合宙Air7XX CSDK luatos_mqtt 使用指南

luatos_mqtt 是luatos 团队根据libemqtt 自行改进与研发，为了方便各位童鞋快速使用与掌握luatos_mqtt,这里luatos 团队提供了使用说明：

注：完整demo 参考example_luatos_mqtt,使用指南配合demo+API 手册一起使用效果更佳

注：使用指南里面代码块里面的注释，重点看一下

1. 使用luatos_mqtt 需要首先在工程对应的 `xmake.lua` 加入库文件，参考如下

```
includes(SDK_TOP .. "/thirdparty/libemqtt")
add_deps("libemqtt")
```

2. 加入luatos_mqtt 需要的net_lwip 依赖代码（主要为了设配网卡）

- 在任务初始化函数里面加入如下代码

```
luat_mobile_event_register_handler(luatos_mobile_event_callback);//注册模块mobile 事件回调函数
net_lwip_init();//初始化net_lwip
net_lwip_register_adapter(NW_ADAPTER_INDEX_LWIP_GPRS);//注册lwip网卡为蜂窝模块GPRS
network_register_set_default(NW_ADAPTER_INDEX_LWIP_GPRS);
```

- 修改luat_mobile_event_register_handler （注：网络事件回调函数luat_mobile_event_register_handler，需要保持全局统一，只有一个网络事件回调函数），加入如下函数
    - `soc_mobile_get_default_pdp_part_info(&type, NULL, NULL, &dns_num, dns_ip)`
    - `NetMgrGetNetInfo(0xff, pNetifInfo)`
    - `net_lwip_set_local_ip6`
    - `network_set_dns_server`
    - `net_lwip_set_link_state`
- 修改注册函数主要为了设置DNS服务器、设置网络状态（net_lwip 需要）

```
static void luatos_mobile_event_callback(LUAT_MOBILE_EVENT_E event, uint8_t index, uint8_t status)
{
    if (LUAT_MOBILE_EVENT_NETIF == event)
    {
        if (LUAT_MOBILE_NETIF_LINK_ON == status)
        {
            ip_addr_t dns_ip[2];
            uint8_t type, dns_num;
            dns_num = 2;
            /*从网络获取默认的DNS服务器*/
            soc_mobile_get_default_pdp_part_info(&type, NULL, NULL, &dns_num, dns_ip);

            if (type & 0x80)
            {
                if (index != 4)
                {
                    return;
                }
                else
                {
                    NmAtiNetifInfo *pNetifInfo = malloc(sizeof(NmAtiNetifInfo));
                    NetMgrGetNetInfo(0xff, pNetifInfo);
                    if (pNetifInfo->ipv6Cid != 0xff)
                    {
                        net_lwip_set_local_ip6(&pNetifInfo->ipv6Info.ipv6Addr);

                    }
                    free(pNetifInfo);
                }
            }
            if (dns_num > 0)
            {
                /*根据得到默认信息，设置DNS服务器*/
                network_set_dns_server(NW_ADAPTER_INDEX_LWIP_GPRS, 2, &dns_ip[0]);
                if (dns_num > 1)
                {
                    network_set_dns_server(NW_ADAPTER_INDEX_LWIP_GPRS, 3, &dns_ip[1]);
                }
            }
            /*设置网络状态*/
            net_lwip_set_link_state(NW_ADAPTER_INDEX_LWIP_GPRS, 1);
```

```
                }
            }
        }
```

3.创建一个MQTT实例

```
int luat_mqtt_init(luat_mqtt_ctrl_t *mqtt_ctrl, int adapter_index);
//使用说明如下
int ret = -1;
luat_mqtt_ctrl_t *luat_mqtt_ctrl = (luat_mqtt_ctrl_t *)luat_heap_malloc(sizeof(luat_mqtt_ctrl_t));
ret = luat_mqtt_init(luat_mqtt_ctrl, NW_ADAPTER_INDEX_LWIP_GPRS);
if (ret)
{
    LUAT_DEBUG_PRINT("mqtt init FAID ret %d", ret);
    return 0;
}
luat_mqtt_ctrl->ip_addr.type = 0xff;
```

4.设置客户端必要的参数

```
mqtt_init(&(luat_mqtt_ctrl->broker), CLIENT_ID);//设置客户端client_id
mqtt_init_auth(&(luat_mqtt_ctrl->broker), USERNAME, PASSWORD);//设置客户端name,password
// luat_mqtt_ctrl->netc->is_debug = 1;// debug信息
luat_mqtt_ctrl->broker.clean_session = 1;
luat_mqtt_ctrl->keepalive = 240;//设置心跳时间
luat_mqtt_ctrl->reconnect = 1;//设置为自动重连
luat_mqtt_ctrl->reconnect_time = 3000;//设置自动重连的时间为3000
```

5.设置服务端信息

```
luat_mqtt_connopts_t opts = {0};
#if (MQTT_DEMO_SSL == 1)
opts.is_tls = 1;
opts.server_cert = testCaCrt;
opts.server_cert_len = strlen(testCaCrt);
opts.client_cert = testclientCert;
opts.client_cert_len = strlen(testclientCert);
opts.client_key = testclientPk;
opts.client_key_len = strlen(testclientPk);
#else
opts.is_tls = 0;
#endif
opts.host = MQTT_HOST;
opts.port = MQTT_PORT;
ret = luat_mqtt_set_connopts(luat_mqtt_ctrl, &opts);//设置服务端信息，具体可以看API手册
```

6.设置回调函数

```
luat_mqtt_set_cb(luat_mqtt_ctrl,luat_mqtt_cb);
```

7.发起连接

```
luat_mqtt_connect(luat_mqtt_ctrl);
```

8.订阅与发布函数

```c
//发布函数
/** Publish a message on a topic.
 * @param MQTT实例对象
 * @param topic 主题名称.
 * @param msg 消息负载.
 * @param msg_len 消息负载长度
 * @param retain Enable or disable the Retain flag (values: 0 or 1).
 * @param qos Quality of Service (values: 0, 1 or 2)
 * @param message_id Variable that will store the Message ID, if the pointer is not NULL.
 *
 * @retval  1 On success.
 * @retval  0 On connection error.
 * @retval -1 On IO error.
 */
int mqtt_publish_with_qos(mqtt_broker_handle_t* broker,
                          const char* topic,
                          const char* msg,
                          uint32_t msg_len,
                          uint8_t retain,
                          uint8_t qos,
                          uint16_t* message_id);
//订阅函数
/** Subscribe to a topic.
 * @param broker Data structure that contains the connection information with the broker.
 * @param topic 主题名称.
 * @param message_id Variable that will store the Message ID, if the pointer is not NULL.
 *
 * @retval  1 On success.
 * @retval  0 On connection error.
 * @retval -1 On IO error.
 */
int mqtt_subscribe(mqtt_broker_handle_t* broker,
                   const char* topic,
                   uint16_t* message_id,
                   uint8_t qos);
```

9.回调函数与事件处理使用方法

```c
//MQTT 回调函数里面不能做大量的数据处理，推荐使用消息队列的方式来出来数据
//处理方法如下
static luat_rtos_queue_t mqtt_queue_handle;
#define MQTT_QUEUE_SIZE 128
//创建消息队列的结构体
typedef struct
{
    luat_mqtt_ctrl_t *luat_mqtt_ctrl;//mqtt 的实例对象
    uint16_t event;//mqtt 事件类型
} mqttQueueData;
//MQTT 回调函数
static void luat_mqtt_cb(luat_mqtt_ctrl_t *luat_mqtt_ctrl, uint16_t event
{
    //回调函数里面，通过消息队列将消息发送到主任务函数
    mqttQueueData mqtt_cb_event = {.luat_mqtt_ctrl = luat_mqtt_ctrl,.event = event};
    luat_rtos_queue_send(mqtt_queue_handle, &mqtt_cb_event, NULL, 0);
    return;
}

static void luat_mqtt_task(void *param)
{

    mqttQueueData mqttQueueRecv = {0};
    //创建MQTT数据处理的消息队列
    luat_rtos_queue_create(&mqtt_queue_handle, MQTT_QUEUE_SIZE, sizeof(mqttQueueData));
    /******省略部分代码，只展示事件处理与数据处理的代码**/
    luat_mqtt_set_cb(luat_mqtt_ctrl,luat_mqtt_cb);
    while(1)
    {
        //主任务里面接收MQTT数据，并进行处理
        if (luat_rtos_queue_recv(mqtt_queue_handle, &mqttQueueRecv, NULL, 5000) == 0)
        {
            switch (mqttQueueRecv.event)
            {
            case MQTT_MSG_CONNACK:{//MQTT连接成功的标志
                LUAT_DEBUG_PRINT("mqtt_connect ok");

                LUAT_DEBUG_PRINT("mqtt_subscribe");
                uint16_t msgid = 0;
                mqtt_subscribe(&(mqttQueueRecv.luat_mqtt_ctrl->broker),
                               mqtt_sub_topic, &msgid, 1);

                LUAT_DEBUG_PRINT("publish");
                uint16_t message_id  = 0;
                mqtt_publish_with_qos(&(mqttQueueRecv.luat_mqtt_ctrl->broker),
                                      mqtt_pub_topic, mqtt_send_payload,
                                      strlen(mqtt_send_payload), 0, 1, &message_id);
                break;
            }
            case MQTT_MSG_PUBLISH : {//收到消息的标志
                const uint8_t* ptr;
                uint16_t topic_len =
                    mqtt_parse_pub_topic_ptr(mqttQueueRecv.luat_mqtt_ctrl->
                                                mqtt_packet_buffer, &ptr);
                LUAT_DEBUG_PRINT("pub_topic: %.*s",topic_len,ptr);
                uint16_t payload_len =
                    mqtt_parse_pub_msg_ptr(mqttQueueRecv.luat_mqtt_ctrl->
                                           mqtt_packet_buffer, &ptr);
                LUAT_DEBUG_PRINT("pub_msg: %.*s",payload_len,ptr);
                break;
            }
            case MQTT_MSG_PUBACK :
            case MQTT_MSG_PUBCOMP : {
                LUAT_DEBUG_PRINT("msg_id: %d",mqtt_parse_msg_id
                                 (mqttQueueRecv.luat_mqtt_ctrl->mqtt_packet_buffer));
                break;
            }
            case MQTT_MSG_RELEASE : {
                LUAT_DEBUG_PRINT("luat_mqtt_cb mqtt release");
                break;
            }
            case MQTT_MSG_DISCONNECT : { // mqtt 断开(只要有断开就会上报,无论是否重连)
                LUAT_DEBUG_PRINT("luat_mqtt_cb mqtt disconnect");
                break;
            }
            case MQTT_MSG_CLOSE : {
                //mqtt 关闭(不会再重连)注意：一定注意和MQTT_MSG_DISCONNECT区别,
                /*如果要做手动重连处理推荐在这里 */
                LUAT_DEBUG_PRINT("luat_mqtt_cb mqtt close");
                if (MQTT_DEMO_AUTOCON == 0)
                {
                    ret = luat_mqtt_connect(mqttQueueRecv.luat_mqtt_ctrl);
                    if (ret) {
                        LUAT_DEBUG_PRINT("mqtt connect ret=%d\n", ret);
```

```c
                        luat_mqtt_close_socket(mqttQueueRecv.luat_mqtt_ctrl);
                        return;
                    }
                }
                break;
            }
            default:
                break;
            }
        }
        else
        {
            if (luat_mqtt_state_get(luat_mqtt_ctrl) == MQTT_STATE_READY)
            {
                uint16_t message_id  = 0;
                mqtt_publish_with_qos(&(luat_mqtt_ctrl->broker),
                                      mqtt_pub_topic,
                                      mqtt_send_payload,
                                      strlen(mqtt_send_payload),
                                      0, 1, &message_id);
            }
        }
    }
}

```