STM32Cube高效开发教程（高级篇）

第1章 FreeRTOS基础

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（高级篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2022年2月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc. edu. cn

第1章 FreeRTOS基础

1.1 FreeRTOS概述

1.2 FreeRTOS入门实例

1.3 FreeRTOS的文件组成和基本原理

1.1 FreeRTOS概述

1.1.1 FreeRTOS发展历史

1.1.2 FreeRTOS的特点和许可方式

1.1.3 FreeRTOS的一些概念和术语

1.1.4 为什么要使用RTOS

1.1.1 FreeRTOS发展历史

- 2003年左右，由Richard Barry开发，并成立公司Real Time Engineers管理和维护，使用开源和商业两种许可模式
- 在2017年，Real Time Engineers公司将FreeRTOS项目的管理权转交给Amazon Web Service（AWS），并且使用了更加开发的MIT许可协议
- FreeRTOS是一个完全免费和开源的嵌入式实时操作系统

FreeRTOS是目前使用最广泛的免费RTOS系统，已经超过μC/OS-II（免费），μC/OS-III（收费）

FreeRTOS官网 https://www.freertos.org/RTOS.html

Quality RTOS & Embedded Software

About Contact Support FAQ Download

Quick Start | Supported MCUs | PDF Books | Trace Tools | Ecosystem | Email List

- Home
- FreeRTOS Books and Manuals
- FreeRTOS
  - About FreeRTOS
    - What is an RTOS/FreeRTOS?
    - A Compelling Free Solution
    - A Better Type of Open Source
    - Coding Standard/MISRA /Testing
    - Features Overview
    - Licensing
    - Site Map
  - Features / Getting Started...
  - More Advanced...
  - Demo Projects
  - Supported Devices & Demos
  - API Reference
  - Contact & Support
- FreeRTOS Interactive!

Quick Start Guide

Support Forum

⬇ Download Source ⬇

History

The FreeRTOS kernel was originally developed by Richard Barry around 2003, and was later developed and maintained by Richard's company, Real Time Engineers Ltd. FreeRTOS was a runaway success, and in 2017 Real Time Engineers Ltd. passed stewardship of the FreeRTOS project to Amazon Web Services (AWS). Richard continues to work on FreeRTOS as part of an AWS team. Read more on the AWS open source blog, and the FAQ question Why Have Amazon Taken Stewardship of FreeRTOS?.

About Amazon Web Services

Amazon Web Services provides a highly reliable, scalable, low-cost cloud infrastructure platform that powers hundreds of thousands of businesses in 190 countries around the world. In 2015 AWS added specific Internet of Things (IoT) capabilities, and now offers Amazon FreeRTOS to help users securely connect their MCU devices to the cloud.

Amazon FreeRTOS uses the FreeRTOS kernel, and adds libraries that make small low-power edge devices easy to program, deploy, secure, connect, and manage. You do not need to be an AWS customer to use Amazon FreeRTOS as the source code is provided under the MIT license.

1.1.2 FreeRTOS的特点和许可方式

FreeRTOS是一个技术上非常完善和成功的RTOS，具有如下的一些标准功能

- 抢占式（pre-emptive）或合作式（co-operative）任务调度方式
- 非常灵活的优先级管理
- 灵活的、快速而轻量化的任务通知（task notification）机制
- 队列（queues）功能
- 二值信号量（binary semaphores）
- 计数信号量（counting semaphores）
- 互斥量（mutexes）
- 递归互斥量（recursive mutexes）
- 软件定时器（software timers）
- 事件组（event groups）
- 任务运行时统计收集（task run-time statics gathering）
- 用于超低功耗的无节拍（tick-less）特性

FreeRTOS的MIT许可协议也为用户扫除了使用FreeRTOS的障碍。FreeRTOS不包含其他任何IP（intellectual property）问题，用户可以完全免费使用FreeRTOS，即使免费用于商业性项目也无需公开自己的源代码，也无需支付任何费用。

FreeRTOS宣称其使命（mission）就是

> ***"Provide a free product that surpasses the quality and service demanded by users of commercial alternatives"***

FreeRTOS还有两个衍生的商业版本：

- OpenRTOS是一个基于FreeRTOS内核的商业许可版本
- SafeRTOS是基于FreeRTOS内核的一个衍生版本，用于安全性要求高的应用

1.1.3 FreeRTOS的一些概念和术语

1. 实时性

RTOS是Real-time Operating System（实时操作系统）

实时性指任务的完成时间是确定的，例如飞机驾驶控制系统，必须在限定的时间内完成对飞行员操作的响应。

FreeRTOS是一个实时操作系统，满足硬实时要求

- 软实时指任务运行要求有一个截止时间，但是超过了这个截止时间不会使系统变得毫无用处。
- 硬实时指任务运行要求有一个截止时间，如果超过了这个截止时间可能导致整个系统的功能失效。例如，轿车的安全气囊控制系统如果在出现撞击时响应缓慢可能导致严重的后果。

2. 任务（Task）

操作系统的主要功能就是实现了多任务管理，FreeRTOS是一个支持多任务的实时操作系统，FreeRTOS将任务称为线程（thread），但我们还是使用常用的名词“任务”。

一般的MCU都是单核的，处理器在任何时刻只能执行一个任务的代码。FreeRTOS的多任务功能是通过其内核中的任务调度器来实现的，FreeRTOS支持基于任务优先级的抢占式任务调度算法，因而能满足硬实时的要求。

3. 移植

FreeRTOS少部分与硬件密切相关的源码需要针对不同架构的MCU进行一些改写，例如针对STM32系列单片机，就需要改写相应的代码，这个过程称为移植。移植好的一套FreeRTOS源码称为一个接口（port）。

STM32Cube固件包里包含移植好了的FreeRTOS源码，例如对于STM32F4系列，其STM32CubeF4中就包含针对STM32F4移植好了的FreeRTOS源码，在CubeMX中作为一个中间件使用。

1.1.4 为什么要使用RTOS

- 裸机系统：初学时使用，功能要求不复杂时使用；
- 使用RTOS：功能要求复杂，必须使用多任务，或对实时性要求高时使用

使用RTOS并且将功能分解为多个任务，可以使程序功能模块化，程序结构更简单，便于维护和扩展，也便于团队协作开发，提高开发效率。

1.2 FreeRTOS入门实例

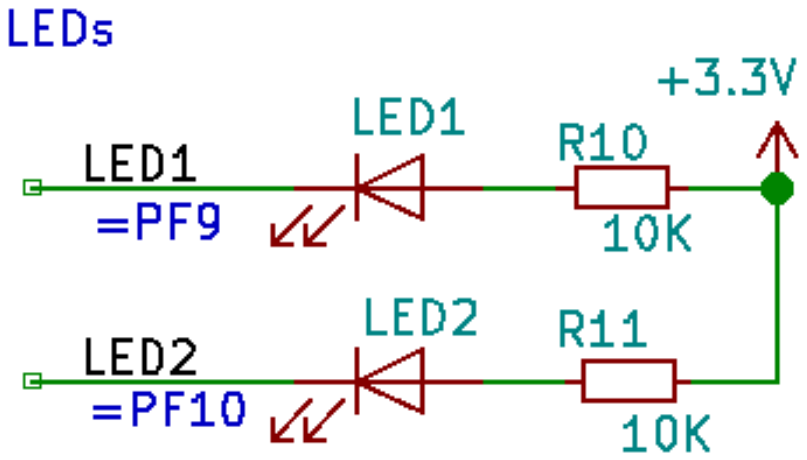
1.2.1 CubeMX项目配置

1.2.2 含FreeRTOS的项目的文件组成

1.2.3 程序分析和功能实现

1.2.1 CubeMX项目配置

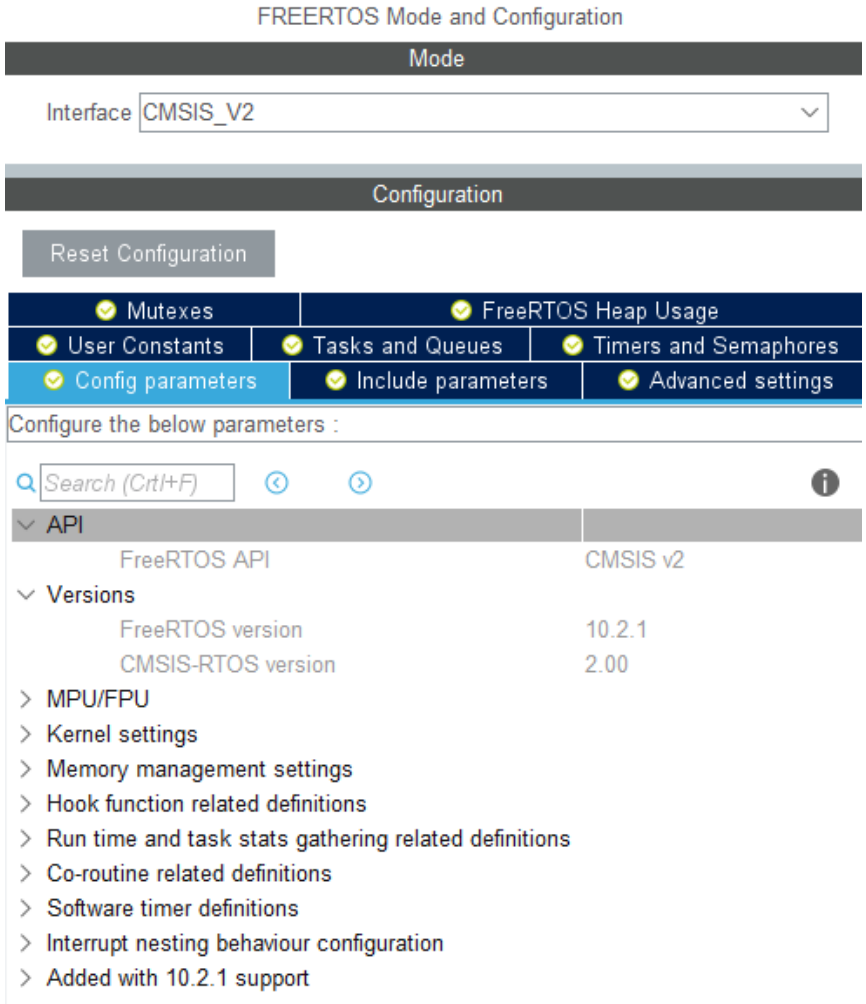
这个实例只用到开发板上的LED1和LED2，它们连接的引脚分别是PF9和PF10



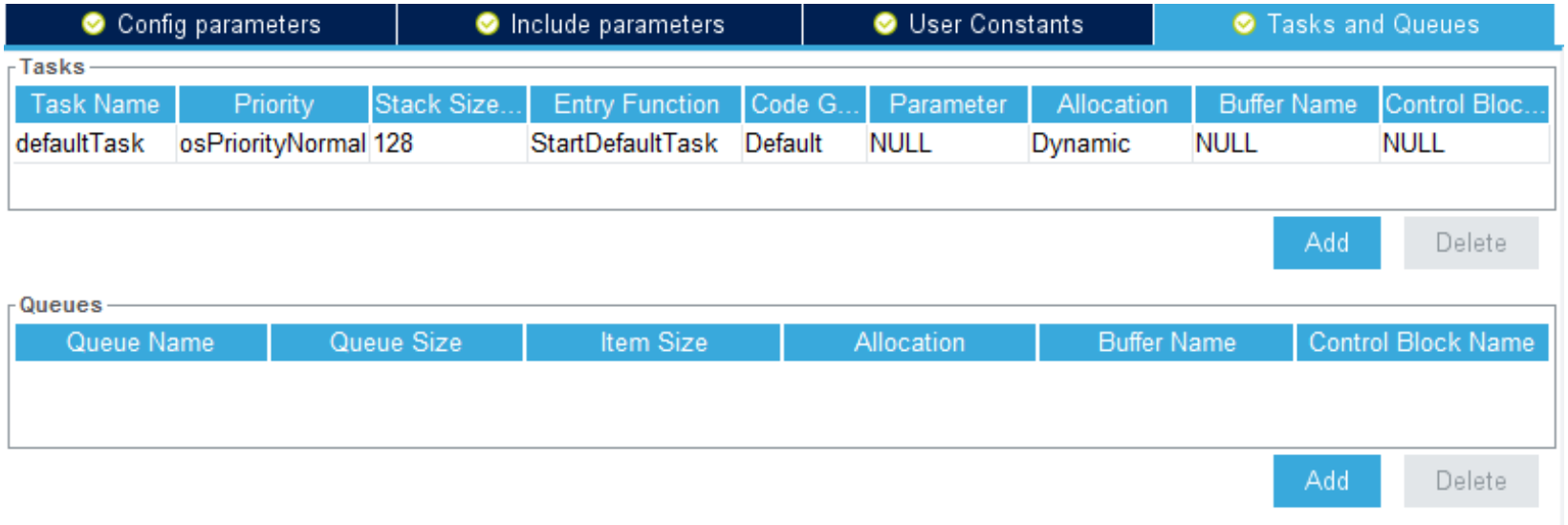
FreeRTOS的模式和参数设置

有多个配置页面

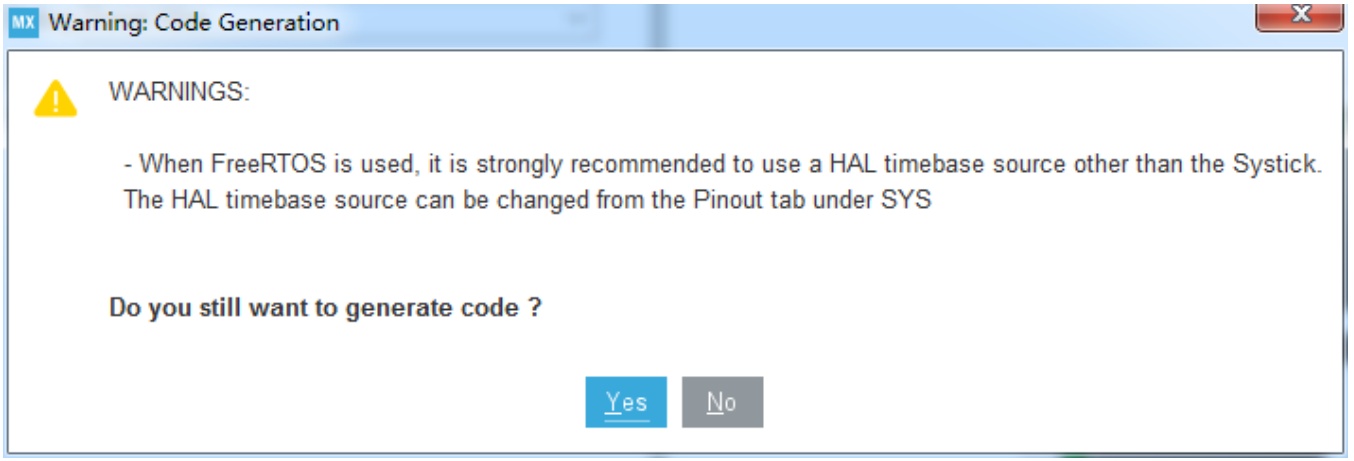
- Configure parameters
- Include parameters
- Tasks and Queues
- Timers and Semaphores
- Mutexes
- FreeRTOS Heap Usage
- User Constants
- MPU Settings



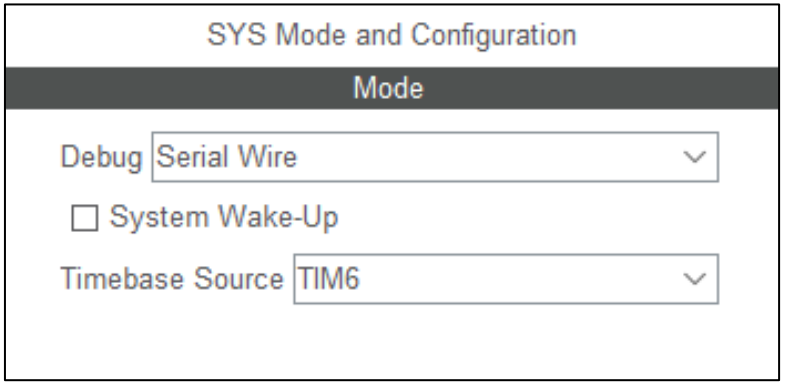
在启用FreeRTOS时就定义了一个缺省的任务defaultTask，任务实现函数名称是StartDefaultTask



在首次生成代码时会出现如图所示的对话框，提示在使用FreeRTOS时应该使用一个独立的定时器作为基础时钟源，而不是使用SysTick



设置Timebase Source为某个定时器，如基础定时器TIM6



启用FreeRTOS后，NVIC优先级分组策略被设置为4位全部用于抢占优先级，所以抢占优先级的设置范围是0至15

- TIM6被设置为HAL基础时钟源，抢占优先级被设置为最高的0级，且不能修改。

NVIC Mode and Configuration

Configuration

NVIC | Code generation

Priority Group: 4 bits for pre-emption priority 0 bits for subpriority | Sort by Premption Priority and Sub Priority

Search: Search (Crtl+F) | Show only enabled interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority	Uses FreeRTOS functions
Non maskable interrupt	☑	0	0	☐
Hard fault interrupt	☑	0	0	☐
Memory management fault	☑	0	0	☐
Pre-fetch fault, memory access fault	☑	0	0	☐
Undefined instruction or illegal state	☑	0	0	☐
System service call via SWI instruction	☑	0	0	☐
Debug monitor	☑	0	0	☐
Pendable request for system service	☑	15	0	☑
System tick timer	☑	15	0	☑
PVD interrupt through EXTI line 16	☐	5	0	☑
Flash global interrupt	☐	5	0	☑
RCC global interrupt	☐	5	0	☑
Time base: TIM6 global interrupt, DAC1 ...	☑	0	0	☐
FPU global interrupt	☐	5	0	☑

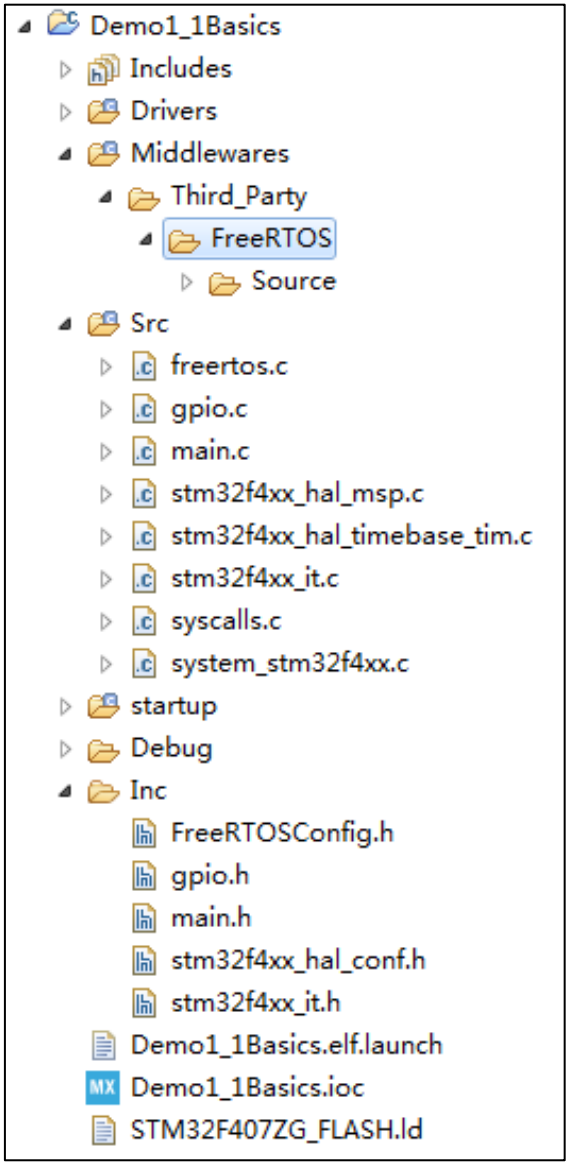
☐ Enabled Preemption Priority 5 Sub Priority 0 ☑ Uses FreeRTOS functions

- System tick timer（SysTick定时器）和Pendable request for system service（可挂起的系统服务请求）中断的抢占优先级被设置为最低的15，且都不能修改

1.2.2 含FreeRTOS的项目的文件组成

用户可修改的文件分布在Inc和Src目录下，包括：

- Inc\FreeRTOSConfig.h，这个文件是FreeRTOS的配置文件
- Src\stm32f4xx_hal_timebase_tim.c，这是设置HAL基础时钟的文件
- Src\freertos.c，这是CubeMX生成的初始化文件，在这个文件里创建任务，编写用户功能代码



1.2.3 程序分析和功能实现

1. 主程序

```
int main(void)
{
  HAL_Init();            //HAL初始化, 调用HAL_InitTick()对基础时钟进行初始化
  SystemClock_Config();                    //配置系统时钟
  MX_GPIO_Init();   //GPIO初始化，LED1和LED2两个引脚的GPIO初始化
   /* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_10, GPIO_PIN_RESET);  //LED2亮
  /* USER CODE END 2 */

  osKernelInitialize();            //CMSIS-RTOS函数，初始化FreeRTOS的调度
  MX_FREERTOS_Init();       //FreeRTOS对象初始化函数，在freertos.c中实现
  osKernelStart();                 //CMSIS-RTOS函数，启动FreeRTOS的任务调度器

    /* 程序不会运行到这里，因为RTOS的任务调度器接管了系统的控制  */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        HAL_GPIO_TogglePin(GPIOF, GPIO_PIN_10);          //LED2闪烁
        HAL_Delay(500);
    /* USER CODE END WHILE */
  }
}
```

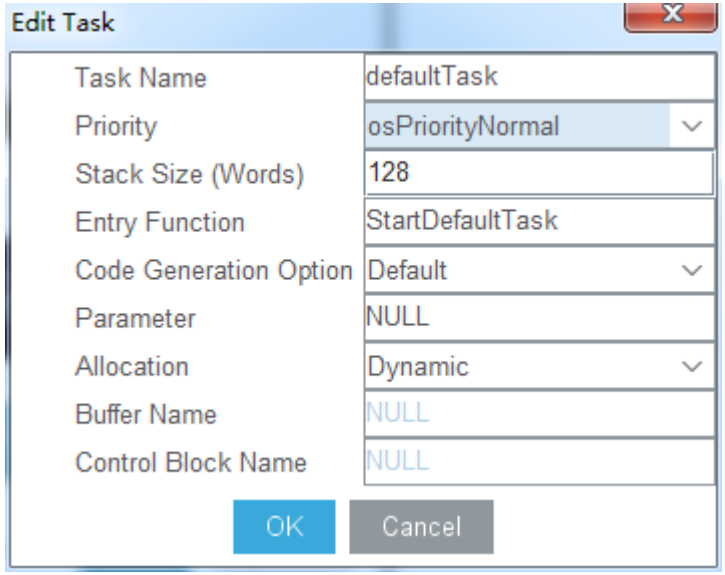
2. FreeRTOS初始化和任务创建

函数MX_FREERTOS_Init()用于对FreeRTOS初始化，创建FreeRTOS中的任务、信号量、队列等对象。

```
/*私有变量定义 ----------------------------*/
osThreadId_t defaultTaskHandle;     //任务defaultTask的句柄变量
 //任务defaultTask的属性
const osThreadAttr_t  defaultTask_attributes = {
          .name = "defaultTask",    //任务的注释名称
          .priority = (osPriority_t) osPriorityNormal,   //任务优先级
          .stack_size = 128 * 4     //栈存储空间大小，即128*4=512字节
       };

void MX_FREERTOS_Init(void) /*  FreeRTOS 初始化   */
{
        /* 创建任务defaultTask */
       defaultTaskHandle = osThreadNew( StartDefaultTask,  NULL,
                                  &defaultTask_attributes);
}
```

定义任务句柄，定义任务属性，创建任务



任务属性结构体

```
typedef struct {
  const char     *name;     //任务的名称,进作备注用途
  uint32_t        attr_bits;     //属性位
  void           *cb_mem;    //用于控制块的存储空间
  uint32_t        cb_size;        // 控制块存储空间大小
  void           *stack_mem;    //栈的存储空间
  uint32_t        stack_size;    //栈存储空间大小,单位:word=4 bytes
  osPriority_t    priority;   //任务的初始优先级，默认值：osPriorityNormal
  TZ_ModuleId_t   tz_module;   // TrustZone 模块标识符
  uint32_t        reserved;   //保留变量，必须是0
} osThreadAttr_t;
```

若设置Allocation为Static，需要静态定义Buffer Name（缓冲区名称）为和Control Block Name（控制块名称）

```
osThreadId_t  defaultTaskHandle;       //任务句柄变量
uint32_t  defaultTaskBuffer[ 128 ];       //栈空间数组
osStaticThreadDef_t  defaultTaskControlBlock;   //控制块结构体变量
//任务属性赋值
const osThreadAttr_t  defaultTask_attributes = {
    .name = "defaultTask",       //任务名称
    .stack_mem = &defaultTaskBuffer[0],         //栈数组地址
    .stack_size = sizeof(defaultTaskBuffer),      //栈数组大小
    .cb_mem = &defaultTaskControlBlock,       //控制块地址
    .cb_size = sizeof(defaultTaskControlBlock),  //控制块大小
    .priority = (osPriority_t) osPriorityNormal,      //任务优先级
};
```

3. 编写任务功能实现代码

- 任务函数的主体就是一个死循环
- osDelay()是毫秒级延时函数，内部调用vTaskDelay()，延时单位是节拍（tick）
- 执行延时函数vTaskDelay()时就会交出处理器的使用权，由FreeRTOS进行任务调度

```
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN StartDefaultTask */
  /* Infinite loop */
  for(;;)
  {
        HAL_GPIO_TogglePin(GPIOF, GPIO_PIN_9);    //LED1闪烁
        osDelay(500);          //延时500个ticks（时钟节拍）
  }
  /* USER CODE END StartDefaultTask */
}
```

4. 运行效果

运行时会发现LED1会闪烁，而LED2只是被点亮。说明任务函数StartDefaultTask()被执行了，而main()函数中后面的while()死循环里的代码没有被执行。

```
int main(void)
{
  osKernelInitialize();          //CMSIS-RTOS函数，初始化FreeRTOS的调度
  MX_FREERTOS_Init();            //FreeRTOS对象初始化函数，在freertos.c中实现
  osKernelStart();               //CMSIS-RTOS函数，启动FreeRTOS的任务调度器

    /* 程序不会运行到这里，因为RTOS的任务调度器接管了系统的控制  */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        HAL_GPIO_TogglePin(GPIOF, GPIO_PIN_10);          //LED2闪烁
        HAL_Delay(500);
    /* USER CODE END WHILE */
  }
}
```

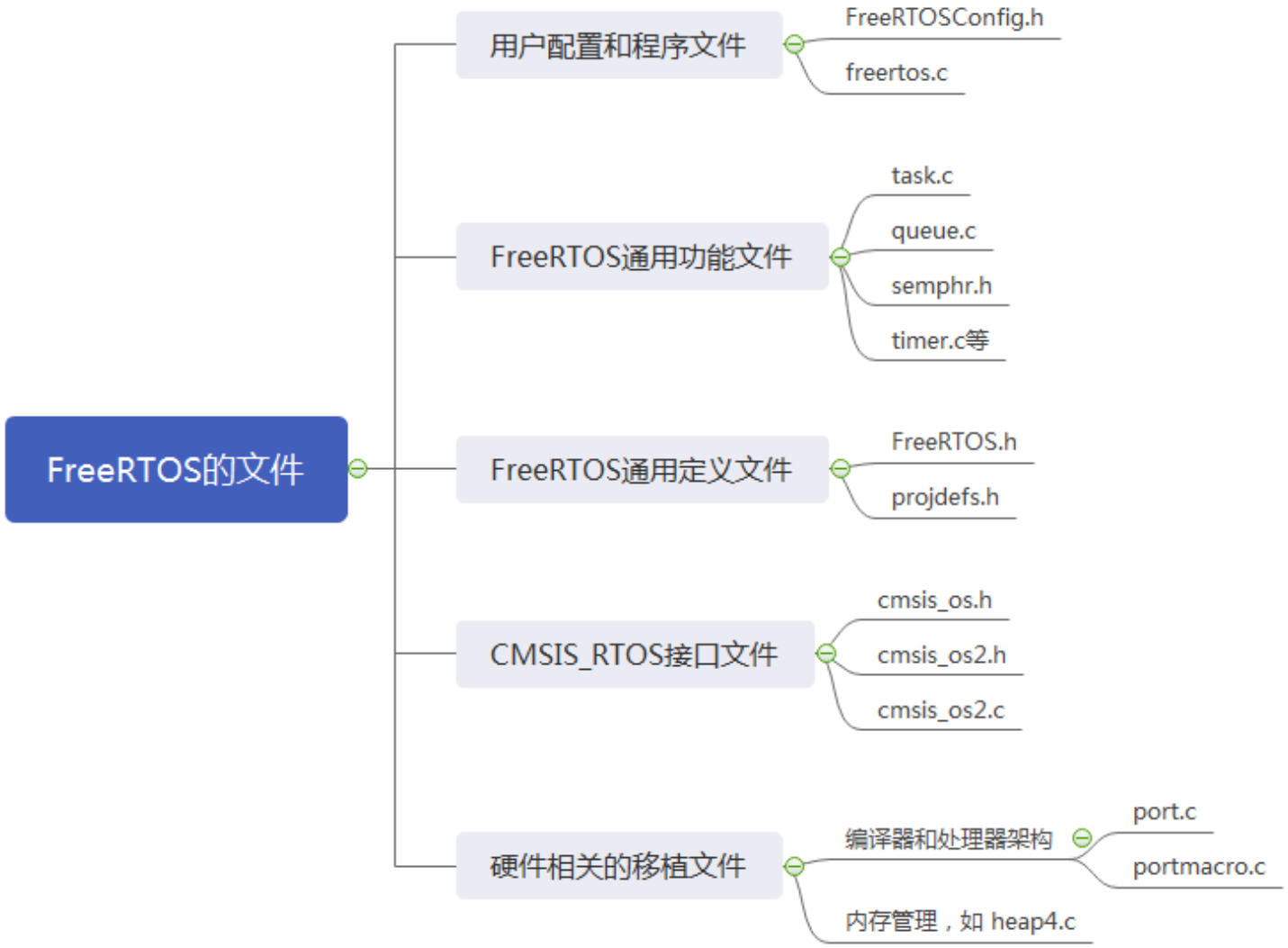
1.3 FreeRTOS的文件组成和基本原理

1.3.1 FreeRTOS的文件组成

1.3.2 FreeRTOS的编码规则

1.3.3 FreeRTOS的配置和功能裁剪

1.3.1 FreeRTOS的文件组成



1. 用户配置和程序文件

包括3个文件，用于对FreeRTOS进行各种配置和功能裁剪，以及实现用户任务的功能。

- FreeRTOSConfig.h是对FreeRTOS进行各种配置的文件，FreeRTOS的功能裁剪就是通过这个文件里的各种宏定义参数的设置实现的。
- stm32f4xx_hal_timebase_tim.c是使用指定的定时器作为基础时钟信号的程序文件。
- freertos.c是进行FreeRTOS初始化，创建用户任务，以及实现用户任务函数的程序文件。

2. FreeRTOS通用功能文件

这些是实现FreeRTOS的任务、队列、信号量、软件定时器、事件组等通用功能的文件，这些功能与硬件无关。

源程序文件在Source目录下，头文件在Source/Include目录下

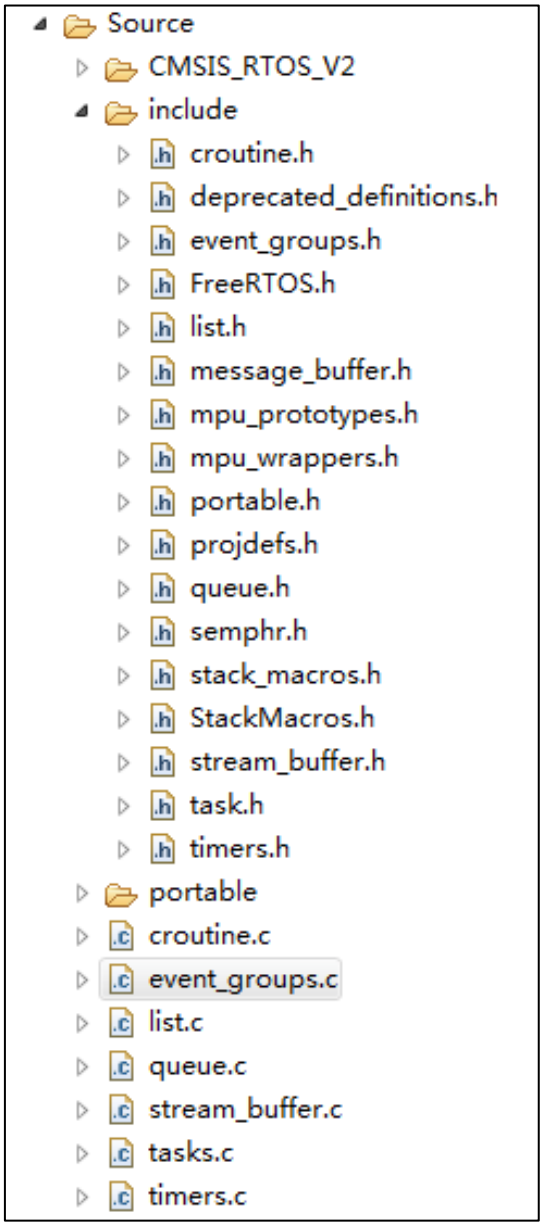


表1-1 FreeRTOS的通用功能程序文件

文件	功能
croutine.h/.c	实现协程（co-routine）功能的程序文件，协程主要用于非常小的MCU，现在已经很少使用
event_groups.h/.c	实现事件组功能的程序文件
list.h/.c	实现链表功能的程序文件，FreeRTOS的任务调度器用到链表
queue.h/.c	实现队列功能的程序文件
semphr.h	实现信号量功能的文件，信号量是基于队列的，信号量操作的函数都是宏定义函数，其实现都是调用队列处理的函数
task.h **tasks.c**	实现任务管理功能的程序文件
timers.h/.c	实现软件定时器功能的程序文件
stream_buffer.h/.c	实现流缓存功能的程序文件。流缓存是一种优化的进程间通讯机制，用于在任务与任务之间、任务与中断服务函数之间传输连续的的流数据。流缓存功能是在FreeRTOS 10版本中才引入的功能
message_buffer.h	实现消息缓存功能的文件。实现消息缓存功能的所有函数都是宏定义函数，因为消息缓存是基于流缓存实现的，都调用流缓存的函数。消息缓存功能是在FreeRTOS 10版本中才引入的功能
mpu_prototypes.h **mpu_wrappers.h**	MPU（消息处理单元）功能的头文件。该文件中定义的函数就是在标准函数前面增加前缀“MPU_”，当应用程序使用MPU功能时，此文件中的函数被FreeRTOS内核优先执行。MPU功能是在FreeRTOS 10版本中才引入的功能

3. FreeRTOS通用定义文件

Source/include目录下有几个与硬件无关的通用定义文件

（1）文件FreeRTOS.h

包含FreeRTOS的一些缺省的宏定义、数据类型定义、接口函数定义。FreeRTOS.h中有用于FreeRTOS功能裁剪的一些缺省的宏定义，例如

```
#ifndef configIDLE_SHOULD_YIELD
    #define configIDLE_SHOULD_YIELD    1
#endif

#ifndef INCLUDE_vTaskDelete
    #define INCLUDE_vTaskDelete 0
#endif
```

两种前缀的宏定义：“config”和“INCLUDE_”

（2）文件projdefs.h

这个文件包含FreeRTOS中的一些通用定义，如错误编号宏定义，逻辑值的宏定义等，文件projdefs.h中常用到的几个宏定义见表1-2。

宏定义	值	功能
pdFALSE	0	表示逻辑值false
pdTRUE	1	表示逻辑值true
pdFAIL	0	表示逻辑值false
pdPASS	1	表示逻辑值true
pdMS_TO_TICKS (xTimeInMs)		这是个宏函数，其功能是将xTimeInMs表示的毫秒数转换为时钟节拍数

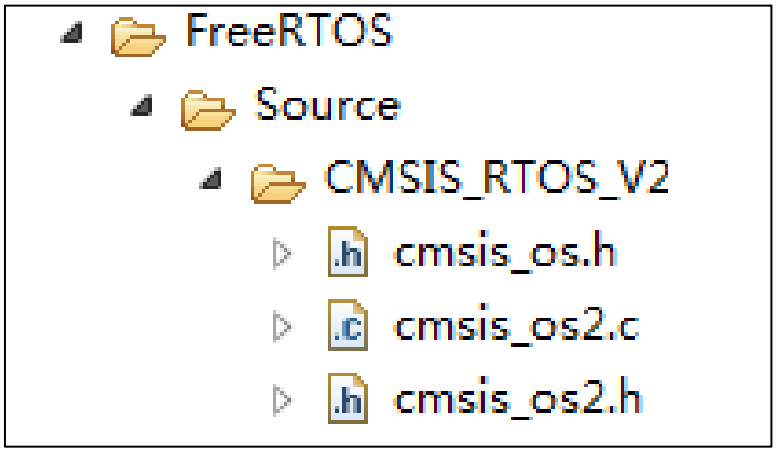
（3）文件stack_macros.h和StackMacros.h

这两个文件的内容完全一样，只是为了向后兼容才出现了两个文件。

这个文件定义了进行栈溢出检查的函数，如果要使用栈溢出检查功能，宏configCHECK_FOR_STACK_OVERFLOW的值需要设置为1或2

4. CMSIS RTOS标准接口文件

Source/CMSIS_RTOS_V2目录下是CMSIS RTOS标准接口文件，这些文件里宏定义、数据类型、函数名称的前缀都是“os”



这些函数和数据类型的名称是与具体的RTOS无关的，它们是CMSIS RTOS标准的定义。在具体实现上，这些前缀为“os”的函数调用具体移植的RTOS的实现函数，例如，若移植的是FreeRTOS，“os”函数就调用FreeRTOS的实现函数。

对于FreeRTOS，这些“os”函数调用的就都是FreeRTOS的函数。例如CMSIS RTOS的延时函数osDelay()内部就是调用了FreeRTOS的延时函数vTaskDelay()，其完整源代码如下

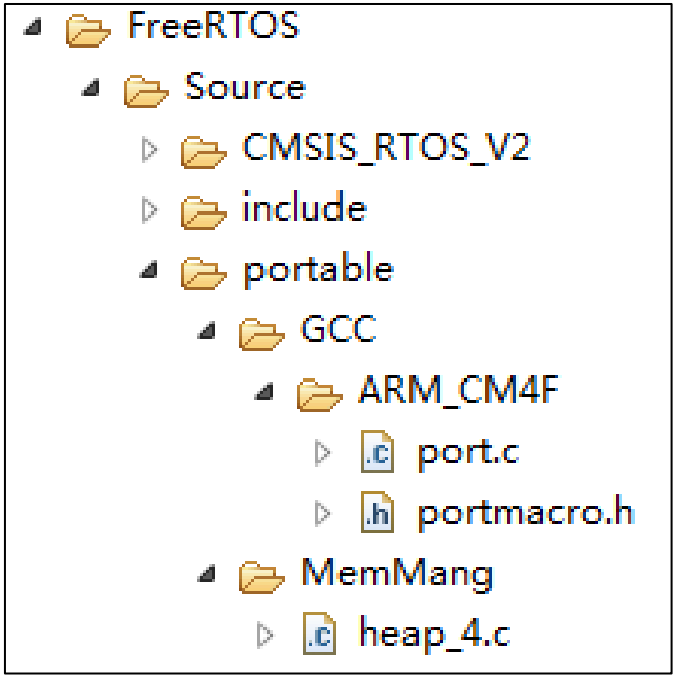
```
osStatus_t osDelay (uint32_t ticks)
{
  osStatus_t stat;
  if (IS_IRQ()) {
    stat = osErrorISR;
  }
  else {
    stat = osOK;

    if (ticks != 0U) {
      vTaskDelay(ticks);
    }
  }

  return (stat);
}
```

5. 硬件相关的移植文件

硬件相关的移植文件就是需要根据硬件类型进行改写的文件，这些文件在Source/portable目录下，又分为架构与编译器、内存管理两个部分



（1）处理器架构和编译器相关文件

有2个文件：portmacro.h和port.c。这两个文件里是一些与硬件相关的基础数据类型、宏定义和函数定义。由于某些函数的功能实现涉及底层操作，其实现代码甚至是用汇编语言写的，所以与硬件密切相关。

文件portmacro.h中对一些基础数据类型重新定义了类型符号，定义的代码如下。Cortex-M4F是32位处理器

```
#define portCHAR                char                //int8_t
#define portFLOAT               float               //4字节浮点数
#define portDOUBLE              double              //8字节浮点数
#define portLONG                long                //int32_t
#define portSHORT               short               //int16_t
#define portSTACK_TYPE          uint32_t            //栈数据类型
#define portBASE_TYPE           long                //int32_t

typedef portSTACK_TYPE StackType_t;  //栈数据类型StackType_t，是uint32_t
typedef long BaseType_t;             //基础数据类型BaseType_t，是int32_t
typedef unsigned long UBaseType_t;   //基础数据类型UBaseType_t,是uint32_t
typedef uint32_t TickType_t;         //节拍数类型TickType_t,是uint32_t
```

重新定义的4个类型符号是为了移植方便，经常用到

（2）内存管理相关文件

内存管理涉及内存动态分配和释放等操作，与具体的处理器密切相关。FreeRTOS提供5种内存管理方案，即heap_1至heap_5，在CubeMX里设置FreeRTOS参数时选择一种。在图1-11中的内存管理文件是heap_4.c，这也是缺省的内存管理方案 。

1.3.2 FreeRTOS的编码规则

1. 变量名

- 对于stdint.h中定义的各种标准类型整数，前缀“c”表示char类型变量，前缀“s”表示int16_t（short）类型变量，前缀“l”表示int32_t类型变量。对于无符号（unsigned）整数，再在前面增加前缀“u”，如“uc”表示uint_8类型，“us”表示uint16_t，“ul”表示uint32_t类型
- BaseType_t和所有其他非标准类型的变量名，如结构体变量、任务句柄、队列句柄等都用前缀“x”
- UBaseType_t类型的变量使用前缀“ux”
- 指针类型变量在前面再增加一个“p”如“pc”表示 char * 类型

2. 函数名

函数名的前缀由返回值类型和函数所在文件组成，若返回值为void类型，则类型前缀是“v”。举例如下：

- xTaskCreate()，返回值为BaseType_t类型，在文件task.h中定义
- vQueueDelete()，返回值为void，在queue.h文件中定义
- pcTimerGetName()，返回值为char *，在timer.h中定义
- pvPortMalloc()，返回值为void *，在文件portable.h中定义

CMSIS RTOS相关文件中定义的函数前缀都是“os”，不包括返回值类型和所在文件的前缀。例如，cmsis_os2.h中的函数osThreadNew()。

3. 宏名称

宏定义和宏函数的名称一般用大写字母，并使用小写字母前缀表示宏定义的功能分组。

前缀	意义	所在文件	举例
config	用于系统功能配置的宏	FreeRTOSConfig.h FreeRTOS.h	configUSE_MUTEXES
INCLUDE_	条件编译某个函数的宏	FreeRTOSConfig.h FreeRTOS.h	INCLUDE_vTaskDelay
task	任务相关的宏	task.h task.c	taskENTER_CRITICAL() taskWAITING_NOTIFICATION
queue	队列相关的宏	queue.h	queueQUEUE_TYPE_MUTEX
pd	项目通用定义的宏	projdefs.h	pdTRUE，pdFALSE
port	移植接口文件定义的宏	portable.h portmacro.h port.c	portBYTE_ALIGNMENT_MASK portCHAR portMAX_24_BIT_NUMBER
tmr	软件定时器相关的宏	timer.h	tmrCOMMAND_START
os	CMSIS RTOS接口相关的宏	cmsis_os.h cmsis_os2.h	osFeature_SysTick osFlagsWaitAll

1.3.3 FreeRTOS的配置和功能裁剪

- FreeRTOS的配置和功能裁剪主要是通过文件FeeRTOSConfig.h和FreeRTOS.h中的一些宏定义实现的
- 前缀为“config”的宏配置FreeRTOS的一些参数
- 前缀为“INCLUDE_”的宏控制是否编译某些函数的源代码
- FreeRTOS.h中的宏定义是系统默认的宏定义，不要直接修改
- FeeRTOSConfig.h是用户可修改的配置文件，如果一个宏没有在文件FeeRTOSConfig.h中重新定义，就使用文件FreeRTOS.h中的默认定义

1. “config”类的宏

前缀为“config”的宏用于对FreeRTOS的一些参数进行配置。

例如，文件FeeRTOSConfig.h中部分这类宏定义代码如下

```
#define configUSE_PREEMPTION                    1
#define configSUPPORT_STATIC_ALLOCATION         1
#define configSUPPORT_DYNAMIC_ALLOCATION        1
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( SystemCoreClock )
#define configTICK_RATE_HZ                      ((TickType_t)1000)
#define configMAX_PRIORITIES                    ( 56 )
#define configMINIMAL_STACK_SIZE                ((uint16_t)128)
#define configTOTAL_HEAP_SIZE                   ((size_t)15360)
#define configMAX_TASK_NAME_LEN                 ( 16 )

/* Software timer definitions. */
#define configUSE_TIMERS                        1
#define configTIMER_TASK_PRIORITY               ( 2 )
```

在CubeMX中修改了值的参数都会在文件FreeRTOSConfig.h中生成语句。有一些默认值的宏定义在文件FreeRTOS.h中，例如文件FreeRTOS.h中有如下的定义。

```
#ifndef configIDLE_SHOULD_YIELD
    #define configIDLE_SHOULD_YIELD                1
#endif
```

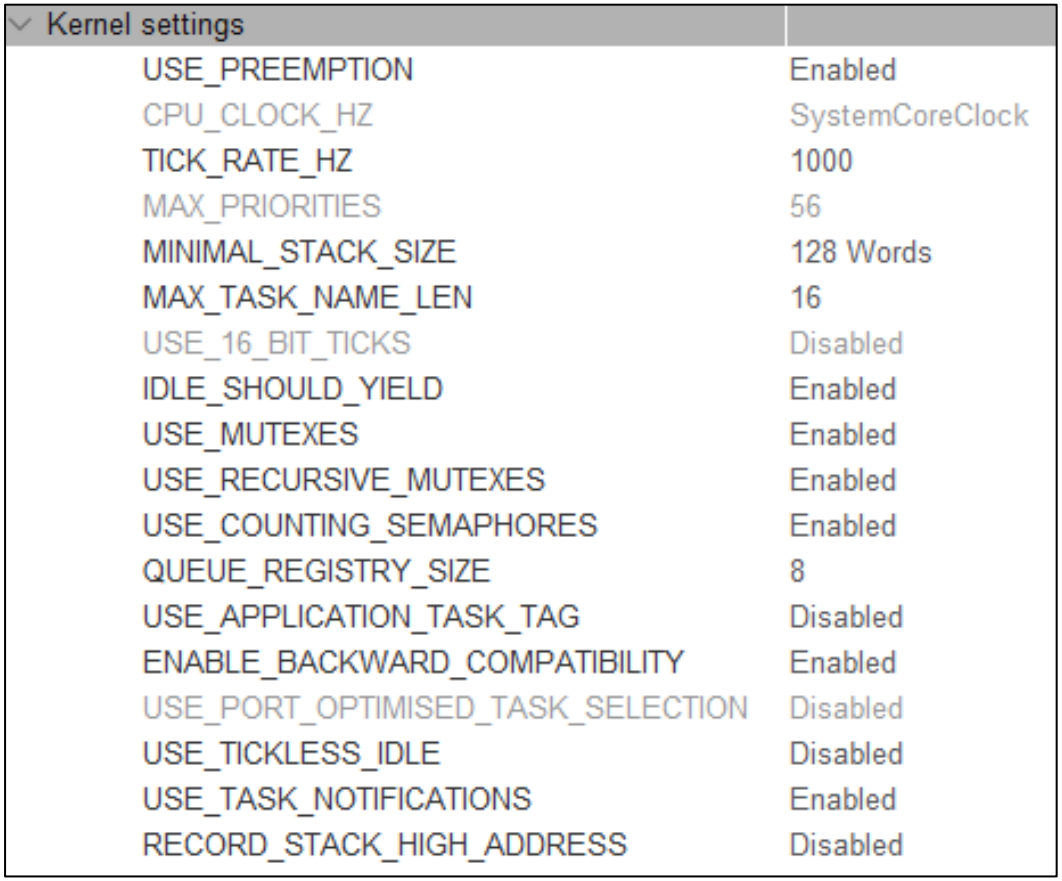
默认情况下，文件FreeRTOSConfig.h中没有定义宏configIDLE_SHOULD_YIELD ，那么就使用文件FreeRTOS.h中的默认定义。如果通过CubeMX修改了这个参数，在FreeRTOSConfig.h中生成了如下的一条语句，就使用FreeRTOSConfig.h中的定义。

```
#define configIDLE_SHOULD_YIELD                0
```

（1）Kernel settings，内核设置

某些参数是不允许修改的，就显示为灰色字体。某些参数只能选择一个参数值

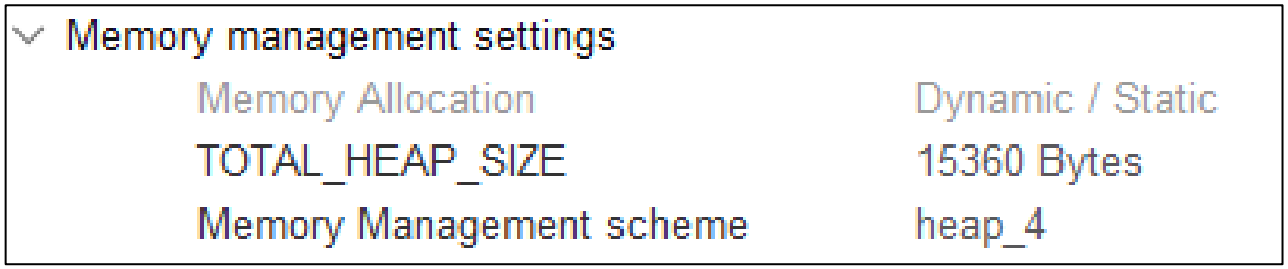
逻辑型参数的可选值是Enabled和Disabled，对应于宏定义的值是1和0



Kernel settings	
USE_PREEMPTION	Enabled
CPU_CLOCK_HZ	SystemCoreClock
TICK_RATE_HZ	1000
MAX_PRIORITIES	56
MINIMAL_STACK_SIZE	128 Words
MAX_TASK_NAME_LEN	16
USE_16_BIT_TICKS	Disabled
IDLE_SHOULD_YIELD	Enabled
USE_MUTEXES	Enabled
USE_RECURSIVE_MUTEXES	Enabled
USE_COUNTING_SEMAPHORES	Enabled
QUEUE_REGISTRY_SIZE	8
USE_APPLICATION_TASK_TAG	Disabled
ENABLE_BACKWARD_COMPATIBILITY	Enabled
USE_PORT_OPTIMISED_TASK_SELECTION	Disabled
USE_TICKLESS_IDLE	Disabled
USE_TASK_NOTIFICATIONS	Enabled
RECORD_STACK_HIGH_ADDRESS	Disabled

每个参数的解释见表1-5

（2）Memory management settings，内存管理设置



- Memory Allocation，内存分配方式，固定为Dynamic/Static
- TOTAL_HEAP_SIZE，FreeRTOS总的堆空间大小，设置范围512至128K字节
- Memory Management Scheme，内存管理方案。有5种可选的内存管理方案，从heap_1到heap_5，默认使用heap_4

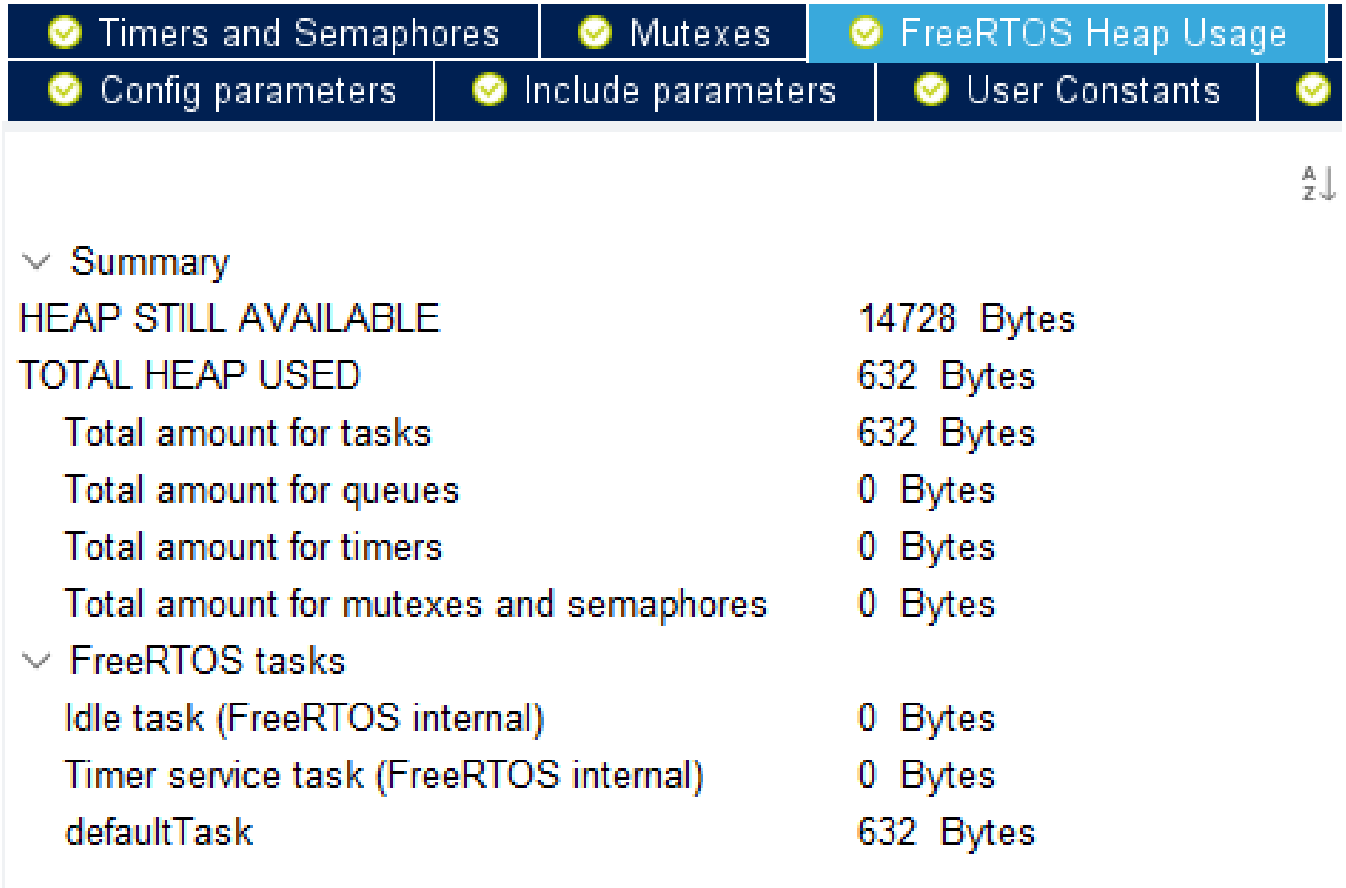


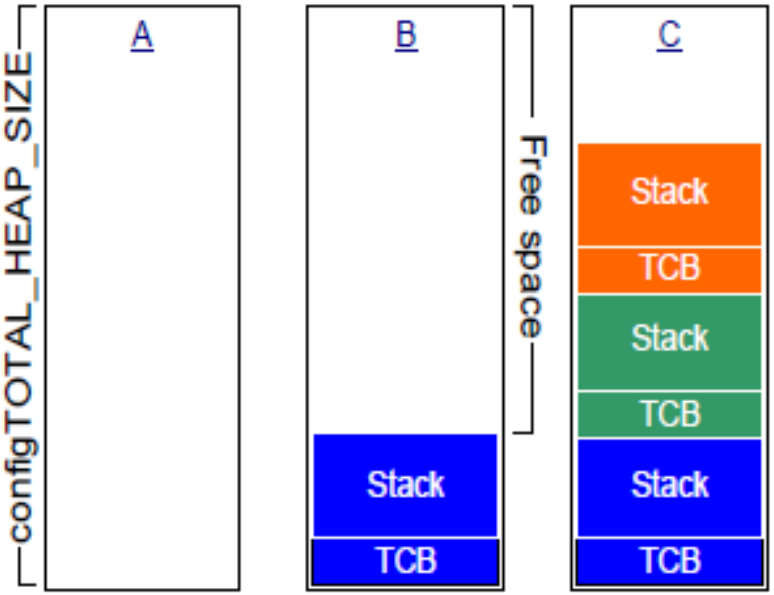
图23-15 FreeRTOS Heap Usage页面显示内存使用信息

FreeRTOS的5种内存管理方案各有特点和适用场合，可参考《Mastering the FreeRTO Real Time Kernel》的第2章

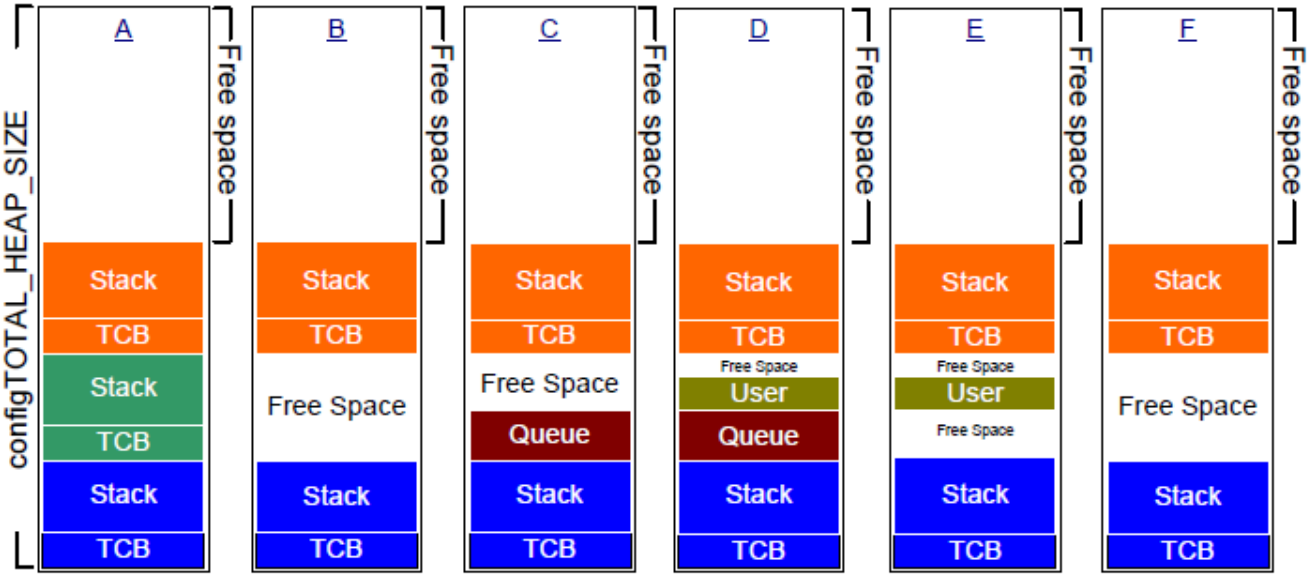
◆ 方案heap_1

方案heap_1只有分配内存的函数pvPortMalloc()，没有释放内存的函数vPortFree()。

先分配一个固定大小的堆（heap），也就是一个固定长度的数组，然后给每个任务分配内存。每个任务占用的内存包括一个栈（stack）和一个任务控制块（task control block, TCB）



◆ 方案heap_4

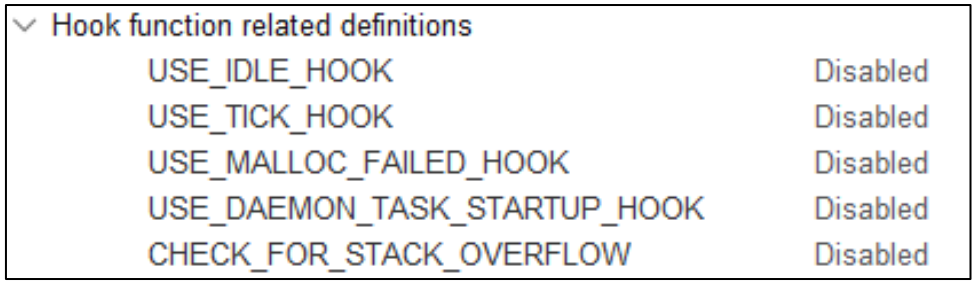


方案heap_4使用第一匹配（first fit）算法分配内存，能将紧邻的空白内存块整合成一个大的空白内存块，可以减少产生内存碎片的风险。

方案heap_4速度比标准库中的函数malloc()和free()要快，是FreeRTOS默认采用的内存管理方案

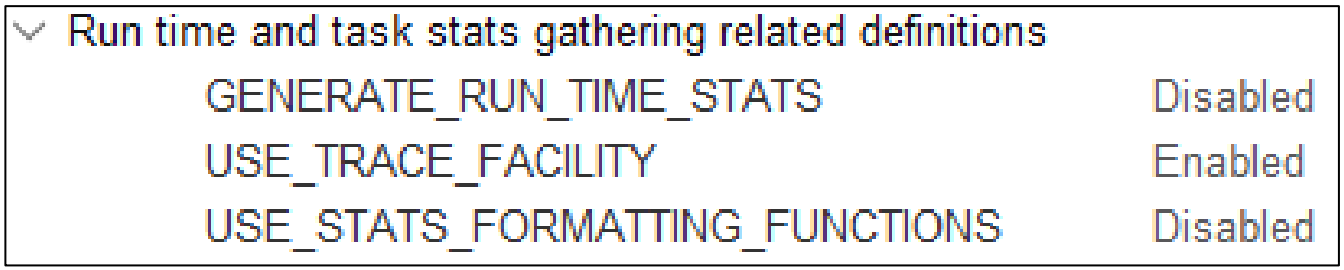
（3）Hook function related definitions，钩子函数相关定义

钩子函数类似于回调函数。如果设置为Enabled，在文件freertos.c中会自动生成相应钩子函数的代码框架。



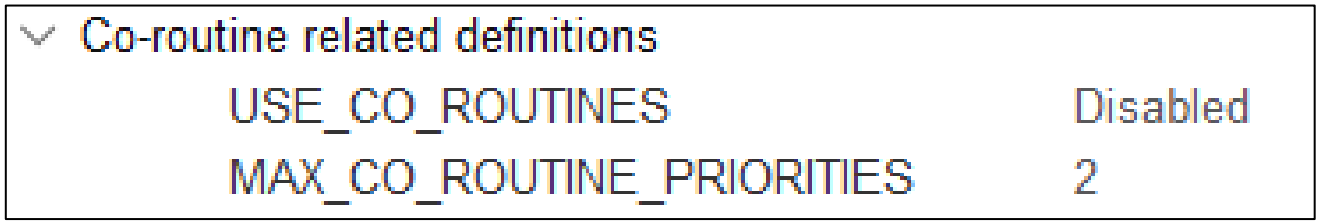
钩子函数配置	调用场合	对应的钩子函数名称
USE_IDLE_HOOK	空闲任务里调用	vApplicationIdleHook()
USE_TICK_HOOK	嘀嗒定时器中断服务函数里调用	vApplicationTickHook()
USE_MALLOC_FAILED_HOOK	使用pvPortMalloc()分配内存失败时调用	vApplicationMallocFailedHook()
USE_DAEMON_TASK_STARTUP_HOOK	守护（Daemon）任务启动时调用	vApplicationDaemonTaskStartupHook()
CHECK_FOR_STACK_OVERFLOW	栈溢出时调用	vApplicationStackOverflowHook()

（4）Run time and task stats gathering related definitions
运行时间和任务状态收集相关定义



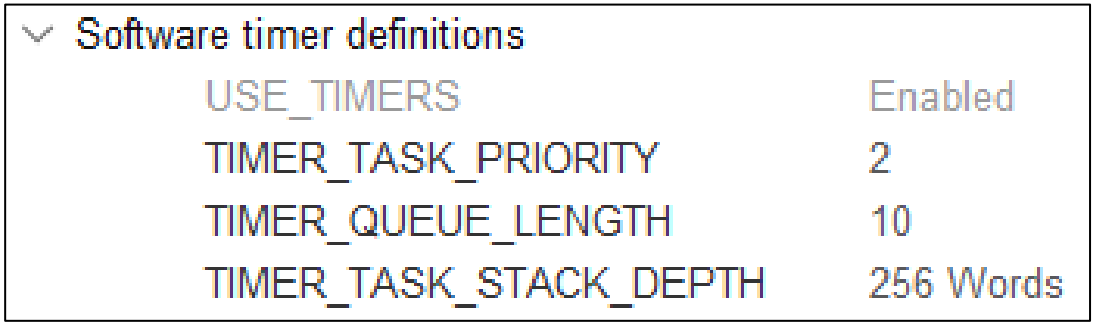
FreeRTOS可以收集任务运行时间和任务状态信息

（5）Co-routine related definitions，协程相关定义



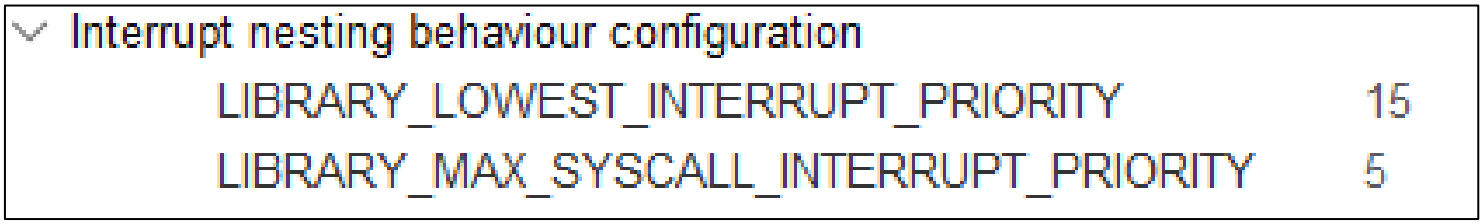
使用协程可以节省内存，主要用于功能有限、内存很小的MCU。现在的MCU内存一般比较充足，就很少使用协程了，所以禁用此功能即可。

（6）Software timer definitions，软件定时器定义



FreeRTOS可以创建软件定时器，其功能类似于高级语言（如C++）中的软件定时器。

（7）Interrupt nesting behaviour configuration，中断嵌套行为配置



- LIBRARY_LOWEST_INTERRUPT_PRIORITY，最低中断优先级，设置范围1到15，默认值是15。
- LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY，系统能管理的最高中断优先级，设置范围1到15，默认值5。

2. “INCLUDE_”类的宏

前缀为“INCLUDE_”的宏用作一些函数的条件编译的条件，若宏的值为1就编译相应的函数，否则就不编译相应的函数，以此可以实现对FreeRTOS的功能裁剪。

文件FeeRTOSConfig.h中部分这类宏定义代码如下。

```
#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           1
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskCleanUpResources       0
#define INCLUDE_vTaskSuspend                1
#define INCLUDE_vTaskDelayUntil             1
#define INCLUDE_vTaskDelay                  1
#define INCLUDE_xTaskGetSchedulerState      1
#define INCLUDE_xTimerPendFunctionCall      1
```

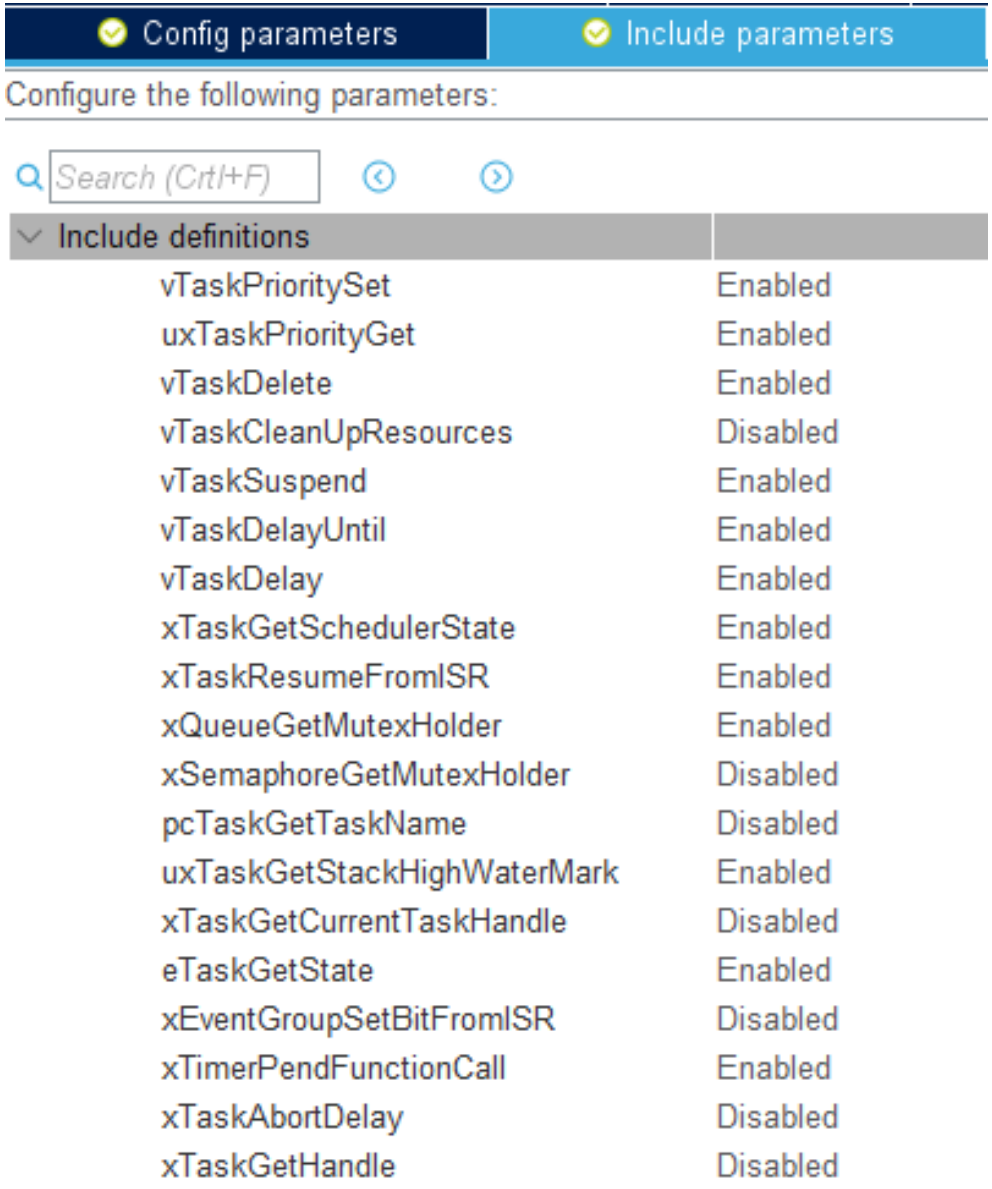


图1-24 Include parameters设置界面

参考资料

- FreeRTOS官方文档，Mastering the FreeRTOS™ Real Time Kernel
- FreeRTOS官方文档，The FreeRTOS™ Reference Manual
- FreeRTOS官方网站，www.freertos.org