第一章 FreeRTOS 简介

本文主要参考安富莱 FreeRTOS 教程和乐鑫 API。乐鑫 SDK 自带 FreeRTOS，项目开发是避免不了多任务的使用，所以 FreeRTOS 还是必须要了解下。

1. 学习目的及目标

- 了解 FreeRTOS 现状
- 了解 FreeRTOS 常用 API
- 不对 API 使用进行讲解

2. FreeRTOS 简介

由于大家都比较关心这个问题，所以把这个问题放在第一章简单的说说。FreeRTOS 最大的优势就是开源免费，商业使用的话不需要用户公开源代码，也不存在任何版权问题，是当前小型嵌入式操作系统市场使用率最高的。去年的全球嵌入式市场分析报告中，FreeRTOS 占据了 20%。并有以下优点：

2.1. 移植容易

FreeRTOS 的移植比较简单，只需要用户添加需要的源码文件，不需要做任何的底层工作，再添加三个宏定义即可。**并且乐鑫已经在 ESP-IDF 移植好了，直接使用 API 接口即可。**

2.2. 上手简单

作为开源免费的 RTOS，官方配套的手册在 RTOS 的基础知识说明、API 函数说明及其举例方面做的都非常好，用户上手比较容易。随着对 ESP32 的学习，大家会体会到这一点。

2.3. 资源丰富

FreeRTOS 的资料丰富程度堪比 STM32F103。

- 入门手册下载地址（版本 V1.3.2）
- API 参考手册（FreeRTOS V8.1.2）

再次感谢安富莱的开源教程。

3. FreeRTOS 任务操作

本节由开源维护者伽岚整理。

3.1. 创建任务函数 xTaskCreate();

创建一个新任务并将其添加到准备运行的任务列表中。

在内部，在 FreeRTOS 实现中，任务使用两块内存。第一个块用于保存任务的数据结构。任务将第二个块用作其堆栈。如果使用 xTaskCreate（）创建任务，则会在 xTaskCreate（）函数内自动动态分配两个内存块。

函数原型	static BaseType_t xTaskCreate(TaskFunction_t pvTaskCode, const char const pcName, const uint32_t usStackDepth, void const pvParameters, UBaseType_t uxPriority, const pvCreatedTask)

函数功能	MQTT Client 取消订阅主题函数
参数	[in] pvTaskCode：指向任务输入功能的指针。必须实现任务以永不返回（即连续循环）。 [in] pcName：任务的描述性名称。这主要用于方便调试。默认值为 16 字节。 [in] usStackDepth：指定为字节数的任务堆栈的大小。请注意，这与 vanilla FreeRTOS 不同。 [in] pvParameters：指针将用作正在创建的任务的参数。 [in] uxPriority：任务应运行的优先级。包含 MPU 支持的系统可以选择通过设置优先级参数的位 portPRIVILEGE_BIT 以特权（系统）模式创建任务。例如，要以优先级 2 创建特权任务，应将 uxPriority 参数设置为（2 | portPRIVILEGE_BIT）。 [in] pvCreatedTask：用于传回一个句柄，通过该句柄可以引用创建的任务。 [in]xCoreID：如果值为 tskNO_AFFINITY，则创建的任务不会固定到任何 CPU，并且调度程序可以在任何可用的核心上运行它。其他值表示任务应固定到的 CPU 的索引号。指定大于（portNUM_PROCESSORS - 1）的值将导致函数失败。
返回值	如果任务已成功创建并添加到就绪列表，则为 pdPASS

3.2. 删除任务函数 vTaskDelete();

从 RTOS 实时内核的管理中删除任务。正在删除的任务将从所有就绪，阻止，暂停和事件列表中删除。

3.3. 延迟函数 vTaskDelay();

延迟给定数量的滴答的任务。任务保持阻塞的实际时间取决于滴答率。常数 portTICK_PERIOD_MS 可用于根据滴答速率计算实时，具有一个滴答周期的分辨率。

- 表示相对延迟 1 个时钟滴答(tick)数,每个时钟滴答 tick 约等于 10 毫秒

  vTaskDelay(1);

- 表示任务延时 1 秒，portTICK_RATE_MS 为 tick 与毫秒的比例，1000/portTICK_RATE_MS=100 tick 滴答

  vTaskDelay(1000/portTICK_RATE_MS);

3.4. 返回&设置任务优先级

- 获取任何任务的优先级 uxTaskPriorityGet();
- 设置任何任务的优先级:uvTaskPrioritySet();

3.5. 返回任务的状态:eTaskGetState();

获取任何任务的状态。 状态由 eTaskState 枚举类型编码。

3.6. 任务挂起与恢复:vTaskSuspend();

挂起任务。挂起后，无论优先级如何，任务都不会获得任何微控制器处理时间。对 vTaskSuspend 的调用不是累积的,即在同一任务上调用 vTaskSuspend（）两次仍然只需要调用 vTaskResume（）来准备挂起的任务。

3.7. 挂起与恢复全部任务:vTaskSuspendAll();

暂停调度程序而不禁用中断。 调度程序挂起时不会发生上下文切换。在调用 vTaskSuspendAll（）之后，调用任务将继续执行，而不会被调换掉，直到调用 xTaskResumeAll（）为止。在调度程序挂起时，不得调用有可能导致上下文切换的 API 函数（例如，vTaskDelayUntil（），xQueueSend（）等）。

3.8. 获取 tick 计数：xTaskGetTickCount();

3.9. 获取任务总数:uxTaskGetNumberOfTasks();

3.10. 获取 CPU 空闲任务句柄:xTaskGetIdleTaskHandle();

获取当前 CPU 的空闲任务句柄。 返回:空闲任务的句柄。在调度程序启动之前调用 xTaskGetIdleTaskHandle（）无效。

3.11. 任务通知的发送:xTaskNotify();

3.12. 任务通知等待:xTaskNotifyWait();

更多更详细接口请参考官方指南。

4. FreeRTOS 事件标志组

事件标志组是实现多任务同步的有效机制之一。也许有不理解的初学者会问采用事件标志组多麻烦，搞个全局变量不是更简单？其实不然，在裸机编程时，使用全局变量的确比较方便，但是在加上 RTOS 后就是另一种情况了。使用全局变量相比事件标志组主要有如下三个问题：

- 使用事件标志组可以让 RTOS 内核有效地管理任务，而全局变量是无法做到的，任务的超时等机制需要用户自己去实现。
- 使用了全局变量就要防止多任务的访问冲突，而使用事件标志组则处理好了这个问题，用户无需担心。
- 使用事件标志组可以有效地解决中断服务程序和任务之间的同步问题。

4.1. 事件标志组常用 API 函数

- 创建事件标志组：xEventGroupCreate()
- 等待事件标志被设置：xEventGroupWaitBits()
- 设置指定的事件标志位为 1：xEventGroupSetBits()

5. FreeRTOS 定时器组

和我们定时器章类似。

6. FreeRTOS 消息队列

消息队列就是通过 RTOS 内核提供的服务，任务或中断服务子程序可以将一个消息（注意，FreeRTOS 消息队列传递的是实际数据，并不是数据地址，RTX，uCOS-II 和 uCOS-III 是传递的地址）放入到队列。同样，一个或者多个任务可以通过 RTOS 内核服务从队列中得到消息。通常，先进入消息队列的消息先传给任务，也就是说，任务先得到的是最先进入到消息队列的消息，即先进先出的原则（FIFO），FreeRTOS 的消息队列支持 FIFO 和 LIFO 两种数据存取方式。

也许有不理解的初学者会问采用消息队列多麻烦，搞个全局数组不是更简单，其实不然。在裸机编程时，使用全局数组的确比较方便，但是在加上 RTOS 后就是另一种情况了。相比消息队列，使用全局数组主要有如下四个问题：

- 使用消息队列可以让 RTOS 内核有效地管理任务，而全局数组是无法做到的，任务的超时等机制需要用户自己去实现。
- 使用了全局数组就要防止多任务的访问冲突，而使用消息队列则处理好了这个问题，用户无需担心。
- 使用消息队列可以有效地解决中断服务程序与任务之间消息传递的问题。

- FIFO 机制更有利于数据的处理。

6.1. 消息队列常用 API 函数

- 创建消息队列：xQueueCreate ()
- 任务中消息发送：xQueueSend ()
- 用于中断服务程序中消息发送：xQueueSendFromISR ()
- 用于接收消息队列中的数据：xQueueReceive ()

7. FreeRTOS 总结

- 更多接口参考安富莱的 1 千页 FreeRTOS 教程。
- 源码地址：https://github.com/HX-IoT/