1. FreeRTOSConfig.h配置文件

INCLUDE\_是FreeRTOS中API编译开关

2.任务状态：运行态，就绪态，阻塞态，挂起态

3.**任务优先级**，范围0~configMAX\_PRIORITIES-1，数值越**大**，优先级越**高**（和UCOS相反）。最低优先级0保留给空闲任务IdleTask使用。最高优先级configMAX\_PRIORITIES-1保留给软件定时器任务TimerTask使用。

4.任务中是一个死循环，一般不退出，如果要退出需要调用vTaskDelete(NULL)删除此任务。

5.任务控制块tskTCB，新版本命名为TCB\_T。采用动态创建时由xTaskCreate()函数自动分配内存。采用静态创建时xTaskCreateStatic()函数，需要传递任务控制块所需的内存地址。

6.任务堆栈StackType\_t为32位，分配堆栈大小以4个字节为基本单位。堆栈分配太小会导致程序无法运行。采用动态创建时xTaskCreate()函数的入口参数指定堆栈大小，函数自动分配堆栈内存。

7.创建任务有三个函数：xTaskCreate()动态方法创建任务，任务创建成功返回pdPASS

；xTaskCreateStatic()静态方法创建任务，任务创建失败返回NULL，其它值表示成功值为任务句柄；xTaskCreateRestricted()创建使用MPU进行限制的任务，使用动态内存分配。

8.删除任务vTaskDelete()，入口参数为任务句柄，如果删除的是任务本身入口参数可以为该任务的句柄或NULL。

9.函数或变量打头小写字母为x表示long，v表示void，e表示enum，c表示char，pc表示char \*，ux表示unsigned long，ul表示unsigned int，us表示unsigned short，

10. 运行态或就绪态的任务暂停运行等待某个条件满足后恢复运行可以进入挂起态，使用任务挂起和恢复。挂起任务恢复后进入就绪态而不是运行态。

任务挂起：vTaskSuspend()，挂起任务本身用vTaskSuspend(NULL)

任务恢复：vTaskResume()

中断服务函数中恢复任务：xTaskResumeFromISR()返回值是pdTRUE说明恢复的任务比正在运行的任务优先级高，必须做任务切换portYIELD\_FROM\_ISR()；返回值是pdFALSE说明恢复的任务比正在运行的任务优先级低，不需要任务切换。

11.PendSV，SysTick两个中断在FreeRTOS中设置的优先级最低是15。在xPortStartScheduler()中设置。

**中断优先级**的数值越**小**，则优先级越**高**。CM支持中断嵌套，使得高优先级异常会抢占低优先级异常。有3 个系统异常：复位，NMI 以及硬fault，它们有固定的优先级，并且它们的优先级号是负数，从而高于所有其它异常。原则上，CM支持3 个固定的高优先级和多达256 级的可编程优先级。

对于STM32使用高4位表示优先级，有效的优先级组：0~15个抢占优先级

分组位置|表达抢占优先级的位段|表达亚优先级的位段

3 [7:4] 无 NVIC\_PriorityGroup\_4全用于抢占，FreeRTOS推荐使用

4 [7:5] [4:4] NVIC\_PriorityGroup\_3

5 [7:6] [5:4] NVIC\_PriorityGroup\_2

6 [7:7] [6:4] NVIC\_PriorityGroup\_1

7 无 [7:4] NVIC\_PriorityGroup\_0

NVIC\_PriorityGroup\_4，设置抢占0~15，子优先级0

NVIC\_PriorityGroup\_3，设置抢占0~7，子优先级0~1

NVIC\_PriorityGroup\_2，设置抢占0~3，子优先级0~3

NVIC\_PriorityGroup\_1，设置抢占0~1，子优先级0~7

NVIC\_PriorityGroup\_0，设置抢占0，子优先级0~15

中断最低优先级configLIBRARY\_LOWEST\_INTERRUPT\_PRIORITY=15

系统可管理的最高中断优先级configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY=5

中断优先级和任务优先级区别

这两个之间没有任何关系，不管中断的优先级是多少，中断的优先级永远高于任何任务的优先级，即任务在执行的过程中，中断来了就开始执行中断服务程序。

对于 STM32来说，中断优先级的数值越小，优先级越高。 而 FreeRTOS的任务优先级是，任务优先级数值越小，任务优先级越低。

12.FreeRTOS开关所有可以管理的中断，使用BASEPRI关闭优先级为configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY~15的中断，

0~configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY-1的中断不可以调用FreeRTOS的API，高于configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY中断可以调用FromISR结尾的API函数。

由于0~configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY-1的优先级不会被FreeRTOS内核屏蔽，因此对于实时性要求高的任务可以使用这些优先级，如四轴飞行器壁障检测。

13. portENABLE\_INTERRUPTS();开中断

portDISABLE\_INTERRUPTS();关中断

14.列表和列表项是FreeRTOS的一个数据结构，是FreeRTOS的基石。

列表类似于链表用于跟踪FreeRTOS任务，包含列表项数量，当前列表项索引号，列表结束。

列表项是存在列表中的项目。分为列表项和迷你列表项。

vListInitialise() 列表初始化

vListInitialiseItem() 列表项初始化

vListInsert() 列表项插入

vListInsertEnd() 列表项末尾插入

uxListRemove() 列表项删除

15.任务相关API

uxTaskPriorityGet(任务句柄);//获取任务优先级

uxTaskGetNumberOfTasks();//获取系统任务数量

uxTaskGetSystemState();//获取系统中所有任务状态

vTaskList(缓冲);//获取所有任务的信息，建立一个表格，缓冲用全局变量，用局部变量任务堆栈需要很大

vTaskGetRunTimeStats(缓冲);//统计任务运行时间信息，建立一张表，缓冲用全局变量。用这个函数查询运行时间长的任务可以拆分几个小的任务运行。

xTaskGetHandle(任务名);//根据任务名获取任务句柄。

vTaskGetInfo();//根据任务句柄获取该单个任务的信息

eTaskGetState(任务句柄); //获取任务壮态

xTaskGetIdleTaskHandle();//获取空闲任务的句柄

uxTaskGetStackHighWaterMark(任务句柄); //获取任务堆栈历史剩余最小值

eTaskGetState(任务句柄) );//返回单个任务状态

xTaskGetTickCount();//查询任务调度器启动到现在运行多少时间

xTaskGetTickCountFromISR();//查询时间在中断中使用

16.内核控制函数

taskYIELD()

taskENTER\_CRITICAL()

taskEXIT\_CRITICAL()

taskENTER\_CRITICAL\_FROM\_ISR()

taskEXIT\_CRITICAL\_FROM\_ISR()

taskDISABLE\_INTERRUPTS()

taskENABLE\_INTERRUPTS()

vTaskStartScheduler()

vTaskEndScheduler()

vTaskSuspendAll()

xTaskResumeAll()

vTaskStepTick()

17.临界段是必须完整运行，不能被打断的代码段。

taskENTER\_CRITICAL();//进入临界区

taskEXIT\_CRITICAL();//退出临界区

taskENTER\_CRITICAL\_FROM\_ISR();//中断中进入临界区，返回值是中断寄存器备份，优先级在configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY~15才能使用

taskEXIT\_CRITICAL\_FROM\_ISR(x) ;//中断中退出临界区，入口参数是需要恢复的中断寄存器值，优先级在configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY~15才能使用

18.延时函数

vTaskDelay( );//相对延时函数，单位毫秒

vTaskDelayUntil( );//绝对延时函数，使用方法《407开发手册》Page196

19.滴答定时器

滴答定时器属于CORTEX-M内核，不是STM32专有。24位向下计数器。实现任务切换等功能。

20.队列用于任务与任务、任务与中断之间传递消息。也称消息队列。FreeRTOS中的信号量也是用队列实现的。数据发送到队列中，队列中存储的是数据的原始值，而不是引用（指针）。FreeRTOS的队列虽然需要数据拷贝，但是原始数据可以删除或修改，可以重复使用缓冲区。对于大数据量也可以存数据指针。队列可以多任务访问。

21. 出队阻塞，0：非阻塞；（1~ portMAX\_DELAY-1）：等待一定节拍时间；portMAX\_DELAY：阻塞等待。

入队阻塞，任务发消息发现队列满，可以采用非阻塞；等一段时间或阻塞等待。

22.队列操作过程：创建队列，返回队列句柄；向队列发消息；从队列读消息。

23.队列函数

xQueueCreate() 创建队列(动态方法) 靠xQueueGenericCreate()实现

xQueueCreateStatic() 创建队列(静态方法) 靠xQueueGenericCreateStatic ()实现

xQueueGenericCreate() 通用队列创建函数(动态方法，系统使用)

xQueueGenericCreateStatic() 通用队列创建函数(静态方法，系统使用)

xQueueSend() 发送消息到队列尾部（后向入队），等同xQueueSendToBack()

xQueueSendToFront() 发送消息到队列头（前向入队）

xQueueOverwrite() 发送消息到队列，带覆盖功能，覆盖旧的消息

xQueueReceive() 从队列中读取队列项（消息），读完删除队列项

xQueuePeek() 从队列中读取队列项（消息），读完队列项保留

中断中的函数加FromISR

24.信号量作用：共享资源访问；任务同步

25. FreeRTOS中信号量分为如下几种：

1、二值信号量

2、计数型信号量

3、互斥信号量

4、递归互斥信号量

二值信号量和互斥信号量区别：互斥信号量拥有优先级继承，二值信号量更适合用于任务之间或任务和中断之间同步，互斥信号量适合用于简单的互斥访问。

二值信号量只有空满2种状态。

创建二值信号量：

vSemaphoreCreateBinary( xSemaphore )动态创建二值信号量，兼容老版本，不再使用

xSemaphoreCreateBinary()动态创建二值信号量，新版本使用，返回SemaphoreHandle\_t型信号量句柄

xSemaphoreCreateBinaryStatic( pxStaticSemaphore ) 静态态创建二值信号量

释放计数型信号量:

xSemaphoreGive( xSemaphore )任务级信号量释放函数

xSemaphoreGiveFromISR( xSemaphore, pxHigherPriorityTaskWoken ) 中断级信号量释放函数

二值信号量、计数型信号量、互斥信号量都用上面的函数释放信号量

获取计数型信号量：

xSemaphoreTake( xSemaphore, xBlockTime ) 任务级获取信号量函数，有阻塞时间。

xSemaphoreTakeFromISR( xSemaphore, pxHigherPriorityTaskWoken ) 中断级获取信号量函数

二值信号量、计数型信号量、互斥信号量都用上面的函数获取信号量

26.计数型信号量，是计数大于1的信号量，用户需要关心信号量是否为空，不需要关心队列中存了什么。应用于事件计数和资源管理。

创建计数型信号量：

xSemaphoreCreateCounting( uxMaxCount, uxInitialCount ) 动态创建计数型信号量

xSemaphoreCreateCountingStatic()静态创建计数型信号量

释放计数型信号量函数和二值信号量相同(Give),判断没有到计数最大值，计数+1；否则不加

获取计数型信号量和二值信号量相同(Take),判断非零，计数-1；判断为0，等待

uxSemaphoreGetCount( xSemaphore )//可以获取计数值

27.优先级翻转，高优先级任务的资源被低优先级占用，被挂起无法运行，低优先级任务在运行时被中优先级任务抢占，导致低优先级任务挂起。结果出现了，先执行中优先级任务，再执行低优先级任务，再执行高优先级任务。高优先级任务实时性变差。

28.用互斥信号量可以避免优先级翻转，互斥信号量是一个拥有优先级继承的二值信号量。当一个互斥信号量正在被一个低优先级任务占用时，高优先级任务尝试获取该互斥信号量被阻塞，高优先级任务会将低优先级任务优先级提高到与自己相同的优先级。互斥信号量不能用于中断服务程序中。

创建互斥信号量：

xSemaphoreCreateMutex() 动态创建计数型信号量

xSemaphoreCreateMutexStatic()静态创建计数型信号量

29.递归互斥信号量和互斥信号量不同之处是已经获取递归互斥信号量的任务可以再次获取这个递归互斥信号量，次数不限。获取多少次就要释放多少次。

30.软件定时器，触发回调函数。 回调函数是在定时器任务中执行的，所以不能调用任何阻塞任务的函数，比如vTaskDelay()，vTaskDelayUnit()，还有一些访问队列或信号量的非零阻塞时间的函数。软件定时器有一个定时器服务任务和定时器命令队列。新创建的软件定时器都是出于休眠状态，需要用xTimerStart开启。

xTimerCreate()//动态创建一个软件定时器

xTimerCreateStatic()//静态创建一个软件定时器

xTimerReset()

xTimerResetFromISR()

xTimerStart()

xTimerStartFromISR()

xTimerStop()

xTimerStopFromISR()

31.事件标志组

事件标志位用来表明某个事件是否发生，事件位通常用作事件标志。一组事件位就是一个事件标志组。EventBits\_t是32位的数据类型，可以存储24个事件位，高8位保留。

xEventGroupCreate()

xEventGroupCreateStatic()

xEventGroupSetBits()

xEventGroupSetBitsFromISR()

xEventGroupClearBits()

xEventGroupClearBitsFromISR()

xEventGroupGetBits()

xEventGroupGetBitsFromISR()

vEventGroupDelete()

xEventGroupWaitBits()

32.任务通知：每个任务有一个32位通知值。可以替代消息邮箱，（二值、计数）信号量，事件标志组，不需要创建队列，信号量或事件标志组。任务通知只能有一个接收任务；接收任务会进入阻塞态，发送任务不会阻塞。（1）任务通知方式仅可以用在只有一个任务等待信号量，消息邮箱或者事件标志组的情况，不过实际项目中这种情况也是最多的。

（2）使用任务通知方式实现的消息邮箱，发送消息的任务不支持超时等待，即消息队列中的数据已经满了，可以等待消息队列有空间可以存新的数据，而任务通知方式实现的消息邮箱不支持超时等待。

xTaskNotify()发送通知，带通知值并且不保留接收任务原通知值

xTaskNotifyFromISR()发送通知，中断版本

xTaskNotifyGive()发送通知，不带通知值并且不保留接收任务原通知值，将通知值+1

xTaskNotifyGiveFromISR ()

xTaskNotifyAndQuery()发送通知，带通知值并且保留接收任务原通知值

xTaskNotifyAndQueryFromISR ()

xTaskNotifyTake()获取任务通知，类似于二值信号量或计数型信号量

xTaskNotifyWait()获取任务通知，全功能版

33.Tickless模式，当处理器处于空闲任务时进入低功耗模式，当需要处理应用层代码时再唤醒。但是滴答定时器中断会频繁唤醒，大量时间消耗在睡眠和唤醒模式中。Tickless模式进入空闲任务后就关闭滴答定时器中断，将休眠和唤醒的时间补偿，启用低功耗定时器(LPTIM)如果没有就还是使用滴答定时器。需要急时响应中断和应用层任务，应用层任务靠设置滴答定时器中断来唤醒，唤醒时间又可以作为时间补偿。

进入低功耗条件：空闲任务是唯一运行的任务，并且休眠时间（当前时刻到下一个任务运行时刻）大于预值。

流程：1.获得下一个任务运行时刻T；2.初始化一个定时器（LPTIM或滴答定时器）设置唤醒时间T；3.降低系统主频，关闭某些外设，以降低功耗；4.WFI进入低功耗模式；5中断退出低功耗模式；6恢复主频，打开外设；7补偿系统计数器，系统时钟；8.处理应用层任务

34.空闲任务，让处理器无事可做时运行的任务，优先级最低(0)。它是调度器自动创建的任务，确保至少有一个任务在运行。某个任务调用vTaskDelete()删除自身，那么任务控制块和任务堆栈等由FreeRTOS分配的内存是在空闲任务中释放掉的，如果删除的是别的任务那么内存会直接释放不需要空闲任务释放。空闲任务的工作：释放自删任务的内存；有其它高优先级任务或同优先级任务释放CPU控制权；没有其它任务就运行钩子函数；完成工作后如果Tickles模式使能就进入低功耗模式。空闲任务钩子函数不能调用可以阻塞空闲任务的API函数，因为系统中至少要保证有一个任务在运行。用钩子函数进入低功耗模式弊端是会被滴答定时器中断频繁唤醒，如果只有2个时钟节拍进入低功耗模式意义不大。

35.任务、队列、信号量等创建都有动态和静态两种方法。FreeRTOS用pvPortMalloc()替代malloc()，用vPortFree()替代free()。FreeRTOS提供了5种内存分配方法。

安富莱[FreeRTOS教程]

1. #define configUSE\_PREEMPTION 1 //1使能抢占式调度器；0使能合作式调度器

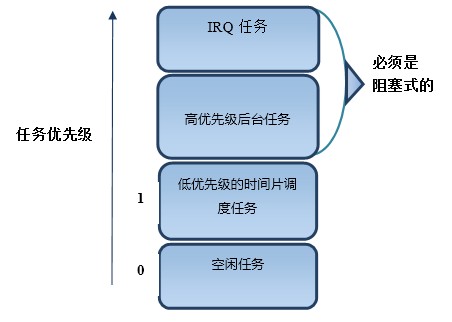
2.空闲任务优先级最低为0，软件定时器优先级最高为(configMAX\_PRIORITIES-1)

3. 函数vTaskResume实现任务挂起态的恢复，多次调用函数vTaskSuspend，只需调用一次vTaskResume即可将任务从挂起态恢复。在中断中使用的是xTaskResumeFromISR()

4.系统栈、任务栈，栈生长方向从高地址向低地址生长。栈溢出。

<http://bbs.armfly.com/read.php?tid=21084&fpage=2>

5.任务优先级设置推荐方式如下图所示：



IRQ任务：指通过中断服务程序进行触发的任务，此类任务应该设置为所有任务里面优先级最高的。

高优先级后台任务：比如按键检测，触摸检测，USB消息处理，串口消息处理等，都可以归为这一类任务。

低优先级的时间片调度任务：比如emWin的界面显示，LED数码管的显示等不需要实时执行的都可以归为这一类任务。实际应用中用户不必拘泥于将这些任务都设置为优先级1的同优先级任务，可以设置多个优先级，只需注意这类任务不需要高实时性。

空闲任务：空闲任务是系统任务。

特别注意：IRQ任务和高优先级任务必须设置为阻塞式（调用消息等待或者延迟等函数即可），只有这样，高优先级任务才会释放CPU的使用权，从而低优先级任务才有机会得到执行。

任务优先级修改：vTaskPrioritySet()

任务优先级获取：vTaskPriorityGet()

6.临界区

任务代码临界段处理:

taskENTER\_CRITICAL()/\* 进入临界区 \*/

临界段代码

taskEXIT\_CRITICAL();/\* 退出临界区 \*/

中断服务程序临界段处理:

uxSavedInterruptStatus = portSET\_INTERRUPT\_MASK\_FROM\_ISR(); /\* 进入临界区 \*/

临界区代码

portCLEAR\_INTERRUPT\_MASK\_FROM\_ISR( uxSavedInterruptStatus ); /\* 退出临界区 \*/

开关中断的实现:FreeRTOS也专门提供了一组开关中断函数，实现比较简单，其实就是前面临界段进入和退出函数的精简版本，主要区别是不支持中断嵌套。具体实现如下：

taskDISABLE\_INTERRUPTS()

taskENABLE\_INTERRUPTS()

7. FreeRTOS调度锁，任务锁和中断锁

调度锁就是RTOS提供的调度器开关函数，如果某个任务调用了调度锁开关函数，处于调度锁开和调度锁关之间的代码在执行期间是不会被高优先级的任务抢占的，即任务调度被禁止。这一点要跟临界段的作用区分开，调度锁只是禁止了任务调度，并没有关闭任何中断，中断还是正常执行的。而临界段进行了开关中断操作。

FreeRTOS的调度锁开启：vTaskSuspendAll()

FreeRTOS的调度锁关闭：xTaskResumeAll ()

使用这个函数要注意以下问题：

1. 调度锁函数只是禁止了任务调度，并没有关闭任何中断。

2. 调度锁开启函数vTaskSuspendAll和调度锁关闭函数xTaskResumeAll一定要成对使用。

3. 切不可在调度锁开启函数vTaskSuspendAll和调度锁关闭函数xTaskResumeAll之间调用任何会引起任务切换的API，比如vTaskDelayUntil、vTaskDelay、xQueueSend等。

中断锁就是RTOS提供的开关中断函数，FreeRTOS没有专门的中断锁函数，使用临界段处理函数就可以实现同样效果。

任务锁，为了防止当前任务的执行被其它高优先级的任务打断而提供的锁机制就是任务锁。FreeRTOS也没有专门的任务锁函数，但是使用FreeRTOS现有的功能有两种实现方法：

（1）通过给调度器加锁实现

（2）通过临界段处理函数关闭任务切换中断PendSV和系统时钟节拍中断Systick