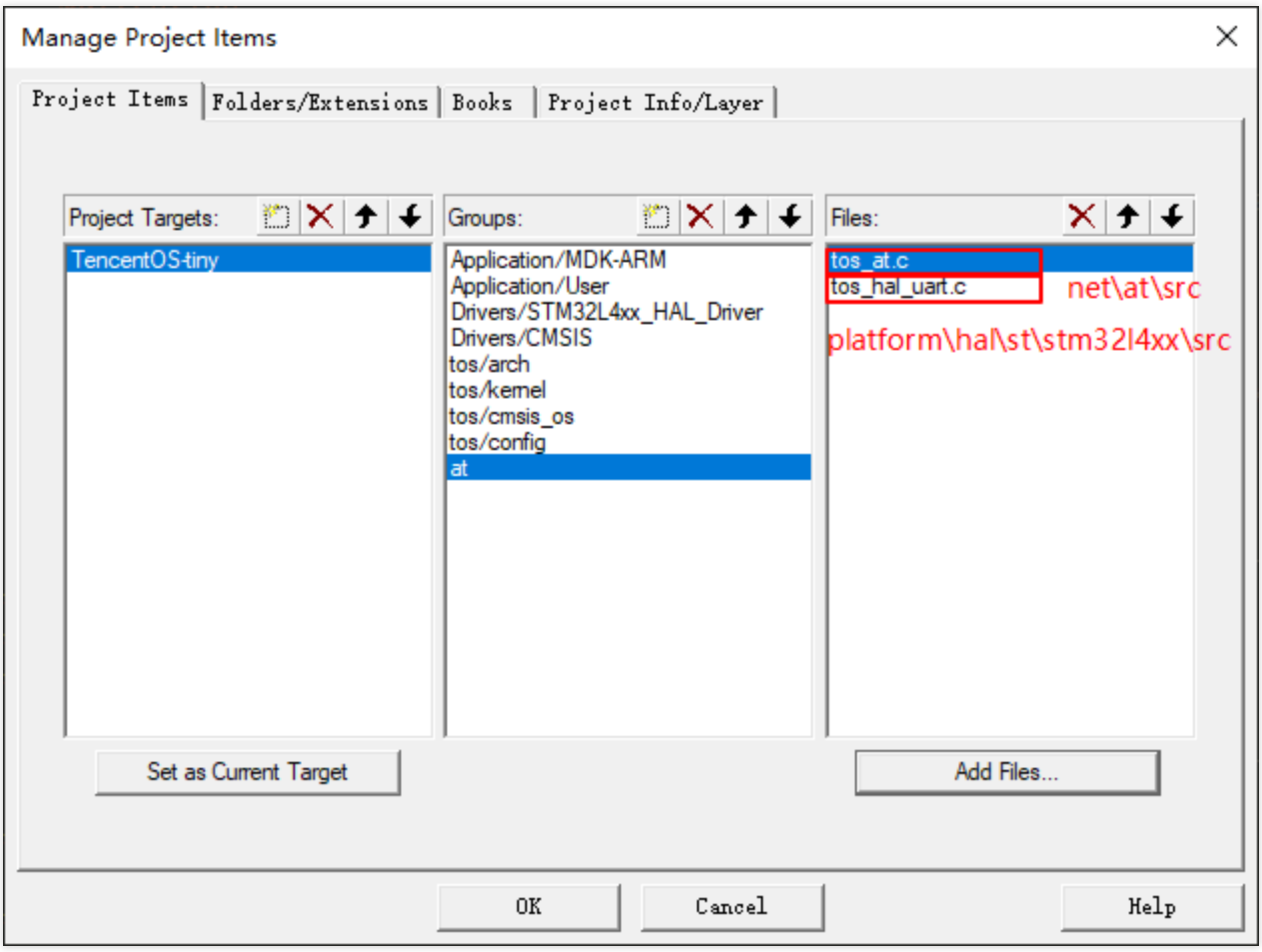
3. 复制 kernel\hal\include 目录下的 tos_hal.h 和 tos_hal_uart.h 文件，两个文件为 TencentOS-tiny AT 框架的部分头文件。



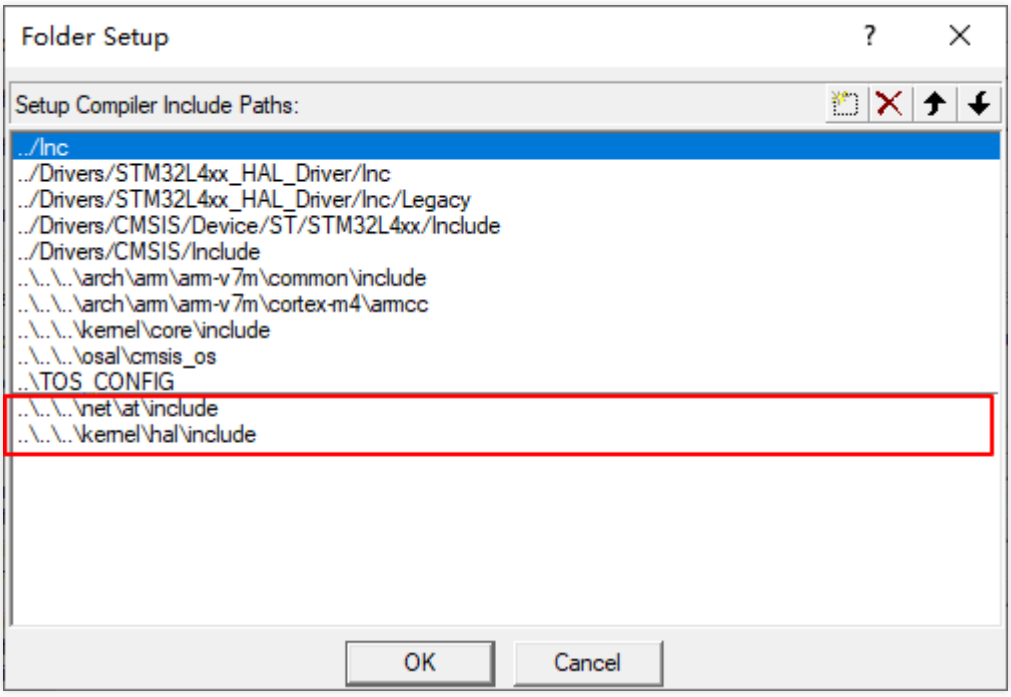
文件复制完成，接下来开始添加到工程中。

版权所有：腾讯云计算（北京）有限责任公司

1. 将以上两个 .c 文件添加到 Keil 工程中。



2. 将 net\at 和 kernel\hal\include 目录下两个头文件路径添加到 Keil MDK 中。



3. 在串口中断中配置调用 AT 框架的字节接收函数，编辑 `stm32l4xx_it.c` 文件。

i. 添加 AT 框架的头文件。

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "tos_at.h"
/* USER CODE END Includes */
```

ii. 在文件最后添加串口中断回调函数。

```
/* USER CODE BEGIN 1 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
extern uint8_t data;
if (huart->Instance == LPUART1) {
HAL_UART_Receive_IT(&hlpuart1, &data, 1);
tos_at_uart_input_byte(data);
}
}
/* USER CODE END 1 */
```

> 注意：
> 在回调函数中声明 data 变量在外部定义，这是因为 STM32 HAL 库的机制，需要在初始化完成之后先调用一次串口接收函数，使能串口接收中断，编辑 `usart.c` 文件。

a. 在文件开头定义 data 变量为全局变量：

```
/* USER CODE BEGIN 0 */
uint8_t data;
/* USER CODE END 0 */
```

b. 在串口初始化完成之后使能接收中断：

```
/* LPUART1 init function */
void MX_LPUART1_UART_Init(void)
{
hlpuart1.Instance = LPUART1;
hlpuart1.Init.BaudRate = 115200;
hlpuart1.Init.WordLength = UART_WORDLENGTH_8B;
hlpuart1.Init.StopBits = UART_STOPBITS_1;
hlpuart1.Init.Parity = UART_PARITY_NONE;
hlpuart1.Init.Mode = UART_MODE_TX_RX;
hlpuart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
```

版权所有：腾讯云计算（北京）有限责任公司

```
hlpuart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
hlpuart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_UART_Init(&hlpuart1) != HAL_OK)
{
Error_Handler();
}
//手动添加，使能串口中断
HAL_UART_Receive_IT(&hlpuart1, &data, 1);
}
```

因为此工程中只配置 LPUART1 和 USART2 两个串口，所以需要将 HAL 驱动中其余的串口屏蔽，否则将会报错。

tos_hal_uart.c

```
#include "tos_k.h"
#include "tos_hal.h"
#include "stm32l4xx_hal.h"
#include "usart.h"

__API__ int tos_hal_uart_init(hal_uart_t *uart, hal_uart_port_t port)
{
    if (!uart) {
        return -1;
    }
    if (port == HAL_UART_PORT_0) {
        uart->private_uart = &hlpuart1;
        MX_LPUART1_UART_Init();
//    } else if (port == HAL_UART_PORT_1) {
//        uart->private_uart = &huart1;
//        MX_USART1_UART_Init();
    } else if (port == HAL_UART_PORT_2) {
        uart->private_uart = &huart2;
        MX_USART2_UART_Init();
//    } else if (port == HAL_UART_PORT_3) {
//        uart->private_uart = &huart3;
//        MX_USART3_UART_Init();
    }

    return 0;
}
```

屏蔽串口1初始化代码

屏蔽串口3初始化代码

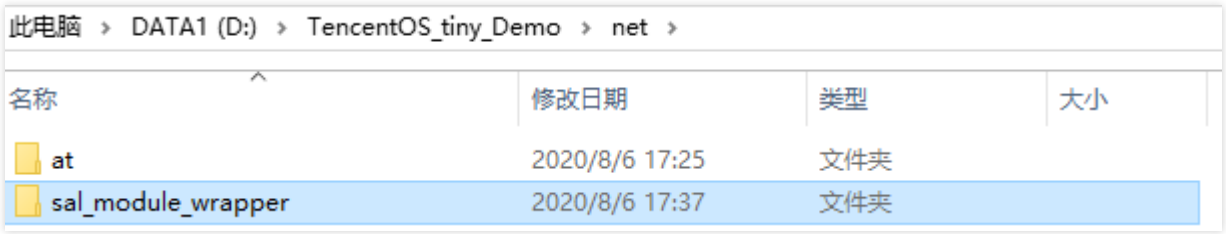
由于 AT 框架使用的缓冲区都是动态内存，所以需要将系统中默认动态内存池的大小至少修改为0x8000。

tos_config.h

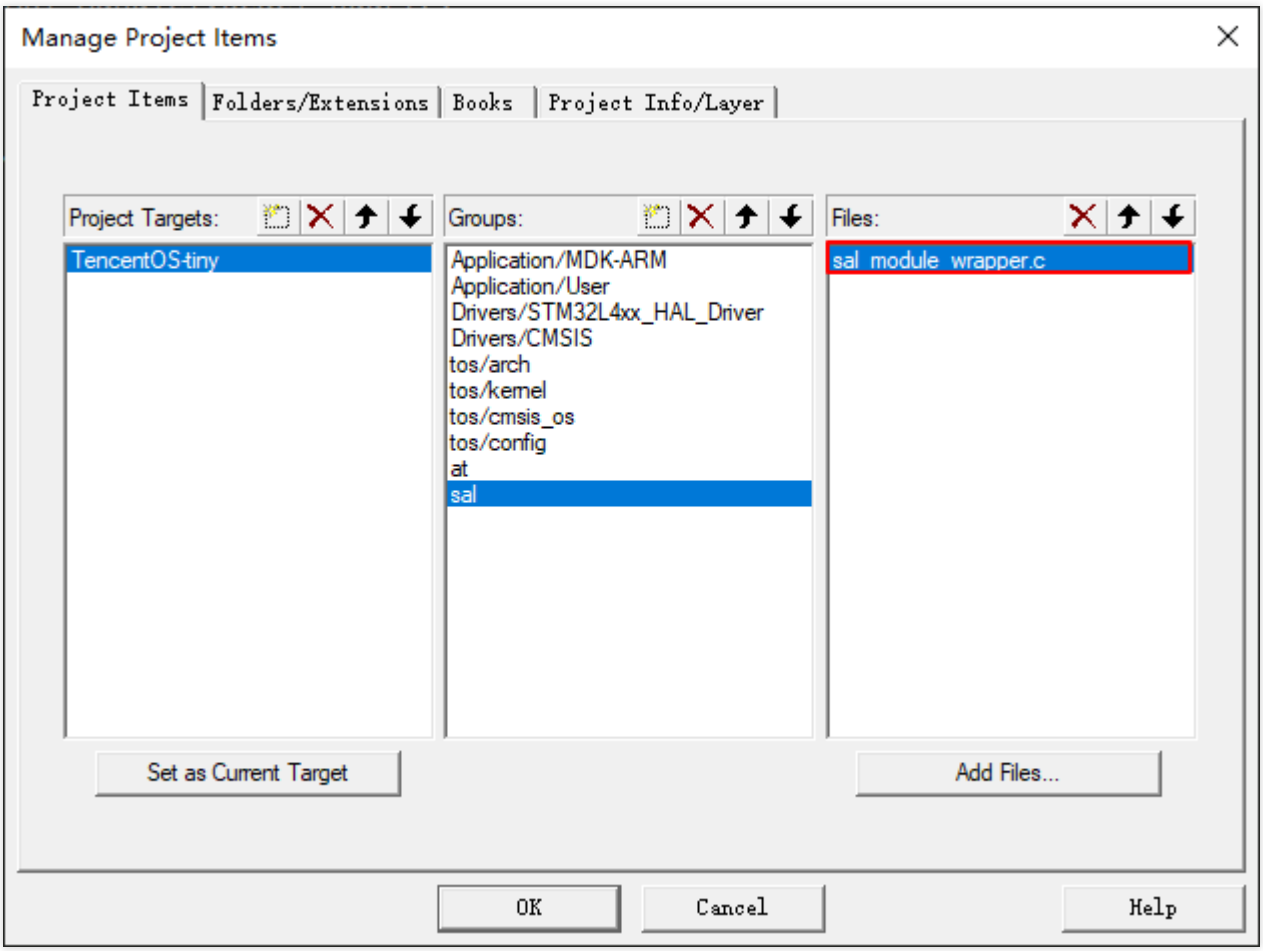
```
#ifndef _TOS_CONFIG_H_
#define  _TOS_CONFIG_H_

#include "stm32l4xx.h"  // 目标芯片头文件，用户需要根据情况更改

#define TOS_CFG_TASK_PRIO_MAX           10u             // 配置TencentOS tiny默认支持的最大优先级数量

#define TOS_CFG_ROUND_ROBIN_EN          0u              // 配置TencentOS tiny的内核是否开启时间片轮转

#define TOS_CFG_OBJECT_VERIFY_EN        1u              // 配置TencentOS tiny是否校验指针合法

#define TOS_CFG_TASK_DYNAMIC_CREATE_EN  1u              // TencentOS tiny 动态任务创建功能宏

#define TOS_CFG_EVENT_EN                1u              // TencentOS tiny 事件模块功能宏

#define TOS_CFG_MMBLK_EN                1u              //配置TencentOS tiny是否开启内存块管理模块

#define TOS_CFG_MMHEAP_EN               1u              //配置TencentOS tiny是否开启动态内存模块

#define TOS_CFG_MMHEAP_DEFAULT_POOL_EN  1u              // TencentOS tiny 默认动态内存池功能宏

#define TOS_CFG_MMHEAP_DEFAULT_POOL_SIZE        0x2000  // 配置TencentOS tiny默认动态内存池大小

#define TOS_CFG_MUTEX_EN                1u              // 配置TencentOS tiny是否开启互斥锁模块

#define TOS_CFG_MESSAGE_QUEUE_EN        1u              // 配置TencentOS tiny是否开启消息队列模块
```

步骤三：移植 SAL 框架

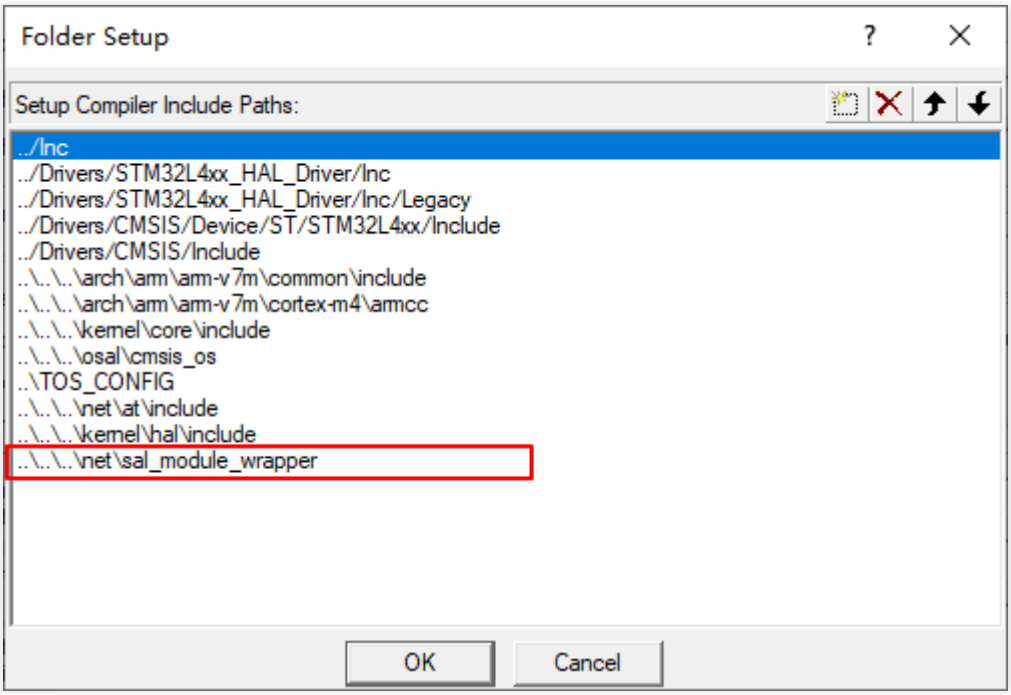
1. TencentOS-tiny SAL 框架的实现在 net\sal_module_wrapper 目录下的 sal_module_wrapper.h 和 sal_module_wrapper.c 两个文件中，将这个文件夹从 TencentOS-tiny 官方仓库复制到工程目录下，并且保持原有架构不变。



2. 将 sal_module_wrapper.c 文件添加到 Keil MDK 工程中。



3. 将头文件 sal_module_wrapper.h 所在目录添加到 Keil MDK 中。



版权所有：腾讯云计算（北京）有限责任公司

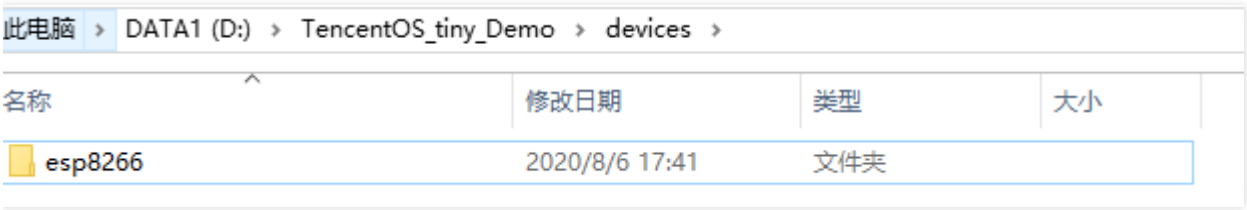
步骤四：移植通信模组驱动

TencentOS-tiny 官方已提供大量的通信模组驱动实现 SAL 框架，覆盖常用的通信方式，例如2G、4G Cat.4、4G Cat.1、NB-IoT 等，在 `devices` 文件夹下,：

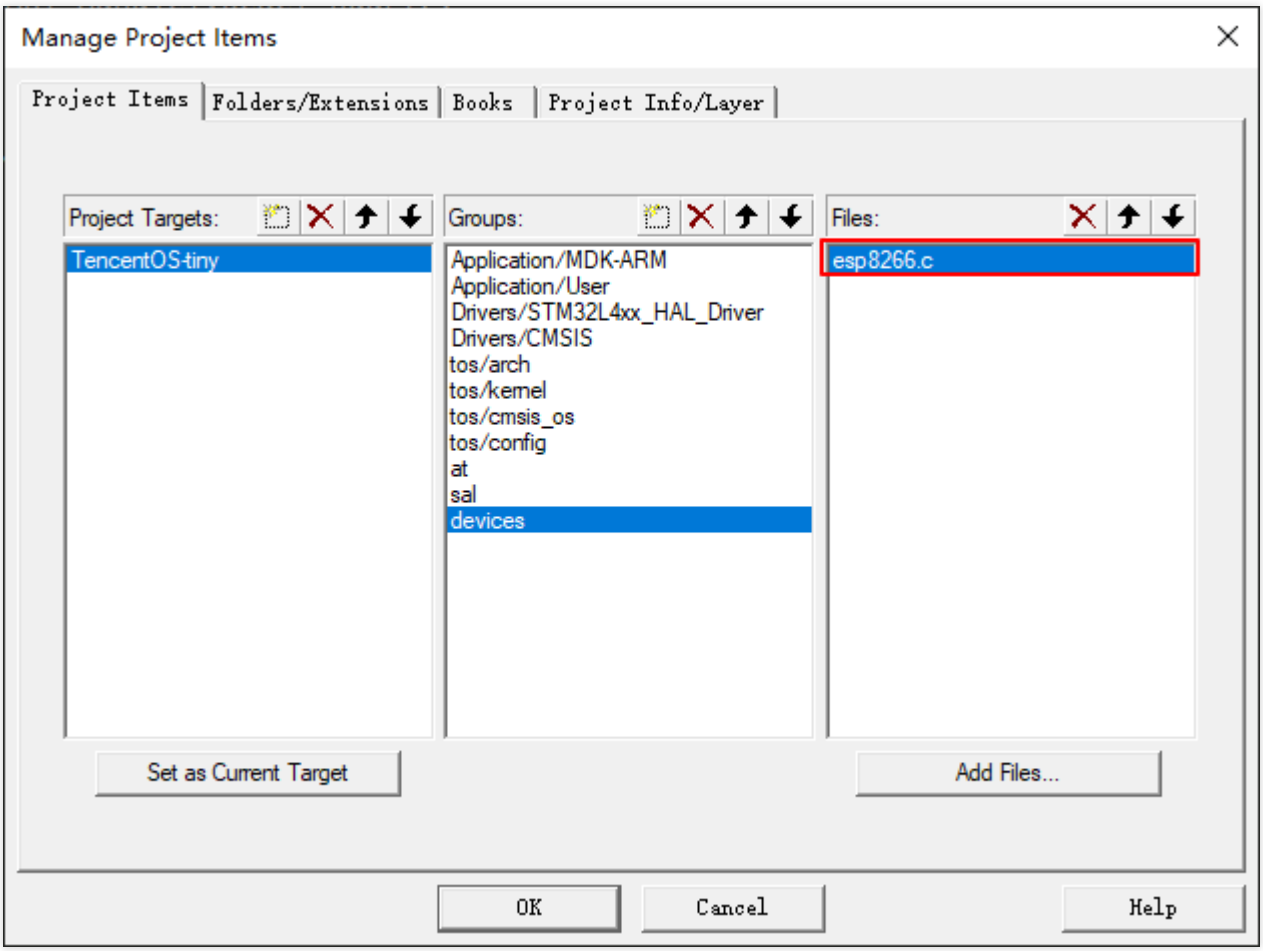
- air724
- bc26
- bc25_28_95
- bc35_28_95_lwm2m
- ec20
- esp8266
- m26
- m5310a
- m6312
- sim800a
- sim7600ce
- 欢迎贡献更多驱动...

因为这些驱动都是 SAL 框架的实现，所以这些通信模组的驱动可以根据实际硬件情况**选择一种加入到工程中**，这里以 Wi-Fi 模组 ESP8266 为例，演示如何加入通信模组驱动到工程中。

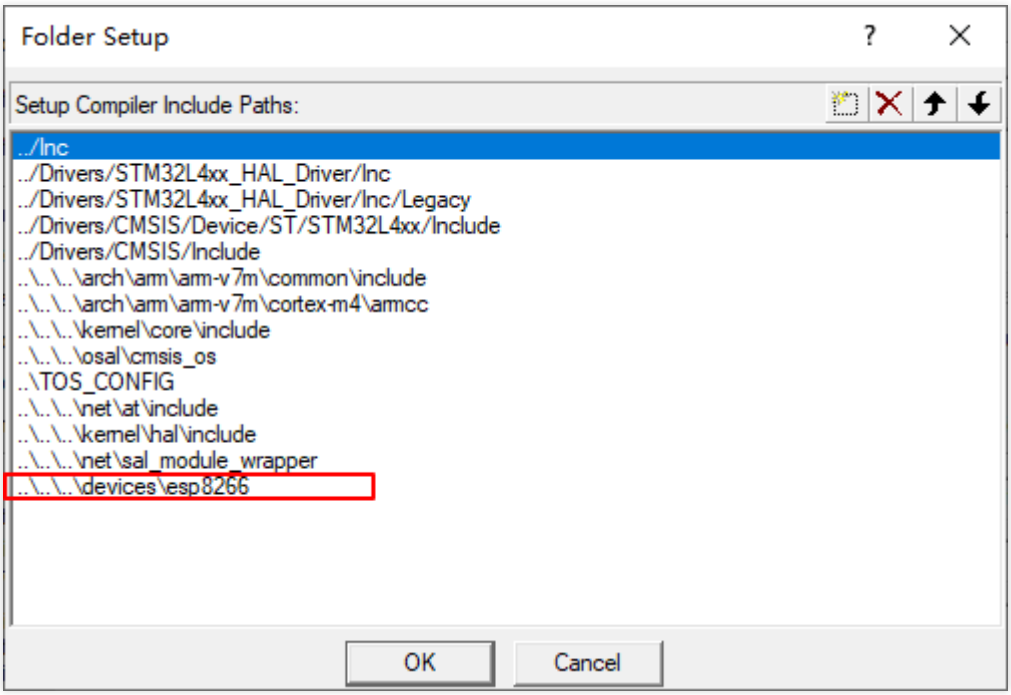
ESP8266 的驱动在 `devices\esp8266` 目录中，将此文件夹从 TencentOS-tiny 官方仓库复制到工程中，保持目录架构不变。



1. 将 esp8266.c 文件加入到 Keil MDK 工程中。



2. 将 esp8266.h 头文件所在路径添加到 Keil MDK 工程中，这样就移植完成。



步骤五：测试网络通信

移植完成之后，修改 main.c，编写代码来测试网络通信是否正常。

添加头文件：

```
#include "sal_module_wrapper.h"
#include "esp8266.h"
```

接着修改之前编写的 task1 的任务函数：

```
#define RECV_LEN 1024
uint8_t recv_data[RECV_LEN];

void task1(void *pdata)
{
int count = 1;
int socket_id = -1;
int recv_len = -1;

/* 通信模组初始化 */
esp8266_sal_init(HAL_UART_PORT_0);

/* 入网 */
esp8266_join_ap("Mculover666","mculover666");

/* 发起socket连接 */
socket_id = tos_sal_module_connect("117.50.111.72", "8001", TOS_SAL_PROTO_TCP);

/* 发送一次数据 */
tos_sal_module_send(socket_id, (uint8_t*)"hello,server!", 12);

/* 接收一次数据 */
recv_len = tos_sal_module_recv_timeout(socket_id, recv_data, sizeof(recv_data), 8000);

if (recv_len < 0) {
printf("receive error\n");
} else if (recv_len == 0) {
printf("receive none\n");
} else {
recv_data[recv_len] = 0;
printf("receive len: %d\nmsg from remote: %s\n", recv_len, recv_data);
}
```

版权所有：腾讯云计算（北京）有限责任公司

```
/* 关闭socket */
tos_sal_module_close(socket_id);

while(1)
{
printf("\r\nHello world!\r\n###This is task1 ,count is %d \r\n", count++);
HAL_GPIO_TogglePin(LED_GPIO_Port,LED_Pin);
osDelay(2000);
}
}
```

注意：
初始化模组时指定的串口号即为 AT 通信模组所使用的串口，在 tos_hal_uart.h 中定义。

```
typedef enum hal_uart_port_en {
HAL_UART_PORT_0 = 0, //对应LPUART1
HAL_UART_PORT_1, //对应USART1
HAL_UART_PORT_2, //依此类推
HAL_UART_PORT_3,
HAL_UART_PORT_4,
HAL_UART_PORT_5,
HAL_UART_PORT_6,
} hal_uart_port_t;
```

测试网络通信时需要的任务栈较大，所以增大 task1 的任务栈大小为512字节：

main.c

```
60  /* USER CODE BEGIN 0 */
61  //task1
62  #define TASK1_STK_SIZE        512
63  void task1(void *pdata);
64  osThreadDef(task1, osPriorityNormal, 1, TASK1_STK_SIZE);
65
66  //task2
67  #define TASK2_STK_SIZE        256
68  void task2(void *pdata);
69  osThreadDef(task2, osPriorityNormal, 1, TASK2_STK_SIZE);
70
71  #define RECV_LEN            1024
72  uint8_t recv_data[RECV_LEN];
```

版权所有：腾讯云计算（北京）有限责任公司

同时，为了避免task2运行打印的信息对网络测试信息产生干扰，将创建 task2 的代码屏蔽：

main.c*

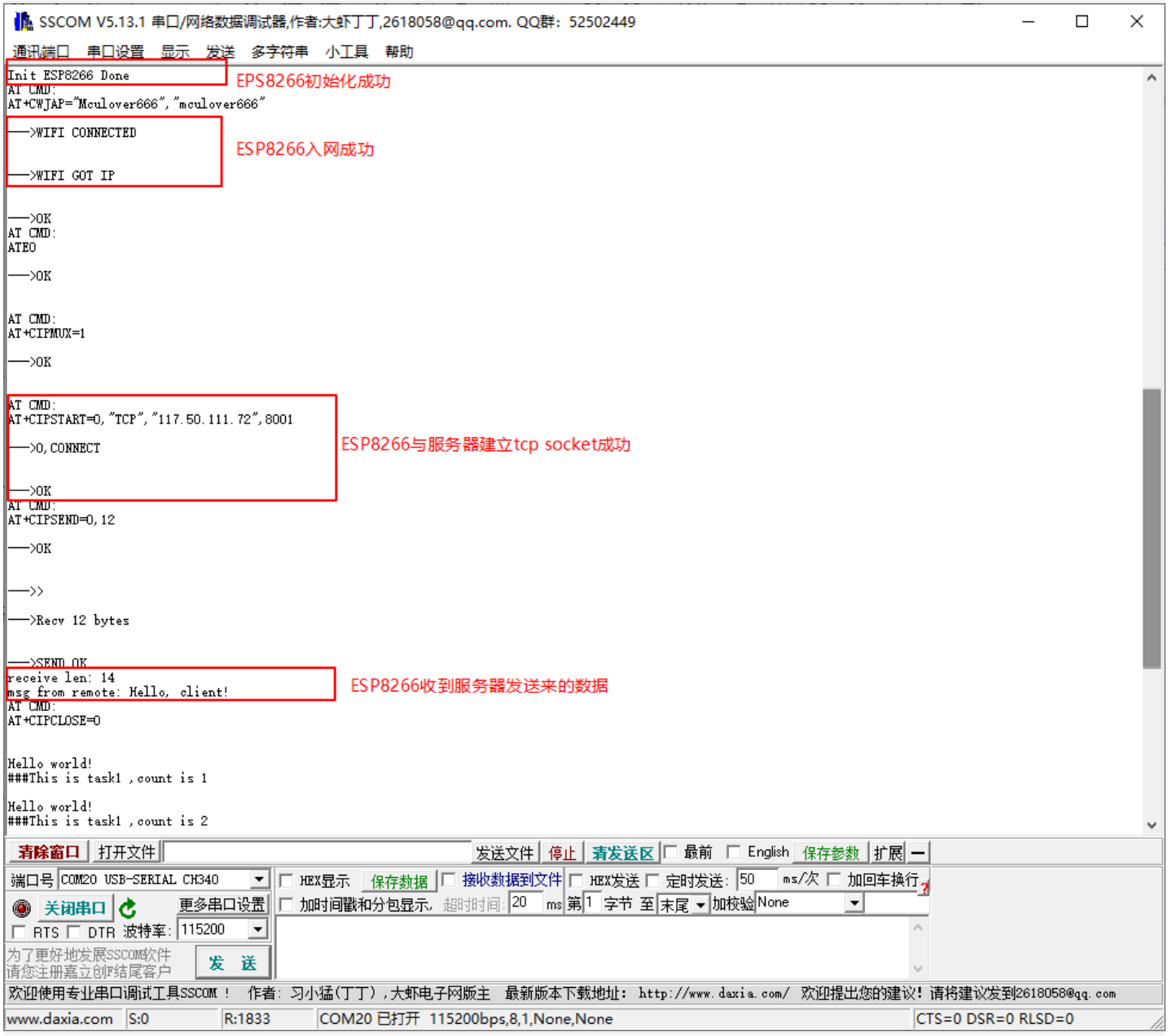
```
135
136 /**
137   * @brief  The application entry point.
138   * @retval int
139   */
140 int main(void)
141 {
142   /* USER CODE BEGIN 1 */
143
144   /* USER CODE END 1 */
145
146   /* MCU Configuration--------------------------------------------------------*/
147
148   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
149   HAL_Init();
150
151   /* USER CODE BEGIN Init */
152
153   /* USER CODE END Init */
154
155   /* Configure the system clock */
156   SystemClock_Config();
157
158   /* USER CODE BEGIN SysInit */
159
160   /* USER CODE END SysInit */
161
162   /* Initialize all configured peripherals */
163   MX_GPIO_Init();
164   MX_LPUART1_UART_Init();
165   MX_USART2_UART_Init();
166   /* USER CODE BEGIN 2 */
167   osKernelInitialize(); //TOS Tiny kernel initialize
168   osThreadCreate(osThread(task1), NULL);// Create task1
169   //osThreadCreate(osThread(task2), NULL);// Create task2
170   osKernelStart();//Start TOS Tiny
171   /* USER CODE END 2 */
```

取消创建任务2的代码

注意：

TCP 测试服务器需要自己搭建或者使用一些小工具，此处不再详述。

编译程序，下载到开发板，在串口查看接收到服务端发送来的消息：



至此，测试完成。

步骤六：下一步操作

请前往 移植腾讯云 C-SDK 进行内核移植操作。

移植腾讯云 C-SDK

最近更新时间：2020-09-11 09:07:17

腾讯云物联网开发平台 IoT Explorer 设备端 C-SDK ，配合平台对设备数据模板化的定义，实现和云端基于数据模板协议的数据交互框架，开发者基于 IoT_Explorer C-SDK 数据模板框架，通过脚本自动生成模板代码，快速实现设备和平台、设备和应用之间的数据交互。

步骤一：云端创建设备

登录腾讯云 物联网开发平台控制台，创建云端设备，详情可参考 基于 TencentOS tiny 接入指引 。

另外，数据模板是将物理实体设备进行数字化描述，构建其数字模型。在物联网开发平台定义数据模板即定义产品功能，完成功能定义后，系统将自动生成该产品的数据模板。
单击【控制台】>【项目】>【产品开发】>【智能灯产品】>【数据模板】>【新建功能】，填写功能信息。

功能类型包含三元种元素：

功能元素	功能描述	功能标识符
属性	包括布尔型、整数型、字符型、浮点型、枚举型和时间型等6种基本数据类型； 用于描述设备的实时状态，支持读取和设置，如模式、亮度、开关等。	PropertiesId
事件	包括告警、故障和信息三种类型，事件型功能属性可以添加具体的事件参数，这些参数可以由属性中6种基本数据类型组成； 用于描述设备运行时的事件，包括告警、信息和故障等三种事件类型，可添加多个输出参数，如环境传感器检测到空气质量很差，空调异常告警等。	EventId

版权所有：腾讯云计算（北京）有限责任公司

功能元素	功能描述	功能标识符
行为	用于描述复杂的业务逻辑，可添加多个调用参数和返回参数，可用于让设备执行某项定特定的任务。例如，开锁动作需要知道是哪个用户在什么时间开锁，锁的状态如何等。并且行为的输入参数和输出参数可添加上述6种属性的基本数据类型。	ActionId

数据类型支持以下6种：

- 布尔型：非真即假的二值型变量。例如，开关功能。
- 整数型：可用于线性调节的整数变量。例如，空调的温度。
- 字符型：以字符串形式表达的功能点，例如，灯的位置。
- 浮点型：精度为浮点型的功能点。例如，压力值的范围：0.0 - 24.0。
- 枚举型：自定义的有限集合值。例如，灯的颜色：白色、红色、黄色等。
- 时间型：string 类型的 UTC 时间戳（毫秒）。

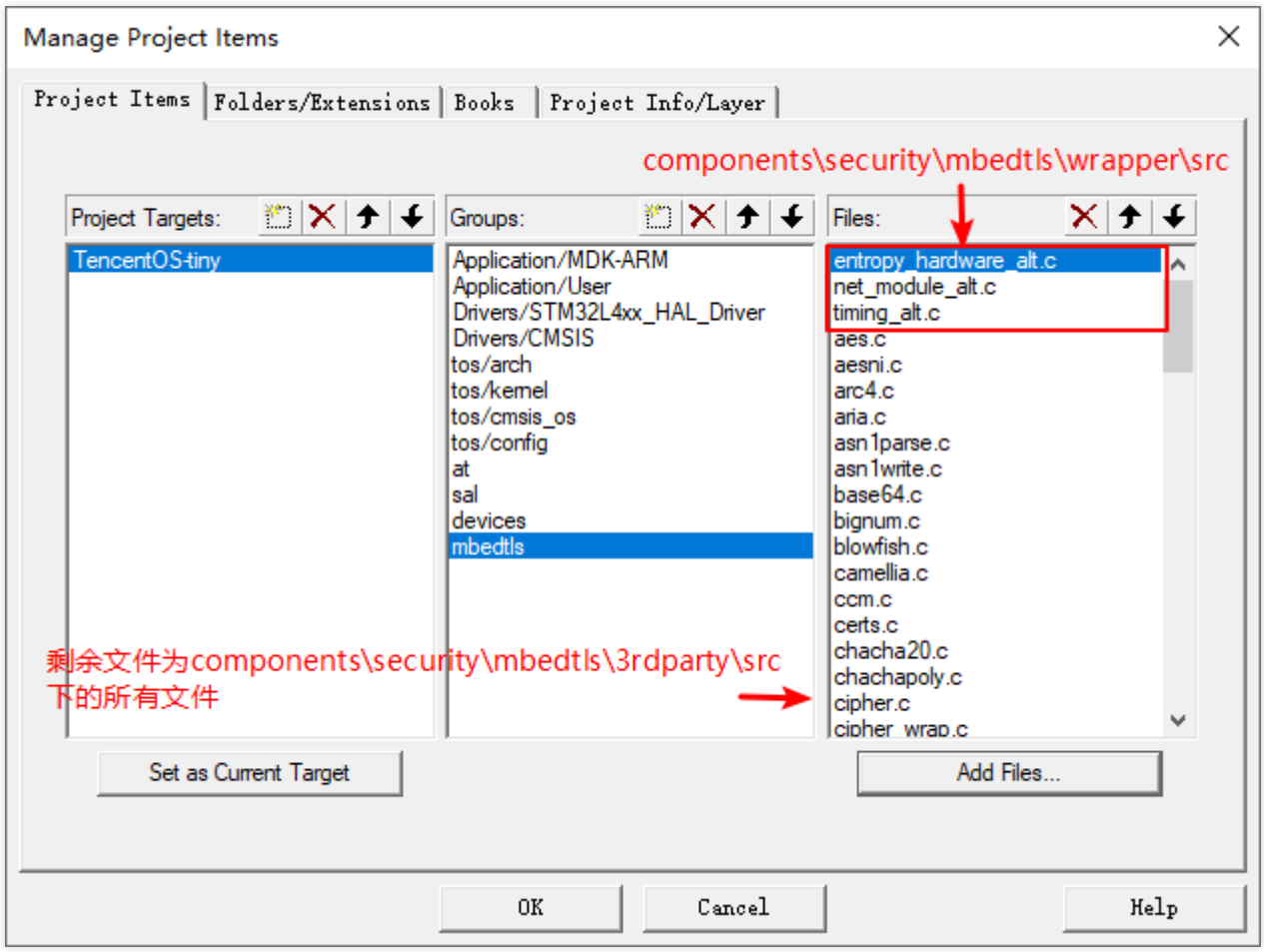
数据模板是一个 JSON 格式的文件，使用数据模板协议，用户的设备需按数据模板定义要求传输设备数据到云端，并可使用基于数据模板的诸多业务功能，单击【查看 JSON】，可查看到已创建功能的 JSON 格式协议。

步骤二：移植 mbedtls

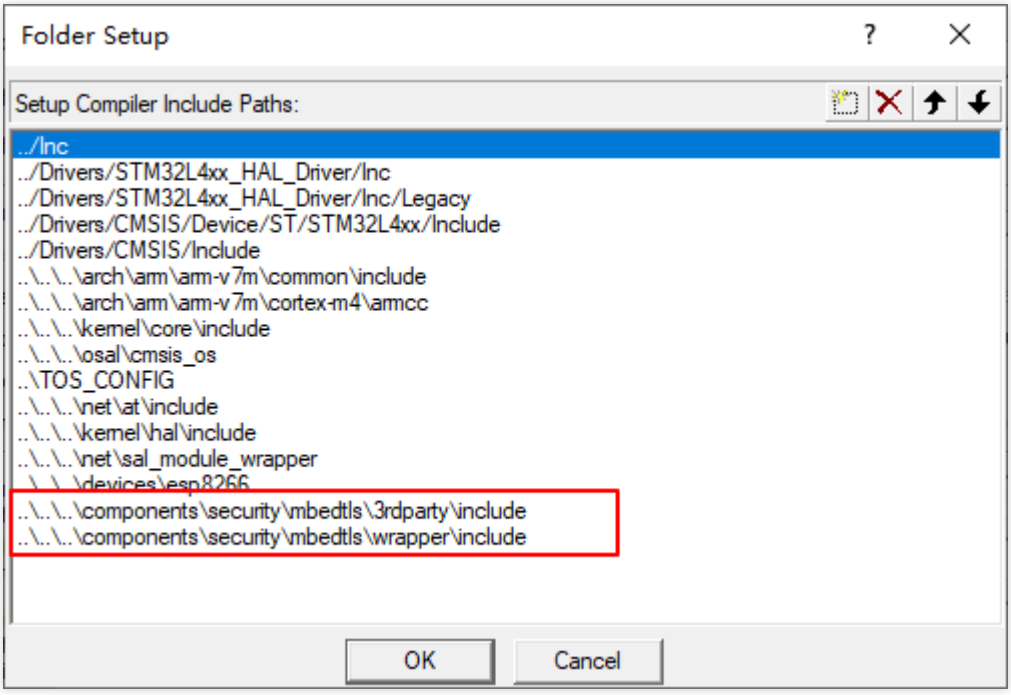
腾讯云 C-SDK 对接云端时如果配置建立安全链接，将会用到 mbedtls 加密库，所以需要先移植 mbedtls。

1. TencentOS-tiny 中已经移植适配好 mbedtls 库，并作为 TencentOS-tiny 的一个组件，在 `components\security\mbedtls` 目录下，将此目录复制到工程目录中，复制过程中保持目录架构不变，并将其余的文件删除。

2. 将 mbedtls 相关的 `.c` 文件添加到 Keil-MDK 工程中。



3. 将 mbedtls 相关的头文件路径都添加到 Keil-MDK 工程中，移植完成。



> 注意：
> 此时还没有指定 mbedtls 配置文件，编译会报错，继续按后续的步骤操作即可。

步骤三：移植腾讯云 C-SDK

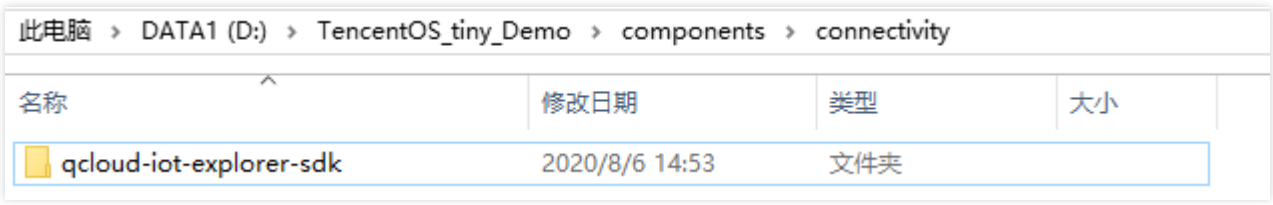
TencentOS-tiny 官方已经将 IoT_Explorer C-SDK 移植适配完成，在 `components\connectivity\qcloud-iot-explorer-sdk` 目录下，其中：

- 3rdparty：IoT_Explorer C-SDK 源码。
- port\TencentOS_tiny：移植适配文件 qcloud/port。

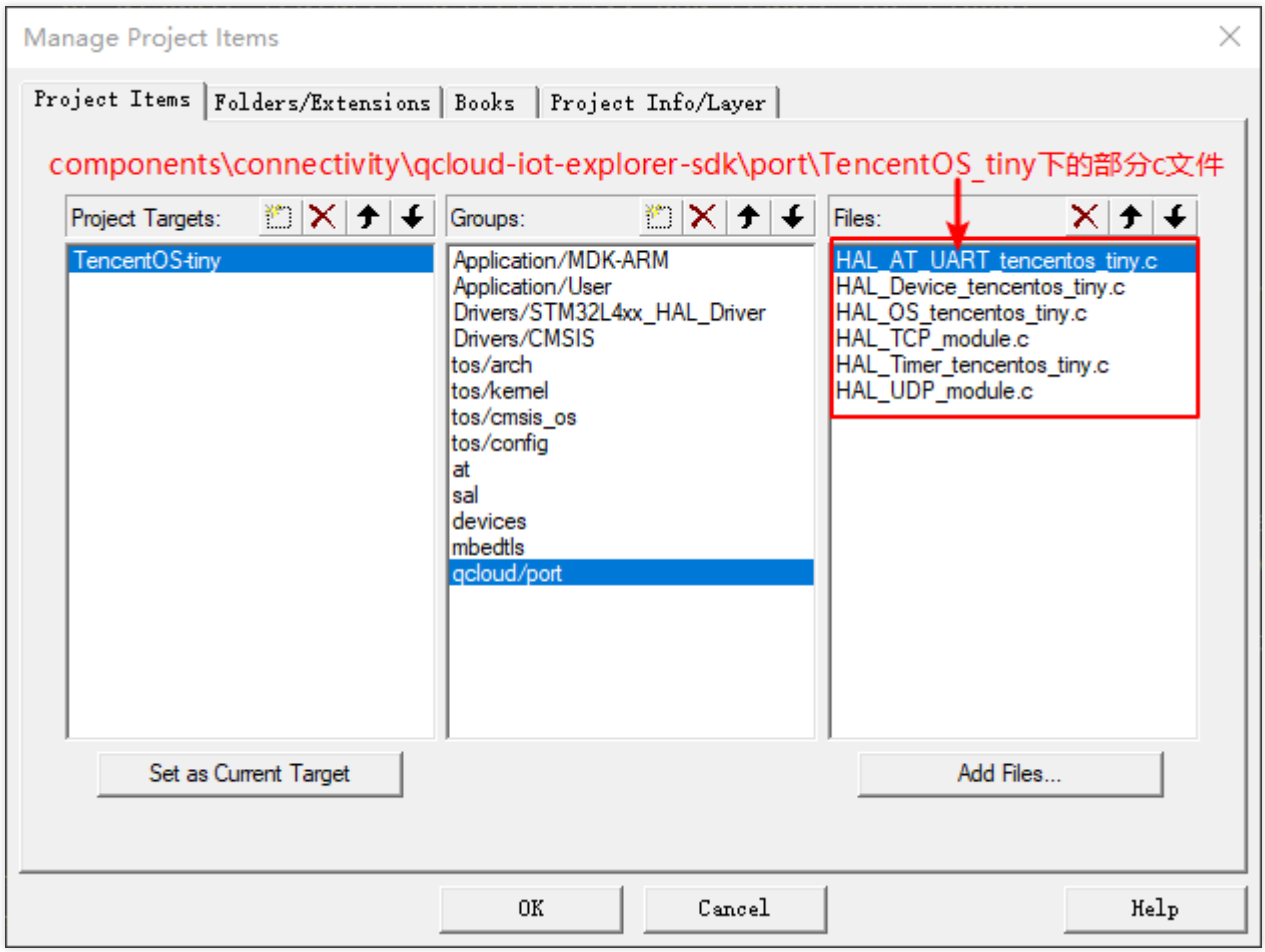
用于使用 TencentOS-tiny 物联网操作系统时，只需要添加相关文件，并修改云端设备对接信息即可，方便快捷。

接下来将介绍基于之前步骤已移植成功的网络工程，讲述如何移植 C-SDK。

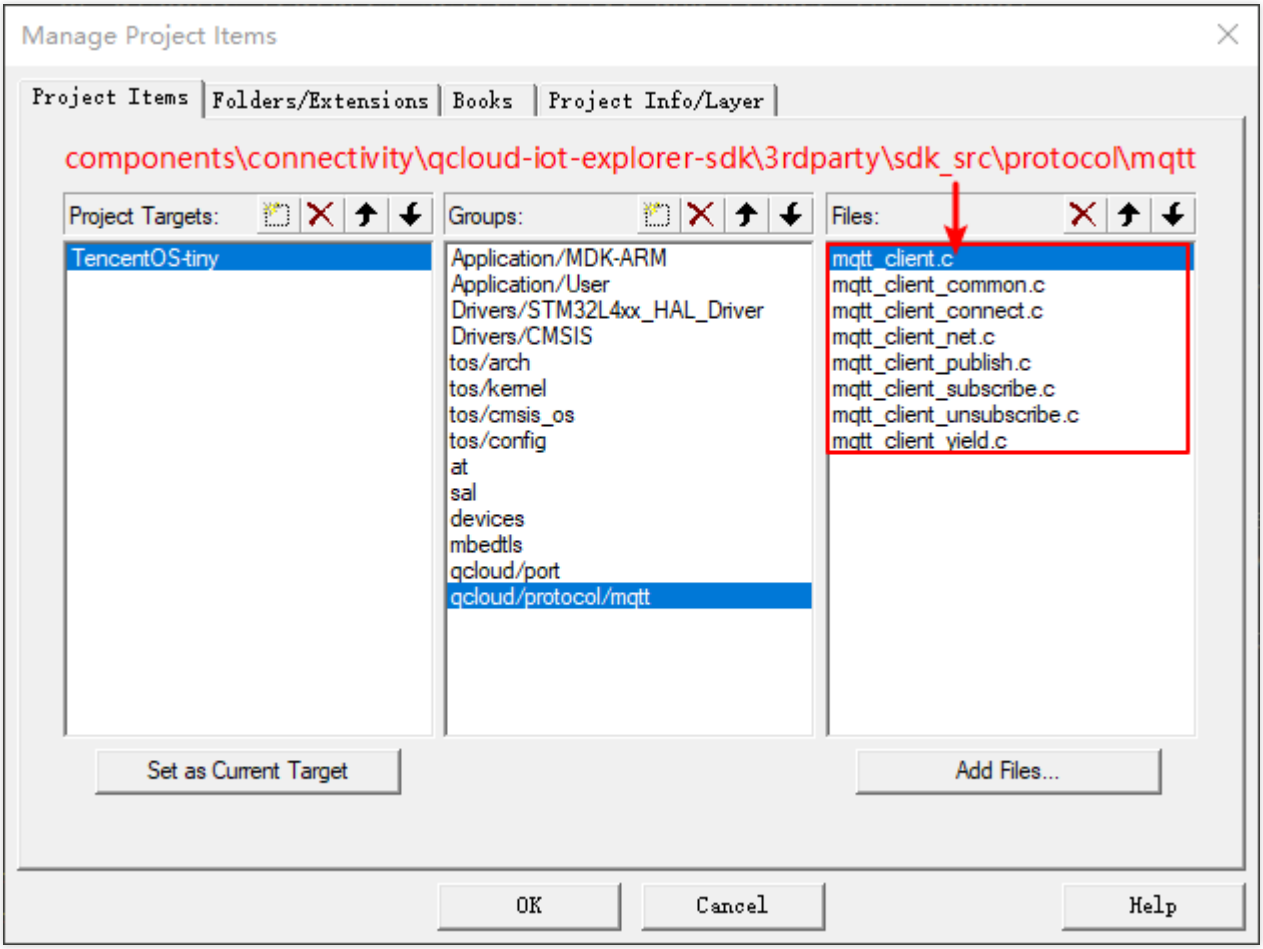
1. 将 TencentOS-tiny 源码中 `qcloud-iot-explorer-sdk` 整个目录复制到工程目录中，保持原有目录架构不变并删除其余的目录。



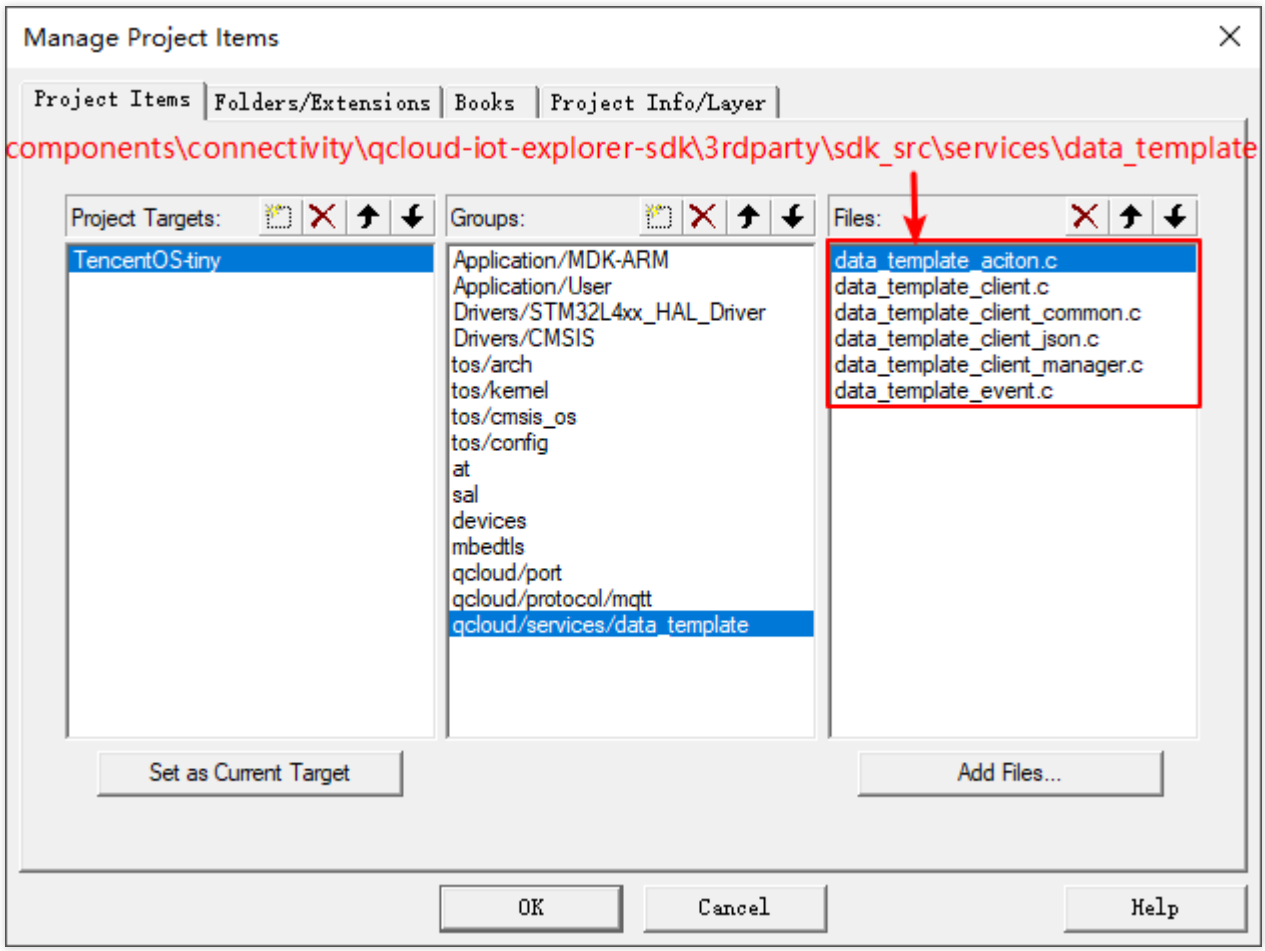
2. 添加腾讯云 C-SDK 移植到 TencentOS-tiny 的适配文件。



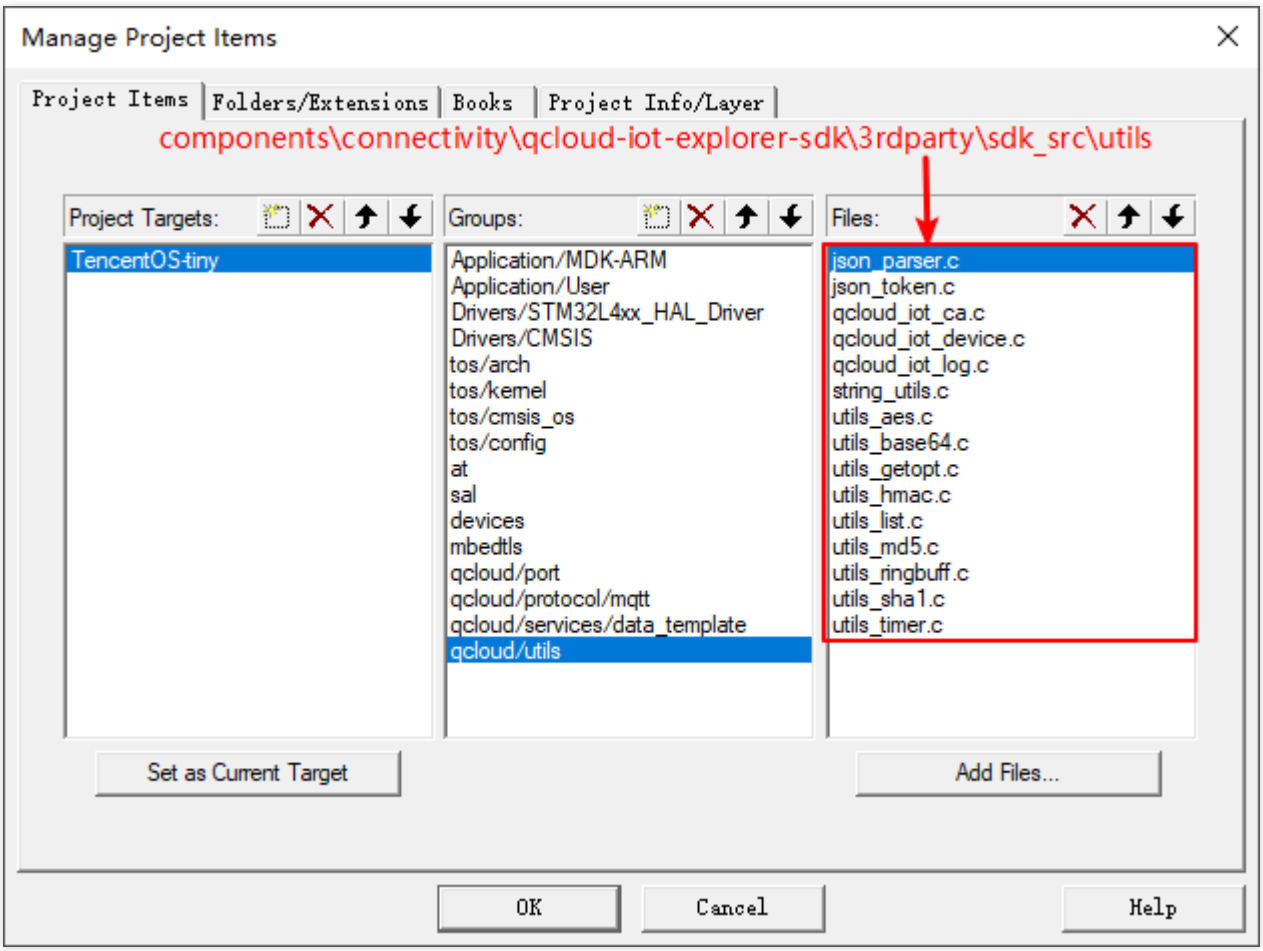
3. 添加腾讯云 C-SDK 中的 mqtt 协议相关源码。



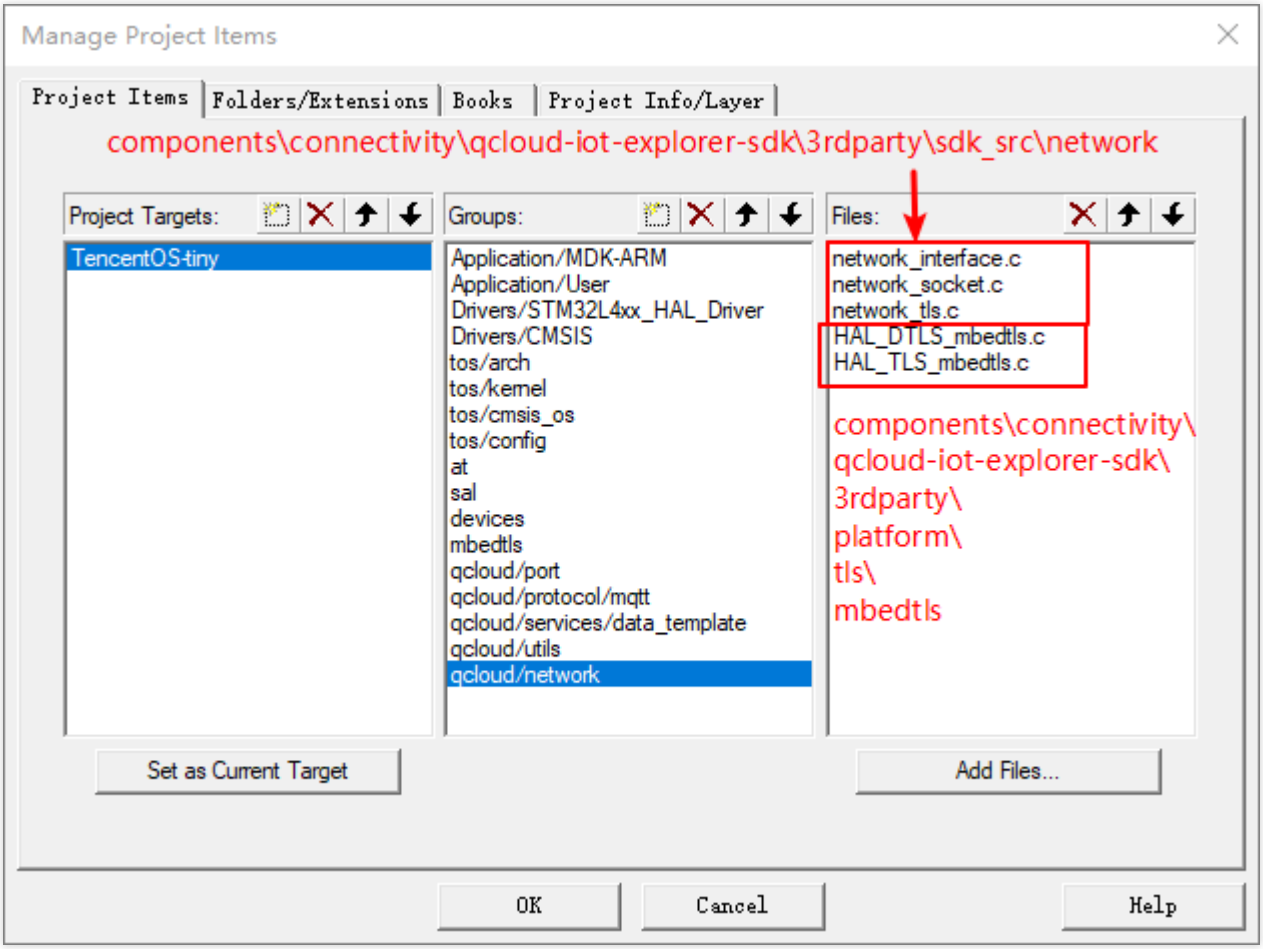
4. 添加腾讯云 C-SDK 中的数据模板相关源码。



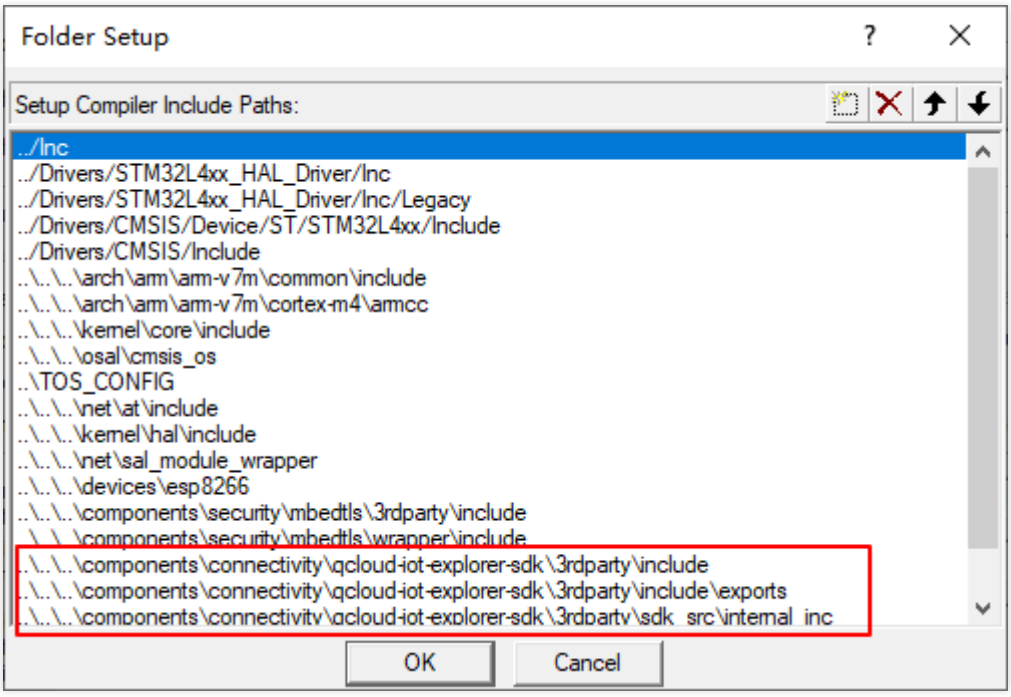
5. 添加腾讯云 C-SDK 中所使用到的工具源码。



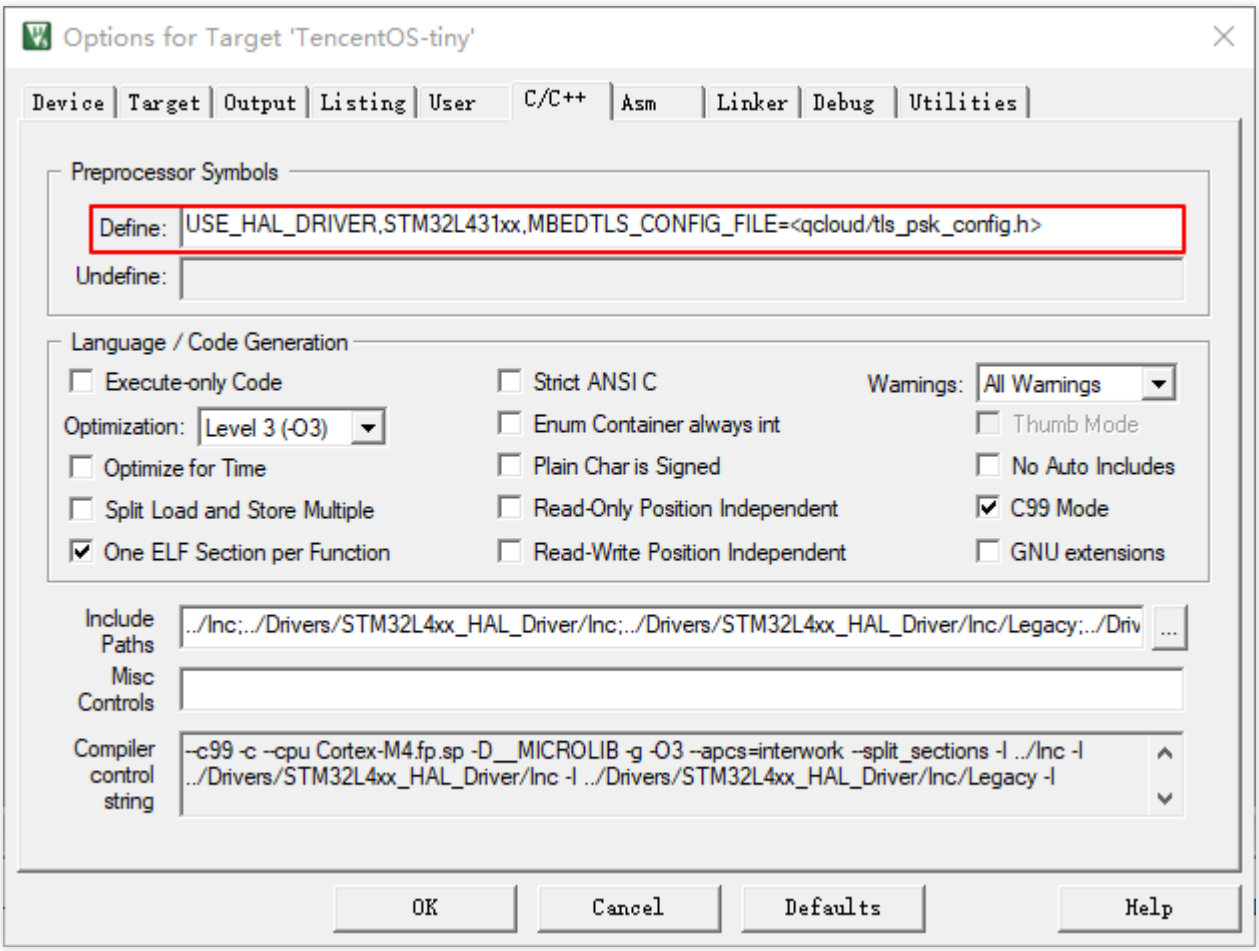
6. 添加腾讯云 C-SDK 中所用到网络封装层源。



7. 添加所有用到的头文件路径。



8. 最后添加宏定义 `MBEDTLS_CONFIG_FILE=<qcloud/tls_psk_config.h>` ，指定 mebedtls 库的配置文件。



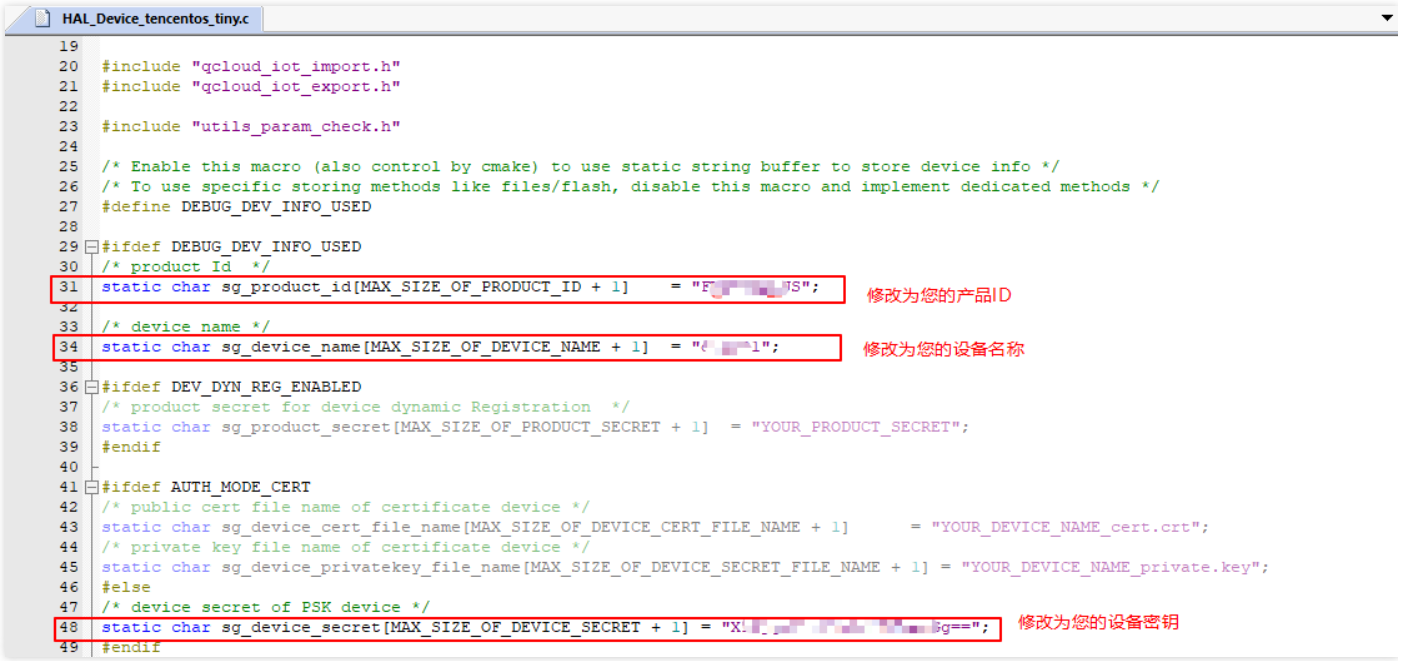
移植完成，此时编译时未发现错误信息，其中警告可暂时忽略。

步骤四：修改端云对接信息

修改 `HAL_Device_tencentos_tiny.c` 文件。在 `TencentOS-tiny\components\connectivity\qcloud-iot-explorer-sdk\port\TencentOS_tiny` 目录中，将下图中的数据分别替换为控制台【设备详情页】中的参数并保存。

- 产品 ID： 将控制台的产品 ID ，复制到上图 sg_product_id。
- 设备名称： 将控制台的设备名称，复制到上图 sg_device_name。

- 设备密钥：将控制台的设备密钥，复制到上图sg_device_secret。




```
#include "qcloud_iot_import.h"
#include "qcloud_iot_export.h"

#include "utils_param_check.h"

/* Enable this macro (also control by cmake) to use static string buffer to store device info */
/* To use specific storing methods like files/flash, disable this macro and implement dedicated methods */
#define DEBUG_DEV_INFO_USED

#ifdef DEBUG_DEV_INFO_USED
/* product Id  */
static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1]     = "F      US";

/* device name */
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1]   = "      1";

#ifdef DEV_DYN_REG_ENABLED
/* product secret for device dynamic Registration  */
static char sg_product_secret[MAX_SIZE_OF_PRODUCT_SECRET + 1]   = "YOUR_PRODUCT_SECRET";
#endif

#ifdef AUTH_MODE_CERT
/* public cert file name of certificate device */
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME + 1]       = "YOUR_DEVICE_NAME_cert.crt";
/* private key file name of certificate device */
static char sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_SECRET_FILE_NAME + 1] = "YOUR_DEVICE_NAME_private.key";
#else
/* device secret of PSK device */
static char sg_device_secret[MAX_SIZE_OF_DEVICE_SECRET + 1] = "X        g==";
#endif
```

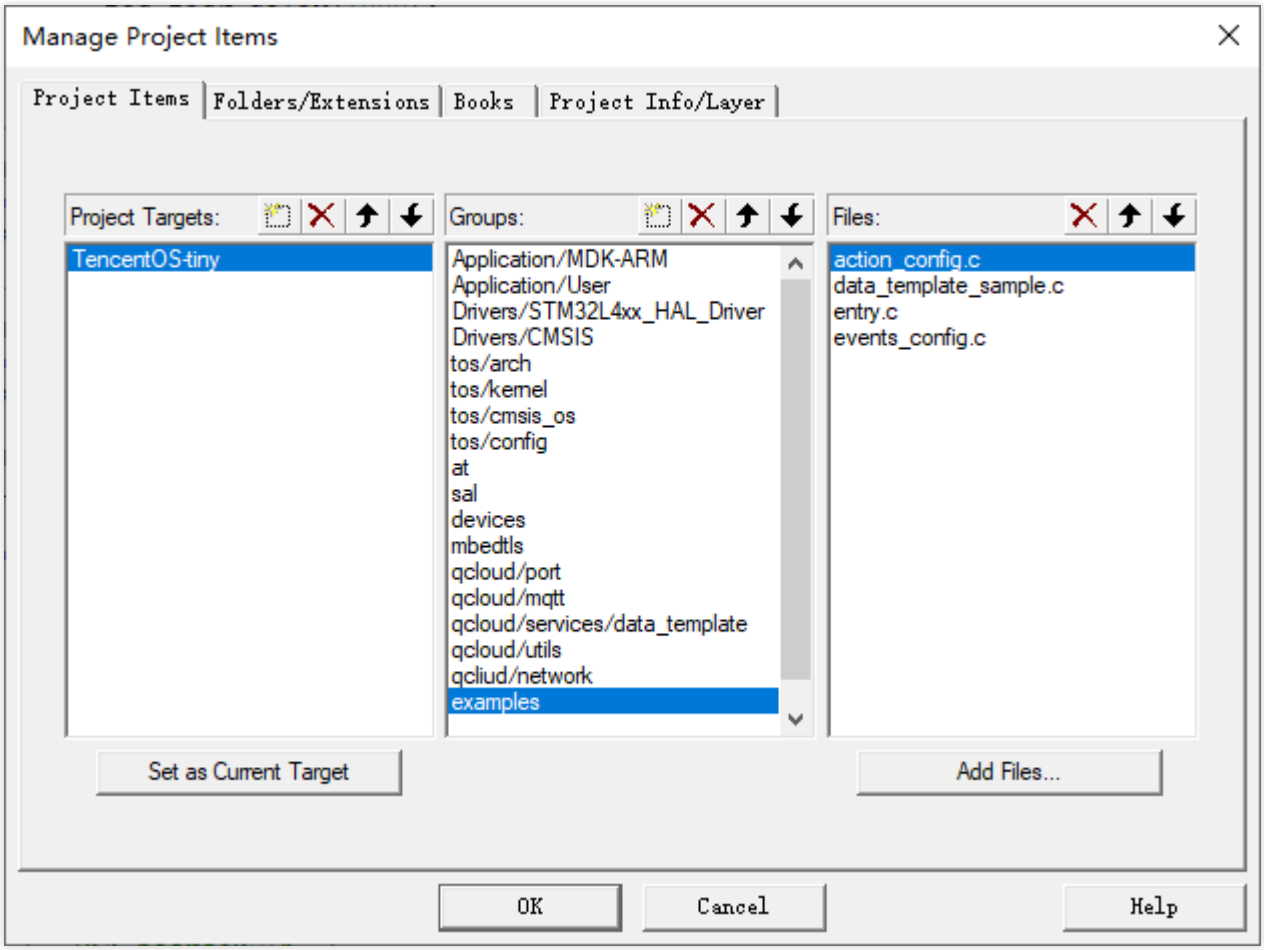
步骤五：加入示例代码

1. 由于腾讯云 C-SDK 的测试代码较多，所以将直接使用官方仓库中提供的示例文件。

   i. 在 examples\qcloud_iot_explorer_sdk_data_template 目录下，将此目录复制到工程目录中，保持原有目录架构不变。

   ii. 将示例代码加入到 Keil-MDK 工程中。

> 注意：
> 请勿将 data_config.c 文件加入！


Manage Project Items
Project Items
Folders/Extensions
Books
Project Info/Layer
Project Targets:
TencentOS-tiny
Groups:
Application/MDK-ARM
Application/User
Drivers/STM32L4xx_HAL_Driver
Drivers/CMSIS
tos/arch
tos/kernel
tos/cmsis_os
tos/config
at
sal
devices
mbedtls
qcloud/port
qcloud/mqtt
qcloud/services/data_template
qcloud/utils
qcliud/network
examples
Files:
action_config.c
data_template_sample.c
entry.c
events_config.c
Set as Current Target
Add Files...
OK
Cancel
Help

iii. 修改 entry.c 中的配置信息。

entry.c

```c
#include "tos_k.h"

/* 用户根据自己的底层通信链路来配置此宏
 * 如果是基于以太网lwip的链路，这里应该定义 USE_LWIP
 * 如果是基于模组的通信链路，这里应该定义相应的模组宏，如使用ESP8266则定义 USE_ESP8266
 * */
#define USE_ESP8266

#ifdef USE_LWIP
#include "lwip/api.h"
#include "lwip/sockets.h"
#include "lwip/err.h"
#include "lwip/sys.h"
#endif

#ifdef USE_ESP8266
#include "esp8266.h"
#endif

void application_entry(void *arg)
{
#ifdef USE_LWIP
    dns_init();
    MX_LWIP_Init();
#endif

#ifdef USE_ESP8266
    extern int esp8266_sal_init(hal_uart_port_t uart_port);
    extern int esp8266_join_ap(const char *ssid, const char *pwd);
    esp8266_sal_init(HAL_UART_PORT_0);
    esp8266_join_ap("Mculover666", "mculover666");
#endif

#ifdef USE_NB_BC35
    extern int bc35_28_95_sal_init(hal_uart_port_t uart_port);
    bc35_28_95_sal_init(HAL_UART_PORT_0);
#endif
```

修改为您的串口号和入网信息

2. 示例代码中的任务入口函数为 application_entry，所以需要将 task1 任务的任务入口函数修改为 application_entry，并再次扩大 task1 的任务栈为4096字节，使示例程序正常运行。

main.c

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
extern void application_entry(void *arg);
__weak void application_entry(void *arg)
{
    while (1) {
        printf("This is a demo task,please use your task entry!\r\n");
        tos_task_delay(1000);
    }
}

//task1
#define TASK1_STK_SIZE        4096
void task1(void *pdata);
osThreadDef(application_entry, osPriorityNormal, 1, TASK1_STK_SIZE);

//task2
#define TASK2_STK_SIZE        256
void task2(void *pdata);
osThreadDef(task2, osPriorityNormal, 1, TASK2_STK_SIZE);
```

3. 然后修改创建任务的代码。

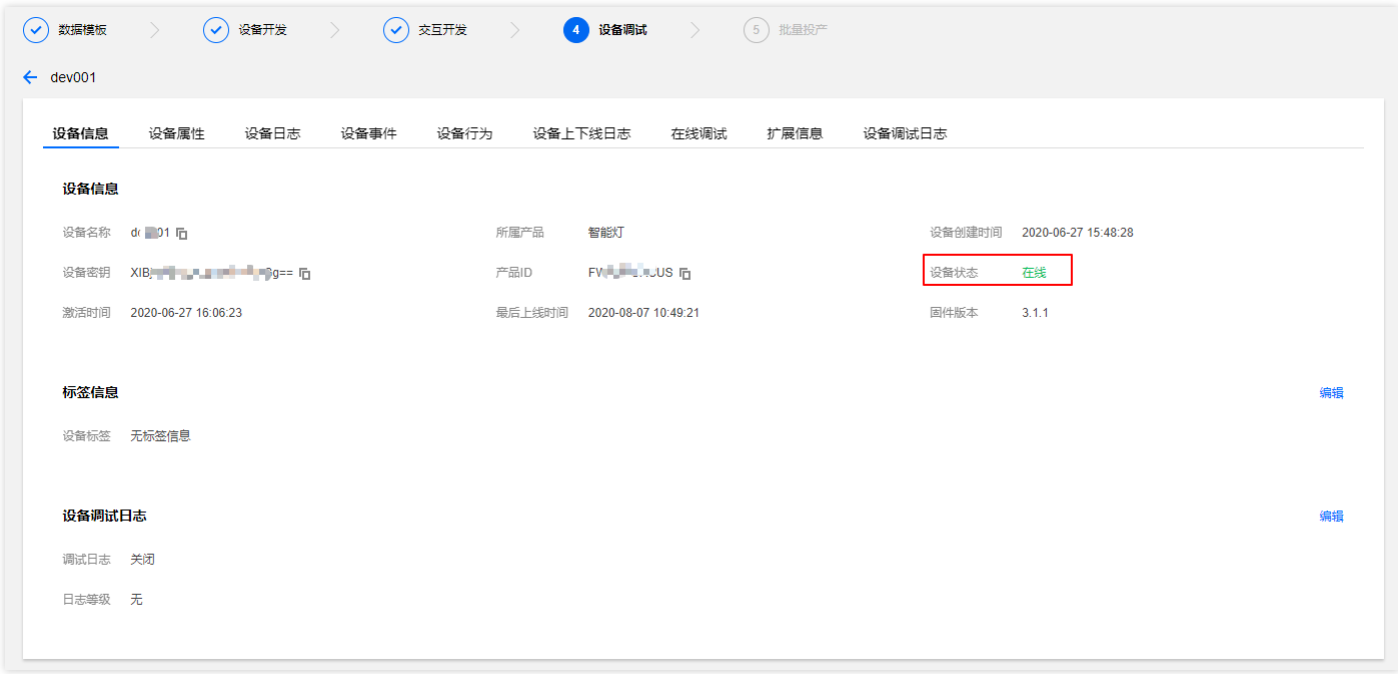
main.c

```
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_LPUART1_UART_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  osKernelInitialize(); //TOS Tiny kernel initialize
  osThreadCreate(osThread(application_entry), NULL);// Create task1   创建任务
  //osThreadCreate(osThread(task2), NULL);// Create task2
  osKernelStart();//Start TOS Tiny
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```

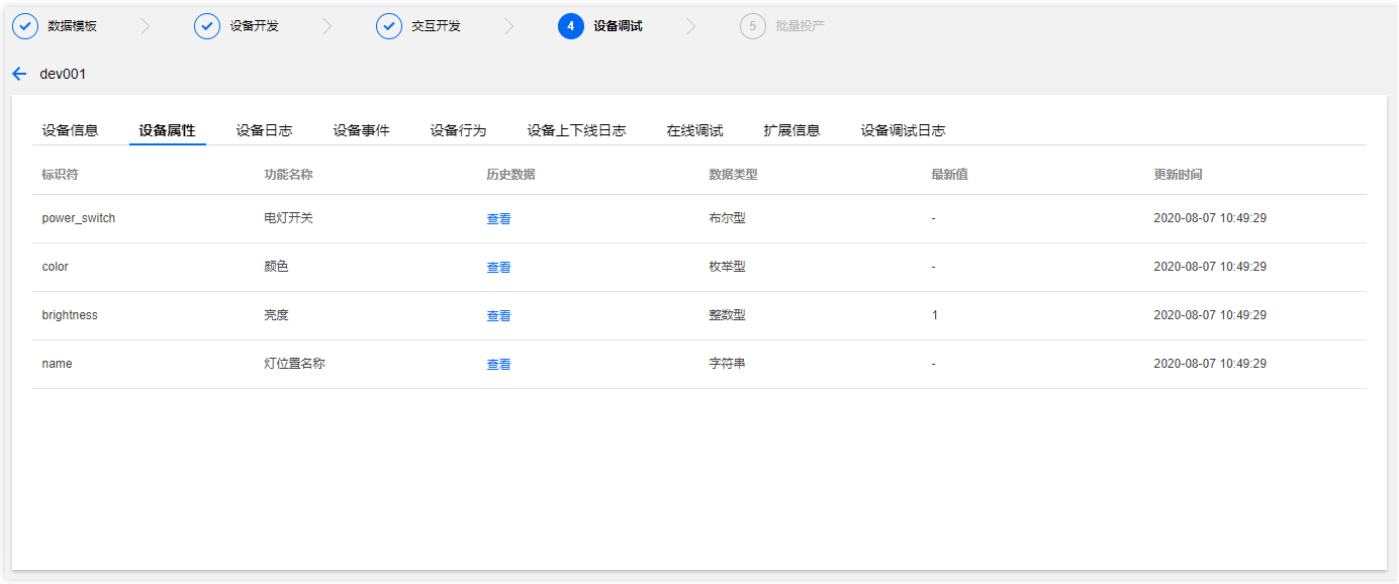
4. 最后进行编译，将程序下载到开发板中，复位开发板后开始运行，便可以在串口助手中查看打印信息。

步骤六：查看设备状态

1. 保持 light Demo 程序为运行状态。
2. 进入【控制台】>【产品开发】>【设备调试】，可查看到设备 "dev001" 的状态为“上线”状态，表示 Demo 程序已成功连接上开发平台。

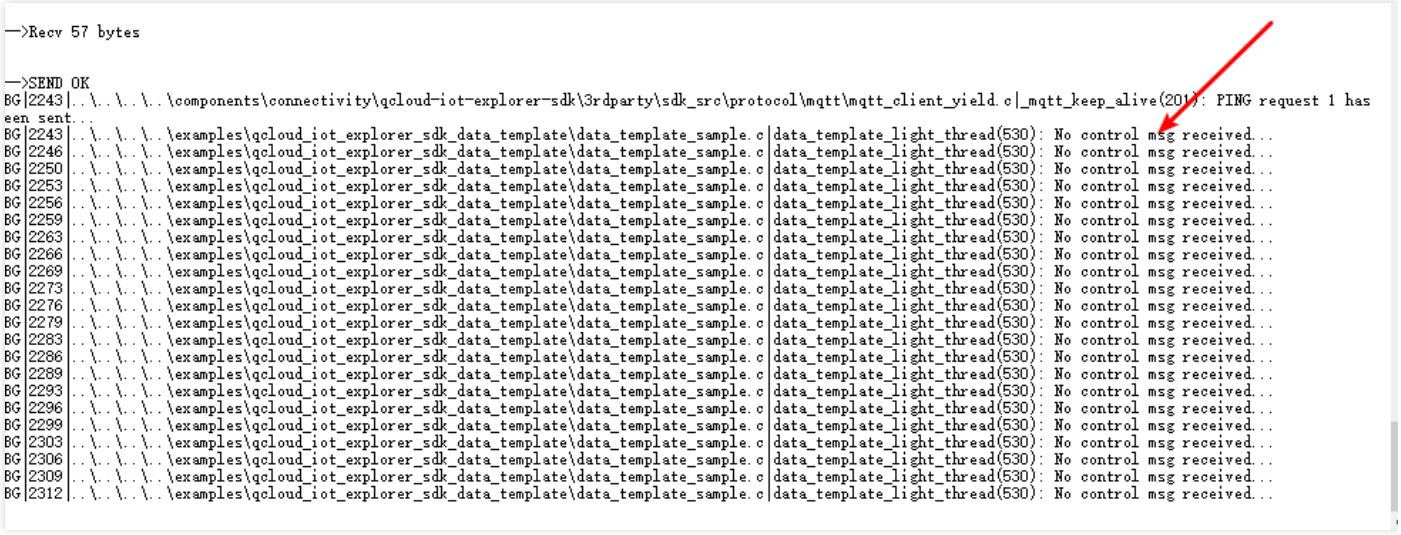


3. 单击【查看】，可进入设备详情页。

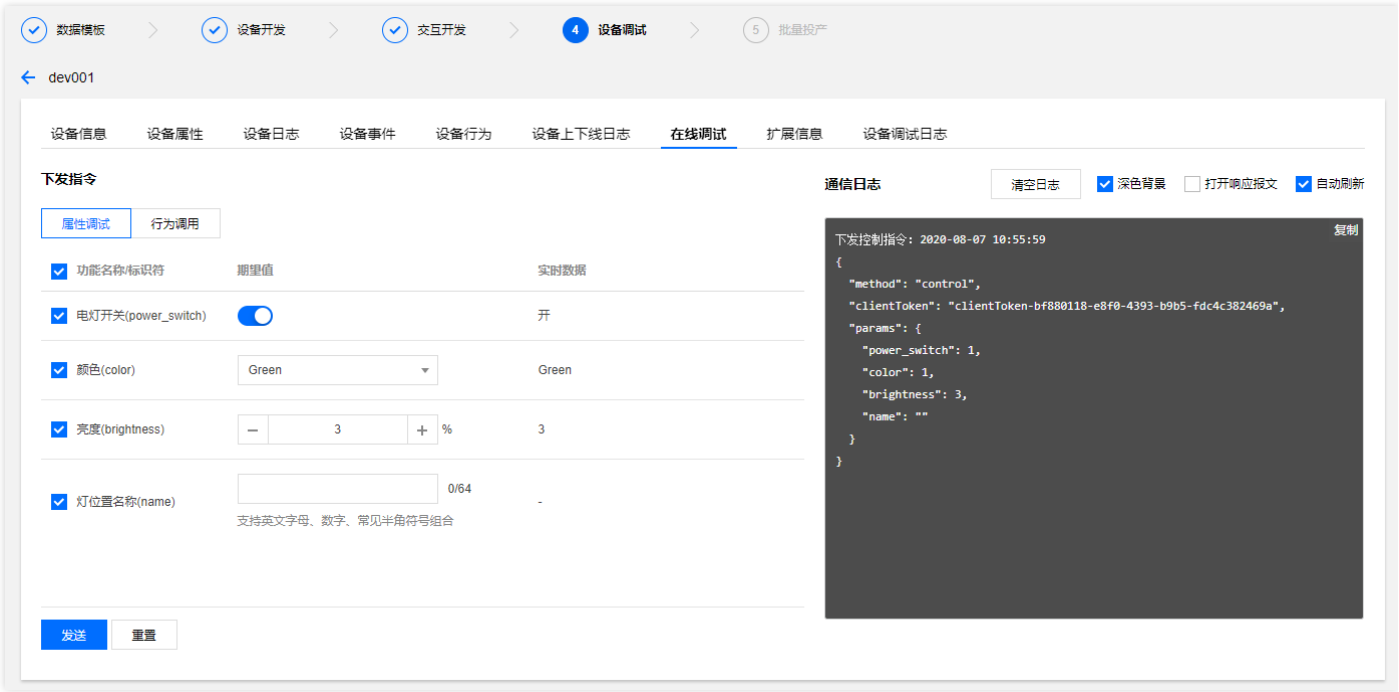


步骤七：下发控制指令

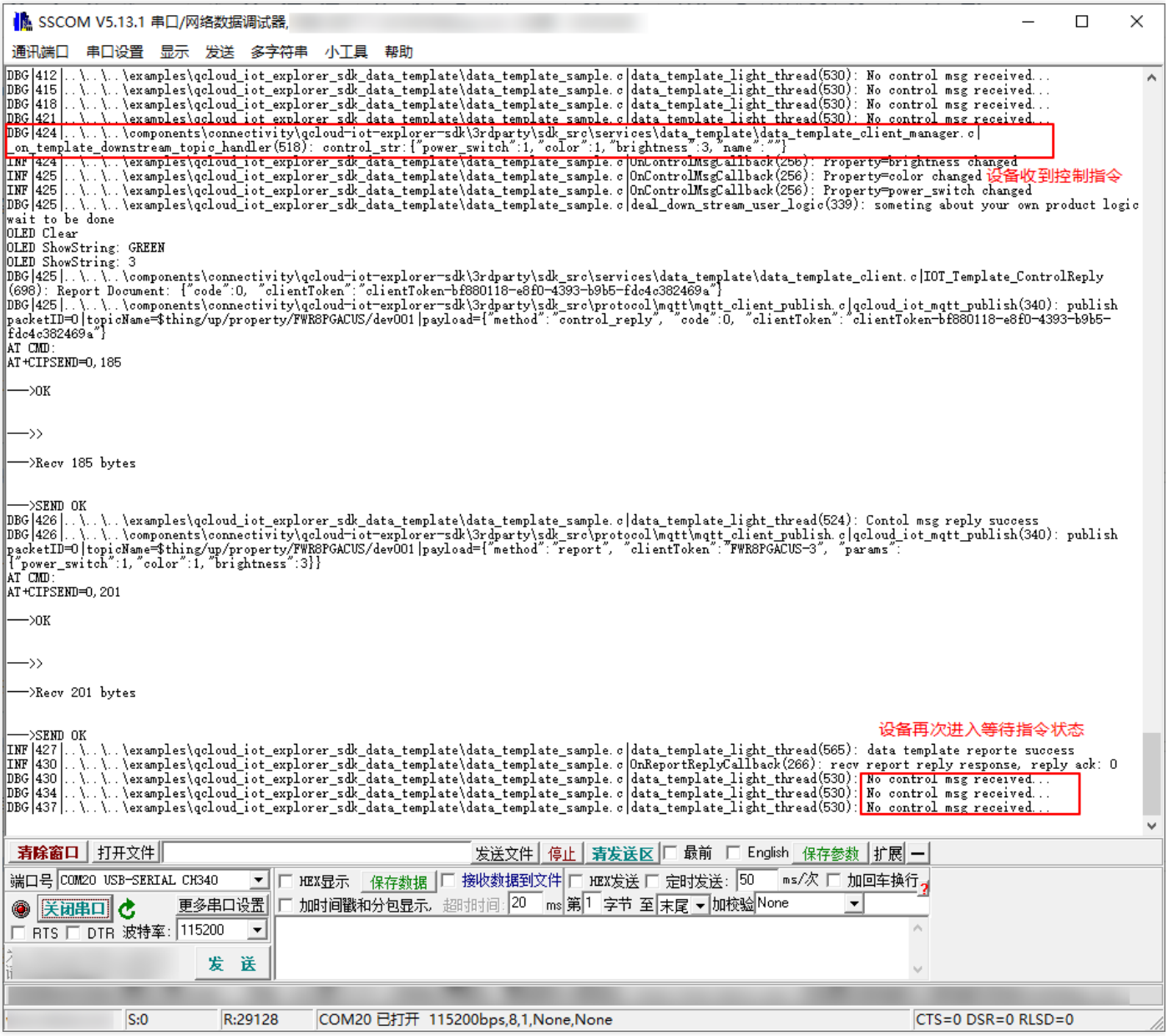
1. 在串口助手中看到设备查看到在等待平台下发控制指令。



2. 然后在云端平台进入设备在线调试，下发控制指令。

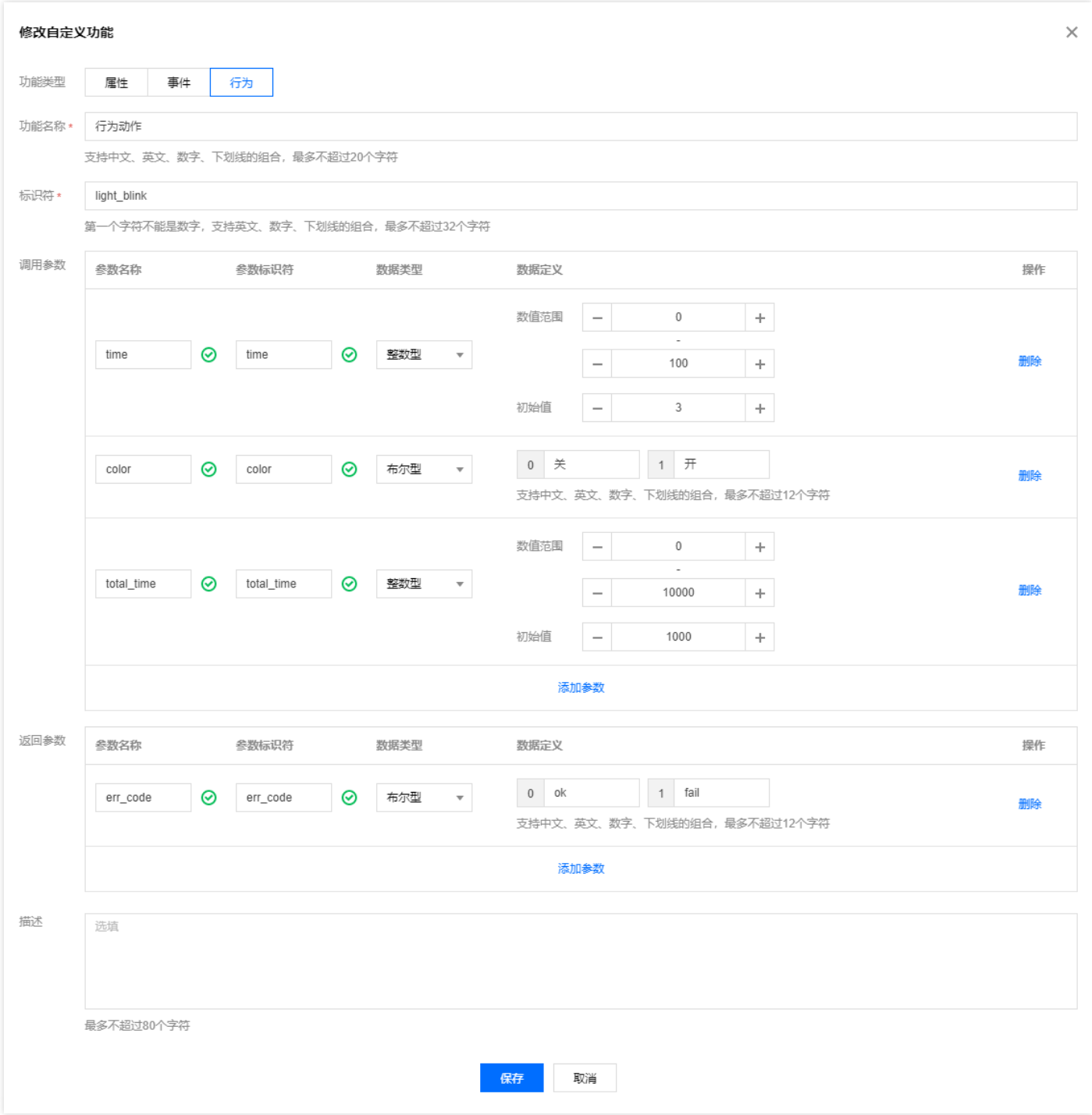


3. 最后可以在串口助手中查看到设备收到后在串口打印控制指令。



步骤八：设备行为调用

1. 在云端的数据模板中手动新建一个设备行为功能。

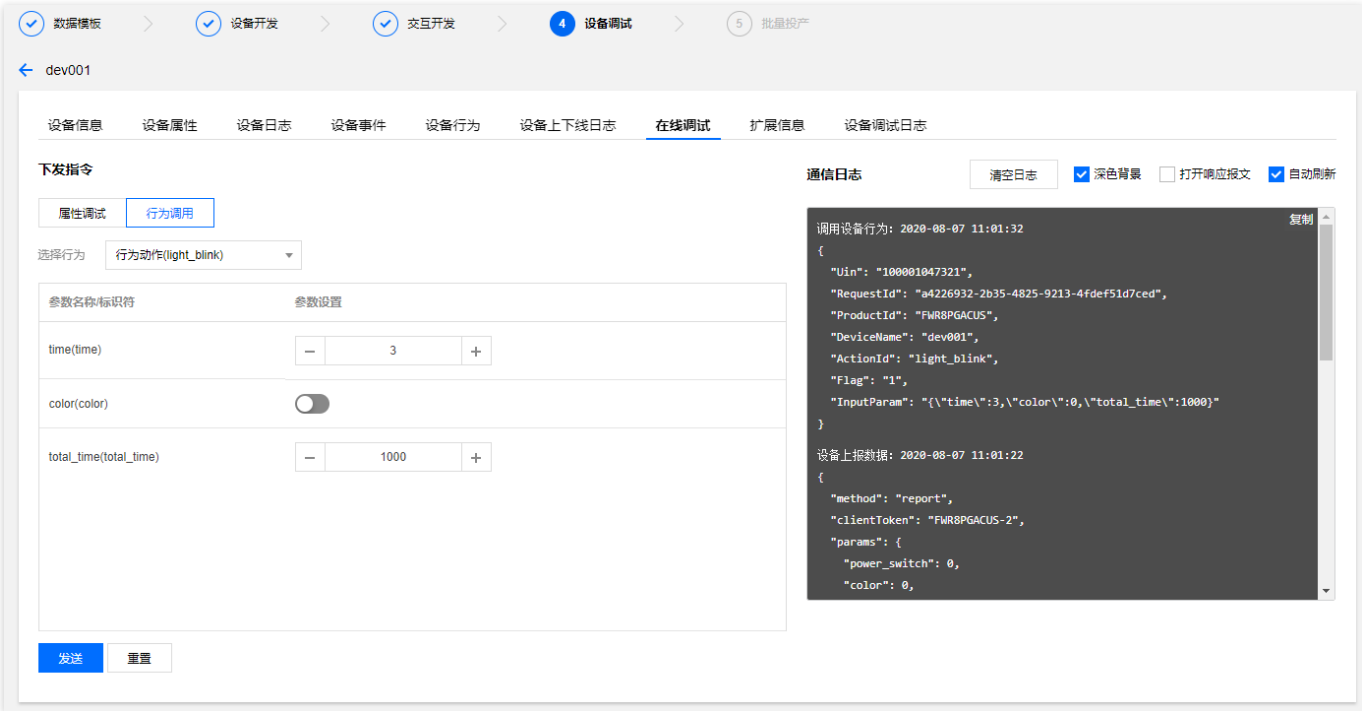


2. 在 data_template_sample.c 文件中使能 Action。

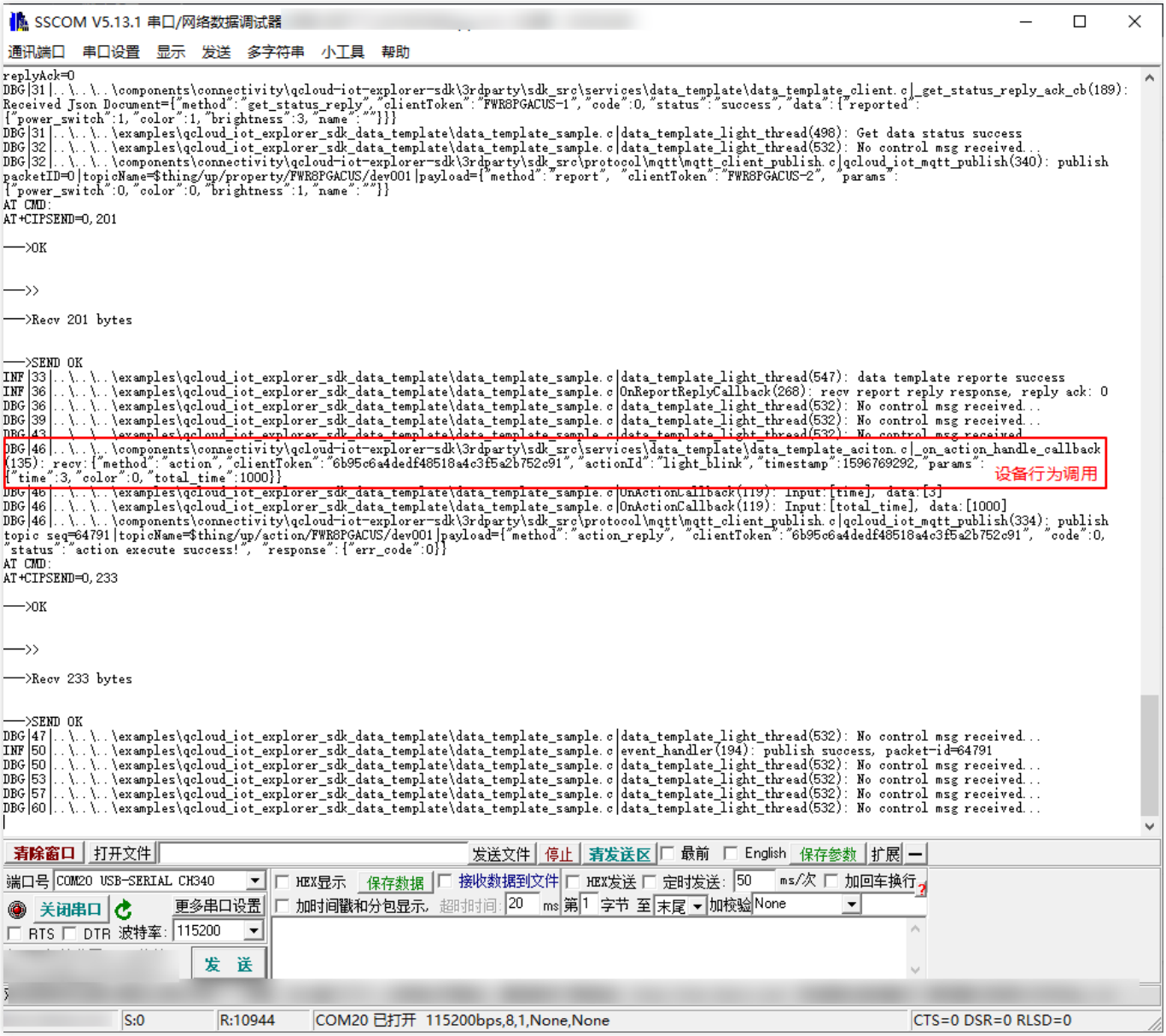
main.c | data_template_sample.c

```
/*
 * Tencent is pleased to support the open source community by making IoT Hub available.
 * Copyright (C) 2016 THL A29 Limited, a Tencent company. All rights reserved.

 * Licensed under the MIT License (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 * http://opensource.org/licenses/MIT

 * Unless required by applicable law or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
 * either express or implied. See the License for the specific language governing permissions and
 * limitations under the License.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>

#include "qcloud_iot_export.h"
#include "qcloud_iot_import.h"
#include "lite-utils.h"
#include "data_config.c"

#define ACTION_ENABLED

```

3. 重新编译下载，按复位运行。
   - 在云端下发设备行为调用：



- 在串口助手中可以看到设备行为被调用：



步骤九：设备事件调用

1. 修改 config.h 文件，将设备行为调用关闭，开启设备事件支持。

main.c | data_template_sample.c | config.h

```
1  /* #undef AUTH_MODE_CERT */
2  #define AUTH_MODE_KEY
3  /* #undef AUTH_WITH_NOTLS */
4  #define GATEWAY_ENABLED
5  /* #undef COAP_COMM_ENABLED */
6  #define OTA_MQTT_CHANNEL
7  /* #undef SYSTEM_COMM */
8  /* #undef EVENT_POST_ENABLED */
9  #define EVENT_POST_ENABLED          开启事件上报支持
10 //#define ACTION_ENABLED            屏蔽行为调用支持
11 /* #undef DEV_DYN_REG_ENABLED */
12 /* #undef LOG_UPLOAD */
13 /* #undef IOT_DEBUG */
14 /* #undef DEBUG_DEV_INFO_USED */
15 /* #undef AT_TCP_ENABLED */
16 #define AT_UART_RECV_IRQ
17 /* #undef AT_OS_USED */
18 /* #undef AT_DEBUG */
19 /* #undef OTA_USE_HTTPS */
20 #define GATEWAY_ENABLED
21 /* #undef MULTITHREAD_ENABLED */
22
```

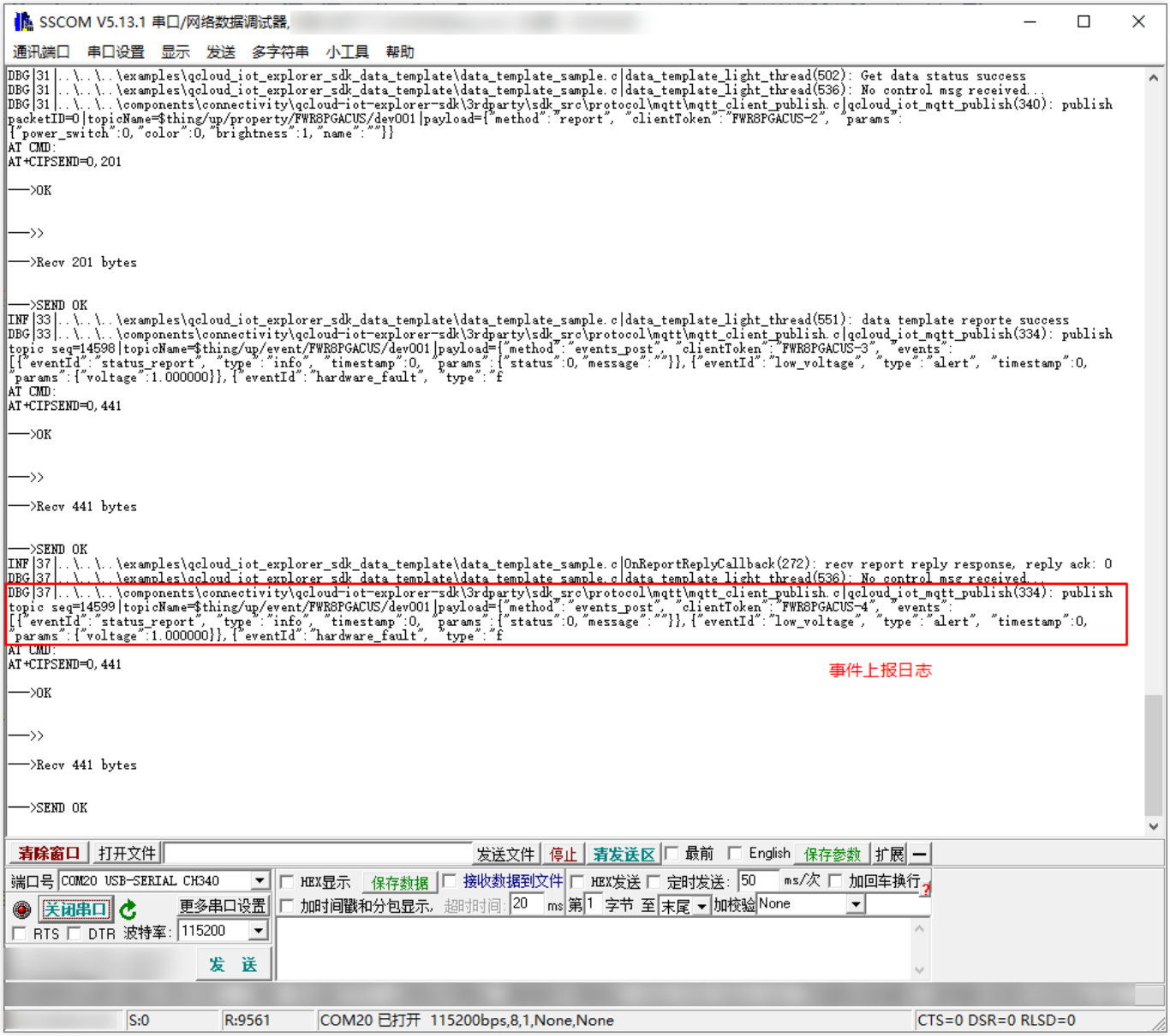
2. 修改 data_template_sample.c 文件，关闭设备行为调用示例，开启事件上报示例。

main.c | data_template_sample.c

```
1  /*
2   * Tencent is pleased to support the open source community by making IoT Hub available.
3   * Copyright (C) 2016 THL A29 Limited, a Tencent company. All rights reserved.
4
5   * Licensed under the MIT License (the "License"); you may not use this file except in
6   * compliance with the License. You may obtain a copy of the License at
7   * http://opensource.org/licenses/MIT
8
9   * Unless required by applicable law or agreed to in writing, software distributed under the License is
10  * distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
11  * either express or implied. See the License for the specific language governing permissions and
12  * limitations under the License.
13  *
14  */
15
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <limits.h>
19 #include <stdbool.h>
20 #include <string.h>
21 #include <time.h>
22
23 #define EVENT_POST_ENABLED          开启设备事件上报示例
24
25 #include "qcloud_iot_export.h"
26 #include "qcloud_iot_import.h"
27 #include "lite-utils.h"
28 #include "data_config.c"
29
30 //#define ACTION_ENABLED            屏蔽设备行为调用示例
31
```

3. 编译程序，下载到开发板中，复位。

- 在串口助手中可以看到设备上报事件的日志：



- 在平台端可以看到设备上报事件的日志：