STM32Cube高效开发教程（高级篇）

第8章 任务通知

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

8.1 任务通知的原理和功能

8.2 任务通知相关函数

8.3 示例一：使用任务通知传递数据

8.4 示例二：将任务通知用作计数信号量

队列、信号量、事件组等IPC技术都需要创建一个中间对象，进程之间通过这些中间对象进行通讯或同步。

创建对象就需要分配内存，占用一定内存。

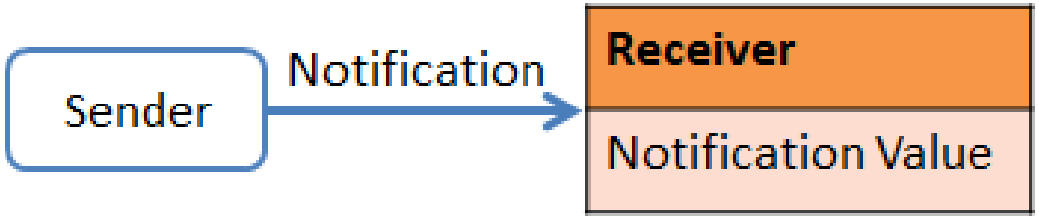
队列、信号量、事件组等

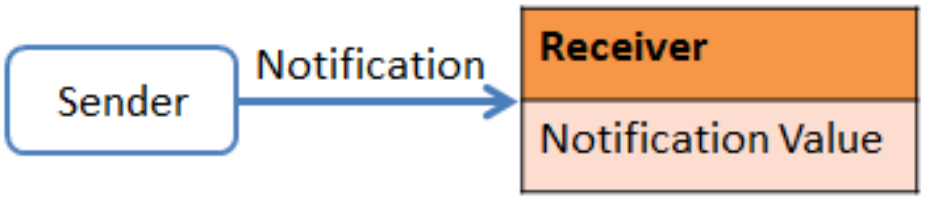
任务通知（Task Notification）： 无需创建中间对象，进程之间直接通讯的方法。

参数configUSE_TASK_NOTIFICATIONS需要设置为1才可以使用任务通知，默认值为1，可以在CubeMX中设置。

当参数configUSE_TASK_NOTIFICATIONS设置为1时，任务的任务控制块中会增加一个uint32_t类型的通知值（Notification Value）变量，并且任务接收通知有2种状况

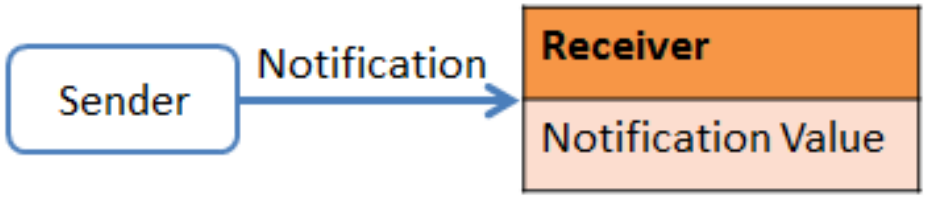
- 挂起（Pending）状态
- 非挂起（Not-Pending）状态





任务通知工作特点如下：

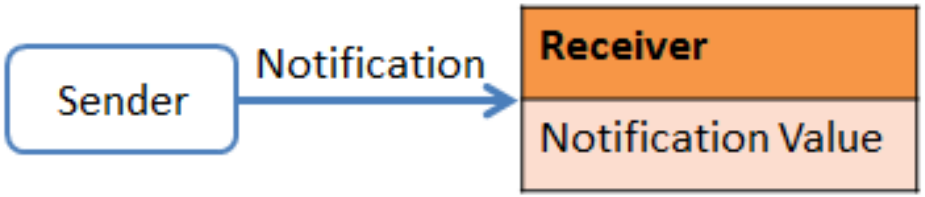
- 一个任务或ISR向另外一个指定的任务发送通知，将发送通知的进程称为发送者，将接收通知的进程称为接收者
- 发送者可以是任务或ISR，接收者只能是任务，不能是ISR
- 发送者发送通知时可以带有一个通知值，或者是使接收者的通知值发生改变的计算方法，例如使通知值加1。发送者只管发送通知，是否接收和处理通知由接收者去决定
- 接收者有未处理的通知时处于挂起状态。接收者可以进入阻塞方式等待通知，接收到通知后再做处理。



任务通知有如下的一些优点：

- 性能更高。使用任务通知方法在进程间传递数据时，比使用队列或信号量等方法的速度快得多。
- 内存开销小。使用任务通知时内存开销小，因为只需在任务控制块中增加几个变量。

使用任务通知可以代替二值信号量、计数信号量、事件组，可以代替只有一个存储单元的队列。任务通知使用比较灵活，而且工作效率高。



任务通知也有一些局限性，包括：

- 不能向ISR发送通知，只能是任务或ISR函数向任务发送通知
- 任务通知指定了接收者，多个发送者可以向同一个接收者发送不同的通知，但是发送者不能将一个通知发送给不同的接收者，也就是不能进行消息广播
- 任务通知一次只能发送或接收一个uint32_t类型的数据，不能像消息队列那样发送多个缓冲数据，因为任务控制块中只有一个uint32_t类型的通知值作为数据缓存

8.2 任务通知相关函数

8.2.1 相关函数概述

8.2.2 函数详解

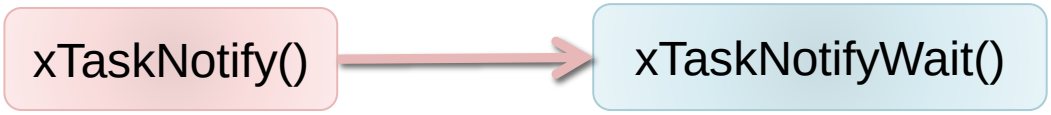
8.2.1 相关函数概述

可以在任务或IS里发送通知，但是只有任务能接收通知

分组	函数	功能
发送通知	xTaskNotify()	向一个任务发送通知的函数，带有通知值，还有值的传递方式设置。适用于利用通知值直接传递数据。
	xTaskNotifyFromISR()	xTaskNotifyFromISR()的ISR版本
	xTaskNotifyAndQuery()	与xTaskNotify()的功能相似，但是可以返回接收者之前的通知值
	xTaskNotifyAndQueryFromISR()	xTaskNotifyAndQuery()的ISR版本
	xTaskNotifyGive()	向一个任务发送通知，不带通知值，只是使接收者的通知值加1。适用于当做二值信号量或计数信号量时使用
	vTaskNotifyGiveFromISR()	xTaskNotifyGive()的ISR版本
接收通知	xTaskNotifyWait()	等待并获取任务通知值的通用函数，可以设置进入和退出等待时的任务通知值的值，例如进入将通知值清零，或退出时将通知值清零
	ulTaskNotifyTake()	等待并获取任务通知值，可在退出时等待时将通知值减1或清零。适适用于作为二值信号量或计数信号量的场合
其他	xTaskNotifyStateClear()	清除任务的等待状态，任务的通知值不变

发送和接收通知的函数可以分组两组：

- 通用版本的函数xTaskNotify()和xTaskNotifyWait()，可以发送任意的通知值，适合于在进程间通过通知值直接传递数据。



通过通知值直接传递数据

- 适用于二值信号量和计数信号量的函数xTaskNotifyGive()和ulTaskNotifyTake()，发送时使接收者的通知值加1，接收时使通知值减1或清零。

用作二值信号量或计数信号量

8.2.2 函数详解

1. 发送通知的函数xTaskNotify()

xTaskNotify()是发送通知值的通用函数，定义为：

```
#define xTaskNotify( xTaskToNotify, ulValue, eAction )
        xTaskGenericNotify( ( xTaskToNotify ), ( ulValue ), ( eAction ), NULL )
```

它实际上是执行了函数xTaskGenericNotify()，其函数原型是：

```
BaseType_t xTaskGenericNotify( TaskHandle_t xTaskToNotify, uint32_t ulValue,
    eNotifyAction eAction, uint32_t *pulPreviousNotificationValue );
```

- xTaskToNotify，是接收者任务的句柄
- ulValue，是发送的通知值
- eAction，是通知值的作用方式，是枚举类型eNotifyAction

枚举类型eNotifyAction的定义

```
typedef enum
{
    eNoAction = 0, /* 只发通知，不改变接收者的通知值 */
    eSetBits,      /* 接收者的通知值与ulValue按位或运算，适用于当做事件组使用 */
    eIncrement,    /* 将接收者的通知值加1，适用于当做二值信号量或计数信号量使用 */
    eSetValueWithOverwrite, /* 用ulValue覆盖接收者的通知值，即使前一次的通知未被处理 */
    eSetValueWithoutOverwrite /* 接收者处于非挂起状态时，用ulValue更新其通知值，否则不
    更新*/
} eNotifyAction;
```

- 参数pulPreviousNotificationValue，返回接收者的通知值被改变之前的值。

xTaskNotify()在调用函数xTaskGenericNotify()时没有传递最后一个参数，所以不能返回接收者更新之前的通知值。函数xTaskNotify()返回的是更新之后的接收者的通知值。

函数xTaskNotifyFromISR()是xTaskNotify()的ISR版本

```
#define xTaskNotifyFromISR( xTaskToNotify, ulValue, eAction,
        pxHigherPriorityTaskWoken )

    xTaskGenericNotifyFromISR( ( xTaskToNotify ), ( ulValue ), ( eAction ), NULL,
        ( pxHigherPriorityTaskWoken ) )
```

前4个参数与函数xTaskGenericNotify()中的参数相同，最后一个参数pxHigherPriorityTaskWoken是一个BaseType_t *类型的指针，实际上是一个返回数据，表示退出中断ISR函数之后是否需要进行上下文切换。

申请进行上下文切换调用函数portYIELD_FROM_ISR()

2. 发送通知且返回先前通知值的函数xTaskNotifyAndQuery()

函数xTaskNotifyAndQuery()与xTaskNotify()的功能相同，但是能返回接收者通知值改变之前的值。

```
#define xTaskNotifyAndQuery( xTaskToNotify, ulValue, eAction, pulPreviousNotifyValue )
  xTaskGenericNotify( ( xTaskToNotify ), ( ulValue ), ( eAction ), ( pulPreviousNotifyValue ) )
```

参数pulPreviousNotifyValue用于获取接收者之前的通知值，是uint32_t *类型的指针变量，调用示意代码如下：

```
uint32_t  previousValue=0;
uint32_t  currentValue=0;
currentValue =xTaskNotifyAndQuery(xTaskToNotify, ulValue, eAction,
      &previousValue);
```

3. 发送通知使通知值加1的函数xTaskNotifyGive()

函数xTaskNotifyGive()是xTaskNotify()的一种功能简化版本

```
#define xTaskNotifyGive( xTaskToNotify )
        xTaskGenericNotify( ( xTaskToNotify ), ( 0 ), eIncrement, NULL )
```

调用函数xTaskGenericNotify()时参数eAction设置为eIncrement，所以，函数xTaskNotifyGive()的功能就是使接收者的通知值加1，这使其适用于将任务通知当做二值信号量或计数信号量使用的场合。

4. 等待通知的函数xTaskNotifyWait()

接收者使用函数xTaskNotifyWait()进入阻塞状态等待任务通知并获取通知值

```
BaseType_t xTaskNotifyWait( uint32_t ulBitsToClearOnEntry, uint32_t
    ulBitsToClearOnExit, uint32_t *pulNotificationValue, TickType_t
    xTicksToWait );
```

- 参数ulBitsToClearOnEntry，在函数进入时需要清零的通知值的位掩码。需要清零的位在掩码中用1表示，否则用0表示

例如，ulBitsToClearOnEntry=0时，不更改当前的通知值

ulBitsToClearOnEntry=0xFFFFFFFF时，所有位清零

```
BaseType_t xTaskNotifyWait( uint32_t ulBitsToClearOnEntry, uint32_t
    ulBitsToClearOnExit, uint32_t *pulNotificationValue, TickType_t
    xTicksToWait );
```

- 参数ulBitsToClearOnExit，是函数在退出时需要清零的通知值的位的掩码。0就是不更改通知值，0xFFFFFFFF就是将通知值设置为0
- 参数pulNotificationValue，是一个uint32_t *类型的指针，用于返回接收到的通知值。
- 参数xTicksToWait，是函数在阻塞状态等待的节拍数

函数的返回值是pdTRUE或pdFALSE，pdTRUE表示接收到了任务通知，包括函数一进入就读取已挂起的任务通知。

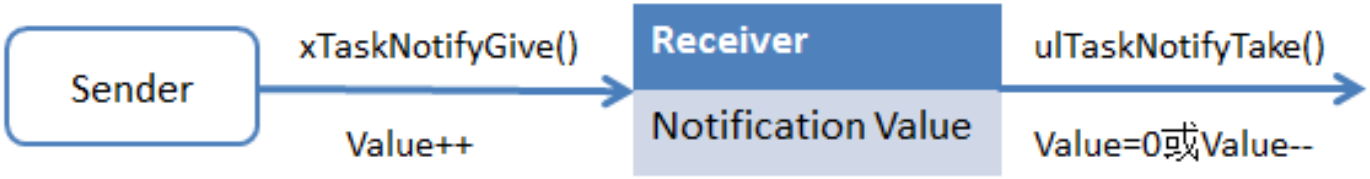
5. 适合用作信号量的等待通知函数ulTaskNotifyTake()

函数ulTaskNotifyTake()也是等待任务通知，它适合于将任务通知当做二值信号量或计数信号量使用的场合

```
uint32_t ulTaskNotifyTake( BaseType_t xClearCountOnExit, TickType_t
    xTicksToWait );
```

- 参数xClearCountOnExit的取值为pdTRUE或pdFALSE
  - 取值为pdTRUE时，函数接收到通知并退出时将通知值清零，这种情况下是将通知值当做二值信号量使用
  - 取值为pdFALSE时，函数接收到通知并退出时将通知值减1，这种情况下是将通知值当做计数信号量使用
- 参数xTicksToWait是在进入阻塞状态等待任务通知的节拍数
- 函数的返回值是减1或清零之前的通知值。

函数ulTaskNotifyTake()一般与函数xTaskNotifyGive()搭配使用，将任务通知值当做二值信号量或计数信号量使用。



将通知值当做计数信号量使用时有如下的特点：

- 接收者的通知值初始值为0
- 使用ulTaskNotifyGive()发送通知时，即使接收者没有接收和处理，通知值也会每次加1
- 执行函数ulTaskNotifyTake()时，如果通知值大于1，函数会立刻使通知值减1后返回，不会等待新的任务通知。如果当前通知值为0，接收者才会进入阻塞状态等待新的任务通知

6. 函数xTaskNotifyStateClear()

函数xTaskNotifyStateClear()的功能是清除接收者的任务通知等待状态，使其变为未挂起状态，但是它不会清零接收者的通知值。函数原型定义如下：

```
BaseType_t  xTaskNotifyStateClear( TaskHandle_t xTask );
```

其中，参数xTask是需要操作的任务的句柄，如果参数xTask设置为NULL表示清除当前任务的通知状态。

8.3 示例一：使用任务通知传递数据

8.3.1 示例功能与CubeMX项目设置

8.3.2 程序功能实现

8.3.1 示例功能和CubeMX项目设置

示例Demo8_1NotifyADC：使用中断方式进行ADC转换，通过任务通知将ADC转换结果作为通知值发送给另外一个任务

- ADC1在定时器TIM3的触发下进行周期为500ms的ADC数据采集，在ADC的ISR函数里通过函数xTaskNotifyFromISR()将转换结果发送给任务Task_Show
- 任务Task_Show总是使用函数xTaskNotifyWait()等待任务通知，读取出通知值后在LCD上显示数据

从项目Demo5_1BinarySemaphore整个复制而来，在原有的基础上进行CubeMX项目设置。

（1）定时器TIM3和ADC1的设置

这两个部分的设置与示例Demo5_1完全相同，其设置完全保留，包括NVIC设置。

TIM3的设置

- TIM3的更新周期为500ms
- 以UEV事件作为TRGO信号

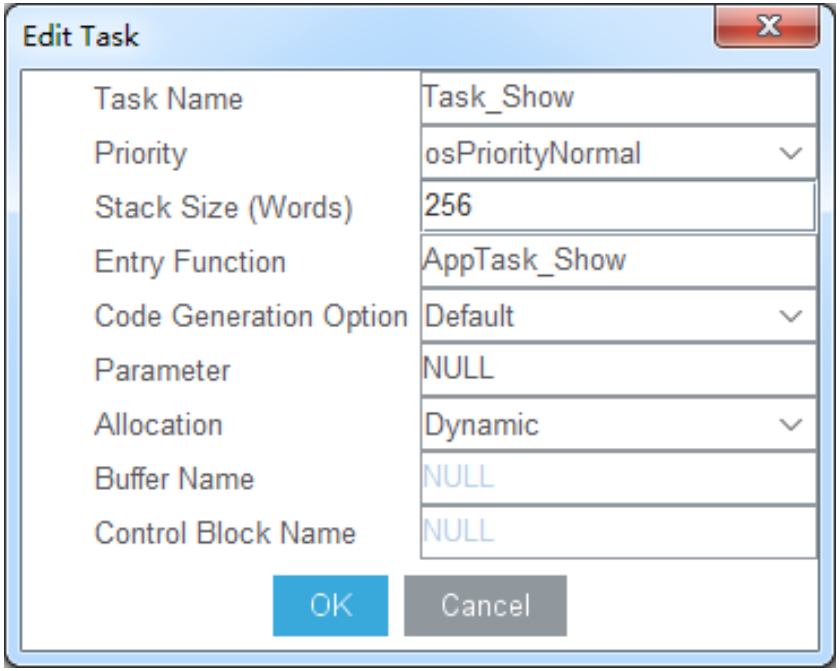
ADC1的设置

- 通道选择IN5
- 12bits精度，靠右对齐
- 外部触发源选择TIM3 TRGO

（2）FreeRTOS的设置

将参数configUSE_TASK_NOTIFICATIONS设置为1。在CubeMX中可以设置这个参数，且默认为Enabled。

保留原来项目中的任务Task_Show，删除原项目中的二值信号量BinSem_DataReady。



8.3.2 程序功能实现

1.主程序

```
int main(void)
{
  HAL_Init();                       //HAL初始化
  SystemClock_Config();             //系统时钟初始化
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_FSMC_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();

  /* USER CODE BEGIN 2 */
  TFTLCD_Init();                    //LCD 初始化
  LCD_ShowString(10, 10, (uint8_t *)"Demo8_1:Task Notification");
  LCD_ShowString(10, 30, (uint8_t *)"Transfer ADC value by notification");
  HAL_ADC_Start_IT(&hadc1);                 //以中断方式启动ADC
  HAL_TIM_Base_Start(&htim3);               //启动定时器
  /* USER CODE END 2 */

   osKernelInitialize();
   MX_FREERTOS_Init();
   osKernelStart();
}
```

2. FreeRTOS初始化

MX_FREERTOS_Init()里只需创建任务，使用任务通知时无需创建任何中间对象

```
/*  任务 Task_Show相关定义  */
osThreadId_t Task_ShowHandle;     //任务句柄变量
const osThreadAttr_t Task_Show_attributes = {   //任务属性
            .name = "Task_Show",
            .priority = (osPriority_t) osPriorityNormal,
            .stack_size = 256 * 4                    //不能使用128*4，会溢出
};

void MX_FREERTOS_Init(void)
{
      /* 创建任务 Task_Show */
      Task_ShowHandle = osThreadNew(AppTask_Show, NULL,
      &Task_Show_attributes);
}
```

3. 在ADC1的中断里发送通知

在文件freertos.c的一个代码沙箱段内实现ADC的中断事件回调函数

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
      if (hadc->Instance == ADC1)
      {
       uint32_t adc_value=HAL_ADC_GetValue(hadc);  //ADC转换原始数据
       if (Task_ShowHandle != NULL)
       {
          BaseType_t  taskWoken=pdFALSE;
          xTaskNotifyFromISR(Task_ShowHandle, adc_value,
                                  eSetValueWithOverwrite, &taskWoken);
          portYIELD_FROM_ISR(taskWoken);  //必须执行这条语句，进行任务切换申请
       }
      }
}
```

4. 任务通知的接收

ADC1中断里发送的通知是发送给任务Task_Show的，这个任务负责接收通知，读取出通知值之后进行处理并显示。

```
void AppTask_Show(void *argument)     //任务Task_Show的任务函数
{
        LCD_ShowString(10, 70, (uint8_t *)"ADC Value=");
        LCD_ShowString(10, 100, (uint8_t *)"Voltage(mV)=");
        uint32_t  notifyValue=0;
        for(;;)
        {
         BaseType_t result=xTaskNotifyWait(0x00, 0xffffffffUL, &notifyValue, portMAX_DELAY);
         if (result==pdTRUE)
         {
            uint32_t   tmpValue=notifyValue;   //ADC原始值
            LCD_ShowUintX(130,70,tmpValue,4);
            uint32_t   Volt=3300*tmpValue;  //mV
            Volt=Volt>>12;      //除以2^12
            LCD_ShowUintX(130,100,Volt,4);
         }
        }
}
```

接收的通知值

进入时不清除

退出时清零通知值

8.4 示例二：将任务通知用作计数信号量

8.4.1 示例功能

8.4.2 CubeMX项目设置

8.4.3 程序功能实现

8.4.1 示例功能

使用函数xTaskNotifyGive()发送通知，使接收者的通知值加1，使用函数ulTaskNotifyTake()读取通知，使通知值减1或清零

实际的计数信号量

任务通知模拟的计数信号量

任务通知模拟计数信号量与实际的计数信号量的细微差别：

- 实际的计数信号量的初始值不为零，一般用于表示可用资源的个数，例如餐厅中空余的餐桌个数（左图）
- 任务通知模拟的计数信号量的初值为0，一般用于表示待处理的事件的个数，例如模拟进入餐厅的排队人数（右图）

本节设计一个示例Demo8_2NotifyCounting，使用任务通知模拟计数信号量，表示如图所示的餐厅前排队的人数变化。

- 在FreeRTOS中创建一个任务Task_CheckIn，其通知值表示当前在排队的人数。
- 在任务Task_CheckIn中连续检测KeyRight键，当KeyRight键按下时执行函数ulTaskNotifyTake()使通知值减1，表示允许1人进店，使排队人数减1。
- 设置RTC周期唤醒周期为2秒，在周期唤醒中断里执行函数vTaskNotifyGiveFromISR()向任务Task_CheckIn发送通知，使其通知值加1，表示又来1人加入排队的队伍。

8.4.2 CubeMX项目设置

（1）RTC的设置

启用RTC的周期唤醒功能，设置唤醒周期为2秒，开启周期唤醒中断，在NVIC中设置其优先级别为5

（2）KeyRight和LED1的设置

Pin Name	GPIO mode	GPIO Pull-up/Pull-down	GPIO output level	User Label
PE2	Input mode	Pull-up	n/a	KeyRight
PF9	Output Push Pull	No pull-up and no pull-down	Low	LED1

（3）FreeRTOS的设置

创建一个任务Task_CheckIn

Tasks

Task Name	Priority	Stack Size (Words)	Entry Function	Allocation
Task_CheckIn	osPriorityNormal	128	AppTask_CheckIn	Dynamic

8.4.3 程序功能实现

1. 主程序

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();                    //GPIO初始化
  MX_FSMC_Init();
  MX_RTC_Init();                     //RTC初始化

  /* USER CODE BEGIN 2 */
  TFTLCD_Init();                     //LCD 初始化
  LCD_ShowString(10, 10, (uint8_t *)"Demo8_2:Task Notification");
  LCD_ShowString(10, 30, (uint8_t *)"Simulating people in wait");
  LCD_ShowString(10, 50, (uint8_t *)"1. People++ each 2sec");
  LCD_ShowString(10, 70, (uint8_t *)"2. Press KeyRight to People--");
  /* USER CODE END 2 */

  osKernelInitialize();
  MX_FREERTOS_Init();
  osKernelStart();
}
```

2. FreeRTOS初始化

使用任务通知时无需创建任何中间对象

```
/* 任务 Task_CheckIn相关定义 */
osThreadId_t Task_CheckInHandle; //任务句柄
const osThreadAttr_t Task_CheckIn_attributes = {   //任务属性
        .name = "Task_CheckIn",
        .priority = (osPriority_t) osPriorityNormal,
        .stack_size = 128 * 4
};

void MX_FREERTOS_Init(void)
{
    /* 创建任务Task_CheckIn */
    Task_CheckInHandle = osThreadNew(AppTask_CheckIn, NULL,
    &Task_CheckIn_attributes);
}
```

3. 在RTC周期唤醒中断里发送通知

回调函数HAL_RTCEx_WakeUpTimerEventCallback()，在文件freertos.c中重新实现这个函数。

```
/* USER CODE BEGIN Application */
/* RTC周期唤醒中断回调函数 */
void HAL_RTCEx_WakeUpTimerEventCallback(RTC_HandleTypeDef *hrtc)
{//发送通知，通知值加1，模拟客人加入排队
    LED1_Toggle();  //使LED1闪烁
    BaseType_t taskWoken=pdFALSE;
    vTaskNotifyGiveFromISR(Task_CheckInHandle,&taskWoken);  //发送通知，加1
    portYIELD_FROM_ISR(taskWoken);  //必须执行这条语句，申请任务调度
}
/* USER CODE END Application */
```

使接收者的通知值加1

4. 任务通知的接收

RTC周期唤醒中断里向任务Task_CheckIn发送通知，这个任务负责接收通知。

```
void AppTask_CheckIn(void *argument)
{
	LCD_ShowString(10, 100, (uint8_t *)"People in wait=");
	for(;;)
	{
	 GPIO_PinState keyState=HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_2); //PE2=KeyRight
	 if (keyState==GPIO_PIN_RESET)  //KeyRight按下表示有空位了，允许1人进店
	 {
	    BaseType_t clearOnExit=pdFALSE;   //退出时通知值减1
	    // 只是在通知值为0时才进入阻塞状态，所以可以多次读取通知值，每次使通知值减1
	    BaseType_t preCount=ulTaskNotifyTake(clearOnExit, portMAX_DELAY);
	    LCD_ShowUintX(170, 100, preCount-1,2);  // preCount是前一次的通知值
	    vTaskDelay(pdMS_TO_TICKS(300)); //延时，消除按键抖动
	 }
	 vTaskDelay(pdMS_TO_TICKS(5));
	}
}
```

注意，函数ulTaskNotifyTake()的执行有如下的两个特点：

- 如果当前通知值大于0，执行ulTaskNotifyTake()时不会进入阻塞状态，而是立刻返回。所以，如果当前通知值为5，可以多次按KeyRight键，即使没有新的任务通知到达，也可以看到当前排队人数在减少。
- 函数ulTaskNotifyTake()返回的是数值减1或清零之前的通知值，所以在程序中如果要显示当前的排队人数，显示的值是preCount-1。

5. 程序运行测试

- 可以看到LED1闪烁，表示RTC的中断ISR函数在执行，每2秒钟发送一次任务通知。
- 按下KeyRight键时，LCD上显示当前排队人数，连续按KeyRight键时会使排队人数减少，直到减少为0，任务Task_CheckIn就会进入阻塞等待状态。
- 除了函数ulTaskNotifyTake()和xTaskNotifyWait()之外，没有其他函数能读取任务的当前通知值，所以程序不能实时显示排队人数，只有在按下KeyRight键执行一次ulTaskNotifyTake()函数后才会显示当前排队人数。

练习任务

1. 看教材，练习本章的示例。