阿熊的FreeRTOS教程系列！

哈喽大家好！我是你们的老朋友阿熊！STM32教程系列更新完结已经有一段时间了，视频反馈还是不错的，从今天开始我们将会更新我们的FreeRTOS的教程

由于东西真的太多了，也纠结了很久要不要讲这个系列，毕竟难度真的很大，怕在难以做到那么通俗易懂，经过一段时间的考虑，还是决定好了给大家做一个入门级的讲解使用，由于FreeRTOS的内容真的很多，作为还是学生的我使用的也相对较少，操作系统层面的东西，我会用最大的能力去让大家理解，主要讲述主要功能，学完以后保证大伙可以理解80%以上的FreeRTOS的使用场景，好了废话不多说，开始我们的课程吧！

第十章：任务通知

任务通知是我们整个FreeRTOS当中的最后一个通信工具，然后他是FreeRTOS第8个大版本的时候才更新出来的产物，然后我们目前是第10个版本了，相对来说还是一个比较新的概念，并且它的功能是比较强大的，它可以一定程度上的代替二值信号量，计数信号量，队列，还有我们的事件组,并且最重要的是他是我们任务中自带的一个属性，不需要我们单独去创建对应的结构体，所以他的执行效率也快很多

壹：任务通知的介绍

每个任务都有一个结构体：TCB(Task Control Block)，里面有 2 个成员：

一个是 uint8_t 类型，用来表示通知状态

一个是 uint32_t 类型，用来表示通知值

任务状态一共有三种：

taskNOT_WAITING_NOTIFICATION：任务没有在等待通知（默认是这个）

taskWAITING_NOTIFICATION：任务在等待通知

taskNOTIFICATION_RECEIVED：任务接收到了通知，也被称为 pending(有数据了，待处理)

```
##define taskNOT_WAITING_NOTIFICATION   ( ( uint8_t ) 0 ) /* 也是初始状态*/
##define taskWAITING_NOTIFICATION       ( ( uint8_t ) 1 )
##define taskNOTIFICATION_RECEIVED      ( ( uint8_t ) 2 )
```

我们的通知值，就可以把他当作一个32位的变量，

如果只用它来存0和1，他就可以当作二值信号量使用

如果只用它来存数字0~N，就可以当作计数信号量

如果用它来存“信息”的话，可以当作队列来使用

如果每一位单独作为标志位，可以当作事件组来使用

大概该就是这样一个使用方法，说的太罗嗦反而难以来理解

贰：任务通知的基本函数

由于他是任务本身自带的属性，所以默认情况是不需要创建的，不过可以在宏定义中关闭，可以节省五个字节的内存，由于我们的任务通知它可以使用的功能有点多，所以它的函数也分为了两个版本，一个简化版和一个全功能版，然后这里我们会分别进行介绍

简化版：

发出通知：

普通任务中使用：

```
BaseType_t xTaskNotifyGive( TaskHandle_t xTaskToNotify );
//传入接收任务的句柄
//使用以后接收的任务其状态将会改为taskNOTIFICATION_RECEIVED，并且使得我们的任务通知加1
//返回pdPASS
```

中断中使用：

```
void vTaskNotifyGiveFromISR( TaskHandle_t xTaskHandle,
                             BaseType_t *pxHigherPriorityTaskWoken
);
```

!!!不论发送是否有效，都是返回pdPASS

接收通知：

```
uint32_t ulTaskNotifyTake( BaseType_t xClearCountOnExit, TickType_t
xTicksToWait );
//xClearCountOnExit:
//                pdTRUE：把通知值清零
//                pdFALSE：如果通知值大于 0，则把通知值减一
//xTicksToWait:
//                0：不等待，即刻返回
//                portMAX_DELAY：一直等待，直到通知值大于0
//返回清除之前的数值
```

从这里可以看出来我们的这个简单版本他只能模拟二值信号量和计数信号量

全功能版：

发出通知：

普通任务中使用：

```
BaseType_t  xTaskNotify( TaskHandle_t xTaskToNotify,
                         uint32_t ulValue,
                         eNotifyAction   eAction);
//xTaskToNotif:任务句柄，给哪个任务发通知
//ulValue:取决于eAction
//eAction:
//      eNoAction：更新通知状态为"pending"，未使用 ulValue，这个选项相当于二进制信号量
//      eSetBits：通知值 = 原来的通知值|ulValue，按位或，相当于事件组
//      eIncrement：通知值 = 原来的通知值 + 1，未使用 ulValue，相当于计数型信号量（二值信号量）
//      eSetValueWithoutOverwrite：如果数据未读状态，则此次调用不做任何事返回pdFAIL，否者写入ulValue
//                                 相当于队列
//      eSetValueWithOverwrite：直接用ulValue覆盖原来的数值，相当于队列的覆盖操作
```

中断中使用：

```
BaseType_t  xTaskNotifyFromISR( TaskHandle_t    xTaskToNotify,
                                uint32_t    ulValue,
                                eNotifyAction   eAction,
                                BaseType_t
*pxHigherPriorityTaskWoken );
```

接收通知：

```
BaseType_t  xTaskNotifyWait( uint32_t ulBitsToClearOnEntry,
                             uint32_t ulBitsToClearOnExit,
                             uint32_t *pulNotificationValue,
                             TickType_t xTicksToWait );
//ulBitsToClearOnEntry:
//                   在 xTaskNotifyWait 入口处，要清除通知值的哪些位？通知状态不是"pending"的情
//                   况下，才会清除。它的本意是我想等待某些事件发生，所以先把"旧数据"的某些位清零。
//ulBitsToClearOnExit:
```

```
//                    在 xTaskNotifyWait 出口处，如果不是因为超时推出，而是
因为得到了数据而退出时
//pulNotificationValue:
//                    用来取出通知值
//xTicksToWait:
//                    0：不等待，即刻返回
//                    portMAX_DELAY：一直等待
//返回：
//       pdPASS：成功
//       pdFAIL：没有得到通知
```

看起来参数有些复杂，但是使用起来其实没有那么复杂，接下啦我们就是带着大家去使用一下我们的任务通知

叁：任务通知的使用

实验一：模拟计数信号量

实验二：模拟事件组

实验室：模拟队列