FreeRTOS 学习之七：软定时器

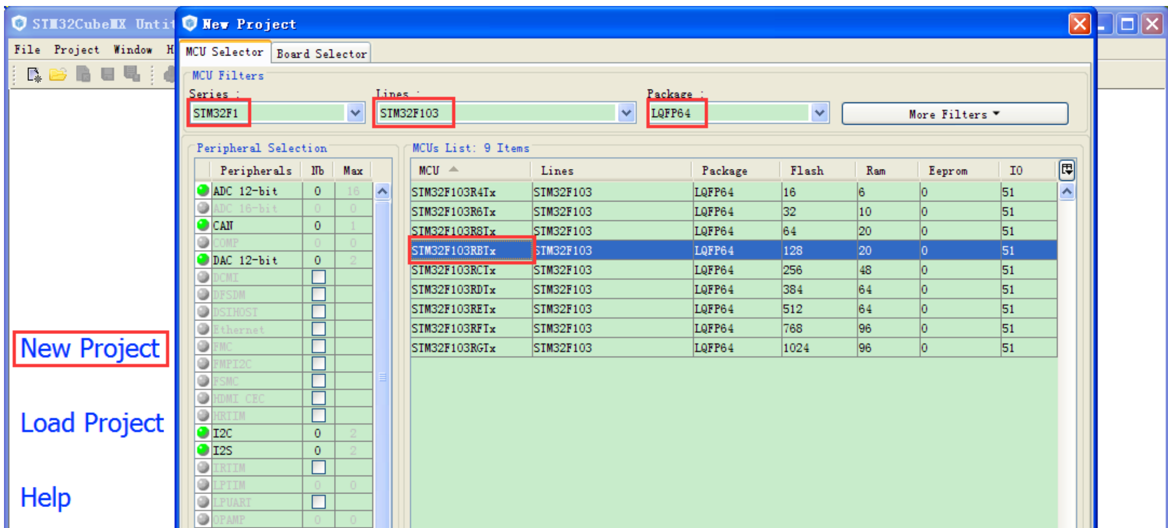
前提：默认已经装好 MDK V5 和 STM32CubeMX，并安装了 STM32F1xx 系列的支持包。

硬件平台：STM32F1xx 系列。

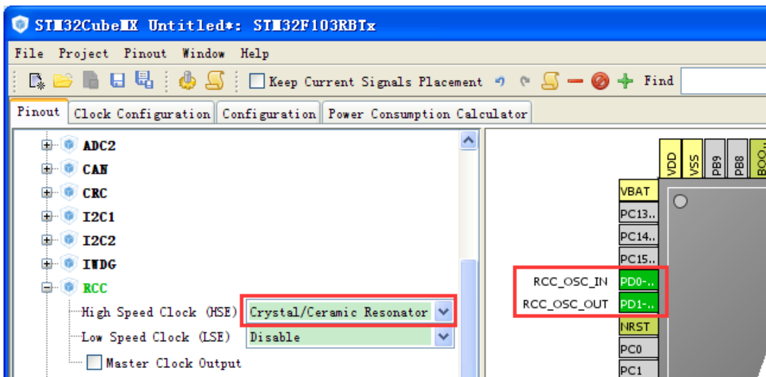
目的：学习软定时器的使用。

有时候我们需要定时进行一些例行处理，FreeRTOS 提供了软定时器来满足这种需求。软定时器不是 FreeRTOS 内核的组成部分，它本质上是一种任务，周期性地调用其回调函数。本文例子使用 STM32CubeMX 配置创建一个软定时器和一个任务，软定时器在回调函数控制 LED0 的状态，任务控制 LED1 的状态。

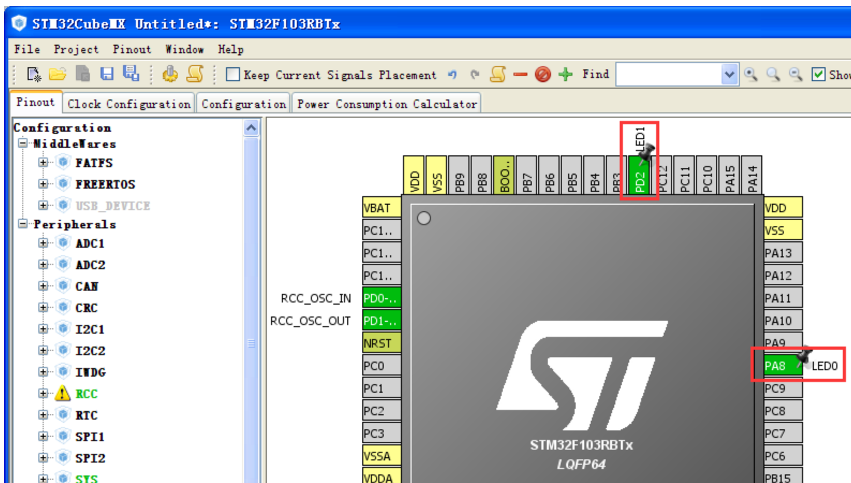
Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F103RBTx。



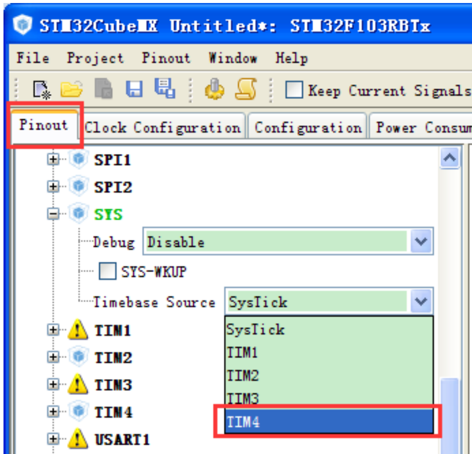
Step2.配置时钟引脚。



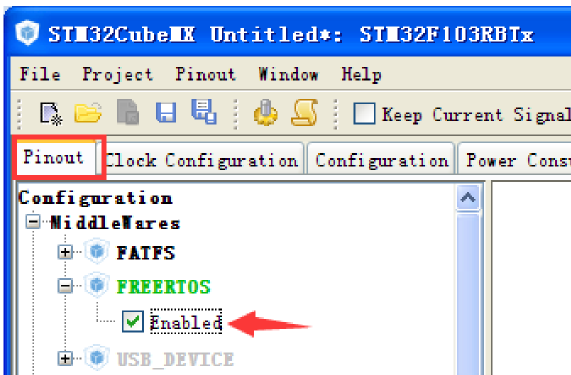
Step3.配置 PA8 和 PD2 为 Output，并把用户标签分别改为 LED0，LED1。



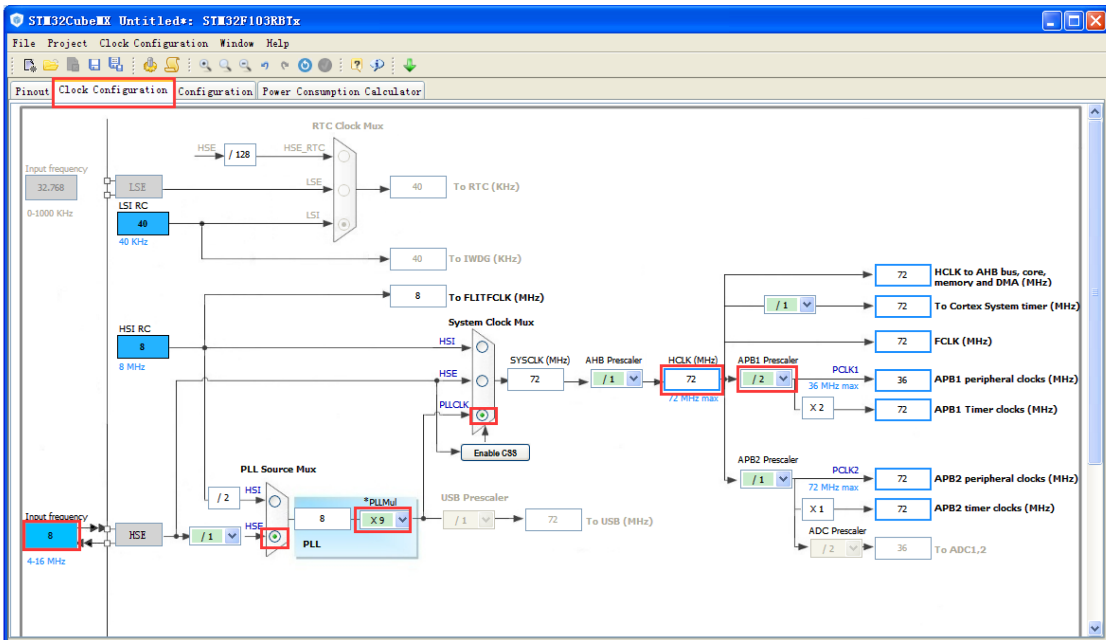
Step4.将系统时基源改为 TIM4。



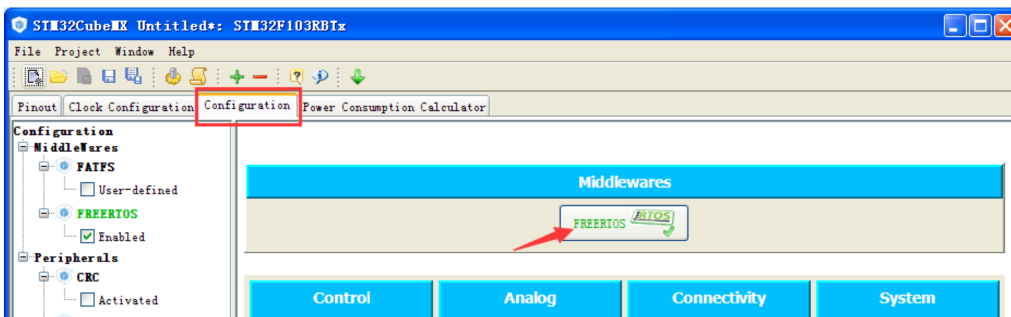
Step5.使能 FreeRTOS。



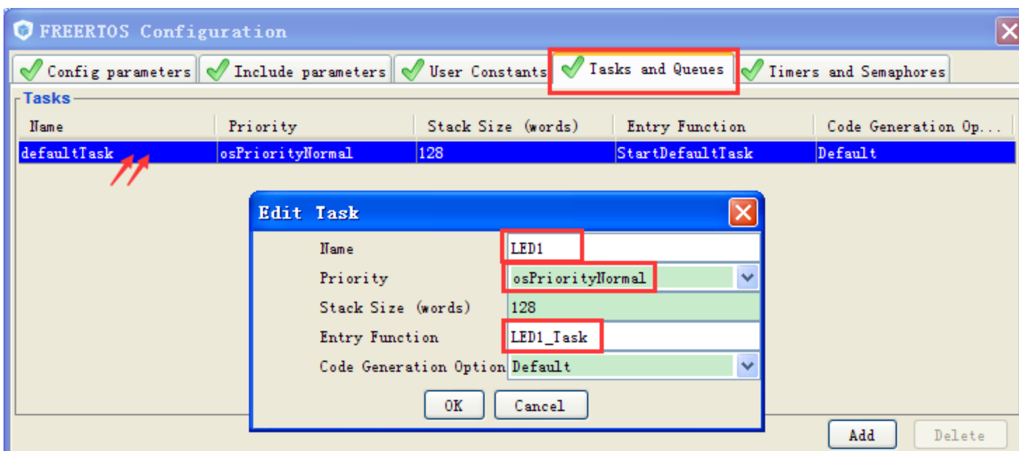
Step6.配置时钟树。8M 输入时，通过 PLL 得到 72M 内部时钟。



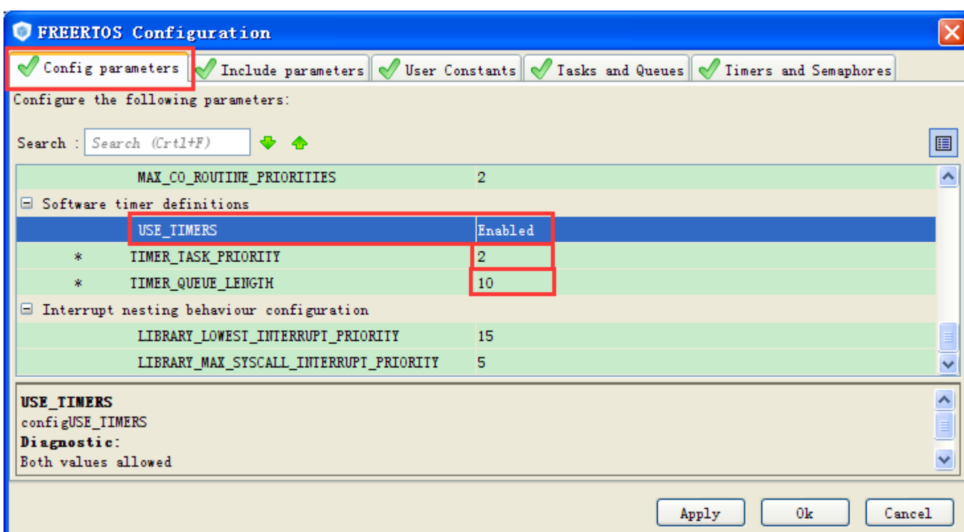
Step7.配置 FreeRTOS。



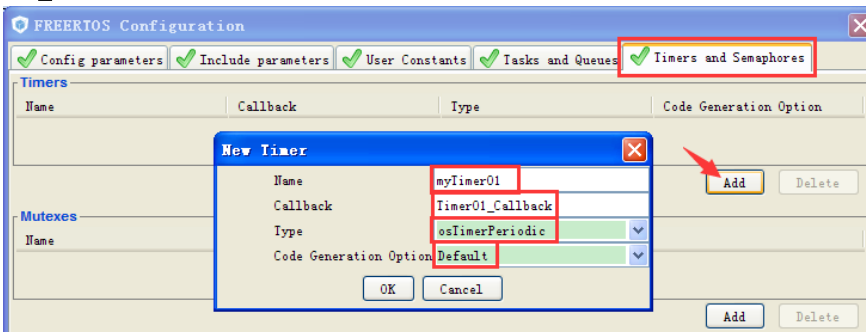
在 Tasks and Queues 选项卡中，默认配置了一个名为 defaultTask 的任务，其优先级为普通，任务堆栈大小为 128 字，任务函数名为 StartDefaultTask。
双击蓝色的地方，弹出对话框，将任务名修改为 LED1，将任务函数名修改为 LED1_Task。



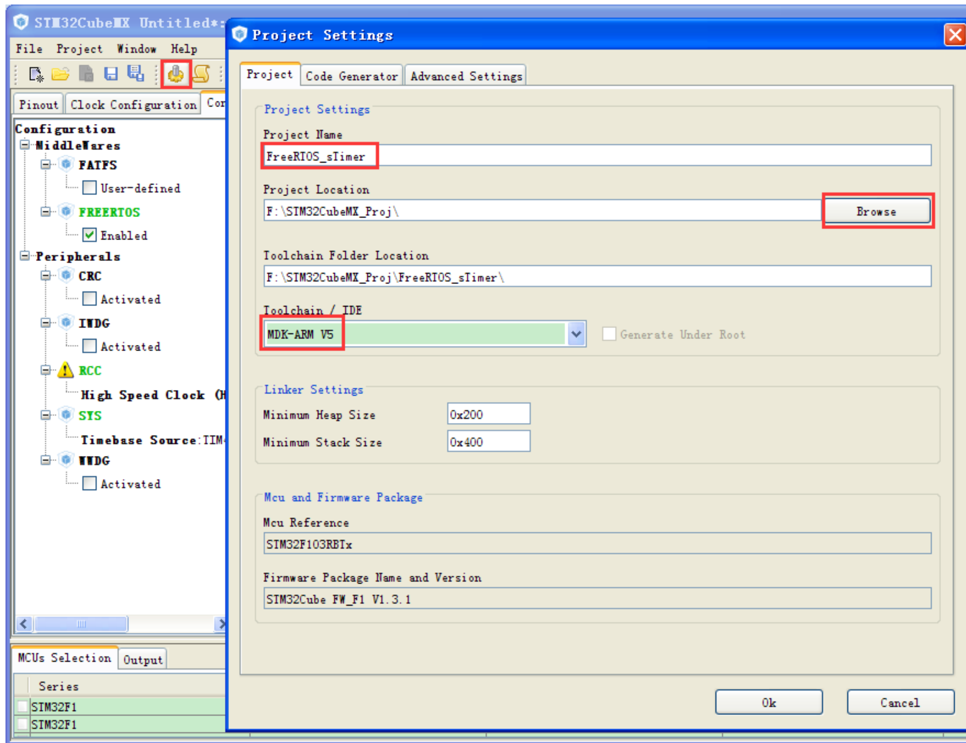
在 Configure parameters 选项卡，使能软定时器，定时器任务的优先级默认为 2，定时器队列深度默认为 10。(后两个参数在后文有详细说明。)



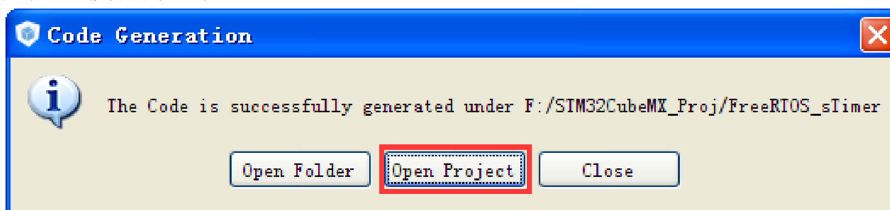
在 Timers and Semaphores 选项卡，点击 Timers 项右边的“Add”按钮，添加一个软定时器，函数改为 Timer01_Callback，类型为周期定时器。



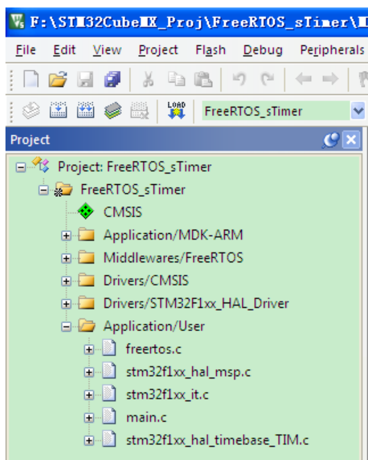
注：其他的都使用默认参数。
Step8.生成代码。



等完成后直接打开工程。



工程基本组织结构如下图，其中 Application/User 组中的文件是用户可以修改的，而其他组中的文件一般不进行修改。



Step9.分析程序结构。

在进入 main 函数之前，先定义了几个变量，声明了几个函数。

```
41 /* Private variables -------------------------
42 osThreadId LED1Handle;
43 osTimerId myTimer01Handle;
44
45 /* USER CODE BEGIN PV */
46 /* Private variables -------------------------
47
48 /* USER CODE END PV */
49
50 /* Private function prototypes ----------------
51 void SystemClock_Config(void);
52 static void MX_GPIO_Init(void);
53 void LED1_Task(void const * argument);
54 void Timer01_Callback(void const * argument);
```

再看 main 函数。将 main 函数整理，删除很多注释之后，得到下图所示内容。

```
65  int main(void)
66  {
67    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
68    HAL_Init();
69    /* Configure the system clock */                ①
70    SystemClock_Config();
71    /* Initialize all configured peripherals */
72    MX_GPIO_Init();
73
74    /* Create the timer(s) */
75    /* definition and creation of myTimer01 */
76    osTimerDef(myTimer01, Timer01_Callback);
77    myTimer01Handle = osTimerCreate(osTimer(myTimer01), osTimerPeriodic, NULL);
78
79    /* USER CODE BEGIN RTOS_TIMERS */
80    /* start timers, add new ones, ... */                              ②
81    /* USER CODE END RTOS_TIMERS */
82
83    /* Create the thread(s) */
84    /* definition and creation of LED1 */
85    osThreadDef(LED1, LED1_Task, osPriorityNormal, 0, 128);
86    LED1Handle = osThreadCreate(osThread(LED1), NULL);
87
88    /* Start scheduler */ ③
89    osKernelStart();
90
91    /* We should never get here as control is now taken by the scheduler */
92
93    /* Infinite loop */
94    /* USER CODE BEGIN WHILE */
95    while (1)
96    {
97    }
98  }
```

其中第①部分，是硬件配置；第②部分，创建一个软定时器和一个任务；第③部分，启动调度器。

Step10.添加代码。

在 main.c 文件中，在任务函数 LED1_Task 添加代码如下。

```
172  /* LED1_Task function */
173  void LED1_Task(void const * argument)
174  {
175
176    /* USER CODE BEGIN 5 */
177    /* Infinite loop */
178    for(;;)
179    {
180        osDelay(500);
181        HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
182    }
183    /* USER CODE END 5 */
184  }
```

在软定时器回调函数中添加代码如下。

```
186  /* Timer01_Callback function */
187  void Timer01_Callback(void const * argument)
188  {
189    /* USER CODE BEGIN Timer01_Callback */
190    HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
191    /* USER CODE END Timer01_Callback */
192  }
```

在 main 函数的 /* USER CODE BEGIN RTOS_TIMERS */和/* USER CODE END RTOS_TIMERS */注释行之间添加启动软定时器的代码。

```
79    /* USER CODE BEGIN RTOS_TIMERS */
80    /* start timers, add new ones, ... */
81    osTimerStart(myTimer01Handle, 250);
82    /* USER CODE END RTOS_TIMERS */
```

Step11.编译下载运行。LED0 和 LED1 分别闪烁，LED0 每秒闪 2 次，LED1 每秒闪 1 次。

特别说明：

软定时器不是 FreeRTOS 内核的组成部分，它本质上是一种任务，周期性地调用其回调函数。要注意的是，在回调函数中，不应该调用造成阻塞的 API 函数，如 vTaskDelay(), vTaskDelayUntil()等。

前面 Step7 配置 FreeRTOS 时，有两个参数 TIMER_TASK_PRIORITY 和 TIMER_QUEUE_LENGTH，下面详细说明一下。

因为软定时器是一种任务，所以它就应该有优先级。和一般的 FreeRTOS 任务一样，它的优先级也是从 0 到(configMAX_PRIORITIES-1)之间。TIMER_TASK_PRIORITY 的作用就是指定软定时器任务的优先级。下图是官网上对该参数的描述：

Home
FreeRTOS Books and Manuals
⊟ FreeRTOS
⊞ About FreeRTOS
⊟ Features / Getting Started...
Quick Start Guide
Tasks & Co-routines
⊞ More About Tasks...
⊞ More About Co-routines...
⊞ Queues, Mutexes, Semaphores...
⊞ Direct To Task Notifications
⊟ Software Timers
Introduction
Timer Service Daemon Task
Timer Daemon Configuration
One-Shot Vs Auto-Reload
Resetting a Timer
Event Groups (or 'flags')
Source Code Organisation
⊞ More Advanced...
⊞ Demo Projects
⊞ Supported Devices & Demos
⊞ API Reference
⊞ Contact, Support, Advertising
⊞ FreeRTOS Interactive!

Quick Start Guide
⇩ Download Source ⇩

FreeRTOS+ Lab Projects
FreeRTOS+TCP: Thread safe TCP/IP stack
FreeRTOS+FAT: Thread aware file system

Software Timers

[More about software timers...]

Configuring an application to use software timers

To make the FreeRTOS software timer API available in an application, simply:

1. Add the FreeRTOS/Source/timers.c source file to your project, and
2. Define the constants detailed in the table below in the applications FreeRTOSConfig.h header file.

Constant	Description
configUSE_TIMERS	Set to 1 to include timer functionality. The timer service task will be automatically created as the RTOS scheduler starts when configUSE_TIMERS is set to 1.
configTIMER_TASK_PRIORITY	Sets the priority of the timer service task. Like all tasks, the timer service task can run at any priority between 0 and (configMAX_PRIORITIES - 1). This value needs to be chosen carefully to meet the requirements of the application. For example, if the timer service task is made the highest priority task in the system, then commands sent to the timer service task (when a timer API function is called) and expired timers will both get processed immediately. Conversely, if the timer service task is given a low priority, then commands sent to the timer service task and expired timers will not be processed until the timer service task is the highest priority task that is able to run. It is worth noting here however, that timer expiry times are calculated relative to when a command is sent, and not relative to when a command is processed.

另外，软定时器不是一般的任务，在创建它时，同时还创建了一个消息队列，成为“定时器命令队列”。软定时器服务任务(或称守护任务)通过这个队列来接收用户发送的命令。当用户的程序调用 xTimerReset() 等定时器 API 函数时，就是向该队列发送命令。TIMER_QUEUE_LENGTH 的作用就是指定软定时器的命令队列的深度。下图是官网上对该参数的描述：

FreeRTOS+ Ecosystem
Internet of Things: Innovative complete solution
Fail Safe File System: Ensures data integrity
FreeRTOS BSPs: 3rd party driver packages
FAT SL File System: Super lean FAT FS
UDP/IP: Thread aware UDP stack
Trace & Visualisation: Tracealyzer for FreeRTOS
CLI: Command line interface
WolfSSL SSL / TLS: Networking security protocols
Safety: TUV certified RTOS
RTOS Training: Delivered online or on-site
IO: read(), write(), ioctl() interface

configTIMER_QUEUE_LENGTH	This sets the maximum number of unprocessed commands that the timer command queue can hold at any one time. Reasons the timer command queue might fill up include: • Making multiple timer API function calls before the RTOS scheduler has been started, and therefore before the timer service task has been created. • Making multiple (interrupt safe) timer API function calls from an interrupt service routine (ISR). • Making multiple timer API function calls from a task that has a priority above that of the timer service task.
configTIMER_TASK_STACK_DEPTH	Sets the size of the stack (in words, not bytes) allocated to the timer service task. Timer callback functions execute in the context of the timer service task. The stack requirement of the timer service task therefore depends on the stack requirements of the timer callback functions.

需要特别注意红框中描述的信息，可能造成定时器命令队列满载的原因包括：

- 在 RTOS 调度器启动前，多次调用定时器 API 函数；
- 在(硬件)中断服务程序(ISR)中多次调用定时器 API 函数；
- 在优先级高于定时器服务任务的任务中，多次调用定时器 API 函数。

更多关于定时器的描述，请访问 FreeRTOS 官网 http://www.freertos.org。

S.D.Lu　于 深圳
2016 年 8 月