

FreeRTOS 任务状态查询

原创

Paradise_Violet

于 2022-08-15 16:16:59 发布

1584

收藏

5

版权


分类专栏:

FreeRTOS笔记

 文章标签:

FreeRTOS

单片机

 FreeRTOS笔记 专栏收录该内容

目录

- 任务相关API函数
- 函数TaskPriorityGet()获取优先级
 - 函数uxTaskGetSystemState()获取任务状态
 - 函数xTaskGetHandle()
 - 函数uxTaskGetStackHighWaterMark()
 - 函数eTaskGetState()
 - 函数vTaskList()

任务相关 API函数

先通过一个表 11.1.1 来看一下这些与任务相关的其他 API 函数都有哪些：

函数	描述
uxTaskPriorityGet()	查询某个任务的优先级。
vTaskPrioritySet()	改变某个任务的的任务优先级。
uxTaskGetSystemState()	获取系统中任务状态。
vTaskGetInfo()	获取某个任务信息。
xTaskGetApplicationTaskTag()	获取某个任务的标签(Tag)值。
xTaskGetCurrentTaskHandle()	获取当前正在运行的任务的的任务句柄。
xTaskGetHandle()	根据任务名字查找某个任务的句柄
xTaskGetIdleTaskHandle()	获取空闲任务的的任务句柄。
uxTaskGetStackHighWaterMark()	获取任务的堆栈的历史剩余最小值,FreeRTOS 中叫做“高水位线”
eTaskGetState()	获取某个任务的壮态, 这个壮态是 eTaskState 类型。
pcTaskGetName()	获取某个任务的的任务名字。
xTaskGetTickCount()	获取系统时间计数器值。
xTaskGetTickCountFromISR()	在中断服务函数中获取时间计数器值
xTaskGetSchedulerState()	获取任务调度器的状态, 开启或未开启。
uxTaskGetNumberOfTasks()	获取当前系统中存在的任务数量。
vTaskList()	以一种表格的形式输出当前系统中所有任务的详细信息。
vTaskGetRunTimeStats()	获取每个任务的运行时间。
vTaskSetApplicationTaskTag()	设置任务标签(Tag)值。
SetThreadLocalStoragePointer()	设置线程本地存储指针
GetThreadLocalStoragePointer()	获取线程本地存储指针

表 11.1.1 任务相关 API 函数 CSDN @Paradise_Violet

函数TaskPriorityGet()获取优先级

此函数用来获取指定任务的优先级, 要使用此函数的话宏 INCLUDE_uxTaskPriorityGet 应该定义为 1, 函数原型如下:

```
UBaseType_t uxTaskPriorityGet( TaskHandle_t xTask )
```

- 参数:
- xTask:

要查找的任务的的任务句柄。
- 返回值:
- 获取到的对应的任务的优先级。
- CSDN @Paradise_Violet

```

44
45 //创建开始任务
46 xTaskCreate((TaskFunction_t )start_task,           //任务函数
47             (const char* )"start_task",           //任务名称
48             (uint16_t )START_STK_SIZE,             //任务堆栈大小
49             (void* )NULL,                           //传递给任务函数的参数
50             (UBaseType_t )START_TASK_PRIO,         //任务优先级
51             (TaskHandle_t* )&StartTask_Handler):   //任务句柄
52 vTaskStartScheduler(); //开启任务调度
53 }
54
55 //开始任务任务函数
56 void start_task(void *pvParameters)
57 {
58     taskENTER_CRITICAL(); //进入临界区
59     //创建LED0任务
60     xTaskCreate((TaskFunction_t )led0_task,
61                 (const char* )"led0_task",
62                 (uint16_t )LED0_STK_SIZE,
63                 (void* )NULL,
64                 (UBaseType_t )LED0_TASK_PRIO,
65                 (TaskHandle_t* )&LED0Task_Handler);
66     //创建LED1任务
67     xTaskCreate((TaskFunction_t )query_task,
68                 (const char* )"query_task",
69                 (uint16_t )QUERY_STK_SIZE,
70                 (void* )NULL,
71                 (UBaseType_t )QUERY_TASK_PRIO,
72                 (TaskHandle_t* )&QUERYTask_Handler);
73     vTaskDelete(StartTask_Handler); //删除开始任务
74     taskEXIT_CRITICAL(); //退出临界区
75 }
76
77 //LED0任务函数

```

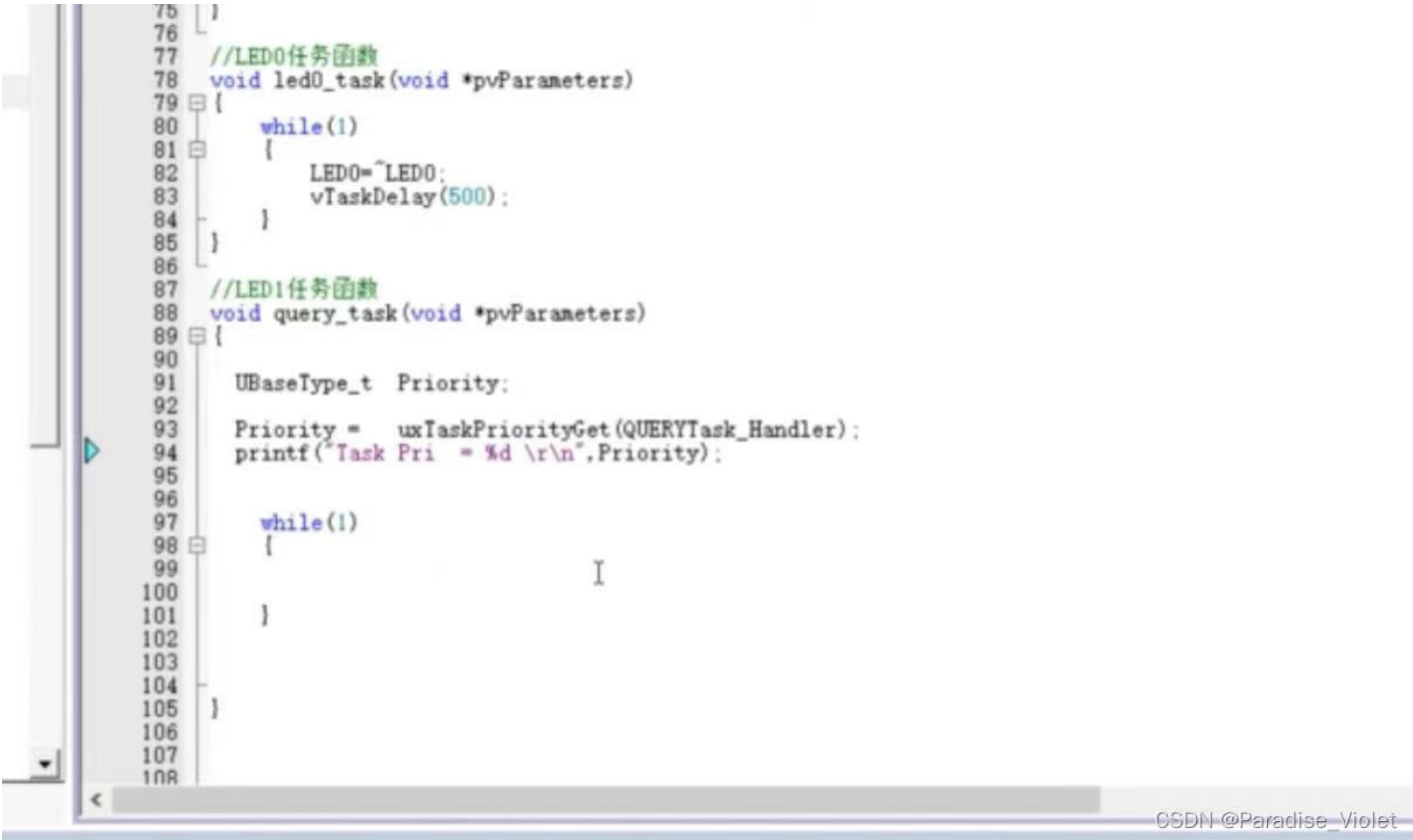
CSDN @Paradise_Violet

```

7
8 //任务优先级
9 #define START_TASK_PRIO 1
10 //任务堆栈大小
11 #define START_STK_SIZE 128
12 //任务句柄
13 TaskHandle_t StartTask_Handler;
14 //任务函数
15 void start_task(void *pvParameters);
16
17 //任务优先级
18 #define LED0_TASK_PRIO 2
19 //任务堆栈大小
20 #define LED0_STK_SIZE 50
21 //任务句柄
22 TaskHandle_t LED0Task_Handler;
23 //任务函数
24 void led0_task(void *pvParameters);
25
26 //任务优先级
27 #define QUERY_TASK_PRIO 3
28 //任务堆栈大小
29 #define QUERY_STK_SIZE 200
30 //任务句柄
31 TaskHandle_t QUERYTask_Handler;
32 //任务函数
33 void query_task(void *pvParameters);
34
35
36

```

CSDN @Paradise_Violet



函数uxTaskGetSystemState()获取任务状态

3、uxTaskGetSystemState()

此函数用于获取系统中所有任务的任务状态，每个任务的状态信息保存在一个 TaskStatus_t 类型的结构体里面，这个结构体里面包含了任务的任务句柄、任务名字、堆栈、优先级等信息，要使用此函数的话宏 configUSE_TRACE_FACILITY 应该定义为 1，函数原型如下：

```
UBaseType_t uxTaskGetSystemState( TaskStatus_t * const    pxTaskStatusArray,
                                   const UBaseType_t      uxArraySize,
                                   uint32_t * const        pulTotalRunTime )
```

参数：

pxTaskStatusArray: 指向 TaskStatus_t 结构体类型的数组首地址，每个任务至少需要一个 TaskStatus_t 结构体，任务的数量可以使用函数 uxTaskGetNumberOfTasks()。结构体 TaskStatus_t 在文件 task.h 中有如下定义：

```
typedef struct xTASK_STATUS
{
    TaskHandle_t    xHandle;          //任务句柄
    const char *    pcTaskName;       //任务名字
    UBaseType_t     xTaskNumber;      //任务编号
    eTaskState      eCurrentState;    //当前任务状态，eTaskState 是一个枚举类型
    UBaseType_t     uxCurrentPriority; //任务当前的优先级
    UBaseType_t     uxBasePriority;    //任务基础优先级
    uint32_t        ulRunTimeCounter;  //任务运行的总时间
    StackType_t *   pxStackBase;      //堆栈基地址
    uint16_t        usStackHighWaterMark; //从任务创建以来任务堆栈剩余的最小大小，此值如果太小的话说明堆栈有溢出的风险。
} TaskStatus_t;
```

uxArraySize: 保存任务状态数组的数组的大小。
pulTotalRunTime: 如果 configGENERATE_RUN_TIME_STATS 为 1 的话此参数用来保存系统总的运行时间。

返回值： 统计到的任务状态的个数，也就是填写到数组 pxTaskStatusArray 中的个数，此值应该等于函数 uxTaskGetNumberOfTasks() 的返回值。如果参数 uxArraySize 太小的话返回值可能为 0。

TaskStatusArray是结构体指针，如果不给它赋予空间的话就是野指针，所以这里需要malloc
FreeRTOS用pvPortMalloc 他是一个数组，每个数组有多少个大小，就是一个结构体大小
判断是否malloc成功
定义和ArraySize同类型的i
用for循环，有多少个任务就打印多少次

\t是补全当前字符串长度到8的整数倍，最少1个最多8个空格，补多少要看你\t前字符串长度。
比如当前字符串长度10，那么\t后长度是16，也就是补6个空格。
如果当前字符串长度12，此时\t后长度是16，补4个空格

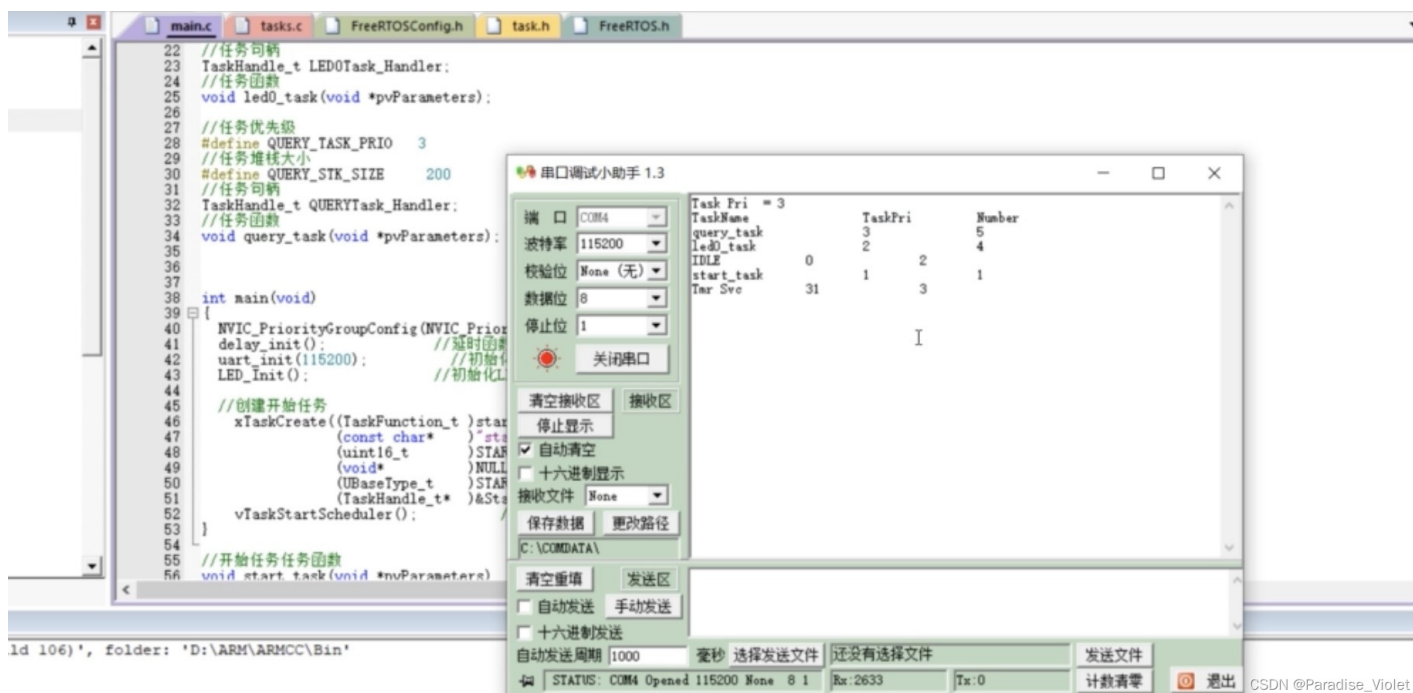
数组里面循环了5次，因为我们有5个任务 一个开始任务，开始任务里又创建两个任务
任务序号就是我们创建任务的过程，第一个任务创建开始任务，第二个是空闲任务，创建开始任务的同时也会创建空闲任务，开始任务创建完后会进入任务调度，在第三个就是定时器任务， 第四个是led0， 第五个是查询任务
任务的优先级和我们设定的一样1,2,3 空闲任务优先级是0，定时器优先级是31 所说0和31优先级我们不能用跟中断优先级不一样，中断里面是数字越小优先级越高。但在我们多任务过程中，是数字越大优先级越大

```

84     }
85 }
86
87 //LED1任务函数
88 void query_task(void *pvParameters)
89 {
90     uint32_t TotalRunTime;
91     UBaseType_t Priority;
92     TaskStatus_t *TaskStatusArray;
93     UBaseType_t ArraySize, i;
94
95     Priority = uxTaskPriorityGet(QUERYTask_Handler);
96     printf("Task Pri = %d \r\n", Priority);
97
98     ArraySize = uxTaskGetNumberOfTasks();
99     TaskStatusArray = pvPortMalloc(ArraySize * sizeof(TaskStatus_t));
100
101     if(TaskStatusArray != NULL)
102     {
103         ArraySize = uxTaskGetSystemState((TaskStatus_t *) TaskStatusArray,
104                                         (UBaseType_t) ArraySize,
105                                         (uint32_t *) &TotalRunTime);
106
107         printf("TaskName\t\tTaskPri\t\tNumber\t\t\r\n");
108
109         for(i=0; i<ArraySize; i++)
110         {
111             printf("%s\t\t%d\t\t%d\t\t\r\n",
112                   TaskStatusArray[i].pcTaskName,
113                   TaskStatusArray[i].uxCurrentPriority,
114                   TaskStatusArray[i].xTaskNumber);
115         }
116     }
117 }

```

CSDN @Paradise_Violet



CSDN @Paradise_Violet

函数xTaskGetHandle()

此函数根据任务名字获取任务的任务句柄，在使用函数 xTaskCreate()或 xTaskCreateStatic() 创建任务的时候都会给任务分配一个任务名，函数 xTaskGetHandle()就是使用这个任务名字来查询其对应的任务句柄的。要使用此函数的话宏 INCLUDE_xTaskGetHandle 应该设置为 1，此函数原型如下：

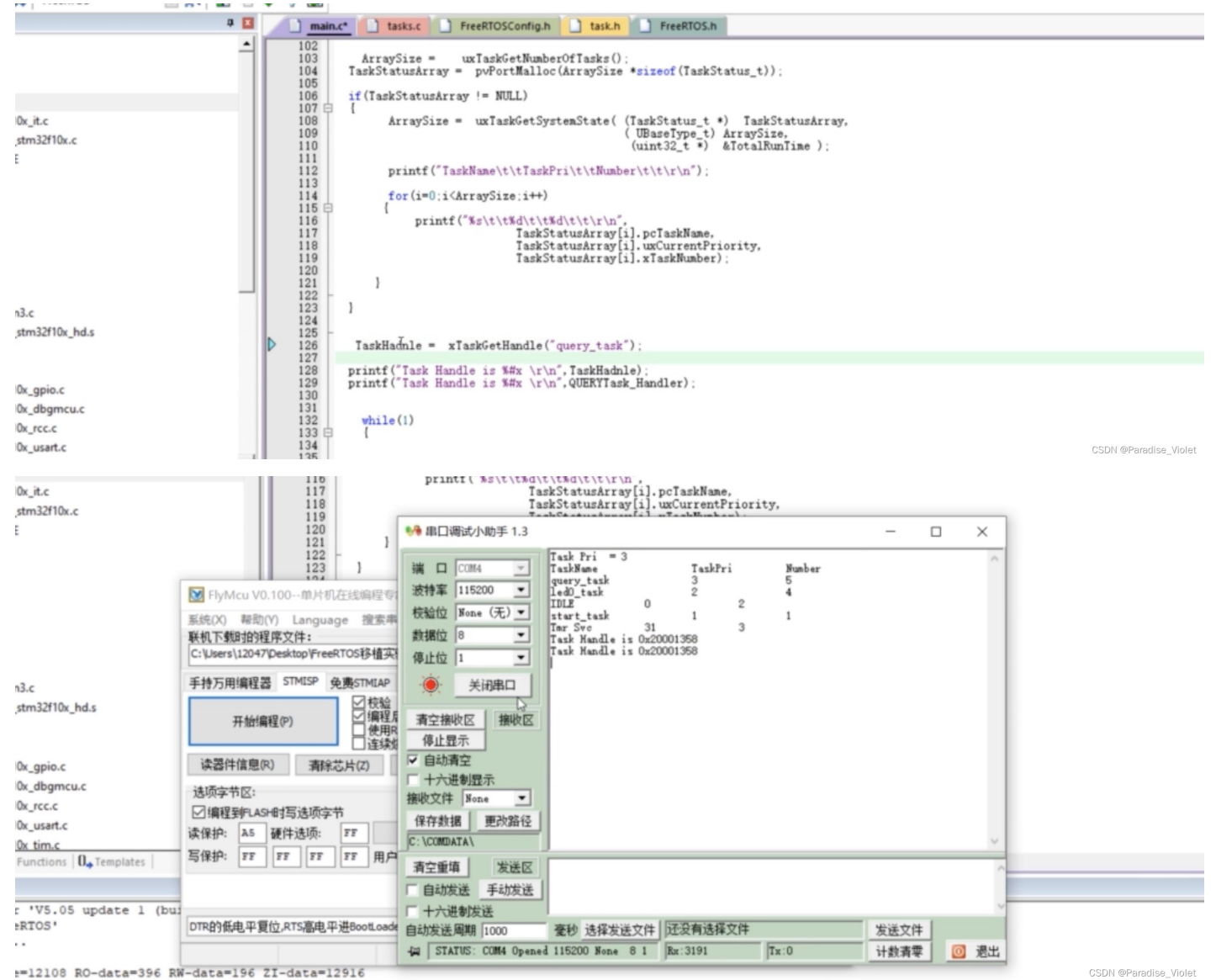
```
TaskHandle_t xTaskGetHandle( const char * pcNameToQuery )
```

参数：

pcNameToQuery: 任务名，C 语言字符串。

CSDN @Paradise_Violet

通过任务名字来获取句柄



两者是一致的

函数uxTaskGetStackHighWaterMark()

每个任务都有自己的堆栈，堆栈的总大小在创建任务的时候就确定了，此函数用于检查任务从创建好到现在的历史剩余最小值，这个值越小说明任务堆栈溢出的可能性就越大！FreeRTOS 把这个历史剩余最小值叫做“高水位线”。此函数相对来说会多耗费一点时间，所以在代码调试阶段可以使用，产品发布的时候最好不要使用。要使用此函数的话宏 INCLUDE_uxTaskGetStackHighWaterMark 必须为 1，此函数原型如下：

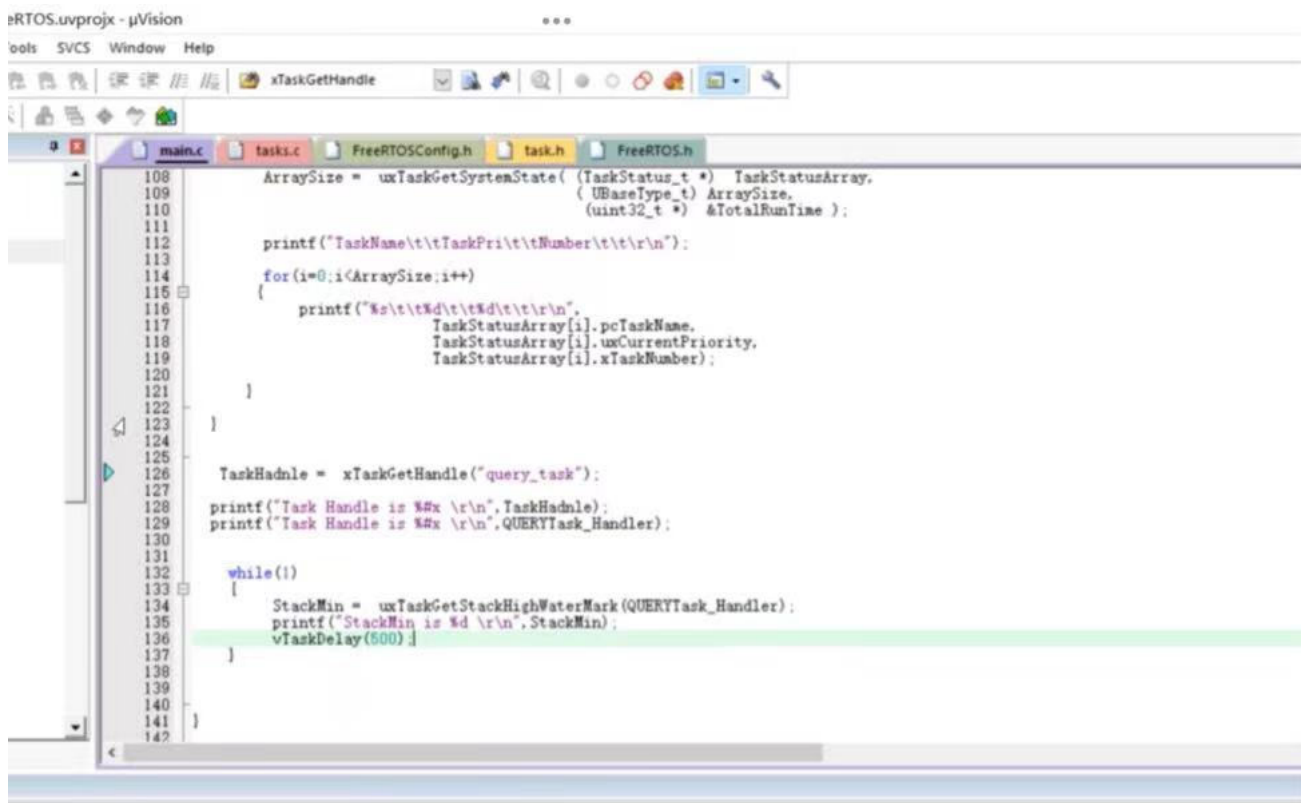
```
UBaseType_t uxTaskGetStackHighWaterMark( TaskHandle_t xTask )
```

参数：

xTask: 要查询的任务的任务句柄，当这个参数为 NULL 的话说明查询自身任务(即调用函数 uxTaskGetStackHighWaterMark()的任务)的“高水位线”。

返回值： 任务堆栈的“高水位线”值，也就是堆栈的历史剩余最小值。

来个延时，不然放在while里面飞的很快



CSDN @Paradise_Violet



CSDN @Paradise_Violet

剩余148，我们定义200，说明我们用了 $(200-148) * 4 = 208$ 个字节

函数eTaskGetState()

10、函数 eTaskGetState()

此函数用于查询某个任务的运行状态，比如：运行态、阻塞态、挂起态、就绪态等，返回值是个枚举类型。要使用此函数的话宏 INCLUDE_eTaskGetState 必须为 1，函数原型如下：

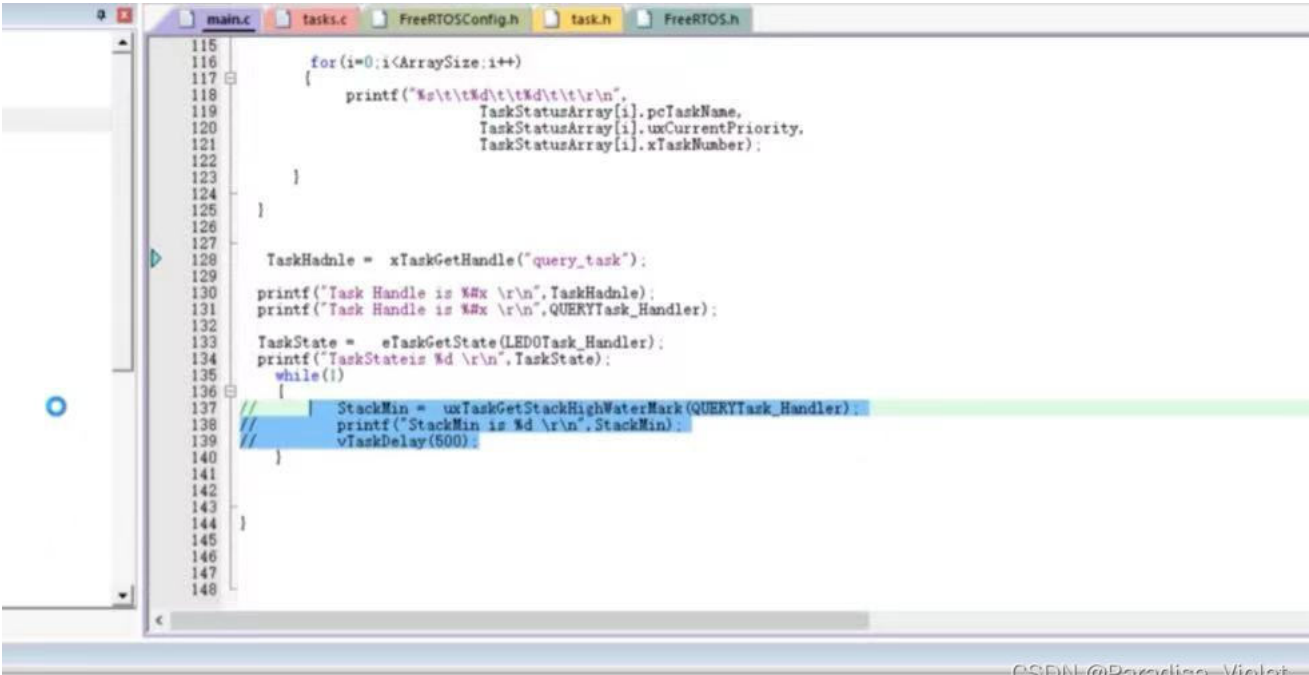
```
eTaskState eTaskGetState( TaskHandle_t xTask )
```

参数：

xTask：要查询的任务的任务句柄。

返回值：返回值为 eTaskState 类型，这是个枚举类型，在文件 task.h 中有定义，前面讲解函数 vTaskGetInfo()的时候已经讲过了。

CSDN @Paradise_Violet



CSDN @Paradise_Violet



CSDN @Paradise_Violet

1---就是就绪态意思

函数vTaskList()

此函数会创建一个表格来描述每个任务的详细信息，如图 11.2.1 所示：

Name	State	Priority	Stack	Num

Print	R	4	331	29
Math7	R	0	417	7
Math8	R	0	407	8
QConsB2	R	0	53	14
QProdB5	R	0	52	17
QConsB4	R	0	53	16
SEM1	R	0	50	27
SEM1	R	0	50	28
IDLE	R	0	64	0
Math1	R	0	436	1
Math2	R	0	436	2

图 11.2.1 任务状态信息表

表中的信息如下：

- Name： 创建任务的时候给任务分配的名字。
- State： 任务的状态信息，B 是阻塞态，R 是就绪态，S 是挂起态，D 是删除态。
- Priority： 任务优先级。
- Stack： 任务堆栈的“高水位线”，就是堆栈历史最小剩余大小。
- Num： 任务编号，这个编号是唯一的，当多个任务使用同一个任务名的时候可以通过此编号来做区分。

函数原型如下：

```
void vTaskList( char * pcWriteBuffer )
```

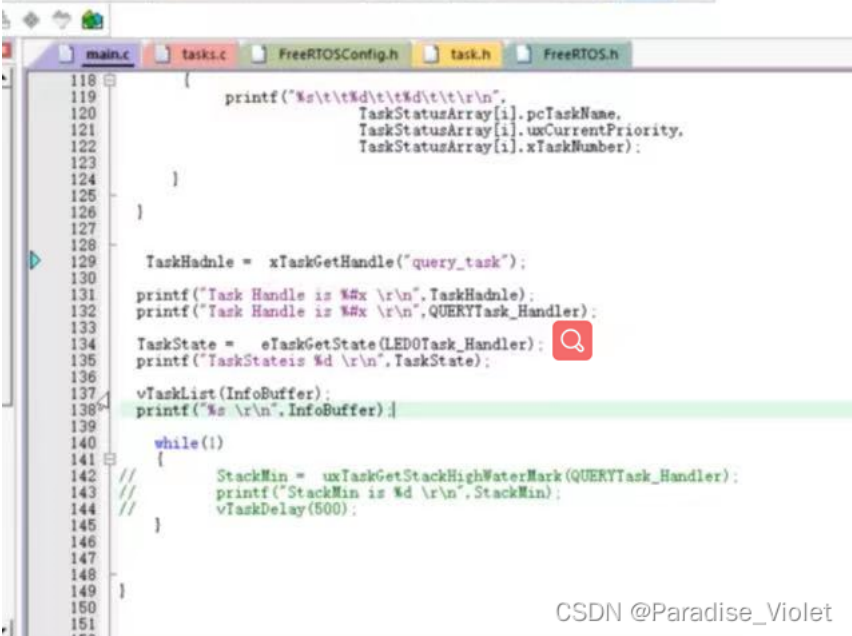
参数：

pcWriteBuffer： 保存任务状态信息表的存储区。存储区要足够大来保存任务状态信息表。

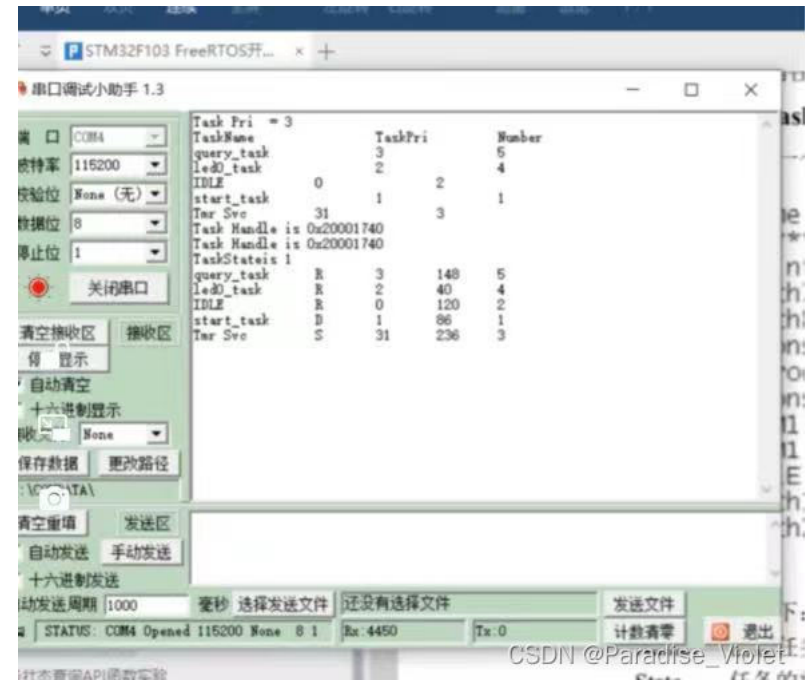
返回值： 无

CSDN @Paradise_Violet

```
char InfoBuffer[1000];
```



CSDN @Paradise_Violet



开始任务是删除态，因为创建后就删除了