



Hi3861V100 / Hi3861LV100 CoAP

开发指南

文档版本 01

发布日期 2020-07-21

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： <https://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文档介绍了基于libcoap的CoAP功能开发实现示例，以及基于lwIP（A Lightweight TCP/IP stack）协议栈对libcoap某些接口进行线程安全封装后的接口说明。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



读者对象

本文档主要适用于以下对象：



- 软件开发工程师
- 技术支持工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2020-07-21	第一次正式版本发布。



目录

前言.....	i
1 API 接口说明.....	1
1.1 概述.....	1
1.2 coap_new_context_lwip.....	2
1.3 coap_free_context_lwip.....	2
1.4 coap_check_notify_lwip.....	3
1.5 coap_send_lwip.....	3
1.6 coap_send_ack_lwip.....	4
1.7 coap_send_error_lwip.....	4
1.8 coap_send_message_type_lwip.....	5
1.9 coap_send_rst_lwip.....	5
1.10 coap_server_sessions_check_idle_lwip.....	6
2 开发指南.....	7
2.1 开发约束.....	7
2.1.1 资源的配置.....	7
2.1.2 创建 session 的说明.....	7
2.1.3 绑定接口.....	8
2.2 代码示例.....	8
2.2.1 服务器.....	8
2.2.2 客户端.....	10



1 API 接口说明

- 1.1 概述
- 1.2 [coap_new_context_lwip](#)
- 1.3 [coap_free_context_lwip](#)
- 1.4 [coap_check_notify_lwip](#)
- 1.5 [coap_send_lwip](#)
- 1.6 [coap_send_ack_lwip](#)
- 1.7 [coap_send_error_lwip](#)
- 1.8 [coap_send_message_type_lwip](#)
- 1.9 [coap_send_rst_lwip](#)
- 1.10 [coap_server_sessions_check_idle_lwip](#)

1.1 概述

CoAP使用开源库libcoap提供基础功能支持，基于CoAP的业务实现请调用libcoap接口。对于开源库自身提供的API的使用说明请参见开源API接口说明信息；对于适配lwip而新提供的接口将会在下面给出接口使用说明。

对于当前提供的libcoap适配的lwip协议栈，开源提供以下接口：

- `coap_new_context`
- `coap_free_context`
- `coap_check_notify`
- `coap_send`
- `coap_send_ack`
- `coap_send_error`
- `coap_send_message_type`
- `coap_send_rst`



在tcpip线程（lwip协议栈线程）内调用不会有线程竞态问题，但在tcpip线程之外的线程中调用则会有线程竞态问题，需要通过锁来消除与tcpip线程的竞态问题，因此封装实现以下接口：

- coap_new_context ↔ coap_new_context_lwip
- coap_free_context ↔ coap_free_context_lwip
- coap_check_notify ↔ coap_check_notify_lwip
- coap_send ↔ coap_send_lwip
- coap_send_ack ↔ coap_send_ack_lwip
- coap_send_error ↔ coap_send_error_lwip
- coap_send_message_type ↔ coap_send_message_type_lwip
- coap_send_rst ↔ coap_send_rst_lwip
- coap_server_sessions_check_idle_lwip

1.2 coap_new_context_lwip

接口定义	coap_context_t *coap_new_context_lwip(const coap_address_t *listen_addr);
描述	本接口实现的功能与coap_new_context()接口相同，但必须在tcpip线程之外的线程中调用。 本接口用于新建一个coap_context_t类型的对象，记录CoAP协议栈的状态信息。
参数	listen_addr：本地监听的地址和端口（不能为NULL）。
返回值	<ul style="list-style-type: none">• 成功：返回coap_context_t类型指针。• 失败：返回NULL。
错误码	无
自从	nStack_N500 1.0.2

1.3 coap_free_context_lwip

接口定义	void coap_free_context_lwip(coap_context_t *context);
描述	本接口实现的功能与coap_free_context()接口相同，但必须在tcpip线程之外的线程中调用。 CoAP协议栈的context必须调用coap_free_context[_lwip]接口进行资源释放。 本接口会将发送队列和接收队列中的所有条目清除，将注册在该context上的resources删除，释放关联在该context上的endpoints。
参数	context：coap_context_t类型指针。



返回值	无
错误码	无
自从	nStack_N500 1.0.2

1.4 coap_check_notify_lwip

接口定义	void coap_check_notify_lwip(coap_context_t *context);
描述	<p>本接口实现的功能与coap_check_notify()接口相同，但必须在tcpip线程之外的线程中调用。</p> <p>本接口用于检查一个context所有已知拥有的资源是否脏（dirty）了，并通知订阅的观察者。</p>
参数	context: coap_context_t类型指针。
返回值	无
错误码	无
自从	nStack_N500 1.0.2

1.5 coap_send_lwip

接口定义	coap_tid_t coap_send_lwip(coap_session_t *session, coap_pdu_t *pdu);
描述	<p>本接口实现的功能与coap_send()接口相同，但必须在tcpip线程之外的线程中调用。</p> <p>本接口用于发送CoAP消息给对端。由pdu申请的内存将会被coap_send_lwip()释放，调用者在调用完coap_send_lwip()后不能再使用pdu。</p>
参数	<ul style="list-style-type: none">• session: 一个CoAP会话（不能为NULL）。• pdu: 将要发送的pdu（不能为NULL）。
返回值	<ul style="list-style-type: none">• 成功: 返回pdu的message id。• 失败: 返回COAP_INVALID_TID。
错误码	COAP_INVALID_TID: 入参非法或内部处理错误。
自从	nStack_N500 1.0.2



1.6 coap_send_ack_lwip

接口定义	coap_tid_t coap_send_ack_lwip(coap_session_t *session, coap_pdu_t *request);
描述	本接口实现的功能与coap_send_ack()接口相同，但必须在tcpip线程之外的线程中调用。 本接口用于向发送“request”消息的端点发送code为0的ACK消息进行确认。
参数	<ul style="list-style-type: none">session：一个CoAP会话（不能为NULL）。request：将被本端点确认的请求pdu（不能为NULL）。
返回值	<ul style="list-style-type: none">成功：返回ACK消息的message id。失败：返回COAP_INVALID_TID。
错误码	COAP_INVALID_TID：入参非法或内部处理错误。
自从	nStack_N500 1.0.2

1.7 coap_send_error_lwip

接口定义	coap_tid_t coap_send_error_lwip(coap_session_t *session, coap_pdu_t *request, unsigned char code, coap_opt_filter_t opts);
描述	本接口实现的功能与coap_send_error()接口相同，但必须在tcpip线程之外的线程中调用。 本接口用于向发送“request”消息的端点发送错误响应消息，响应码是code，将会携带opts标记的option列表。
参数	<ul style="list-style-type: none">session：一个CoAP会话（不能为NULL）。request：将要被响应的请求消息（不能为NULL）。code：响应码。opts：一个过滤器，指定要从“request”中复制的option列表。
返回值	<ul style="list-style-type: none">成功：返回发送消息的message id。失败：返回COAP_INVALID_TID。
错误码	COAP_INVALID_TID：入参非法或内部处理错误。
自从	nStack_N500 1.0.2



1.8 coap_send_message_type_lwip

接口定义	coap_tid_t coap_send_message_type_lwip(coap_session_t *session, coap_pdu_t *request, unsigned char type);
描述	本接口实现的功能与coap_send_message_type()接口相同，但必须在tcpip线程之外的线程中调用。 本接口用于向发送“request”消息的端点发送消息类型是type的响应消息。
参数	<ul style="list-style-type: none">• session：一个CoAP会话（不能为NULL）。• request：将要被响应的请求消息（不能为NULL）。• type：消息类型（通常是ACK or RST）。
返回值	<ul style="list-style-type: none">• 成功：返回发送消息的message id。• 失败：返回COAP_INVALID_TID。
错误码	COAP_INVALID_TID：入参非法或内部处理错误。
自从	nStack_N500 1.0.2

1.9 coap_send_rst_lwip

接口定义	coap_tid_t coap_send_rst_lwip(coap_session_t *session, coap_pdu_t *request);
描述	本接口实现的功能与coap_send_rst()接口相同，但必须在tcpip线程之外的线程中调用。 本接口用于向发送“request”消息的端点发送消息类型是RST的响应消息。 本接口实际调用的是coap_send_message_type_lwip()，并将type设置为COAP_MESSAGE_RST。
参数	<ul style="list-style-type: none">• session：一个CoAP会话（不能为NULL）。• request：将要被响应的请求消息（不能为NULL）。
返回值	<ul style="list-style-type: none">• 成功：返回pdu的message id。• 失败：返回COAP_INVALID_TID。
错误码	COAP_INVALID_TID：入参非法或内部处理错误。
自从	nStack_N500 1.0.2



1.10 coap_server_sessions_check_idle_lwip

接口定义	void coap_server_sessions_check_idle_lwip(coap_context_t *ctx)
描述	本接口属于新增API，必须在tcpip线程之外的线程中调用。 本接口用于检测服务器的任何会话由于没有收发包处于空闲状态，空闲时间可以通过ctx->session_timeout设置（单位：s）。 如果设置ctx->session_timeout为0，则默认的时间是COAP_DEFAULT_SESSION_TIMEOUT。
参数	ctx：一个CoAP会话（不能为NULL）。
返回值	void
错误码	NA
自从	nStack_N500 1.0.2



2 开发指南

2.1 开发约束

2.2 代码示例

2.1 开发约束

2.1.1 资源的配置

当前libcoap中使用的资源是在lwip中预分配的，资源包括支持的节点个数、context/endpoint/session/pdu/resource等个数的限制。请开发者根据需要的实际场景进行合理配置。具体配置的值在lwippools.h文件中进行设置，以宏的形式给出，开发者可以在该文件中进行适配修改。

配置示例：

```
#define MEMP_NUM_COAPCONTEXT 1 /* 最大支持的context个数 */  
#define MEMP_NUM_COAPENDPOINT 1 /* 最大支持的endpoint个数 */  
#define MEMP_NUM_COAPPACKET 1 /* 最大需要同时处理的packet个数 */  
#define MEMP_NUM_COAPNODE 4 /* 最大支持的coap节点个数 */  
#define MEMP_NUM_COAPPDU MEMP_NUM_COAPNODE /* 最大支持的同时进行收发消息的个数 */  
#define MEMP_NUM_COAPSESSION 4 /* 最大支持的session个数 */  
#define MEMP_NUM_COAP_SUBSCRIPTION 1 /* 最大支持的订阅者个数 */  
#define MEMP_NUM_COAPRESOURCE 1 /* 最大支持的资源个数 */  
#define MEMP_NUM_COAPRESOURCEATTR 2 /* 最大支持的资源特性个数 */  
#define MEMP_NUM_COAPOPTLIST 1 /* 最大支持的option个数 */  
#define MEMP_LEN_COAPOPTLIST 12 /* 最大支持的单个option长度 */  
#define MEMP_NUM_COAPSTRING 2 /* 最大支持的coap string个数 */  
#define MEMP_LEN_COAPSTRING 16 /* 最大支持的单个coap string长度 */
```

2.1.2 创建 session 的说明

本次提供的libcoap在适配lwip时直接调用lwip协议栈的接口，而不是调用POSIX接口，由于开源libcoap在这样的适配下创建客户端session存在的问题，需要增加额外的代码保证功能正常：

```
session = coap_new_client_session(context, &local_addr, dst_addr, COAP_PROTO_UDP);  
session->sock.pcb = context->endpoint->sock.pcb;  
SESSIONS_ADD(context->endpoint->sessions, session);
```



在创建完session后需要额外配置session关联的UDP pcb，然后将session扩展到endpoint的sessions列表中，此时该session才能正常收发包。

2.1.3 绑定接口

当前版本默认配置不绑定网络接口，如果需要支持绑定特定网络接口的功能，则在“coap_config.h”中定义WITH_LWIP_SPECIFIC_IFINDEX宏，且在创建session成功后，指定session需要绑定的网络接口：

```
session->ifindex = netif->ifindex;
```

2.2 代码示例

2.2.1 服务器

```
#define TEST_IPV4 1
static u32_t coap_test_taskid = -1;
static int serv_running = 0;
static coap_context_t *serv_ctx = NULL;

/*
 * The resource 'hello' GET method handler
 */
static void
hello_handler(coap_context_t *ctx, struct coap_resource_t *resource,
              coap_session_t *session,
              coap_pdu_t *request, coap_binary_t *token,
              coap_string_t *query,
              coap_pdu_t *response)
{
    unsigned char buf[3];
    /* response with text "Hello World!" */
    const char* response_data = "Hello World!";
    size_t len = 0;
    unsigned char *data = NULL;

    (void)ctx;
    (void)resource;
    (void)session;
    (void)token;
    (void)query;

    response->code = COAP_RESPONSE_CODE(205);
    coap_add_option(response, COAP_OPTION_CONTENT_TYPE, coap_encode_var_safe(buf, 3,
COAP_MEDIATYPE_TEXT_PLAIN), buf);
    coap_add_data(response, strlen(response_data), (unsigned char *)response_data);

    if (coap_get_data(request, &len, &data)) {
        printf("[%s][%d] len : %d, data : %.*s\n", __FUNCTION__, __LINE__, len, len, data);
    }
}

void coap_server_thread(UINT32 uwParam1,
                        UINT32 uwParam2,
                        UINT32 uwParam3,
                        UINT32 uwParam4)
{
    coap_context_t* ctx;
```



```
(void)uwParam2;
(void)uwParam3;
(void)uwParam4;

printf("[%s][%d] thread running\n", __FUNCTION__, __LINE__);
ctx = (coap_context_t*)uwParam1;
while (serv_running == 1) {
    // printf("coap_server sleep 1s\n");
    hi_sleep(1000);
    coap_check_notify_lwip(ctx);
}
if (serv_ctx != NULL) {
    coap_free_context_lwip(serv_ctx);
    serv_ctx = NULL;
}
printf("[%s][%d] thread exit\n", __FUNCTION__, __LINE__);
return;
}

int coap_server_start(int argc, char* argv[])
{
    TSK_INIT_PARAM_S stappTask;
    UINT32 ret;

    (void)argc;
    (void)argv;

    coap_address_t serv_addr;
    coap_resource_t* hello_resource;

    if (serv_running == 1) {
        return 0;
    }
    serv_running = 1;
    /* Prepare the CoAP server socket */
    coap_address_init(&serv_addr);
#ifdef TEST_IPV4
    ip_addr_set_any(false, &(serv_addr.addr));
#else
    ip_addr_set_any(true, &(serv_addr.addr));
#endif
    serv_addr.port = COAP_DEFAULT_PORT;
    serv_ctx = coap_new_context_lwip(&serv_addr);
    if (!serv_ctx) {
        return -1;
    }
    /* Initialize the hello resource */
    hello_resource = coap_resource_init(coap_make_str_const("hello"), 0);
    coap_register_handler(hello_resource, COAP_REQUEST_GET, hello_handler);
    coap_add_resource(serv_ctx, hello_resource);

    /* create a thread task */
    stappTask.pfnTaskEntry = (TSK_ENTRY_FUNC)coap_server_thread;
    stappTask.uwStackSize = 10*LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    stappTask.pcName = "coap_test_task";
    stappTask.usTaskPrio = 11;
    stappTask.uwResved = LOS_TASK_STATUS_DETACHED;
    stappTask.auwArgs[0] = (UINT32)serv_ctx;

    printf("task create coap_server_thread\n");
    ret = LOS_TaskCreate(&coap_test_taskid, &stappTask);
}
```



```
    if (0 != ret ) {
        dprintf("coap_server_thread create failed ! \n");
        return -1;
    }

    return 0;
}

void coap_server_stop(void)
{
    if (serv_running == 0) {
        printf("[%s][%d] not running\n", __FUNCTION__, __LINE__);
        return;
    }
    if (serv_ctx != NULL) {
        coap_free_context_lwip(serv_ctx);
        serv_ctx = NULL;
    }
    serv_running = 0;
    printf("[%s][%d] stopped\n", __FUNCTION__, __LINE__);
    return;
}
```

2.2.2 客户端

```
#define TEST_IPV4 1
#define DHCP_COAP_TOKEN_LEN (4)
static coap_context_t* cli_ctx = NULL;
/* The response handler */
static void message_handler(struct coap_context_t *ctx,
                           coap_session_t *session,
                           coap_pdu_t *sent,
                           coap_pdu_t *received,
                           const coap_tid_t id)
{
    unsigned char* data;
    size_t data_len;
    (void)ctx;
    (void)session;
    (void)sent;
    (void)received;
    (void)id;
    if (COAP_RESPONSE_CLASS(received->code) == 2) {
        if (coap_get_data(received, &data_len, &data)) {
            printf("Received: %.*s\n", data_len, data);
        }
    }
}

int coap_client_start(int argc, char* argv[])
{
    coap_address_t src_addr;
    (void)argc;
    (void)argv;
    if (cli_ctx != NULL) {
        return 0;
    }
    /* Prepare coap socket*/
    coap_address_init(&src_addr);
#ifdef TEST_IPV4
    ip_addr_set_any(false, &(src_addr.addr));
#else
```



```
    ip_addr_set_any(true, &(src_addr.addr));
#endif
    src_addr.port = 23456;
    cli_ctx = coap_new_context_lwip(&src_addr);
    if (!cli_ctx) {
        return -1;
    }
    /* Set the response handler*/
    coap_register_response_handler(cli_ctx, message_handler);
    return 0;
}

void coap_client_stop(void)
{
    if (cli_ctx != NULL) {
        coap_free_context_lwip(cli_ctx);
        cli_ctx = NULL;
    }
    printf("[%s][%d] stopped\n", __FUNCTION__, __LINE__);
    return;
}

/* to create a new token value depending on time */
s32_t coap_new_token(u16_t msg_id, u8_t *token, u8_t token_len)
{
    u32_t now_ms;
    if ((token == NULL) || (token_len < DHCP_COAP_TOKEN_LEN)) {
        return -1;
    }
    now_ms = sys_now();
    token[0] = (u8_t)(msg_id);
    token[1] = (u8_t)(msg_id >> 8);
    token[2] = (u8_t)(now_ms);
    token[3] = (u8_t)(now_ms >> 8);
    return 0;
}

int coap_client_send_msg(char* dst)
{
    coap_address_t dst_addr, listen_addr;
    static coap_uri_t uri;
    coap_pdu_t* request;
    coap_session_t *session = NULL;
    char server_uri[128] = {0};
    u8_t temp_token[DHCP_COAP_TOKEN_LEN] = {0};
    unsigned char get_method = COAP_REQUEST_GET;
    /* The destination endpoint */
    coap_address_init(&dst_addr);
    printf("[%s][%d] server : %s\n", __FUNCTION__, __LINE__, dst);
#ifdef TEST_IPV4
    if (!ipaddr_aton(dst, &(dst_addr.addr))) {
        printf("invalid ip4 addr\n");
        return -1;
    }
#else
    if (!ip6addr_aton(dst, &(dst_addr.addr.u_addr.ip6))) {
        printf("invalid ip6 addr\n");
        return -1;
    }
    IP_SET_TYPE_VAL(dst_addr.addr, IPADDR_TYPE_V6);
#endif
    dst_addr.port = COAP_DEFAULT_PORT;
    /* try to reuse existed session */
}
```




```
    session = coap_session_get_by_peer(cli_ctx, &dst_addr, 0);
    if (session == NULL) {
        coap_address_init(&listen_addr);
#ifdef TEST_IPV4
        ip_addr_set_any(false, &(listen_addr.addr));
#else
        ip_addr_set_any(true, &(listen_addr.addr));
#endif
        listen_addr.port = 23456;
        session = coap_new_client_session(cli_ctx, &listen_addr, &dst_addr, COAP_PROTO_UDP);
        if (session == NULL) {
            printf("[%s][%d] new client session failed\n", __FUNCTION__, __LINE__);
            return -1;
        }
        session->sock.pcb = cli_ctx->endpoint->sock.pcb;
        SESSIONS_ADD(cli_ctx->endpoint->sessions, session);
    }
    /* Prepare the request */
    strcpy(server_uri, "/hello");
    coap_split_uri((unsigned char *)server_uri, strlen(server_uri), &uri);
    request = coap_new_pdu(session);
    if (request == NULL) {
        printf("[%s][%d] get pdu failed\n", __FUNCTION__, __LINE__);
        return -1;
    }
    request->type = COAP_MESSAGE_CON;
    request->tid = coap_new_message_id(session);
    (void)coap_new_token(request->tid, temp_token, DHCP_COAP_TOKEN_LEN);
    if (coap_add_token(request, DHCP_COAP_TOKEN_LEN, temp_token) == 0) {
        printf("[%s][%d] add token failed\n", __FUNCTION__, __LINE__);
    }
    request->code = get_method;
    coap_add_option(request, COAP_OPTION_URI_PATH, uri.path.length, uri.path.s);
    char request_data[64] = {0};
    (void)snprintf_s(request_data, sizeof(request_data), sizeof(request_data)-1, "%s", "luminais");
    coap_add_data(request, 4 + strlen((const char *) (request_data+4)), (unsigned char *)request_data);
    coap_send_lwip(session, request);
    return 0;
}

int coap_client_send(int argc, char* argv[])
{
    (void)argc;
    (void)argv;
    if (argc < 2) {
        printf("coap client {start | stop | send server_addr}\n");
        return -1;
    }
    if (cli_ctx == NULL) {
        return -1;
    }
    /* argv[0] is server_addr */
    coap_client_send_msg(argv[0]);
    return 0;
}
```