FreeRTOS学习笔记（六）

FreeRTOS的延时函数（INCLUDE_vTaskDelay置1）

- **相对模式（vTaskDelay();）**

```
/**
* 描述：系统的延时函数
* 参数：需要延时的时间
*/
void vTaskDelay(const TickType_t xTicksToDelay);
```

- 函数的执行过程
  1. 延时时间由参数xTicksToDelay来确定，参数要大于0，否则直接调用portYIELD();进行任务切换
  2. 使用vTaskSuspendAll()挂起任务调度器
  3. prvAddCurrentTaskToDelayedList()将要延时的任务添加到延时列表中
  4. xTaskResumeAll()恢复任务调度器
  5. 如果xTaskResumeAll()没有进行任务的调度，就在这里进行任务的调度portYIELD_WITHIN_API()

- **绝对模式（vTaskDelayUntil();）**

```
/**
* 描述：延时一个绝对时间
* 参数：pxPreviousWakeTime：上一次任务延时结束被唤醒的时间点
*       xTimeIncrement    ：时钟节拍数
*/
void vTaskDelayUntil(TickType_t * const pxPreviousWakeTime, const TickType_t xTimeIncrement);
```

- 函数的执行过程
  1. 使用vTaskSuspendAll()挂起任务调度器
  2. 记录进入延时函数的时间点值，保存在xConstTickCount中
  3. 计算下一次要唤醒的时间点，保存在xTimeToWake中
  4. prvAddCurrentTaskToDelayedList()将要延时的任务添加到延时列表中
  5. xTaskResumeAll()恢复任务调度器

FreeRTOS系统时钟节拍

简述：xTickCount就是时钟节拍计数器，滴答定时器每中断一次，计数器就加一

FreeRTOS队列

队列的特性

任务间通信，任务与中断间的通信，都可以用消息队列实现。

- 多任务访问

- 同一个消息队列是可以被多个任务访问的

- 出队阻塞
  - 也就是，任务在读取队列中的消息的时候，如果队列中没有消息，任务可以选择等和不等或者等多长时间

- 入队阻塞
  - 同出队阻塞，任务在给队列发送消息的时候，如果队列消息满了，任务可以选择等和不等或者等多长时间

队列的API函数

```
/**
* 描述：队列创建函数（一个宏，调用xQueueGenericCreate）
* 参数：uxQueueLength：创建队列的长度
*      uxItemSize   ：队列中每一个项的函数
* 返回：null  ：创建失败
*      其他值：创建成功之后，队列句柄
*/
QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength,
                           UBaseType_t uxItemSize);

/**
* 描述：静态创建队列函数（一个宏，调用xQueueGenericCreateStatic）
* 参数：uxQueueLength        ：创建队列的长度
*      uxItemSize          ：队列中每一个项的长度
*      pucQueueStorageBuffer：指向消息的存储区
*      pxQueueBuffer       ：StaticQueue_t类型的指针，保存队列结构体
* 返回：null  ：创建失败
*      其他值：创建成功之后，队列句柄
*/
QueueHandle_t xQueueCreateStatic(UBaseType_t    uxQueueLength,
                                 UBaseType_t    uxItemSize,
                                 uint8_t       *pucQueueStorageBuffer,
                                 StaticQueue_t *pxQueueBuffer);

/**
* 描述：动态创建消息队列
* 参数：uxQueueLength：创建队列的长度
*      uxItemSize   ：队列中每一个项的函数
*      ucQueueType  ：队列的类型（信号量，队列集，消息队列等）
* 返回：null  ：创建失败
*      其他值：创建成功之后，队列句柄
*/
QueueHandle_t xQueueGenericCreate(const UBaseType_t uxQueueLength,
                                  const UBaseType_t uxItemSize,
                                  const uint8_t     ucQueueType);

/**
* 描述：静态创建队列函数
* 参数：uxQueueLength  ：创建队列的长度
*      uxItemSize     ：队列中每一个项的长度
```

```
*      pucQueueStorage：指向消息的存储区
*      pxStaticQueue  ：StaticQueue_t类型的指针，保存队列结构体
*      ucQueueType    ：队列类型
* 返回：null  ：创建失败
*      其他值：创建成功之后，队列句柄
*/
QueueHandle_t xQueueGenericCreateStatic(const UBaseType_t uxQueueLength,
                                        const UBaseType_t uxItemSize,
                                        uint8_t          *pucQueueStorage,
                                        StaticQueue_t    *pxStaticQueue,
                                        const uint8_t     ucQueueType )
```

入队API

```
/**
* 描述：入队函数（尾部入队）
* 参数：xQueue        ：队列句柄
*      pvItemToQueue：要发送的消息
*      xTicksToWait  ：阻塞时间
* 返回：pdPASS        ：发送成功
*      errQUEUE_FULL：队列已满，发送失败
*/
BaseType_t xQueueSend(QueueHandle_t xQueue,
                      const void   *pvItemToQueue,
                      TickType_t    xTicksToWait);
/**
* 描述：入队函数（尾部入队）
* 参数：xQueue        ：队列句柄
*      pvItemToQueue：要发送的消息
*      xTicksToWait  ：阻塞时间
* 返回：pdPASS        ：发送成功
*      errQUEUE_FULL：队列已满，发送失败
*/
BaseType_t xQueueSendToBack(QueueHandle_t xQueue,
                            const void *pvItemToQueue,
                            TickType_t         xTicksToWait);
/**
* 描述：入队函数（头部入队）
* 参数：xQueue        ：队列句柄
*      pvItemToQueue：要发送的消息
*      xTicksToWait  ：阻塞时间
* 返回：pdPASS        ：发送成功
*      errQUEUE_FULL：队列已满，发送失败
*/
BaseType_t xQueueSendToToFront(QueueHandle_t xQueue,
                               const void     *pvItemToQueue,
                               TickType_t    xTicksToWait);
```

```
/*-----------------------------------------------------------*/
/* 上面三个函数后面加上FromISR，就是在中断服务函数中使用的函数 */
/*-----------------------------------------------------------*/

/**
* 描述：队列上锁
*/
prvLockQueue(Queue_t pxQueue);

/**
* 描述：队列解锁
*/
prvUnlockQueue(Queue_t pxQueue);

/**
* 描述：从队列中读取消息（读取后删除）
* 参数：xQueue       ：队列句柄
*       pvBuffer     ：保存数据的缓存区
*       xTicksToWait：阻塞时间
* 返回：pdTRUE ：读取成功
*       pdFALSE：读取失败
*/
BaseType_t xQueueReceive(QueueHandle_t xQueue,
                         void         *pvBuffer,
                         TickType_t    xTicksToWait);
/**
* 描述：从队列中读取消息（读取后不删除）
* 参数：xQueue       ：队列句柄
*       pvBuffer     ：保存数据的缓存区
*       xTicksToWait：阻塞时间
* 返回：pdTRUE ：读取成功
*       pdFALSE：读取失败
*/
BaseType_t xQueuePeek(QueueHandle_t xQueue,
                      void         *pvBuffer,
                      TickType_t    xTicksToWait);

/*-----------------------------------------------------------*/
/* 上面两个函数后面加上FromISR，就是在中断服务函数中使用的函数 */
/*-----------------------------------------------------------*/
```