title: 【TencentOS tiny】深度源码分析（4）——消息队列 author: 杰杰 top: false cover: false toc: true mathjax: false date: 2019-10-12 23:16:54 img: coverImg: password: summary: tags: - TencentOS tiny - RTOS - 操作系统 - 物联网 categories: - 操作系统 - TencentOS tiny

消息队列

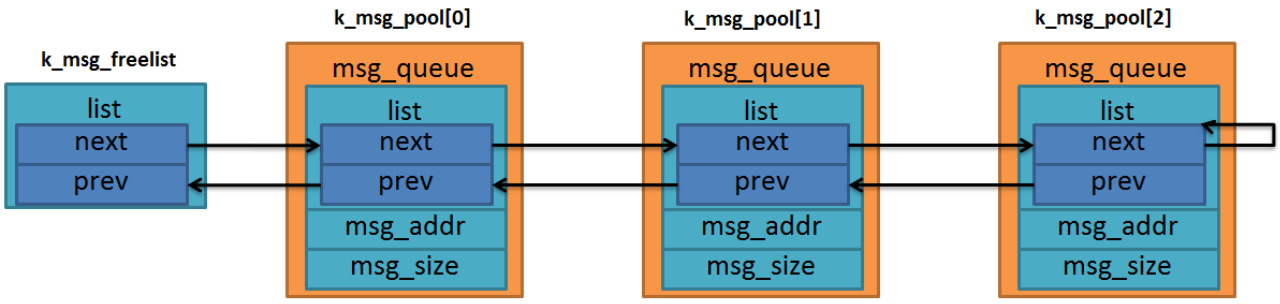
在前一篇文章中【TencentOS tiny学习】源码分析（3）——队列 我们描述了TencentOS tiny的队列实现，同时也点出了TencentOS tiny的队列是依赖于消息队列的，那么我们今天来看看消息队列的实现。

其实消息队列是TencentOS tiny的一个**基础组件**，作为队列的底层。 所以在`tos_config.h`中会用以下宏定义：

```
#if (TOS_CFG_QUEUE_EN > 0u)
#define TOS_CFG_MSG_EN      1u
#else
#define TOS_CFG_MSG_EN      0u
#endif
```

系统消息池初始化

在系统初始化（`tos_knl_init()`）的时候，系统就会将消息池进行初始化，其中， `msgpool_init()`函数就是用来初始化消息池的，该函数的定义位于 tos_msg.c文件中，函数的实现主要是通过一个`for`循环，将消息池`k_msg_pool[TOS_CFG_MSG_POOL_SIZE]`的成员变量进行初始化，初始化对应的列表节点，并且将它挂载到空闲消息列表上`k_msg_freelist` 初始化完成示意图：（假设只有3个消息）



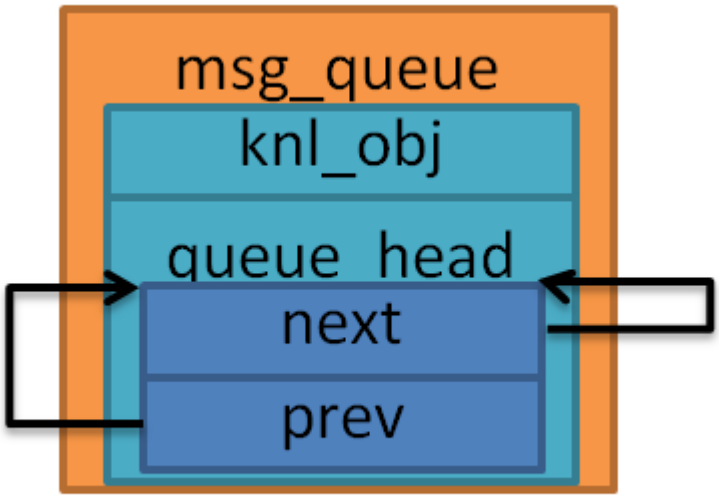

```
__KERNEL__ void msgpool_init(void)
{
    uint32_t i;

    for (i = 0; i < TOS_CFG_MSG_POOL_SIZE; ++i) {
        tos_list_init(&k_msg_pool[i].list);
        tos_list_add(&k_msg_pool[i].list, &k_msg_freelist);
```

```
    }
}
```

```
__API__ k_err_t tos_knl_init(void)
{
        ...
        #if (TOS_CFG_MSG_EN) > 0
            msgpool_init();
        #endif
        ...
}
```

消息队列创建

这个函数在队列创建中会被调用，当然他也可以自己作为用户API接口提供给用户使用，而非仅仅是内核API接口。 这个函数的本质上就是初始化消息队列中的消息列表queue_head。 初始化完成示意图：




```
__API__ k_err_t tos_msg_queue_create(k_msg_queue_t *msg_queue)
{
    TOS_PTR_SANITY_CHECK(msg_queue);

#if TOS_CFG_OBJECT_VERIFY_EN > 0u
    knl_object_init(&msg_queue->knl_obj, KNL_OBJ_TYPE_MSG_QUEUE);
#endif

    tos_list_init(&msg_queue->queue_head);
    return K_ERR_NONE;
}
```

消息队列销毁

tos_msg_queue_destroy()函数用于销毁一个消息队列，当消息队列不在使用是可以将其销毁，销毁的本质其实是将消息队列控制块的内容进行清除，首先判断一下消息队列控制块的类型是KNL_OBJ_TYPE_MSG_QUEUE，这个函数只能销毁队列类型的控制块。然后调用tos_msg_queue_flush()函数将队列的消息列表的消息全部“清空”，“清空”的意思是将挂载到队列上的消息释放回消息池（如果消息队列的消息列表存在消息，使用msgpool_free()函数释放消息）。并且通过tos_list_init()函数将消息队列的消息列表进行初始化，knl_object_deinit()函数是为了确保消息队列已经被销毁，此时消息队列控制块的pend_obj成员变量中的type 属性标识为KNL_OBJ_TYPE_NONE。

> 但是有一点要注意，因为队列控制块的RAM是由编译器静态分配的，所以即使是销毁了队列，这个内存也是没办法释放的~

```
__API__ k_err_t tos_msg_queue_destroy(k_msg_queue_t *msg_queue)
{
    TOS_PTR_SANITY_CHECK(msg_queue);

#if TOS_CFG_OBJECT_VERIFY_EN > 0u
    if (!knl_object_verify(&msg_queue->knl_obj, KNL_OBJ_TYPE_MSG_QUEUE)) {
        return K_ERR_OBJ_INVALID;
    }
#endif

    tos_msg_queue_flush(msg_queue);
    tos_list_init(&msg_queue->queue_head);

#if TOS_CFG_OBJECT_VERIFY_EN > 0u
    knl_object_deinit(&msg_queue->knl_obj);
#endif

    return K_ERR_NONE;
}
```

```
__API__ void tos_msg_queue_flush(k_msg_queue_t *msg_queue)
{
    TOS_CPU_CPSR_ALLOC();
    k_list_t *curr, *next;

    TOS_CPU_INT_DISABLE();

    TOS_LIST_FOR_EACH_SAFE(curr, next, &msg_queue->queue_head) {
        msgpool_free(TOS_LIST_ENTRY(curr, k_msg_t, list));
    }

    TOS_CPU_INT_ENABLE();
}
```

从消息队列获取消息

tos_msg_queue_get()函数用于从消息队列中获取消息，获取到的消息通过msg_addr参数返回，获取到消息的大小通过msg_size参数返回给用户，当获取成功是返回K_ERR_NONE，否则返回对应的错误代码。 这个从消息队列中获取消息的函数是不会产生阻塞的，如果有消息则获取成功，否则就获取失败，它的实现过程如下：TOS_CFG_OBJECT_VERIFY_EN 宏定义使能了，就调用knl_object_verify()函数确保是从消息队列中获取消息，然后通过TOS_LIST_FIRST_ENTRY_OR_NULL判断一下是消息队列的消息列表否存在消息，如果不存在则返回K_ERR_MSG_QUEUE_EMPTY表示消息队列是空的，反正将获取成功，获取成功后需要使用msgpool_free()函数将消息释放回消息池。

```
__API__ k_err_t tos_msg_queue_get(k_msg_queue_t *msg_queue, void **msg_addr,
size_t *msg_size)
{
    TOS_CPU_CPSR_ALLOC();
    k_msg_t *msg;

#if TOS_CFG_OBJECT_VERIFY_EN > 0u
    if (!knl_object_verify(&msg_queue->knl_obj, KNL_OBJ_TYPE_MSG_QUEUE)) {
        return K_ERR_OBJ_INVALID;
    }
#endif

    TOS_CPU_INT_DISABLE();

    msg = TOS_LIST_FIRST_ENTRY_OR_NULL(&msg_queue->queue_head, k_msg_t, list);
    if (!msg) {
        TOS_CPU_INT_ENABLE();
        return K_ERR_MSG_QUEUE_EMPTY;
    }

    *msg_addr = msg->msg_addr;
    *msg_size = msg->msg_size;
    msgpool_free(msg);

    TOS_CPU_INT_ENABLE();

    return K_ERR_NONE;
}
```

向消息队列写入消息

当发送消息时，TencentOS tiny会从消息池（空闲消息列表）中取出一个空闲消息，挂载到消息队列的消息列表中，可以通过opt参数选择挂载到消息列表的末尾或者是头部，因此消息队列的写入是支持FIFO与LIFO方式的，msg_queue是要写入消息的消息队列控制块，msg_addr、msg_size则是要写入消息的地址与大小。

写入消息的过程非常简单，直接通过msgpool_alloc()函数从消息池取出一个空闲消息，如果系统不存在空闲的消息，则直接返回错误代码K_ERR_MSG_QUEUE_FULL表示系统可用的消息已经被使用完。如果取出空闲消息

成功则将要写入的消息地址与大小记录到消息池的msg_addr 与 msg_size 成员变量中，然后通过opt参数选择将消息挂载到消息列表的位置（头部或者是尾部）。

```
__API__ k_err_t tos_msg_queue_put(k_msg_queue_t *msg_queue, void *msg_addr, size_t
msg_size, k_opt_t opt)
{
    TOS_CPU_CPSR_ALLOC();
    k_msg_t *msg;

#if TOS_CFG_OBJECT_VERIFY_EN > 0u
    if (!knl_object_verify(&msg_queue->knl_obj, KNL_OBJ_TYPE_MSG_QUEUE)) {
        return K_ERR_OBJ_INVALID;
    }
#endif

    TOS_CPU_INT_DISABLE();

    msg = msgpool_alloc();
    if (!msg) {
        TOS_CPU_INT_ENABLE();
        return K_ERR_MSG_QUEUE_FULL;
    }

    msg->msg_addr = msg_addr;
    msg->msg_size = msg_size;

    if (opt & TOS_OPT_MSG_PUT_LIFO) {
        tos_list_add(&msg->list, &msg_queue->queue_head);
    } else {
        tos_list_add_tail(&msg->list, &msg_queue->queue_head);
    }

    TOS_CPU_INT_ENABLE();

    return K_ERR_NONE;
}
```

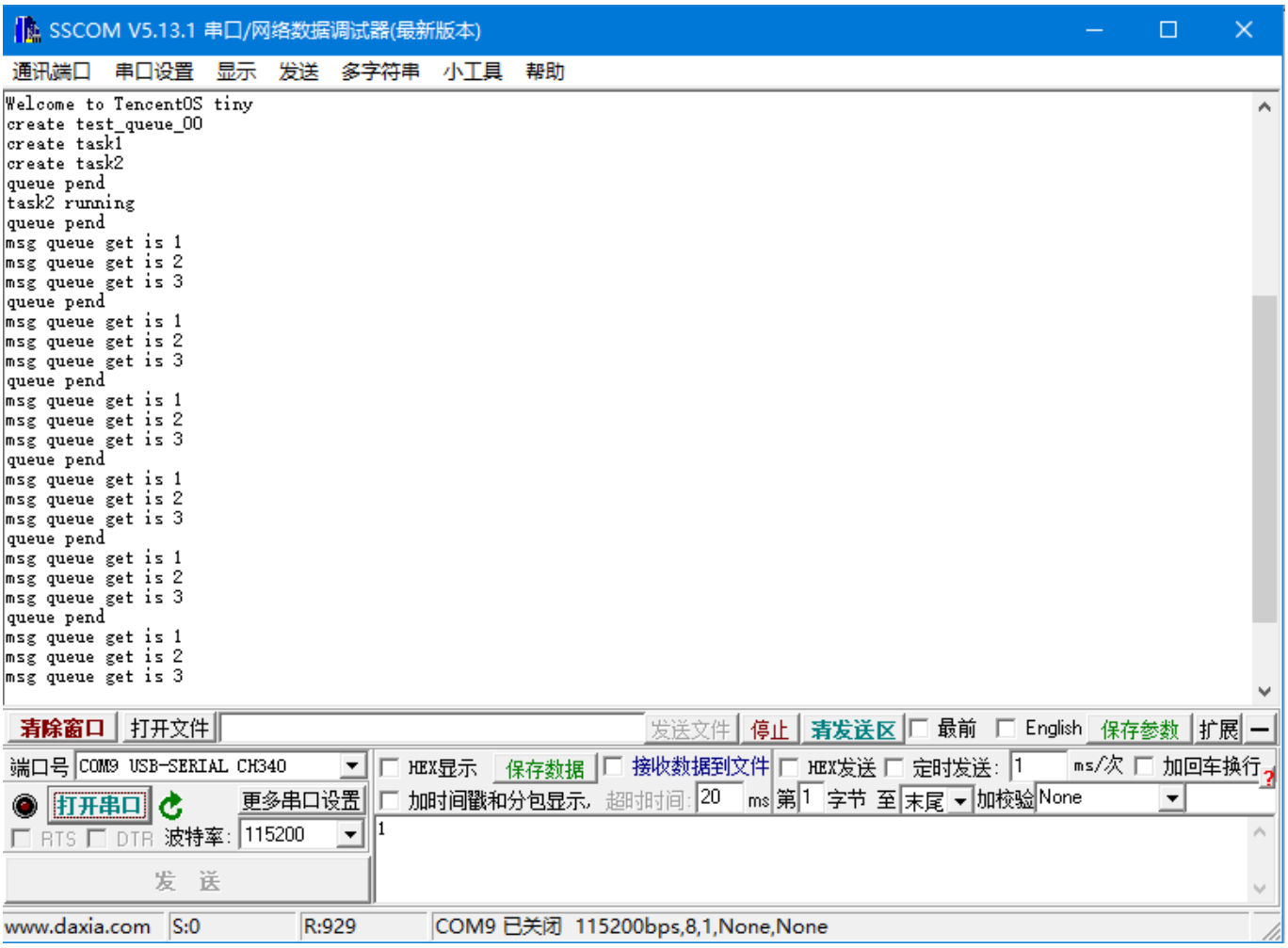
实验测试代码

```
#include "stm32f10x.h"
#include "bsp_usart.h"
#include "tos.h"


k_msg_queue_t test_msg_queue_00;

k_task_t task1;
k_task_t task2;
k_stack_t task_stack1[1024];
```

```
k_stack_t task_stack2[1024];


void test_task1(void *Parameter)
{
    k_err_t err;
    int i = 0;
    int msg_received;
    size_t msg_size = 0;

    while(1)
    {
        printf("queue pend\r\n");
        for (i = 0; i < 3; ++i)
        {
            err = tos_msg_queue_get(&test_msg_queue_00, (void **)&msg_received,
&msg_size);
            if (err == K_ERR_NONE)
                printf("msg queue get is %d \r\n",msg_received);

            if (err == K_ERR_PEND_DESTROY)
            {
                printf("queue is destroy\r\n");
                tos_task_delay(TOS_TIME_FOREVER - 1);
            }
        }
        tos_task_delay(1000);
    }
}

void test_task2(void *Parameter)
{
    k_err_t err;

    int i = 0;
    uint32_t msgs[3] = { 1, 2, 3 };

    printf("task2 running\r\n");

    while(1)
    {
        for (i = 0; i < 3; ++i)
        {
            err = tos_msg_queue_put(&test_msg_queue_00, (void *)(msgs[i]),
sizeof(uint32_t), TOS_OPT_MSG_PUT_FIFO);
            if (err != K_ERR_NONE)
                printf("msg queue put fail! code : %d \r\n",err);
        }

        tos_task_delay(1000);
    }
}
/**
  * @brief  主函数
```

```
  * @param  无
  * @retval 无
  */
int main(void)
{
    k_err_t err;

    /*初始化USART 配置模式为 115200 8-N-1，中断接收*/
    USART_Config();

    printf("Welcome to TencentOS tiny\r\n");

    tos_knl_init(); // TOS Tiny kernel initialize

    tos_robin_config(TOS_ROBIN_STATE_ENABLED, (k_timeslice_t)500u);

    printf("create test_queue_00\r\n");
    err = tos_msg_queue_create(&test_msg_queue_00);
    if(err != K_ERR_NONE)
        printf("TencentOS Create test_msg_queue_00 fail! code : %d \r\n",err);

    printf("create task1\r\n");
    err = tos_task_create(&task1,
                          "task1",
                          test_task1,
                          NULL,
                          3,
                          task_stack1,
                          1024,
                          20);
    if(err != K_ERR_NONE)
        printf("TencentOS Create task1 fail! code : %d \r\n",err);

    printf("create task2\r\n");
    err = tos_task_create(&task2,
                          "task2",
                          test_task2,
                          NULL,
                          4,
                          task_stack2,
                          1024,
                          20);
    if(err != K_ERR_NONE)
        printf("TencentOS Create task2 fail! code : %d \r\n",err);

    tos_knl_start(); // Start TOS Tiny

}
```

现象



喜欢就关注我吧！

相关代码可以在公众号后台回复“19”获取。