Freertos中的List是双向循环链表

但是， 在多任务系统中，每个任务都是独立的，互不干扰的，所以要为每个任务都分  
配独立的栈空间，这个栈空间通常是一个预先定义好的全局数组， 也可以是动态分配的一  
段内存空间，但它们都存在于 RAM 中。

系统为了顺利的调度任务，为每个任务都额外定义了一个任务控制块，  
这个任务控制块就相当于任务的身份证，里面存有任务的所有信息，比如任务的栈指针，  
任务名称， 任务的形参等。有了这个任务控制块之后，以后系统对任务的全部操作都可以  
通过这个任务控制块来实现。TCB

1 typedef struct tskTaskControlBlock  
2 {  
3 volatile StackType\_t \*pxTopOfStack; /\* 栈顶 \*/ (1)  
4  
5 ListItem\_t xStateListItem; /\* 任务节点 \*/ (2)  
6  
7 StackType\_t \*pxStack; /\* 任务栈起始地址 \*/ (3)  
8 /\* 任务名称，字符串形式 \*/(4)  
9 char pcTaskName[ configMAX\_TASK\_NAME\_LEN ];  
10 } tskTCB;

栈顶指针，作为 TCB 的第一个成员

任务节点，这是一个内置在 TCB 控制块中的链表节点，通过这个  
节点，可以将任务控制块挂接到各种链表中。

任务栈起始地址。

FreeRTOS 中，任务的创建有两种方法，一种是使用动态创建，一  
种是使用静态创建。 动态创建时，任务控制块和栈的内存是创建任务时动态分配的， 任务  
删除时，内存可以释放。 静态创建时，任务控制块和栈的内存需要事先定义好，是静态的  
内 存 ， 任 务 删 除 时 ， 内 存 不 能 释 放 。

配置 PendSV 和 SysTick 的中断优先级为最低。 SysTick 和  
PendSV 都会涉及到系统调度，系统调度的优先级要低于系统的其它硬件中断优先级， 即优  
先相应系统中的外部硬件中断， 所以 SysTick 和 PendSV 的中断优先级配置为最低。

prvStartFirstTask()函数用于开始第一个任务，主要做了两个动作，一个是更新 MSP 的  
值，二是产生 SVC 系统调用，然后去到 SVC 的中断服务函数里面真正切换到第一个任务。

\_\_asm void prvStartFirstTask( void )  
9 {  
10 PRESERVE8 **(2)**  
11  
12 /\* 在 Cortex-M 中， 0xE000ED08 是 SCB\_VTOR 这个寄存器的地址， **(3)**  
13 里面存放的是向量表的起始地址，即 MSP 的地址 \*/  
14 ldr r0, =0xE000ED08 **(4)**  
15 ldr r0, [r0] **(5)**  
16 ldr r0, [r0] **(6)**  
17  
18 /\* 设置主堆栈指针 msp 的值 \*/  
19 msr msp, r0 **(7)**  
20  
21 /\* 使能全局中断 \*/ **(8)**  
22 cpsie i  
23 cpsie f  
24 dsb  
25 isb  
26  
27 /\* 调用 SVC 去启动第一个任务 \*/  
28 svc 0 **(9)**  
29 nop  
30 nop  
31 }

1 #define xPortPendSVHandler PendSV\_Handler  
2 #define xPortSysTickHandler SysTick\_Handler  
**3 #define vPortSVCHandler SVC\_Handler**

vPortSVCHandler()函数开始真正启动第一个任务，不再返回，

xPortPendSVHandler中断服务函数是真正实现任务切换的地方

（在整个系统中，中断使用的是主堆栈，  
栈指针使用的是 MSP），因为接下来要调用函数 vTaskSwitchContext，调用函数时，返回  
地址自动保存到 R14 中，所以一旦调用发生， R14 的值会被覆盖（ PendSV 中断服务函数执  
行完毕后，返回的时候需要根据 R14 的值来决定返回处理器模式还是任务模式，出栈时使  
用的是 PSP 还是 MSP）

/\* 带返回值的关中断函数，可以嵌套，可以在中断里面使用 \*/ **(2)**  
16 #define portSET\_INTERRUPT\_MASK\_FROM\_ISR() ulPortRaiseBASEPRI()  
17 ulPortRaiseBASEPRI( void )  
18 {  
19 uint32\_t ulReturn, ulNewBASEPRI = configMAX\_SYSCALL\_INTERRUPT\_PRIORITY; **(2)-**①  
20 \_\_asm  
21 {  
22 mrs ulReturn, basepri **(2)-**②  
23 msr basepri, ulNewBASEPRI **(2)-**③  
24 dsb  
25 isb  
26 }  
27 return ulReturn; **(2)-**④  
28 }

中断里的关中断函数和任务里的关中断函数的用途？？？？

操作系统里面的最小的时间单位就是  
SysTick 的中断周期，我们称之为一个 tick， SysTick 中断服务函数在 port.c.c 中实现，

void xPortSysTickHandler( void )  
2 {  
3 /\* 关中断 \*/  
4 vPortRaiseBASEPRI(); **(1)**  
5  
6 /\* 更新系统时基 \*/  
7 xTaskIncrementTick(); **(2)**  
8  
9 /\* 开中断 \*/

查 找 最 高 优 先 级 的 就 绪 任 务 有 两 种 方 法 ， 具 体 由  
configUSE\_PORT\_OPTIMISED\_TASK\_SELECTION 这个宏控制， 定义为 0 选择通用方法，  
定义为 1 选择根据处理器优化的方法，该宏默认在 portmacro.h 中定义为 1，即使用优化过  
的方法

任务延时列表表维护着一条双向链表，每个节点代表了正在延时的任务，节点按照延  
时时间大小做升序排列。 当每次时基中断（ SysTick 中断） 来临时， 就拿系统时基计数器的  
值 xTickCount 与下一个任务的解锁时刻变量 xNextTaskUnblockTime 的值相比较， 如果相  
等， 则表示有任务延时到期， 需要将该任务就绪， 否则只是单纯地更新系统时基计数器  
xTickCount 的值， 然后进行任务切换。

FreeRTOS 定义了两个任务延时列表，当系统时基计数器  
xTickCount 没有溢出时，用一条列表，当 xTickCount 溢出后， 用另外一条列表。

延时的到期时间比xTickCount 最大值还大时，溢出。则将到期时间插入溢出列表

在 RTOS 中，最小的时间单位为一个 tick，即 SysTick 的中断周期，  
RT-Thread 和 μC/OS 可以指定时间片的大小为多个 tick，但是 FreeRTOS 不一样，时间片只  
能是一个 tick。 与其说 FreeRTOS 支持时间片，倒不如说它的时间片就是正常的任务调度。

listGET\_OWNER\_OF\_NEXT\_ENTRY()函数的妙处在于它并不是获取链表下的第一个  
节点的 OWNER，而且用于获取下一个节点的 OWNER。有下一个那么就会有上一个的说  
法，怎么理解？假设当前链表有 N 个节点，当第 N 次调用该函数时， pxIndex 则指向第 N  
个节点， 即每调用一次， 节点遍历指针 pxIndex 则会向后移动一次，用于指向下一个节点。

（实现了相同优先级的任务，时间片轮转调度）

taskRESET\_READY\_PRIORITY() 函 数 的 妙 处 在 于 清 除 优 先 级 位 图 表  
uxTopReadyPriority 中相应的位时候，会先判断当前优先级链表下是否还有其它任务，如果  
有则不清零。 假设当前实验中，任务 1 会调用 vTaskDelay()，会将自己挂起，只能是将任  
务 1 从就绪列表删除，不能将任务 1 在优先级位图表 uxTopReadyPriority 中对应的位清 0，  
因为该优先级下还有任务 2，否则任务 2 将得不到执行