腾讯物联网终端操作系统

SDK 文档

产品文档

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

SDK 文档

最近更新时间：2020-12-14 18:23:35

内核 API

系统管理

tos_knl_init

```
k_err_t tos_knl_init(void);
```

- 功能描述：初始化内核。
- 参数解释：无。
- 返回值
  - K_ERR_NONE 内核，初始化成功。
  - 非 K_ERR_NONE 内核，初始化失败。

tos_knl_start

```
k_err_t tos_knl_start(void);
```

- 功能描述：启动内核。
- 参数解释：无。
- 返回值
  - K_ERR_NONE 内核，启动成功。
  - 非 K_ERR_NONE 内核，启动失败。

tos_knl_is_running

```
int tos_knl_is_running(void);
```

- 功能描述：判断内核是否正在运行。
- 参数解释：无。
- 返回值
  - 0内核，不在运行。
  - 非0内核，正在运行。

tos_knl_irq_enter

```
void tos_knl_irq_enter(void);
```

- 功能描述：此函数应该在中断服务函数开始前被调用。
- 参数解释：无。
- 返回值：无。

tos_knl_irq_leave

```
void tos_knl_irq_leave(void);
```

- 功能描述：此函数应该在中断服务函数结束前被调用。
- 参数解释：无。
- 返回值：无。

tos_knl_sched_lock

```
k_err_t tos_knl_sched_lock(void);
```

- 功能描述： 锁内核调度。
- 参数解释：无。
- 返回值
  - K_ERR_NONE 内核，调度锁定成功。
  - K_ERR_KNL_NOT_RUNNING 内核，并未处于运行状态。
  - K_ERR_LOCK_NESTING_OVERFLOW 调度锁溢出。

tos_knl_sched_unlock

```
k_err_t tos_knl_sched_unlock(void);
```

- 功能描述：解锁内核调度。
- 参数解释：无。
- 返回值
  - K_ERR_NONE 内核，调度解锁成功。
  - K_ERR_KNL_NOT_RUNNING 内核，并未处于运行状态。
  - K_ERR_SCHED_NOT_LOCKED 内核，调度并未处于锁定状态。

任务管理

tos_task_create

```
k_err_t tos_task_create(k_task_t *task,
char *name,
k_task_entry_t entry,
void *arg,
k_prio_t prio,
k_stack_t *stk_base,
size_t stk_size,
k_timeslice_t timeslice);
```

功能描述

创建任务。

参数解释

IN/OUT	参数名	描述
[in]	task	任务结构体描述符
[in]	name	任务名称
[in]	entry	任务入口函数
[in]	arg	任务入口函数参数
[in]	prio	任务优先级
[in]	stk_base	任务栈空间首地址
[in]	stk_size	任务栈空间的大小
[in]	timeslice	时间片轮转调度侧策略中，时间片的大小设置，0表示设置为系统默认值

返回值

- K_ERR_NONE ，任务创建成功。
- K_ERR_TASK_STK_SIZE_INVALID， 非法的任务栈大小。
- K_ERR_TASK_PRIO_INVALID ，非法的任务优先级。

tos_task_destroy

```
k_err_t tos_task_destroy(k_task_t *task)
```

功能描述

销毁任务。

参数解释

IN/OUT	参数名	描述
[in]	task	任务结构体描述符

返回值

- K_ERR_NONE 任务销毁成功。
- K_ERR_TASK_DESTROY_IDLE 试图销毁 idle 任务（非法）。

tos_task_delay

```
k_err_t tos_task_delay(k_tick_t delay);
```

功能描述

睡眠当前任务，以系统时钟滴答数为单位。

参数解释

IN/OUT	参数名	描述
[in]	delay	任务睡眠时间

返回值

- K_ERR_NONE，任务延迟成功。
- K_ERR_DELAY_ZERO delay，值为零（非法）。

tos_task_delay_abort

```
k_err_t tos_task_delay_abort(k_task_t *task);
```

功能描述

取消一个任务的 delay 状态。

参数解释

IN/OUT	参数名	描述
[in]	task	任务结构体描述符

返回值

- K_ERR_NONE，取消 delay 成功。
- K_ERR_TASK_NOT_DELAY task，并未处于 delay 状态。
- K_ERR_TASK_SUSPENDED task 被挂起（suspend）。

tos_task_suspend

```
k_err_t tos_task_suspend(k_task_t *task);
```

功能描述

挂起一个任务（剥夺一个任务的运行调度）。

参数解释

IN/OUT	参数名	描述
[in]	task	任务结构体描述符

返回值

- TOS_ERR_NONE 挂起任务成功。
- K_ERR_TASK_SUSPEND_IDLE 试图挂起 idle 任务（非法）。

tos_task_resume

```
k_err_t tos_task_resume(k_task_t *task);
```

功能描述

恢复一个任务的调度运行。

参数解释

IN/OUT	参数名	描述

IN/OUT	参数名	描述
[in]	task	任务结构体描述符

返回值

- TOS_ERR_NONE 恢复任务运行成功。
- K_ERR_TASK_RESUME_SELF 试图恢复当前任务（非法）。

tos_task_prio_change

```
k_err_t tos_task_prio_change(k_task_t *task, k_prio_t prio_new);
```

功能描述

更改一个任务的优先级。

参数解释

IN/OUT	参数名	描述
[in]	task	任务结构体描述符
[in]	prio_new	新的优先级

返回值

- K_ERR_NONE 任务优先级更新成功。
- K_ERR_TASK_PRIO_INVALID 优先级非法。

tos_task_yield

```
void tos_task_yield(void);
```

- **功能描述**：当前任务主动放弃 CPU。
- **参数解释**：无。
- **返回值**：无。

堆内存管理

tos_mmheap_pool_add

```
k_err_t tos_mmheap_pool_add(void *pool_start, size_t pool_size);
```

功能描述

向堆内存中添加内存池。

参数解释

IN/OUT	参数名	描述
[in]	pool_start	待添加内存池起始地址
[in]	pool_size	待添加内存池大小

返回值

- K_ERR_NONE 添加成功。
- K_ERR_MMHEAP_INVALID_POOL_ADDR 内存池起始地址非法。
- K_ERR_MMHEAP_INVALID_POOL_SIZE 内存池大小非法。

tos_mmheap_pool_rmv

```
k_err_t tos_mmheap_pool_rmv(void *pool_start);
```

功能描述

从堆内存中删除内存池。

参数解释

IN/OUT	参数名	描述
[in]	pool_start	待删除的内存池起始地址

返回值

无。

tos_mmheap_alloc

```
void *tos_mmheap_alloc(size_t size);
```

功能描述

从堆内存中分配一块内存。

参数解释

IN/OUT	参数名	描述
[in]	size	期望分配的内存大小

返回值

分配到的内存起始地址（返回 K_NULL 表示分配失败）。

tos_mmheap_aligned_alloc

```
void *tos_mmheap_aligned_alloc(size_t size, size_t align);
```

功能描述

从堆内存中分配一块内存，此内存起始地址按 align 参数对齐。

参数解释

IN/OUT	参数名	描述
[in]	size	期望分配的内存大小
[in]	align	对齐参数

返回值

分配到的内存起始地址（返回 K_NULL 表示分配失败）。

tos_mmheap_realloc

```
void *tos_mmheap_realloc(void *ptr, size_t size);
```

功能描述

从堆内存中重新分配一块内存。

参数解释

IN/OUT	参数名	描述
[in]	ptr	原内存起始地址

IN/OUT	参数名	描述
[in]	size	期望重新分配的内存大小

返回值

分配到的内存起始地址（返回 K_NULL 表示分配失败）。

tos_mmheap_free

```
void *tos_mmheap_free(void *ptr);
```

功能描述

释放一片从堆内存中分配到的内存。

参数解释

IN/OUT	参数名	描述
[in]	ptr	待释放的内存起始地址

返回值

无。

块内存管理

tos_mmblk_pool_create

```
k_err_t tos_mmblk_pool_create(k_mmblk_pool_t *mbp, void *pool_start, size_t blk_num,
size_t blk_size);
```

功能描述

创建一个块内存池。

参数解释

IN/OUT	参数名	描述
[in]	mbp	待创建的块内存句柄
[in]	pool_start	块内存池起始地址

IN/OUT	参数名	描述
[in]	blk_num	内存块数量
[in]	blk_size	单个内存块大小

返回值

- K_ERR_NONE 块内存创建成功。
- K_ERR_MMBLK_INVALID_POOL_ADDR 块内存池起始地址非法。
- K_ERR_MMBLK_INVALID_BLK_SIZE 块内存大小非法。

tos_mmblk_alloc

```
k_err_t tos_mmblk_alloc(k_mmblk_pool_t *mbp, void **blk);
```

功能描述

从内存池中分配一个内存块。

参数解释

IN/OUT	参数名	描述
[in]	mbp	块内存句柄
[out]	blk	返回的块内存起始地址

返回值

- K_ERR_NONE 块内存分配成功，blk 指向分配到的内存块起始地址。
- K_ERR_MMBLK_POOL_EMPTY 内存池是空的，blk 为 K_NULL。

tos_mmblk_free

```
k_err_t tos_mmblk_free(k_mmblk_pool_t *mbp, void *blk);
```

功能描述

释放一个内存块。

参数解释

IN/OUT	参数名	描述
[in]	mbp	块内存句柄
[in]	blk	待释放的块内存起始地址

返回值

- K_ERR_NONE 块内存释放成功。
- K_ERR_MMBLK_POOL_FULL 内存池是满的。

互斥量 mutex

tos_mutex_create

```
k_err_t tos_mutex_create(k_mutex_t *mutex);
```

功能描述

创建一个互斥量。

参数解释

IN/OUT	参数名	描述
[in]	mutex	互斥量句柄

返回值

- K_ERR_NONE 互斥量创建成功。
- K_ERR_OBJ_PTR_NULL mutex 为空指针。

tos_mutex_destroy

```
k_err_t tos_mutex_destroy(k_mutex_t *mutex);
```

功能描述

销毁一个互斥量。

参数解释

IN/OUT	参数名	描述

IN/OUT	参数名	描述
[in]	mutex	互斥量句柄

返回值

- K_ERR_NONE 互斥量销毁成功。
- K_ERR_OBJ_PTR_NULL mutex 为空指针。
- K_ERR_OBJ_INVALID mutex 指向的不是一个合法的互斥量。

tos_mutex_pend

```
k_err_t tos_mutex_pend(k_mutex_t *mutex);
```

功能描述

尝试获取一个互斥量（永久阻塞式等待）。

参数解释

IN/OUT	参数名	描述
[in]	mutex	互斥量句柄

返回值

- K_ERR_NONE 获取互斥量成功。
- K_ERR_MUTEX_NESTING_OVERFLOW 互斥量拥有者嵌套获取溢出。
- K_ERR_MUTEX_NESTING 互斥量拥有者嵌套获取。
- K_ERR_PEND_SCHED_LOCKED 此互斥量被其他任务持有，且系统调度处于锁定状态。
- K_ERR_PEND_DESTROY 当前任务试图获取的互斥量被销毁（tos_mutex_destroy）了。

tos_mutex_pend_timed

```
k_err_t tos_mutex_pend(k_mutex_t *mutex, k_tick_t timeout);
```

功能描述

尝试获取一个互斥量（有限时间内的阻塞等待）。

参数解释

IN/OUT	参数名	描述
[in]	mutex	互斥量句柄
[in]	timeout	等待超时参数

返回值

- K_ERR_NONE 获取互斥量成功。
- K_ERR_MUTEX_NESTING_OVERFLOW 互斥量拥有者嵌套获取溢出。
- K_ERR_MUTEX_NESTING 互斥量拥有者嵌套获取。
- K_ERR_PEND_NOWAIT 此互斥量被其他任务持有，同时 timeout 参数为 TOS_TIME_NOWAIT（表示获取不到互斥量时立即返回）。
- K_ERR_PEND_SCHED_LOCKED 此互斥量被其他任务持有（获取失败），且系统调度处于锁定状态。
- K_ERR_PEND_TIMEOUT 在 timeout 时间范围内未获取到互斥量。
- K_ERR_PEND_DESTROY 当前任务试图获取的互斥量被销毁（tos_mutex_destroy）了。

tos_mutex_post

```
k_err_t tos_mutex_post(k_mutex_t *mutex);
```

功能描述

释放互斥量。

参数解释

IN/OUT	参数名	描述
[in]	mutex	互斥量句柄

返回值

- K_ERR_NONE 互斥量释放成功。
- K_ERR_MUTEX_NOT_OWNER 当前任务并非此互斥量的拥有者。
- K_ERR_MUTEX_NESTING_OVERFLOW 互斥量拥有者嵌套释放溢出。
- K_ERR_MUTEX_NESTING 互斥量拥有者嵌套释放。

信号量 semaphore

tos_sem_create

```
k_err_t tos_sem_create(k_sem_t *sem, k_sem_cnt_t init_count);
```

功能描述

创建一个信号量。

参数解释

IN/OUT	参数名	描述
-	[in]	sem
[in]	init_count	信号量初始量

返回值

- K_ERR_NONE 信号量创建成功。
- K_ERR_OBJ_PTR_NULL sem 指针为空。

tos_sem_destroy

```
k_err_t tos_sem_destroy(k_sem_t *sem);
```

功能描述

销毁一个信号量。

参数解释

IN/OUT	参数名	描述
[in]	sem	信号量数据结构指针

返回值

- K_ERR_NONE 信号量销毁成功。
- K_ERR_OBJ_INVALID sem 指向的不是一个合法的信号量。
- K_ERR_OBJ_PTR_NULL sem 指针为空。

tos_sem_pend

```
k_err_t tos_sem_pend(k_sem_t *sem, k_tick_t timeout);
```

功能描述

尝试获取一个信号量。

参数解释

IN/OUT	参数名	描述
[in]	sem	信号量句柄
[in]	timeout	等待超时参数

返回值

- K_ERR_NONE 获取信号量成功。
- K_ERR_PEND_NOWAIT 获取信号量失败（信号量资源不足），同时 timeout 参数为 TOS_TIME_NOWAIT（表示获取不到信号量时立即返回）。
- K_ERR_PEND_SCHED_LOCKED 获取信号量失败，同时系统调度处于锁定状态。
- K_ERR_PEND_TIMEOUT 在 timeout 超时范围内未获取到信号量。
- K_ERR_PEND_DESTROY 试图获取的信号量被销毁了（tos_sem_destroy）。

tos_sem_post

```
k_err_t tos_sem_post(k_sem_t *sem);
```

功能描述

释放信号量，并唤醒等待队列上的一个任务。如果有多个任务在此信号量的等待队列上，唤醒优先级最高的任务。

参数解释

IN/OUT	参数名	描述
[in]	sem	信号量句柄

返回值

- K_ERR_NONE 信号量释放成功。
- K_ERR_SEM_OVERFLOW 信号量值溢出。

tos_sem_post_all

```
k_err_t tos_sem_post_all(k_sem_t *sem);
```

功能描述

释放信号量，并唤醒等待队列上的所有任务。

参数解释

IN/OUT	参数名	描述
[in]	sem	信号量句柄

返回值

- K_ERR_NONE 信号量释放成功。
- K_ERR_SEM_OVERFLOW 信号量值溢出。

队列 queue

tos_queue_create

```
k_err_t tos_queue_create(k_queue_t *queue);
```

功能描述

创建一个队列。

参数解释

IN/OUT	参数名	描述
[in]	queue	队列句柄

返回值

- K_ERR_NONE 队列创建成功。
- K_ERR_OBJ_PTR_NULL queue 指针为空。

tos_queue_destroy

```
k_err_t tos_queue_destroy(k_queue_t *queue);
```

功能描述

销毁一个队列。

参数解释

IN/OUT	参数名	描述
[in]	queue	队列句柄

返回值

- K_ERR_NONE 队列销毁成功。
- K_ERR_OBJ_PTR_NULL queue 指针为空。
- K_ERR_OBJ_INVALID queue 指向的不是一个合法的队列。

tos_queue_flush

```
k_err_t tos_queue_flush(k_queue_t *queue);
```

功能描述

冲洗队列（丢弃队列中的所有消息）。

参数解释

IN/OUT	参数名	描述
[in]	queue	队列句柄

返回值

- K_ERR_NONE 队列冲洗成功。
- K_ERR_OBJ_PTR_NULL queue 指针为空。
- K_ERR_OBJ_INVALID queue 指向的不是一个合法的队列。

tos_queue_pend

```
k_err_t tos_queue_pend(k_queue_t *queue, void **msg_addr, size_t *msg_size, k_tick_t timeout);
```

功能描述

尝试从队列中获取消息。

参数解释

IN/OUT	参数名	描述

IN/OUT	参数名	描述
[in]	queue	队列句柄
[out]	msg_addr	获取到的消息地址
[out]	msg_size	获取到的消息长度
[in]	timeout	等待超时参数

返回值

- K_ERR_NONE 从队列获取消息成功，msg_addr 和 msg_size 分别是获取到的消息地址和消息长度。
- K_ERR_PEND_NOWAIT 未能从队列中获取到消息，并且 timeout 参数为 TOS_TIME_NOWAIT（表示获取不到消息时立即返回）。
- K_ERR_PEND_SCHED_LOCKED 未能从队列中获取到消息，并且系统调度处于锁定状态。
- K_ERR_PEND_TIMEOUT 在 timeout 超时范围内未能从队列中获取到消息。
- K_ERR_PEND_DESTROY 尝试获取消息的队列被销毁了（tos_queue_destroy）。

tos_queue_post

```
k_err_t tos_queue_post(k_queue_t *queue, void *msg_addr, size_t msg_size);
```

功能描述

向队列中放入一个消息，并唤醒等待队列上的一个任务。如果有多个任务在此队列的等待队列上，唤醒优先级最高的任务。

参数解释

IN/OUT	参数名	描述
[in]	queue	队列句柄
[in]	msg_addr	消息地址
[in]	msg_size	消息长度

返回值

- K_ERR_NONE 向队列中放入消息成功。
- K_ERR_QUEUE_FULL 队列已满。

tos_queue_post_all

```
k_err_t tos_queue_post_all(k_queue_t *queue, void *msg_addr, size_t msg_size);
```

功能描述

向队列中放入一个消息，并唤醒等待队列上的所有任务。

参数解释

IN/OUT	参数名	描述
[in]	queue	队列句柄
[in]	msg_addr	消息地址
[in]	msg_size	消息长度

返回值

- K_ERR_NONE 向队列中放入消息成功。
- K_ERR_QUEUE_FULL 队列已满。

事件 event

tos_event_create

```
k_err_t tos_event_create(k_event_t *event, k_event_flag_t init_flag);
```

功能描述

创建一个事件。

参数解释

IN/OUT	参数名	描述
[in]	event	事件句柄
[in]	init_flag	事件初始旗标

返回值

- K_ERR_NONE 事件创建成功。

- K_ERR_OBJ_PTR_NULL event 指针为空。

tos_event_destroy

```
k_err_t tos_event_destroy(k_event_t *event);
```

功能描述

销毁一个事件。

参数解释

IN/OUT	参数名	描述
[in]	event	事件句柄

返回值

- K_ERR_NONE 事件销毁成功。
- K_ERR_OBJ_PTR_NULL event 指针为空。
- K_ERR_OBJ_INVALID event 指向的不是一个合法的事件。

tos_event_pend

```
k_err_t tos_event_pend(k_event_t *event, k_event_flag_t flag_expect, k_event_flag_t
*flag_match, k_tick_t timeout, k_opt_t opt);
```

功能描述

尝试获取一个特定的事件。

参数解释

IN/OUT	参数名	描述
[in]	event	事件句柄
[in]	flag_expect	期望获取到的事件旗标
[out]	flag_match	实际获取到的事件旗标
[in]	timeout	等待超时参数
[in]	opt	选项

返回值

- K_ERR_NONE 读取特定的事件成功。
- K_ERR_EVENT_PEND_OPT_INVALID opt 参数非法。
- K_ERR_PEND_NOWAIT 未能从 event 中获取到期望的旗标，并且 timeout 参数为 TOS_TIME_NOWAIT（表示未获取到期望旗标时立即返回）。
- K_ERR_PEND_SCHED_LOCKED 未能从 event 中获取到期望的旗标，并且系统调度处于锁定的状态。
- K_ERR_PEND_TIMEOUT 在超时范围内未能从 event 中获取到期望的旗标。
- K_ERR_PEND_DESTROY 尝试获取旗标的事件被销毁了（tos_event_destroy）。

tos_event_post

```
k_err_t tos_event_post(k_event_t *event, k_event_flag_t flag);
```

功能描述

向事件中放置一组旗标。

参数解释

IN/OUT	参数名	描述
[in]	event	事件句柄
[in]	flag	要放置的旗标

> ⚠ 注意：
>
> - 调用此接口放置一组旗标时，原 event 中包含的旗标将被覆盖。
> - 例如，如果原事件的旗标为0x1，并且要放置的旗标为0x8，此接口调用结束后，事件的旗标将被覆写为0x8。

返回值

- K_ERR_NONE 旗标放置成功。
- K_ERR_OBJ_PTR_NULL event 为空指针。
- K_ERR_OBJ_INVALID event 指向的并不是一个合法的事件。

tos_event_post_keep

```
k_err_t tos_event_post_keep(k_event_t *event, k_event_flag_t flag);
```

功能描述

向事件中放置一组旗标，原 event 中包含的旗标将被保留。

例如，如果原事件的旗标为0x1，并且要放置的旗标为0x8，此接口调用结束后，事件的旗标将被修改为0x9(0x8 | 0x1)，即原事件的包含的旗标被保留。

参数解释

IN/OUT	参数名	描述
[in]	event	事件句柄
[in]	flag	要放置的旗标

返回值

- K_ERR_NONE 旗标放置成功。
- K_ERR_OBJ_PTR_NULL event 为空指针。
- K_ERR_OBJ_INVALID event 指向的并不是一个合法的事件。

消息队列 msg_queue

tos_msg_queue_create

```
k_err_t tos_msg_queue_create(k_msg_queue_t *msg_queue);
```

功能描述

创建一个消息队列。

参数解释

IN/OUT	参数名	描述
[in]	msg_queue	消息队列句柄

> ⚠ **注意：**
> 消息队列 msg_queue 与队列 queue 的区别在于，queue 提供了一种同步等待机制。实际上，queue 的底层实现采用了 msg_queue 的消息管理机制。

返回值

- K_ERR_NONE 获取消息成功。
- K_ERR_OBJ_PTR_NULL msg_queue 是空指针。

tos_msg_queue_destroy

```
k_err_t tos_msg_queue_destroy(k_msg_queue_t *msg_queue);
```

功能描述

销毁一个消息队列。

参数解释

IN/OUT	参数名	描述
[in]	msg_queue	消息队列句柄

返回值

- K_ERR_NONE 获取消息成功。

- K_ERR_OBJ_PTR_NULL msg_queue 是空指针。
- K_ERR_OBJ_INVALID msg_queue 指向的不是一个合法的消息队列。

tos_msg_queue_get

```
k_err_t tos_msg_queue_get(k_msg_queue_t *msg_queue, void **msg_addr, size_t *msg_size);
```

功能描述

从消息队列中获取一个消息。

参数解释

IN/OUT	参数名	描述
[in]	msg_queue	消息队列句柄
[out]	msg_addr	获取到的消息地址
[out]	msg_size	获取到的消息长度

返回值

- K_ERR_NONE 获取消息成功。
- K_ERR_QUEUE_EMPTY 消息队列已满。

tos_msg_queue_put

```
k_err_t tos_msg_queue_put(k_msg_queue_t *msg_queue, void *msg_addr, size_t msg_size, k_opt_t opt);
```

功能描述

向消息队列中放置一个消息。

参数解释

IN/OUT	参数名	描述
[in]	msg_queue	消息队列句柄
[in]	msg_addr	待放置的消息地址

IN/OUT	参数名	描述
[in]	msg_size	待放置的消息长度
[in]	opt	选项

说明：

- TOS_OPT_MSG_PUT_LIFO：消息队列采用 last in first out 策略，即最后入队的消息最先出队。
- TOS_OPT_MSG_PUT_FIFO：消息队列采用 first in first out 策略，即最先入队的消息最先出队。

返回值

- K_ERR_NONE 消息放置成功。
- K_ERR_QUEUE_FULL 消息队列已满。

tos_msg_queue_flush

```
void tos_msg_queue_flush(k_msg_queue_t *msg_queue);
```

功能描述

冲洗消息队列（复位消息队列，丢弃消息队列中的所有消息）。

参数解释

IN/OUT	参数名	描述
[in]	msg_queue	消息队列句柄

返回值

无。

字符流先入先出队列 fifo

tos_fifo_create

```
k_err_t tos_fifo_create(k_fifo_t *fifo, uint8_t *buffer, size_t size);
```

功能描述

创建一个字符流先入先出队列。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄
[in]	buffer	字符流先入先出队列内存池
[in]	size	字符流先入先出队列内存池大小

返回值

- K_ERR_NONE 队列创建成功。
- K_ERR_OBJ_PTR_NULL fifo是空指针。

tos_fifo_destroy

```
k_err_t tos_fifo_destroy(k_fifo_t *fifo);
```

功能描述

销毁一个字符流先入先出队列。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄

返回值

- K_ERR_NONE 队列创建成功。
- K_ERR_OBJ_PTR_NULL fifo 是空指针。
- K_ERR_OBJ_INVALID fifo 指向的不是一个合法的先入先出队列。

tos_fifo_push

```
k_err_t tos_fifo_push(k_fifo_t *fifo, uint8_t data);
```

功能描述

向字符流先入先出队列压入一个字符。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄
[in]	data	压入的字符

返回值

- K_ERR_NONE 字符压入成功。
- K_ERR_FIFO_FULL 字符流先入先出队列已满。

tos_fifo_push_stream

```
k_err_t tos_fifo_push_stream(k_fifo_t *fifo, uint8_t *stream, size_t size);
```

功能描述

向字符流先入先出队列压入一个字符流。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄
[in]	stream	压入的字符流
[in]	size	字符流长度

返回值

实际压入的字符流长度。

tos_fifo_pop

```
k_err_t tos_fifo_pop(k_fifo_t *fifo, uint8_t *out);
```

功能描述

从字符流先入先出队列弹出一个字符。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄
[out]	out	弹出的字符

返回值

- K_ERR_NONE 字符弹出成功。
- K_ERR_FIFO_EMPTY 字符流先入先出队列内存池已空。

tos_fifo_pop_stream

```
int tos_fifo_pop_stream(k_fifo_t *fifo, uint8_t *buffer, size_t size);
```

功能描述

从字符流先入先出队列弹出一个字符流。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄
[out]	stream	弹出的字符流
[in]	size	字符流长度

返回值

实际弹出的字符流长度。

tos_fifo_flush

```
void tos_fifo_flush(k_fifo_t *fifo);
```

功能描述

冲洗字符流先入先出队列。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄

返回值

无。

tos_fifo_is_empty

```
int tos_fifo_is_empty(k_fifo_t *fifo);
```

功能描述

判断字符流先入先出队列是否为空。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄

返回值

- 0，fifo 不为空。
- 非0值，fifo 为空。

tos_fifo_is_full

```
int tos_fifo_is_full(k_fifo_t *fifo);
```

功能描述

判断字符流先入先出队列是否为满。

参数解释

IN/OUT	参数名	描述
[in]	fifo	字符流先入先出队列句柄

返回值

- 0，fifo 不为满。

- 非0值，fifo 为满。

定时器 timer

tos_timer_create

```
k_err_t tos_timer_create(k_timer_t *tmr,
k_tick_t delay,
k_tick_t period,
k_timer_callback_t callback,
void *cb_arg,
k_opt_t opt);
```

功能描述

创建一个定时器。

参数解释

IN/OUT	参数名	描述
[in]	tmr	定时器句柄
[in]	delay	定时器延迟多久后执行
[in]	period	周期性 timer 的周期
[in]	callback	定时器回调
[in]	cb_arg	定时器回调参数
[in]	opt	选项

> 说明：
>
> - TOS_OPT_TIMER_ONESHOT：一次性定时器，创建定时器时传入此参数，表示该定时器是一次性的，只会执行一次。
> - TOS_OPT_TIMER_PERIODIC：周期性定时器，创建定时器时传入此参数，表示该定时器是周期性的，定时器到期后，会按period参数开启下一个周期。

返回值

- K_ERR_NONE 定时器创建成功。

- K_ERR_TIMER_INVALID_PERIOD 非法的 period 参数。
- K_ERR_TIMER_INVALID_DELAY 非法的 delay 参数。

tos_timer_destroy

```
k_err_t tos_timer_destroy(k_timer_t *tmr);
```

功能描述

销毁一个定时器。

参数解释

IN/OUT	参数名	描述
[in]	tmr	定时器句柄

返回值

- K_ERR_NONE 定时器启动成功。
- K_ERR_TIMER_INACTIVE 定时器并未被创建。

tos_timer_start

```
k_err_t tos_timer_start(k_timer_t *tmr);
```

功能描述

启动一个定时器。

参数解释

IN/OUT	参数名	描述
[in]	tmr	定时器句柄

返回值

- K_ERR_NONE 定时器启动成功。
- K_ERR_TIMER_INACTIVE 定时器并未被创建。
- K_ERR_TIMER_INVALID_STATE 定时器状态非法。

tos_timer_stop

```
k_err_t tos_timer_stop(k_timer_t *tmr);
```

功能描述

停止一个定时器。

参数解释

IN/OUT	参数名	描述
[in]	tmr	定时器句柄

返回值

- K_ERR_NONE 停止定时器成功。
- K_ERR_TIMER_INACTIVE 定时器并未被创建。
- K_ERR_TIMER_STOPPED 定时器已经处于停止状态。

时间管理

tos_systick_get

```
k_tick_t tos_systick_get(void);
```

功能描述：获取系统时钟滴答数。

参数解释： 无。

返回值：系统自启动为止到目前为止的时钟滴答数。

tos_systick_set

```
void tos_systick_set(k_tick_t tick);
```

功能描述

设置系统时钟滴答数。

参数解释

IN/OUT	参数名	描述
[in]	tick	系统时钟滴答数

返回值

无。

tos_tick2millisec

```
k_time_t tos_tick2millisec(k_tick_t tick);
```

功能描述

系统时钟滴答数转化为毫秒。

参数解释

IN/OUT	参数名	描述
[in]	tick	系统时钟滴答数

返回值

毫秒数。

tos_millisec2tick

```
k_tick_t tos_millisec2tick(k_time_t millisec);
```

功能描述

毫秒转化为系统时钟滴答数。

参数解释

IN/OUT	参数名	描述
[in]	millisec	毫秒数

返回值

系统时钟滴答数。

tos_sleep_ms

```
k_err_t tos_sleep_ms(k_time_t millisec);
```

功能描述

睡眠当前任务，以毫秒为单位。

参数解释

IN/OUT	参数名	描述
[in]	millisec	任务睡眠毫秒数

返回值

- K_ERR_NONE 睡眠成功。
- K_ERR_DELAY_ZERO 毫秒数为0。

tos_sleep_hmsm

```
k_err_t tos_sleep_hmsm(k_time_t hour, k_time_t minute, k_time_t second, k_time_t mil
lisec);
```

功能描述

睡眠当前任务，睡眠时长以特定的时分秒毫秒度量。

参数解释

IN/OUT	参数名	描述
[in]	hour	小时数
[in]	minute	分钟数
[in]	second	秒数
[in]	millisec	毫秒数

返回值

- K_ERR_NONE 睡眠成功。
- K_ERR_DELAY_ZERO 毫秒数为0。

功耗管理

tos_pm_cpu_lpwr_mode_set

```
k_err_t tos_pm_cpu_lpwr_mode_set(k_cpu_lpwr_mode_t cpu_lpwr_mode);
```

功能描述

设置 CPU 的低功耗模式。

参数解释

IN/OUT	参数名	描述
[in]	cpu_lpwr_mode	CPU 的低功耗模式

返回值

- K_ERR_PM_WKUP_SOURCE_NOT_INSTALL 对应低功耗模式下的唤醒源没有安装。
- K_ERR_NONE 低功耗模式设置成功。

tos_pm_device_register

```
int tos_pm_device_register(k_pm_device_t *device);
```

功能描述

注册一个低功耗管理设备。

参数解释

IN/OUT	参数名	描述
[in]	device	低功耗管理设备句柄

返回值

- K_ERR_NONE 注册成功。
- K_ERR_OBJ_PTR_NULL device 为空。
- K_ERR_PM_DEVICE_ALREADY_REG 设备已注册过。
- K_ERR_PM_DEVICE_OVERFLOW 注册设备数量太多。

tos_tickless_wkup_alarm_install

```
void tos_tickless_wkup_alarm_install(k_cpu_lpwr_mode_t mode, k_tickless_wkup_alarm_t
*wkup_alarm);
```

功能描述

安装一个低功耗模式下的唤醒时钟。

参数解释

IN/OUT	参数名	描述
[in]	mode	低功耗模式
[in]	wkup_alarm	唤醒时钟

返回值

无。

tos_tickless_wkup_alarm_init

```
int tos_tickless_wkup_alarm_init(k_cpu_lpwr_mode_t mode);
```

功能描述

初始化特定低功耗模式下的唤醒时钟。

参数解释

IN/OUT	参数名	描述
[in]	mode	低功耗模式

返回值

- K_ERR_TICKLESS_WKUP_ALARM_NOT_INSTALLED 对应低功耗模式的唤醒闹钟没有被安装。
- K_ERR_TICKLESS_WKUP_ALARM_NO_INIT 对应低功耗模式的唤醒闹钟没有初始化函数。

组件 API

MQTT 端云对接

tos_mqtt_connect

```
int tos_mqtt_connect(char *host, const char *port, mqtt_con_param_t *param);
```

功能描述

连接 MQTT 服务器。

参数解释

IN/OUT	参数名	描述
[in]	host	服务器 IP 地址或域名
[in]	port	服务器端口
[in]	param	连接参数

返回值

- 成功，返回 socket fd。
- 失败，返回-1。

tos_mqtt_publish

```
int tos_mqtt_publish(int sock, mqtt_pub_param_t *param);
```

功能描述

发布 MQTT 消息。

参数解释

IN/OUT	参数名	描述
[in]	sock	socket fd，由 tos_mqtt_connect 获取。
[in]	param	消息发布参数

返回值

- 0，发布成功。
- -1，发布失败。

tos_mqtt_subscribe

```
int tos_mqtt_subscribe(int sock, mqtt_sub_param_t *param);
```

功能描述

订阅 MQTT 消息。

参数解释

IN/OUT	参数名	描述
[in]	sock	socket fd，由 tos_mqtt_connect 获取。
[in]	param	消息订阅参数

返回值

- 0，订阅成功。
- -1，订阅失败。

tos_mqtt_receive

```
int tos_mqtt_receive(char *topic, int topic_len, unsigned char *payload, int payload_len);
```

功能描述

收取 MQTT 消息。

参数解释

IN/OUT	参数名	描述
[out]	topic	收取到的 MQTT topic
[in]	topic_len	MQTT topic buffer 长度
[out]	payload	收取到的 payload
[in]	payload_len	payload buffer 长度

返回值

- 成功，返回收取到的 payload 长度。
- 失败，返回-1。

CMSIS 适配层

osKernelStart

```
osStatus osKernelStart(void);
```

功能描述

启动内核。

参数解释

无。

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osKernelInitialize

```
osStatus osKernelInitialize(void);
```

功能描述

初始化内核。

参数解释

无。

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osKernelRunning

```
int32_t osKernelRunning(void);
```

功能描述

返回内核是否正在运行。

参数解释

无。

返回值

- 0，内核不在运行。
- 非0，内核正在运行。

osKernelSysTick

```
uint32_t osKernelSysTick(void);
```

功能描述

获取系统时钟滴答数。

参数解释

无。

返回值

系统自启动开始到目前的时钟滴答数。

osThreadCreate

```
osThreadId osThreadCreate(const osThreadDef_t *thread_def, void *argument);
```

功能描述

创建任务。

参数解释

IN/OUT	参数名	描述
[in]	thread_def	任务初始化参数
[in]	argument	传递给任务入参

返回值

- 成功，则返回创建的任务句柄。
- 失败，返回 NULL。

osThreadGetId

```
osThreadId osThreadGetId(void);
```

功能描述

获取当前任务句柄。

参数解释

无。

返回值

当前任务句柄。

osThreadTerminate

```
osStatus osThreadTerminate(osThreadId thread_id);
```

功能描述

终止任务运行并删除任务。

参数解释

IN/OUT	参数名	描述
[in]	thread_id	任务句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osThreadSetPriority

```
osStatus osThreadSetPriority(osThreadId thread_id, osPriority priority);
```

功能描述

设置任务优先级。

参数解释

IN/OUT	参数名	描述
[in]	thread_id	任务句柄
[in]	priority	优先级

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osThreadGetPriority

```
osPriority osThreadGetPriority(osThreadId thread_id);
```

功能描述

获取任务优先级。

参数解释

IN/OUT	参数名	描述
[in]	thread_id	任务句柄

返回值

任务优先级。

osDelay

```
osStatus osDelay(uint32_t millisec);
```

功能描述

睡眠当前任务，以毫秒为单位。

参数解释

IN/OUT	参数名	描述
[in]	millisec	睡眠时间毫秒数

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osTimerCreate

```
osTimerId osTimerCreate(const osTimerDef_t *timer_def, os_timer_type type, void *argument);
```

功能描述

创建一个定时器。

参数解释

IN/OUT	参数名	描述
[in]	timer_def	定时器初始化参数
[in]	type	定时器类型
[in]	argument	定时器回调入参

返回值

- 成功，则返回创建的定时器句柄。
- 失败，返回 NULL。

osTimerStart

```
osStatus osTimerStart(osTimerId timer_id, uint32_t millisec);
```

功能描述

启动定时器。

参数解释

IN/OUT	参数名	描述
[in]	timer_id	定时器句柄
[in]	millisec	执行延迟参数

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osTimerStop

```
osStatus osTimerStop(osTimerId timer_id);
```

功能描述

停止定时器。

参数解释

IN/OUT	参数名	描述
[in]	timer_id	定时器句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osTimerDelete

```
osStatus osTimerDelete(osTimerId timer_id);
```

功能描述

销毁定时器。

参数解释

IN/OUT	参数名	描述
[in]	timer_id	定时器句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osMutexCreate

```
osStatus osMutexCreate(const osMutexDef_t *mutex_def);
```

功能描述

创建互斥量。

参数解释

IN/OUT	参数名	描述
[in]	mutex_def	互斥量初始化参数

返回值

- 成功，则返回创建的互斥量句柄。
- 失败，返回 NULL。

osMutexWait

```
osStatus osMutexWait(osMutexId mutex_id, uint32_t millisec);
```

功能描述

尝试获取一个互斥量。

参数解释

IN/OUT	参数名	描述
[in]	mutex_id	互斥量句柄
[in]	millisec	超时参数

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osMutexRelease

```
osStatus osMutexRelease(osMutexId mutex_id);
```

功能描述

释放互斥量。

参数解释

IN/OUT	参数名	描述
[in]	mutex_id	互斥量句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osMutexDelete

```
osStatus osMutexDelete(osMutexId mutex_id);
```

功能描述

销毁互斥量。

参数解释

IN/OUT	参数名	描述
[in]	mutex_id	互斥量句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osSemaphoreCreate

```
osSemaphoreId osSemaphoreCreate(const osSemaphoreDef_t *semaphore_def, int32_t coun
t);
```

功能描述

创建一个信号量。

参数解释

IN/OUT	参数名	描述
[in]	semaphore_def	信号量初始化参数

IN/OUT	参数名	描述
[in]	count	信号量初始值

返回值

- 成功，则返回创建的信号量句柄。
- 失败，返回 NULL。

osSemaphoreWait

```
int32_t osSemaphoreWait(osSemaphoreId semaphore_id, uint32_t millisec);
```

功能描述

等待一个信号量。

参数解释

IN/OUT	参数名	描述
[in]	semaphore_id	信号量句柄
[in]	millisec	超时参数

返回值

- 成功，则返回信号量可用的资源数。
- 失败，返回-1。

osSemaphoreRelease

```
osStatus osSemaphoreRelease(osSemaphoreId semaphore_id);
```

功能描述

释放信号量。

参数解释

IN/OUT	参数名	描述
[in]	semaphore_id	信号量句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osSemaphoreDelete

```
osStatus osSemaphoreDelete(osSemaphoreId semaphore_id);
```

功能描述

销毁信号量。

参数解释

IN/OUT	参数名	描述
[in]	semaphore_id	信号量句柄

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osPoolCreate

```
osPoolId osPoolCreate(const osPoolDef_t *pool_def);
```

功能描述

创建一个内存池。

参数解释

IN/OUT	参数名	描述
[in]	pool_def	内存池初始化参数

返回值

- 成功，则返回创建的内存池句柄。
- 失败，返回 NULL。

osPoolAlloc

```
void *osPoolAlloc(osPoolId pool_id);
```

功能描述

从内存池中获取一个内存块。

参数解释

IN/OUT	参数名	描述
[in]	pool_id	内存池句柄

返回值

- 成功，则返回分配到的内存块起始地址。
- 失败，返回 NULL。

osPoolCAlloc

```
void *osPoolCAlloc(osPoolId pool_id);
```

功能描述

从内存池中获取一个内存块，并将此内存块清空为0。

参数解释

IN/OUT	参数名	描述
[in]	pool_id	内存池句柄

返回值

- 成功，则返回分配到的内存块起始地址。
- 失败，返回 NULL。

osPoolFree

```
osStatus osPoolFree(osPoolId pool_id, void *block);
```

功能描述

释放一个内存块。

参数解释

IN/OUT	参数名	描述
[in]	pool_id	内存池句柄
[in]	block	内存块起始地址

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osMessageCreate

```
osMessageQId osMessageCreate(const osMessageQDef_t *queue_def, osThreadId thread_id)
;
```

功能描述

创建一个队列。

参数解释

IN/OUT	参数名	描述
[in]	queue_def	队列初始化参数
[in]	thread_id	任务句柄

返回值

- 成功，则返回创建的队列句柄。
- 失败，返回 NULL。

osMessagePut

```
osStatus osMessagePut(osMessageQId queue_id, uint32_t info, uint32_t millisec);
```

功能描述

向队列中放置一个消息。

参数解释

IN/OUT	参数名	描述
[in]	queue_id	队列句柄
[in]	info	要放置的消息体
[in]	millisec	超时参数

返回值

- osOK，返回成功。
- osErrorOS，返回失败。

osMessageGet

```
osEvent osMessageGet(osMessageQId queue_id, uint32_t millisec);
```

功能描述

从内存池中获取一个消息。

参数解释

IN/OUT	参数名	描述
[in]	queue_id	队列句柄
[in]	millisec	超时参数

返回值

一个包含了返回状态的 osEvent 信息结构体。

网络 API

sal 模组联网接口

tos_sal_module_register

```
int tos_sal_module_register(sal_module_t *module);
```

功能描述

注册一个联网模组。

参数解释

IN/OUT	参数名	描述
[in]	module	联网模组句柄

返回值

- 0，返回成功。
- -1，返回失败。

tos_sal_module_init

```
int tos_sal_module_init(void);
```

功能描述

初始化模组。

参数解释

无。

返回值

- 0，返回成功。
- -1，返回失败。

tos_sal_module_parse_domain

```
int tos_sal_module_parse_domain(const char *host_name, char *host_ip);
```

功能描述

域名解析，将一个域名转换为 IP 地址。

参数解释

IN/OUT	参数名	描述
[in]	host_name	待解析的域名
[out]	host_ip	解析后的 IP 地址

返回值

- 0，返回成功。
- -1，返回失败。

tos_sal_module_connect

```
int tos_sal_module_connect(const char *ip, const char *port, sal_proto_t proto);
```

功能描述

向远端发起连接。

参数解释

IN/OUT	参数名	描述
[in]	ip	IP 地址
[in]	port	端口
[in]	proto	TCP/UDP 协议

返回值

- 成功，则返回连接的 socket id。
- 失败，返回-1。

tos_sal_module_send

```
int tos_sal_module_send(int sock, const void *buf, size_t len);
```

功能描述

向远端发送数据。（TCP 协议栈）

参数解释

IN/OUT	参数名	描述
[in]	sock	socket id（由 tos_sal_module_connect 获取）
[in]	buf	发送的数据起始地址
[in]	len	发送的数据长度

返回值

发送的数据长度。

tos_sal_module_recv

```
int tos_sal_module_recv(int sock, void *buf, size_t len);
```

功能描述

从远端读取数据（TCP 协议栈）。

参数解释

IN/OUT	参数名	描述
[in]	sock	socket id（由 tos_sal_module_connect 获取）
[out]	buf	接收数据 buffer
[in]	len	接收数据 buffer 长度

返回值

实际接收到的数据长度。

tos_sal_module_recv_timeout

```
int tos_sal_module_recv_timeout(int sock, void *buf, size_t len, uint32_t timeout);
```

功能描述

从远端接收数据（TCP 协议栈）。

参数解释

IN/OUT	参数名	描述

IN/OUT	参数名	描述
[in]	sock	socket id（由 tos_sal_module_connect 获取）
[in]	buf	数据起始地址
[in]	len	数据长度
[in]	timeout	超时参数

返回值

实际接收到的数据长度。

tos_sal_module_sendto

```
int tos_sal_module_sendto(int sock, char *ip, char *port, void *buf, size_t len);
```

功能描述

向远端发送数据（UDP 协议栈）。

参数解释

IN/OUT	参数名	描述
[in]	sock	socket id（由 tos_sal_module_connect 获取）
[in]	ip	IP 地址
[in]	port	端口
[in]	buf	待发送数据起始地址
[in]	len	待发送数据长度

返回值

发送的数据长度。

tos_sal_module_recvfrom

```
int tos_sal_module_recvfrom(int sock, char *ip, char *port, void *buf, size_t len);
```

功能描述

从远端接收数据（UDP 协议栈）。

参数解释

IN/OUT	参数名	描述
[in]	sock	socket id（由 tos_sal_module_connect 获取）
[in]	ip	IP 地址
[in]	port	端口
[in]	buf	接收数据 buffer 起始地址
[in]	len	接收数据 buffer 长度

返回值

实际接收到的数据长度。

tos_sal_module_close

```
int tos_sal_module_close(int sock);
```

功能描述

关闭与远端的连接。

参数解释

IN/OUT	参数名	描述
[in]	sock	socket id（由 tos_sal_module_connect 获取）

返回值

- 0，返回成功。
- -1，返回失败。

lora 模组联网接口

tos_lora_module_register

```
int tos_lora_module_register(lora_module_t *module);
```

功能描述

注册一个 lora 模组。

参数解释

IN/OUT	参数名	描述
[in]	module	lora 模组句柄

返回值

- 0，返回成功。
- -1，返回失败。

tos_lora_module_init

```
int tos_lora_module_init(void);
```

功能描述

初始化 lora 模组。

参数解释

无

返回值

- 0，返回成功。
- -1，返回失败。

tos_lora_module_join

```
int tos_lora_module_join(void);
```

功能描述

加入 lora 网关。

参数解释

无

返回值

- 0，返回成功。
- -1，返回失败。

tos_lora_module_send

```
int tos_lora_module_send(const void *buf, size_t len);
```

功能描述

通过 lora 模组发送数据。

参数解释

IN/OUT	参数名	描述
[in]	buf	要发送的数据起始地址
[in]	len	要发送的数据长度

返回值

- 0，返回成功。
- -1，返回失败。

tos_lora_module_recv

```
int tos_lora_module_recv(void *buf, size_t len);
```

功能描述

通过 lora 模组接收数据。

参数解释

IN/OUT	参数名	描述
[out]	buf	接收数据的 buffer 地址
[in]	len	buffer 长度

返回值

- 0，返回成功。
- -1，返回失败。

tos_lora_module_recv_timeout

```
int tos_lora_module_recv_timeout(void *buf, size_t len, uint32_t timeout);
```

功能描述

通过 lora 模组接收数据。

参数解释

IN/OUT	参数名	描述
[in]	buf	接收数据的 buffer 地址
[in]	len	buffer 长度
[in]	timeout	超时参数

返回值

- 0，返回成功。
- -1，返回失败。

tos_lora_module_close

```
int tos_lora_module_close(void);
```

功能描述

关闭 lora 模组

参数解释

无。

返回值

- 0，返回成功。
- -1，返回失败。

腾讯定制固件模组联网接口

tos_tf_module_register

```
int tos_tf_module_register(tencent_firmware_module_t *module);
```

功能描述

注册一个腾讯定制固件模组。

参数解释

IN/OUT	参数名	描述
[in]	module	腾讯定制固件模组句柄

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_init

```
int tos_tf_module_init(void);
```

功能描述

初始化腾讯定制固件模组。

参数解释

无。

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_info_set

```
int tos_tf_module_info_set(device_info_t *info, tls_mode_t tls_mode);
```

功能描述

设置腾讯定制固件模组设备信息。

参数解释

IN/OUT	参数名	描述
[in]	device_info_t	腾讯定制固件模组设备信息描述结构体

IN/OUT	参数名	描述
[in]	tls_mode	tls 校验模式

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_conn

```
int tos_tf_module_mqtt_conn(mqtt_param_t init_params);
```

功能描述

通过腾讯定制固件模组发起 MQTT 连接。

参数解释

IN/OUT	参数名	描述
[in]	init_params	MQTT 连接参数

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_discon

```
int tos_tf_module_mqtt_discon(void);
```

功能描述

断开腾讯定制固件模组的 MQTT 连接。

参数解释

无。

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_pub

```
int tos_tf_module_mqtt_pub(const char *topic, qos_t qos, char *payload);
```

功能描述

发布主题。

**参数解释

IN/OUT	参数名	描述
[in]	topic	主题
[in]	qos	qos
[in]	payload	主题的消息负载

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_publ

```
int tos_tf_module_mqtt_publ(const char *topic, qos_t qos, char *payload);
```

功能描述

发布长消息负载（> 200）主题。

参数解释

IN/OUT	参数名	描述
[in]	topic	主题
[in]	qos	qos
[in]	payload	主题的消息负载

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_sub

```
int tos_tf_module_mqtt_sub(char *topic, qos_t qos);
```

功能描述

订阅主题。

参数解释

IN/OUT	参数名	描述
[in]	topic	主题
[in]	qos	qos

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_unsub

```
int tos_tf_module_mqtt_unsub(char *topic);
```

功能描述

取消订阅主题。

参数解释

IN/OUT	参数名	描述
[in]	topic	主题

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_mqtt_state

```
int tos_tf_module_mqtt_state(mqtt_state_t *state);
```

功能描述

查询 MQTT 连接状态。

参数解释

IN/OUT	参数名	描述
[out]	state	连接状态

返回值

- 0，返回成功。
- -1，返回失败。

tos_tf_module_debug_level_set

```
int tos_tf_module_debug_level_set(int log_level);
```

功能描述

设置模组调试日志级别。

参数解释

IN/OUT	参数名	描述
[in]	log_level	调试日志级别

返回值

- 0，返回成功。
- -1，返回失败。