任务的创建

(1) 函数原型

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                        const char * const pcName,
                        unsigned short usStackDepth,
                        void *pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t *pxCreatedTask );
```

(2) 函数参数说明

<1> **pvTaskCode**，任务实现部分，任务函数原型如下

```
typedef void (*TaskFunction_t)( void * );
```

从任务函数的原型可以看出，任务函数是一个无返回值，可以传递一个参数的函数。所需要实现的功能全部在任务函数中，因此任务函数一般都执行一个无限循环，在无限循环中的代码只要该任务还存在，那么就会一直执行。

<2> **pcName**，任务函数的名字，它表示创建的任务函数名称。为了省事，一般来说任务函数的名称和任务函数同名即可。需要注意的是任务函数的名称具有长度限制，在配置文件FreeRTOSConfig.h中configMAX_TASK_NAME_LEN定义，可以根据需要修改。使用函数xTaskGetHandle可以任务函数的名称获取到当前任务的句柄pxCreatedTask，其中xTaskGetHandle函数原型如下

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

<3> **usStackDepth**，任务函数需要的堆栈空间。每一个任务都有自己唯一的堆栈空间，栈空间的大小由该参数指定。需要注意的是，该参数的单位是字长，而非字节。如果在一个系统架构中一个字长由四个字节，那么创建任务时该参数值为256时，那么表示给任务函数分配的栈空间为256*4=1024 bytes。如果一个系统存在比较多的任务，那么就需要对不同的任务进行分配相应的堆栈空间，如果堆栈空间不够的话，那么就需要调整堆栈空间的大小，在配置文件FreeRTOSConfig.h中configTOTAL_HEAP_SIZE可以调整堆栈空间大小，堆栈空间可调整的大小和当前系统息息相关，configTOTAL_HEAP_SIZE定义的堆栈空间必须小于当前系统所支持的堆栈空间。额外说明，嵌入式操作系统FreeRTOS，具有5个堆栈空间管理文件，根据实际需要选择某个堆栈管理文件即可，它们的区别如下：

- heap_1.c -最简单的，不允许释放内存。
- heap_2.c -允许释放内存，但不合并相邻的空闲块。
- heap_3.c -简单地包装了标准的malloc()和free()以保证线程安全。
- heap_4.c -合并相邻的空闲块以避免碎片。 包含绝对地址放置选项。
- heap_5.c -与heap_4相同，能够跨多个不相邻的内存区域跨越堆。

一般情况下，建议使用heap_4.c来管理堆栈空间。详情参照"https://www.freertos.org/a00111.html"

<4> **pvParameters**，任务函数参数。在创建任务时，如果需要给任务函数传递信息，那么就需要使用该参数；如果创建任务时不需要传递信息给任务，那么使用NULL即可。

示例一：

A、创建任务，并传递信息

```
uint32_t value = 1024;
BaseType_t create_success = xTaskCreate(test_task_1,
                                        "test_task_1",
                                        1000,
                                        (void*)value,
                                        1,
                                        NULL);
if (create_success != pdPASS)
{
    Log(FATAL, RED"Create test_task_1 failed.");
}
```

B、任务接收信息

```
static void test_task_1(void* arg)
{
    uint32_t value = (uint32_t)arg;

    while (1)
    {
        Log(DEBUG, BLUE"value = %d\n", value);

        /* Delay one second. */
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

示例二:

A、信息类型定义

```
typedef struct
{
    const char* name;
    uint8_t age;
}person_t;
```

B、创建任务，并传递消息

```
person_t person =
{
    .name = "Candy",
    .age  = 18,
};
xTaskCreate(test_task_2, "test_task_2", 1000, &person, 1, NULL);
```

C、任务接收消息

```
static void test_task_2(void* arg)
{
    person_t* p = (person_t*)arg;

    while (1)
    {
        Log(DEBUG, MAGENTA"%s", "name: %s, age: %hhu\n", p->name, p->age);

        /* Delay one second. */
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

<5> **uxPriority**，任务的优先级。指的是给创建的任务指定执行优先级，在多任务的项目中，不同任务往往有不同的执行等级，即优先级。优先级值越高，表示执行等级越高，该任务就越容易被执行。最低优先级为0，最高优先级为configMAX_PRIORITIES-1，最大优先级的值默认是5，在配置文件FreeRTOSConfig.h中configMAX_PRIORITIES定义。需要注意的是，在多任务项目中，往往会因为各个任务优先级分配不好，会出现意料不到的执行结果，如果出现该情况，可以尝试的调整各个任务的优先级。如果在创建任务时不知道优先级给多高，默认设置为2即可，后面任务多了，业务复杂时，再根据实际需求调整。

<6> **pxCreatedTask**，任务句柄。该参数用来表示创建任务的标识。它被其它API所使用的，比如xTaskNotify。如果在其它地方不需要使用任务句柄，那么可以直接使用NULL替代。比如在<4>创建任务时，句柄为NULL。

(3) 函数返回值说明

<1> **pdPASS**，表示任务创建成功。其中pdPASS的定义如下：

```
typedef long BaseType_t;
#define pdTRUE          ( ( BaseType_t ) 1 )
#define pdPASS          ( pdTRUE )
```

<2> errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY，表示创建任务失败，没有足够的堆栈空间给任务执行。此时就需要去调整参数usStackDepth的大小。如果该参数满足条件，但还是无法创建任务，那么就需要检查下在配置文件FreeRTOSConfig.h中总共的堆栈空间大小宏定义configTOTAL_HEAP_SIZE，根据实际情况调整大小即可。

(4) 任务创建失败捕获

配置文件FreeRTOSConfig.h中configUSE_MALLOC_FAILED_HOOK值为1时，要实现内存分配失败函数vApplicationMallocFailedHook，它的函数原型为：

```
void vApplicationMallocFailedHook( void )
```

该函数一般用于警示作用，当存在分配内存失败时，就会调用该函数。它是一个钩子函数，所谓的钩子函数，指的是它不是必须存在的，但是也可以存在让开发者去实现处理一些事情的函数。虽然钩子函数的存在对功能没有任何影响，可有可无，但是钩子函数随处可见，不管是Windows，Linux，还是其它嵌入式操作系统。下面给出一种最简单的实现：

```
void vApplicationMallocFailedHook( void )
{
    Log(FATAL, RED"@@@@@@@@@@ Malloc memory failed @@@@@@@@@@\n");
    Log(FATAL, RED"@@@@@@@@@@ Malloc memory failed @@@@@@@@@@\n");
    Log(FATAL, RED"@@@@@@@@@@ Malloc memory failed @@@@@@@@@@\n");

    while(1);
}
```

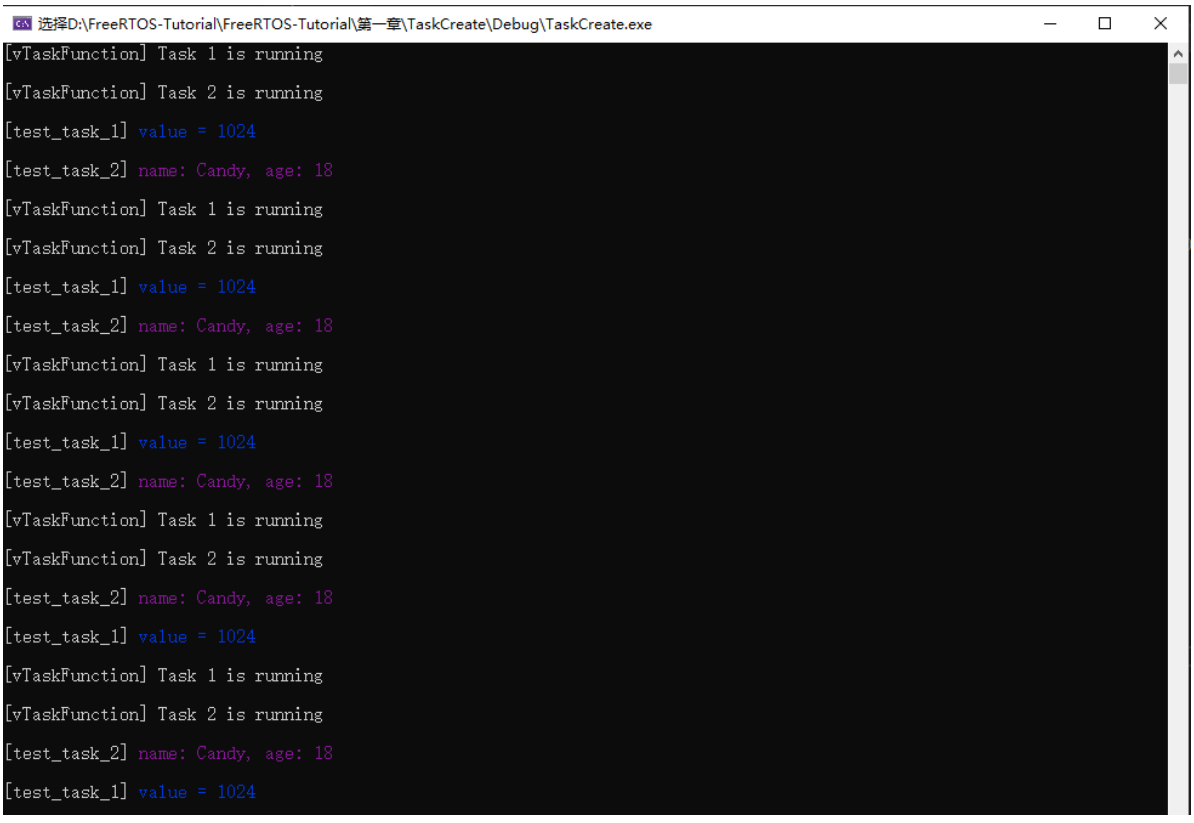
(5) 多任务示例程序

```
/* FreeRTOS.org includes. */
#include "FreeRTOS.h"
#include "task.h"
#include <stdio.h>

/* Demo includes. */
#include "supporting_functions.h"
#include "common.h"

/* Used as a loop counter to create a very crude delay. */
#define mainDELAY_LOOP_COUNT        ( 0xffffff )

/* The task function. */
void vTaskFunction(void* pvParameters);

/* Define the strings that will be passed in as the task parameters.  These are
defined const and off the stack to ensure they remain valid when the tasks are
executing. */
const char* pcTextForTask1 = "Task 1 is running\r\n";
const char* pcTextForTask2 = "Task 2 is running\r\n";

typedef struct
{
    const char* name;
    uint8_t age;
}person_t;

static void test_task_1(void* arg)
{
    uint32_t value = (uint32_t)arg;

    while (1)
    {
        Log(DEBUG, BLUE"value = %d", value);

        /* Delay one second. */
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

static void test_task_2(void* arg)
{
    person_t* p = (person_t*)arg;

    while (1)
```

```
    {
        Log(DEBUG, MAGENTA"%s", "name: %s, age: %hhu\n", p->name, p->age);

        /* Delay one second. */
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

static void test_task_3(void* arg)
{
    char* buffer = (char*)pvPortMalloc(4096);

    while (1)
    {
        Log(DEBUG, CYAN"test_task_3");

        /* Delay one second. */
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

/*-----------------------------------------------------------*/

int main(void)
{
    /* Create one of the two tasks. */
    xTaskCreate(vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL);
     xTaskCreate(vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 1, NULL);

    uint32_t value = 1024;
    BaseType_t create_success = xTaskCreate(test_task_1, "test_task_1", 1000,
(void*)value, 1, NULL);
    if (create_success != pdPASS)
    {
        Log(FATAL, RED"Create test_task_1 failed.");
    }

    person_t person =
    {
        .name = "Candy",
        .age = 18,
    };
    create_success = xTaskCreate(test_task_2, "test_task_2", 1000, &person, 1,
NULL);
    if (create_success != pdPASS)
    {
        Log(FATAL, RED"Create test_task_2 failed.");
    }

#if 0
    create_success = xTaskCreate(test_task_3, "test_task_3", 128, NULL, 1,
NULL);
    if (create_success != pdPASS)
    {
        Log(FATAL, RED"Create test_task_3 failed.");
    }
#endif
```

```
    /* Start the scheduler to start the tasks executing. */
    vTaskStartScheduler();

    /* The following line should never be reached because vTaskStartScheduler()
    will only return if there was not enough FreeRTOS heap memory available to
    create the Idle and (if configured) Timer tasks.  Heap management, and
    techniques for trapping heap exhaustion, are described in the book text. */
    while (1);

    return 0;
}
/*-----------------------------------------------------------*/

void vTaskFunction(void* pvParameters)
{
    char* pcTaskName;
    volatile uint32_t ul;

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = (char*)pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    while(1)
    {
        /* Print out the name of this task. */
        Log(DEBUG, WHITE"%s", pcTaskName);

        /* Delay one second. */
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

执行结果：



本教程示例源代码地址："https://github.com/Andy001847/FreeRTOS-Tutorial"