---

title: 继续学习FreeRTOS消息队列 author: 杰杰 top: false cover: false toc: true mathjax: false date: 2019-10-05 09:34:58 img: coverImg: password: summary: tags:

- FreeRTOS
- RTOS
- 操作系统 categories: 操作系统

> 写在前面：杰杰这个月很忙~所以并没有时间更新，现在健身房闭馆装修，晚上有空就更新一下！其实在公众号没更新的这段日子，每天都有兄弟在来关注我的公众号，这让我受宠若惊，在这里谢谢大家的支持啦！！谢谢^

在这里我们就跟着火哥的书来学习一下FreeRTOS的消息队列，这本书我觉得写得很好，基本都讲解到了，关于什么是消息队列，就请大家去看书，基础知识我暂时不说了。

声明：本书绝大部分内容来自《FreeRTOS 内核实现与应用开发实战指南—基于野火 STM32 全系列（M3/4/7）开发板》，如涉及侵权请联系杰杰删除

FreeRTOS的消息队列支持

- FreeRTOS 中使用队列数据结构实现任务异步通信工作，具有如下特性：
- 消息支持先进先出方式排队，支持异步读写工作方式。
- 读写队列均支持超时机制。
- 消息支持后进先出方式排队， 往队首发送消息（LIFO） 。
- 可以允许不同长度（不超过队列节点最大值）的任意类型消息。
- 一个任务能够从任意一个消息队列接收和发送消息。
- 多个任务能够从同一个消息队列接收和发送消息。
- 当队列使用结束后，可以通过删除队列函数进行删除。

FreeRTOS队列的特点

一般来说，鱼与熊掌不可兼得，如果数据太多，那数据传输的速度必然是会慢下来，而如果采用引用传递的方式，当原始数据被修改的时候，数据有变得不安全，但是FreeRTOS支持拷贝与引用的方式进行数据的传输，变得更加灵活。 队列是通过拷贝传递数据的，但这并不妨碍队列通过引用来传递数据。当信息的大小到达一个临界点后，逐字节拷贝整个信息是不实际的，可以定义一个指向数据区域的指针，将指针传递即可。这种方法在物联网中是非常常用的。

消息队列控制块

其实消息队列不仅仅是用于当做消息队列，FreeRTOS还把他当做信号量的数据结构来使用

```
typedef struct QueueDefinition
{
    int8_t *pcHead;             /* 指向队列存储区起始位置,即第一个队列项 */
    int8_t *pcTail;             /* 指向队列存储区结束后的下一个字节 */
    int8_t *pcWriteTo;          /* 指向下队列存储区的下一个空闲位置 */


    union                       /* 使用联合体用来确保两个互斥的结构体成员不会同时出现
```

```
*/
    {
        int8_t *pcReadFrom;     /* 当结构体用于队列时,这个字段指向出队项目中的最后一个. */
        UBaseType_t uxRecursiveCallCount;/* 当结构体用于互斥量时,用作计数器,保存递归互斥量被"获取"的次数. */
    } u;


    List_t xTasksWaitingToSend;      /* 因为等待入队而阻塞的任务列表,按照优先级顺序存储 */
    List_t xTasksWaitingToReceive;   /* 因为等待队列项而阻塞的任务列表,按照优先级顺序存储 */


    volatile UBaseType_t uxMessagesWaiting;/*< 当前队列的队列项数目 */
    UBaseType_t uxLength;            /* 队列项的数目 */
    UBaseType_t uxItemSize;          /* 每个队列项的大小 */


    volatile BaseType_t xRxLock;   /* 队列上锁后,存储从队列收到的列表项数目，如果队列没有上锁，设置为queueUNLOCKED */
    volatile BaseType_t xTxLock;   /* 队列上锁后,存储发送到队列的列表项数目，如果队列没有上锁，设置为queueUNLOCKED */

    /* 删除部分源码 */

} xQUEUE;

typedef xQUEUE Queue_t;
```

先过一遍消息队列的数据结构，其实没啥东西的，记不住也没啥大问题，下面会用到就行了。

创建消息队列

FreeRTOS创建队列API函数是xQueueCreate()，但其实这是一个宏。真正被执行的函数是xQueueGenericCreate()，我们称这个函数为通用队列创建函数。

```
        QueueHandle_t xQueueGenericCreate( const UBaseType_t uxQueueLength, const UBaseType_t uxItemSize, const uint8_t ucQueueType )
        {
        Queue_t *pxNewQueue;
        size_t xQueueSizeInBytes;
        uint8_t *pucQueueStorage;

                configASSERT( uxQueueLength > ( UBaseType_t ) 0 );

                if( uxItemSize == ( UBaseType_t ) 0 )
                {
                        /* 如果 uxItemSize 为 0，也就是单个消息空间大小为 0，这样子就不
```

```
需要申请内存了，那么 xQueueSizeInBytes 也设置为 0 即可，设置为 0 是可以的，用作信号
量的时候这个就可以设置为 0。*/
                        xQueueSizeInBytes = ( size_t ) 0;
                }
                else
                {
                        /* 分配足够消息存储空间，空间的大小为队列长度*单个消息大小 */
                        xQueueSizeInBytes = ( size_t ) ( uxQueueLength *
uxItemSize ); /*lint !e961 MISRA exception as the casts are only redundant for
some ports. */
                }
                /* FreeRTOS 调用 pvPortMalloc()函数向系统申请内存空间，内存大
小为消息队列控制块大小加上消息存储空间大小，因为这段内存空间是需要保证连续的 */
                pxNewQueue = ( Queue_t * ) pvPortMalloc( sizeof( Queue_t ) +
xQueueSizeInBytes );

                if( pxNewQueue != NULL )
                {
                        /* 计算出消息存储空间的起始地址 */
                        pucQueueStorage = ( ( uint8_t * ) pxNewQueue ) + sizeof(
Queue_t );

                        #if( configSUPPORT_STATIC_ALLOCATION == 1 )
                        {
                                pxNewQueue->ucStaticallyAllocated = pdFALSE;
                        }
                        #endif /* configSUPPORT_STATIC_ALLOCATION */

                        prvInitialiseNewQueue( uxQueueLength, uxItemSize,
pucQueueStorage, ucQueueType, pxNewQueue );
                }

                return pxNewQueue;
        }
```

真正的初始化在下面这个函数中：

```
BaseType_t xQueueGenericReset( QueueHandle_t xQueue, BaseType_t xNewQueue )
{
Queue_t * const pxQueue = ( Queue_t * ) xQueue;

        configASSERT( pxQueue );

        taskENTER_CRITICAL();
        {
                /* 消息队列数据结构的相关初始化 */
                pxQueue->pcTail = pxQueue->pcHead + ( pxQueue->uxLength * pxQueue-
>uxItemSize );
                pxQueue->uxMessagesWaiting = ( UBaseType_t ) 0U;
                pxQueue->pcWriteTo = pxQueue->pcHead;
                pxQueue->u.pcReadFrom = pxQueue->pcHead + ( ( pxQueue->uxLength -
( UBaseType_t ) 1U ) * pxQueue->uxItemSize );
```

```
                pxQueue->cRxLock = queueUNLOCKED;
                pxQueue->cTxLock = queueUNLOCKED;

                if( xNewQueue == pdFALSE )
                {
                        if( listLIST_IS_EMPTY( &( pxQueue->xTasksWaitingToSend ) )
== pdFALSE )
                        {
                                if( xTaskRemoveFromEventList( &( pxQueue-
>xTasksWaitingToSend ) ) != pdFALSE )
                                {
                                        queueYIELD_IF_USING_PREEMPTION();
                                }
                                else
                                {
                                        mtCOVERAGE_TEST_MARKER();
                                }
                        }
                        else
                        {
                                mtCOVERAGE_TEST_MARKER();
                        }
                }
                else
                {
                        /* Ensure the event queues start in the correct state. */
                        vListInitialise( &( pxQueue->xTasksWaitingToSend ) );
                        vListInitialise( &( pxQueue->xTasksWaitingToReceive ) );
                }
        }
        taskEXIT_CRITICAL();

        return pdPASS;
}
```

初始化完成之后，为了让大家理解，消息队列是怎么样的，就给出一个示意图，黄色部分是消息队列的控制块，而绿色部分则是消息队列的存放消息的地方，在创建的时候，我们知道的消息队列长度与单个消息空间大

小。

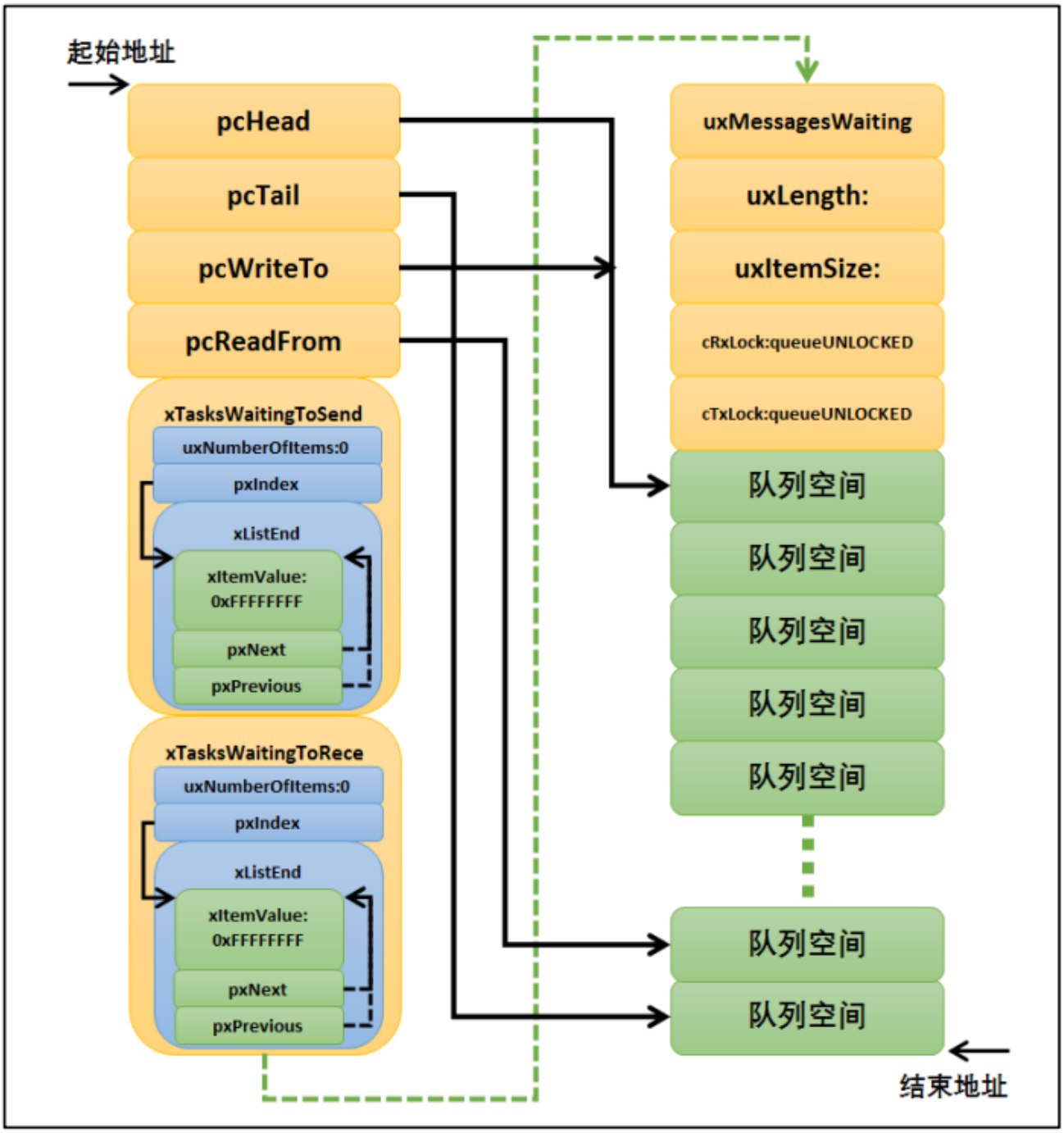


图 17-3 消息队列创建完成示意图

消息队列发送

任务或者中断服务程序都可以给消息队列发送消息，当发送消息时，如果队列未满或者允许覆盖入队，FreeRTOS 会将消息拷贝到消息队列队尾，否则，会根据用户指定的阻塞超时时间进行阻塞，在这段时间中，如果队列一直不允许入队，该任务将保持阻塞状态以等待队列允许入队。当其它任务从其等待的队列中读取入了数据（队列未满），该任务将自动由阻塞态转为就绪态。当任务等待的时间超过了指定的阻塞时间，即使队列中还不允许入队，任务也会自动从阻塞态转移为就绪态，此时发送消息的任务或者中断程序会收到一个错误码 errQUEUE_FULL。 发送紧急消息的过程与发送消息几乎一样，唯一的不同是，当发送紧急消息时，发送的位置是消息队列队头而非队尾，这样，接收者就能够优先接收到紧急消息，从而及时进行消息处理。 下面是消息队

列的发送API接口，函数中有FromISR则表明在中断中使用的。

入队方式	API 接口	实际执行函数
从队列尾部入队	xQueueSend()	xQueueGenericSend()
	xQueueSendToBack()	
	xQueueOverWrite()	
从队列首部入队	xQueueSendToFront()	
从队列尾部入队（带中断保护）	xQueueSendFromISR()	xQueueGenericSendFromISR ()
	xQueueSendToBackFromISR ()	
	xQueueOverWriteFromISR ()	
从队列首部入队（带中断保护）	xQueueSendToFront()	

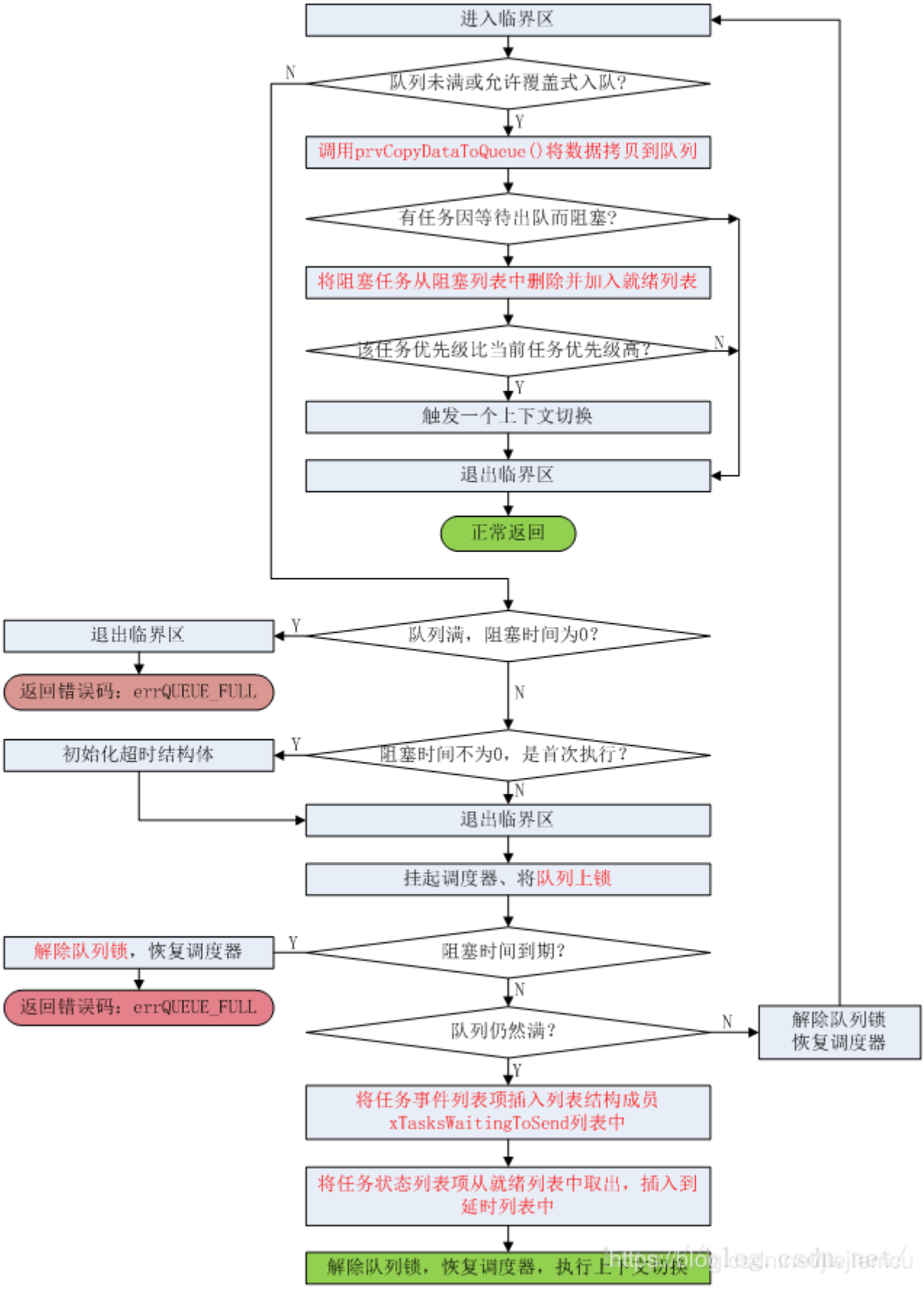

```
1 /*-----------------------------------------------------------*/
2 BaseType_t xQueueGenericSend( QueueHandle_t xQueue,          (1)
3                               const void * const pvItemToQueue,      (2)
4                               TickType_t xTicksToWait,               (3)
5                               const BaseType_t xCopyPosition )       (4)
6 {
7     BaseType_t xEntryTimeSet = pdFALSE, xYieldRequired;
8     TimeOut_t xTimeOut;
9     Queue_t * const pxQueue = ( Queue_t * ) xQueue;
10
11    /* 已删除一些断言操作 */
12
13    for ( ;; ) {
14        taskENTER_CRITICAL();                                          (5)
15        {
16            /* 队列未满 */
17            if ( ( pxQueue->uxMessagesWaiting < pxQueue->uxLength )
18                 || ( xCopyPosition == queueOVERWRITE ) ) {   (6)
19                traceQUEUE_SEND( pxQueue );
20                xYieldRequired =
21          prvCopyDataToQueue( pxQueue, pvItemToQueue, xCopyPosition ); (7)
22
23                /* 已删除使用队列集部分代码 */
24                /* 如果有任务在等待获取此消息队列 */
25      if ( listLIST_IS_EMPTY(&(pxQueue->xTasksWaitingToReceive))==pdFALSE){ (8)
26                    /* 将任务从阻塞中恢复 */
27            if ( xTaskRemoveFromEventList(
28                  &( pxQueue->xTasksWaitingToReceive ) )!=pdFALSE) { (9)
29                        /* 如果恢复的任务优先级比当前运行任务优先级还高，
30                        那么需要进行一次任务切换 */
31                        queueYIELD_IF_USING_PREEMPTION();     (10)
32                    } else {
33                        mtCOVERAGE_TEST_MARKER();
34                    }
35                } else if ( xYieldRequired != pdFALSE ) {
36                    /* 如果没有等待的任务，拷贝成功也需要任务切换 */
37                    queueYIELD_IF_USING_PREEMPTION();                (11)
```

```
38                  } else {
39                      mtCOVERAGE_TEST_MARKER();
40                  }
41
42                  taskEXIT_CRITICAL();                            (12)
43                  return pdPASS;
44              }
45              /* 队列已满 */
46              else {                                              (13)
47                  if ( xTicksToWait == ( TickType_t ) 0 ) {
48                      /* 如果用户不指定阻塞超时时间，退出 */
49                      taskEXIT_CRITICAL();                        (14)
50                      traceQUEUE_SEND_FAILED( pxQueue );
51                      return errQUEUE_FULL;
52                  } else if ( xEntryTimeSet == pdFALSE ) {
53                       /* 初始化阻塞超时结构体变量，初始化进入
54               阻塞的时间xTickCount和溢出次数xNumOfOverflows */
55                      vTaskSetTimeOutState( &xTimeOut );                  (15)
56                      xEntryTimeSet = pdTRUE;
57                  } else {
58                      mtCOVERAGE_TEST_MARKER();
59                  }
60              }
61          }
62          taskEXIT_CRITICAL();                                    (16)
63          /* 挂起调度器 */
64          vTaskSuspendAll();
65          /* 队列上锁 */
66          prvLockQueue( pxQueue );
67
68          /* 检查超时时间是否已经过去了 */
69          if (xTaskCheckForTimeOut(&xTimeOut, &xTicksToWait)==pdFALSE){ (17)
70              /* 如果队列还是满的 */
71              if ( prvIsQueueFull( pxQueue ) != pdFALSE ) {    (18)
72                  traceBLOCKING_ON_QUEUE_SEND( pxQueue );
73                  /* 将当前任务添加到队列的等待发送列表中
74                     以及阻塞延时列表，延时时间为用户指定的超时时间xTicksToWait */
75                  vTaskPlaceOnEventList(
76                     &( pxQueue->xTasksWaitingToSend ), xTicksToWait );(19)
77                  /* 队列解锁 */
78                  prvUnlockQueue( pxQueue );                      (20)
79
80                  /* 恢复调度器 */
81                  if ( xTaskResumeAll() == pdFALSE ) {
82                      portYIELD_WITHIN_API();
83                  }
84              } else {
85                  /* 队列有空闲消息空间，允许入队 */
86                  prvUnlockQueue( pxQueue );                      (21)
87                  ( void ) xTaskResumeAll();
88              }
89          } else {
90              /* 超时时间已过，退出 */
91              prvUnlockQueue( pxQueue );                          (22)
```

```
92              ( void ) xTaskResumeAll();
93
94              traceQUEUE_SEND_FAILED( pxQueue );
95              return errQUEUE_FULL;
96          }
97      }
98  }
99  /*-----------------------------------------------------------*/
```

如果阻塞时间不为 0，任务会因为等待入队而进入阻塞， 在将任务设置为阻塞的过程中， 系统不希望有其它任务和中断操作这个队列的 xTasksWaitingToReceive 列表和 xTasksWaitingToSend 列表，因为可能引起其它任务解除阻塞，这可能会发生优先级翻转。比如任务 A 的优先级低于当前任务，但是在当前任务进入阻塞的过程中，任务 A 却因为其它原因解除阻塞了，这显然是要绝对禁止的。因此FreeRTOS 使用挂起调度器禁止其它任务操作队列，因为挂起调度器意味着任务不能切换并且不准调用可能引起任务切换的 API 函数。但挂起调度器并不会禁止中断，中断服务函数仍然可以操作队列事件列表，可能会解除任务阻塞、可能会进行上下文切换，这也是不允许的。于是，解决办法是不但挂起调度器，还要给队列上锁，禁止任何中断来操作队列。 再借用朱

工精心制作的流程图加以理解：图片出自：https://blog.csdn.net/zhzht19861011/article/details/51510384



消息队列出队的API函数接口：

出队方式	API 接口	实际执行函数
出队并删除队列项	xQueueReceive ()	xQueueGenericReceive ()
出队不删除队列项	xQueuePeek ()	
出队并删除队列项（带中断保护）	xQueueReceiveFromISR()	xQueueReceiveFromISR ()
出队不删除队列项（带中断保护）	xQueuePeekFromISR()	xQueuePeekFromISR()

消息队列出队过程分析，其实跟入队差不多，请看注释：

```
1  /*-----------------------------------------------------------*/
2  BaseType_t xQueueGenericReceive( QueueHandle_t xQueue,                (1)
3                                   void * const pvBuffer,               (2)
4                                   TickType_t xTicksToWait,     (3)
5                                   const BaseType_t xJustPeeking )      (4)
6  {
7      BaseType_t xEntryTimeSet = pdFALSE;
8      TimeOut_t xTimeOut;
9      int8_t *pcOriginalReadPosition;
10     Queue_t * const pxQueue = ( Queue_t * ) xQueue;
11
12     /* 已删除一些断言 */
13     for ( ;; ) {
14         taskENTER_CRITICAL();                                          (5)
15         {
16             const UBaseType_t uxMessagesWaiting = pxQueue->uxMessagesWaiting;
17
18             /* 看看队列中有没有消息 */
19             if ( uxMessagesWaiting > ( UBaseType_t ) 0 ) {    (6)
20                 /*防止仅仅是读取消息，而不进行消息出队操作*/
21                 pcOriginalReadPosition = pxQueue->u.pcReadFrom;       (7)
22                 /* 拷贝消息到用户指定存放区域pvBuffer */
23                 prvCopyDataFromQueue( pxQueue, pvBuffer );    (8)
24
25                 if ( xJustPeeking == pdFALSE ) {              (9)
26                     /* 读取消息并且消息出队 */
27                     traceQUEUE_RECEIVE( pxQueue );
28
29                     /* 获取了消息，当前消息队列的消息个数需要减一 */
30                     pxQueue->uxMessagesWaiting = uxMessagesWaiting - 1;  (10)
31                     /* 判断一下消息队列中是否有等待发送消息的任务 */
32                     if ( listLIST_IS_EMPTY(                    (11)
33                             &( pxQueue->xTasksWaitingToSend ) ) == pdFALSE ) {
34                         /* 将任务从阻塞中恢复 */
35                         if ( xTaskRemoveFromEventList(                (12)
36                                 &( pxQueue->xTasksWaitingToSend ) ) != pdFALSE
) {
37                             /* 如果被恢复的任务优先级比当前任务高，会进行一次任务切
换 */
38                             queueYIELD_IF_USING_PREEMPTION();         (13)
```

```
39                          } else {
40                              mtCOVERAGE_TEST_MARKER();
41                          }
42                      } else {
43                          mtCOVERAGE_TEST_MARKER();
44                      }
45                  } else {                                        (14)
46                      /* 任务只是看一下消息（peek），并不出队 */
47                      traceQUEUE_PEEK( pxQueue );
48
49                      /* 因为是只读消息 所以还要还原读消息位置指针 */
50                      pxQueue->u.pcReadFrom = pcOriginalReadPosition; (15)
51
52                      /* 判断一下消息队列中是否还有等待获取消息的任务 */
53                      if ( listLIST_IS_EMPTY(                     (16)
54                              &( pxQueue->xTasksWaitingToReceive ) ) == pdFALSE
) {
55                          /* 将任务从阻塞中恢复 */
56                          if ( xTaskRemoveFromEventList(
57                                  &( pxQueue->xTasksWaitingToReceive ) ) != pdFALSE
) {
58                              /* 如果被恢复的任务优先级比当前任务高，会进行一次任务切
换 */
59                              queueYIELD_IF_USING_PREEMPTION();
60                          } else {
61                              mtCOVERAGE_TEST_MARKER();
62                          }
63                      } else {
64                          mtCOVERAGE_TEST_MARKER();
65                      }
66                  }
67
68                  taskEXIT_CRITICAL();                            (17)
69                  return pdPASS;
70              } else {                                            (18)
71                  /* 消息队列中没有消息可读 */
72                  if ( xTicksToWait == ( TickType_t ) 0 ) {       (19)
73                      /* 不等待，直接返回 */
74                      taskEXIT_CRITICAL();
75                      traceQUEUE_RECEIVE_FAILED( pxQueue );
76                      return errQUEUE_EMPTY;
77                  } else if ( xEntryTimeSet == pdFALSE ) {
78                      /* 初始化阻塞超时结构体变量，初始化进入
79                      阻塞的时间xTickCount和溢出次数xNumOfOverflows */
80                      vTaskSetTimeOutState( &xTimeOut );                  (20)
81                      xEntryTimeSet = pdTRUE;
82                  } else {
83                      mtCOVERAGE_TEST_MARKER();
84                  }
85              }
86          }
87          taskEXIT_CRITICAL();
88
89          vTaskSuspendAll();
```

```
90          prvLockQueue( pxQueue );                                    (21)
91
92          /* 检查超时时间是否已经过去了*/
93          if ( xTaskCheckForTimeOut( &xTimeOut, &xTicksToWait ) == pdFALSE )
{(22)
94              /* 如果队列还是空的 */
95              if ( prvIsQueueEmpty( pxQueue ) != pdFALSE ) {
96                  traceBLOCKING_ON_QUEUE_RECEIVE( pxQueue );    (23)
97                  /* 将当前任务添加到队列的等待接收列表中
98                     以及阻塞延时列表，阻塞时间为用户指定的超时时间xTicksToWait */
99                  vTaskPlaceOnEventList(
100                      &( pxQueue->xTasksWaitingToReceive ), xTicksToWait );
101                 prvUnlockQueue( pxQueue );
102                 if ( xTaskResumeAll() == pdFALSE ) {
103                     /* 如果有任务优先级比当前任务高，会进行一次任务切换 */
104                     portYIELD_WITHIN_API();
105                 } else {
106                     mtCOVERAGE_TEST_MARKER();
107                 }
108             } else {
109                 /* 如果队列有消息了，就再试一次获取消息 */
110                 prvUnlockQueue( pxQueue );                    (24)
111                 ( void ) xTaskResumeAll();
112             }
113         } else {
114             /* 超时时间已过，退出 */
115             prvUnlockQueue( pxQueue );                                (25)
116             ( void ) xTaskResumeAll();
117
118             if ( prvIsQueueEmpty( pxQueue ) != pdFALSE ) {
119                 /* 如果队列还是空的，返回错误代码errQUEUE_EMPTY */
120                 traceQUEUE_RECEIVE_FAILED( pxQueue );
121                 return errQUEUE_EMPTY;                            (26)
122             } else {
123                 mtCOVERAGE_TEST_MARKER();
124             }
125         }
126     }
127 }
128 /*-----------------------------------------------------------*/
```

喜欢就关注我吧！

相关代码可以在公众号后台获取。