

合宙Air7XX CSDK luatos_mqtt 使用指南

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
2. 加入luatos_mqtt 需要的net_lwip 依赖代码（主要为了设配网卡）
3. 创建一个MQTT实例
4. 设置客户端必要的参数
5. 设置服务端信息
6. 设置回调函数
7. 发起连接
8. 订阅与发布函数
9. 回调函数与事件处理使用方法

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 `lua` 加入库文件，参考如下

```
1 includes(SDK_TOP .. "/thirdparty/libemqtt")
2 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

```

1  luat_mobile_event_register_handler(luatos_mobile_event_callback); //注册模块mobile 事件回调函数
2  net_lwip_init(); //初始化net_lwip
3  net_lwip_register_adapter(NW_ADAPTER_INDEX_LWIP_GPRS); //注册lwip网卡为蜂窝模块GPRS
4  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_name, 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）

```

1  static void luatos_mobile_event_callback(LUAT_MOBILE_EVENT_E event, uint8_
    t index, uint8_t status)
2  {
3      if (LUAT_MOBILE_EVENT_NETIF == event)
4      {
5          if (LUAT_MOBILE_NETIF_LINK_ON == status)
6          {
7              ip_addr_t dns_ip[2];
8              uint8_t type, dns_num;
9              dns_num = 2;
10             /*从网络获取默认的DNS服务器*/
11             soc_mobile_get_default_pdp_part_info(&type, NULL, NULL, &dns_n
                um, dns_ip);
12
13             if (type & 0x80)
14             {
15                 if (index != 4)
16                 {
17                     return;
18                 }
19                 else
20                 {
21                     NmAtiNetifInfo *pNetifInfo = malloc(sizeof(NmAtiNetifI
                        nfo));
22                     NetMgrGetNetInfo(0xff, pNetifInfo);
23                     if (pNetifInfo->ipv6Cid != 0xff)
24                     {
25                         net_lwip_set_local_ip6(&pNetifInfo->ipv6Info.ipv6A
                            ddr);
26
27                     }
28                     free(pNetifInfo);
29                 }
30             }
31             if (dns_num > 0)
32             {
33                 /*根据得到默认信息，设置DNS服务器*/
34                 network_set_dns_server(NW_ADAPTER_INDEX_LWIP_GPRS, 2, &dns
                    _ip[0]);
35                 if (dns_num > 1)
36                 {
37                     network_set_dns_server(NW_ADAPTER_INDEX_LWIP_GPRS, 3,
                        &dns_ip[1]);
38                 }
39             }
40             /*设置网络状态*/
41             net_lwip_set_link_state(NW_ADAPTER_INDEX_LWIP_GPRS, 1);

```

```
42     }  
43 }  
44 }
```

3.创建一个MQTT实例

```
1  int luat_mqtt_init(luat_mqtt_ctrl_t *mqtt_ctrl, int adapter_index);  
2  //使用说明如下  
3  int ret = -1;  
4  luat_mqtt_ctrl_t *luat_mqtt_ctrl = (luat_mqtt_ctrl_t *)luat_heap_malloc(sizeof(luat_mqtt_ctrl_t));  
5  ret = luat_mqtt_init(luat_mqtt_ctrl, NW_ADAPTER_INDEX_LWIP_GPRS);  
6  if (ret)  
7  {  
8      LUAT_DEBUG_PRINT("mqtt init FAID ret %d", ret);  
9      return 0;  
10 }  
11 luat_mqtt_ctrl->ip_addr.type = 0xff;
```

4.设置客户端必要的参数

```
1  mqtt_init(&(luat_mqtt_ctrl->broker), CLIENT_ID); //设置客户端client_id  
2  mqtt_init_auth(&(luat_mqtt_ctrl->broker), USERNAME, PASSWORD); //设置客户端name, password  
3  // luat_mqtt_ctrl->netc->is_debug = 1; // debug信息  
4  luat_mqtt_ctrl->broker.clean_session = 1;  
5  luat_mqtt_ctrl->keepalive = 240; //设置心跳时间  
6  luat_mqtt_ctrl->reconnect = 1; //设置为自动重连  
7  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

```

1  luat_mqtt_connopts_t opts = {0};
2  #if (MQTT_DEMO_SSL == 1)
3  opts.is_tls = 1;
4  opts.server_cert = testCaCrt;
5  opts.server_cert_len = strlen(testCaCrt);
6  opts.client_cert = testclientCert;
7  opts.client_cert_len = strlen(testclientCert);
8  opts.client_key = testclientPk;
9  opts.client_key_len = strlen(testclientPk);
10 #else
11 opts.is_tls = 0;
12 #endif
13 opts.host = MQTT_HOST;
14 opts.port = MQTT_PORT;
15 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

```

1  luat_mqtt_set_cb(luat_mqtt_ctrl, luat_mqtt_cb);

```

7.发起连接

```

1  luat_mqtt_connect(luat_mqtt_ctrl);

```

8.订阅与发布函数

```

1  //发布函数
2  /** Publish a message on a topic.
3   * @param MQTT实例对象
4   * @param topic 主题名称.
5   * @param msg 消息负载.
6   * @param msg_len 消息负载长度
7   * @param retain Enable or disable the Retain flag (values: 0 or 1).
8   * @param qos Quality of Service (values: 0, 1 or 2)
9   * @param message_id Variable that will store the Message ID, if the point
   er is not NULL.
10  *
11  * @retval 1 On success.
12  * @retval 0 On connection error.
13  * @retval -1 On IO error.
14  */
15  int mqtt_publish_with_qos(mqtt_broker_handle_t* broker,
16                           const char* topic,
17                           const char* msg,
18                           uint32_t msg_len,
19                           uint8_t retain,
20                           uint8_t qos,
21                           uint16_t* message_id);
22  //订阅函数
23  /** Subscribe to a topic.
24   * @param broker Data structure that contains the connection information w
   ith the broker.
25   * @param topic 主题名称.
26   * @param message_id Variable that will store the Message ID, if the point
   er is not NULL.
27  *
28  * @retval 1 On success.
29  * @retval 0 On connection error.
30  * @retval -1 On IO error.
31  */
32  int mqtt_subscribe(mqtt_broker_handle_t* broker,
33                    const char* topic,
34                    uint16_t* message_id,
35                    uint8_t qos);

```

9.回调函数与事件处理使用方法

```

1 //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
2 //处理方法如下
3 static luat_rtos_queue_t mqtt_queue_handle;
4 #define MQTT_QUEUE_SIZE 128
5 //创建消息队列的结构体
6 typedef struct
7 {
8     luat_mqtt_ctrl_t *luat_mqtt_ctrl;//mqtt 的实例对象
9     uint16_t event;//mqtt 事件类型
10 } mqttQueueData;
11 //MQTT 回调函数
12 static void luat_mqtt_cb(luat_mqtt_ctrl_t *luat_mqtt_ctrl, uint16_t event
13 {
14     //回调函数里面，通过消息队列将消息发送到主任务函数
15     mqttQueueData mqtt_cb_event = {.luat_mqtt_ctrl = luat_mqtt_ctrl,.event
16     t = event};
17     luat_rtos_queue_send(mqtt_queue_handle, &mqtt_cb_event, NULL, 0);
18     return;
19 }
20 static void luat_mqtt_task(void *param)
21 {
22
23     mqttQueueData mqttQueueRecv = {0};
24     //创建MQTT数据处理的消息队列
25     luat_rtos_queue_create(&mqtt_queue_handle, MQTT_QUEUE_SIZE, sizeof(mq
26     ttQueueData));
27     /*******省略部分代码，只展示事件处理与数据处理的代码**/
28     luat_mqtt_set_cb(luat_mqtt_ctrl,luat_mqtt_cb);
29     while(1)
30     {
31         //主任务里面接收MQTT数据，并进行处理
32         if (luat_rtos_queue_recv(mqtt_queue_handle, &mqttQueueRecv, NULL
33         , 5000) == 0)
34         {
35             switch (mqttQueueRecv.event)
36             {
37                 case MQTT_MSG_CONNACK:{//MQTT连接成功的标志
38                     LUAT_DEBUG_PRINT("mqtt_connect ok");
39
40                     LUAT_DEBUG_PRINT("mqtt_subscribe");
41                     uint16_t msgid = 0;
42                     mqtt_subscribe(&(mqttQueueRecv.luat_mqtt_ctrl->broker),
43                     mqtt_sub_topic, &msgid, 1);
44
45                     LUAT_DEBUG_PRINT("publish");
46                     uint16_t message_id = 0;

```

```

45         mqtt_publish_with_qos(&(mqttQueueRecv.luamqtt_ctrl->bro
46         ker),
47         mqtt_pub_topic, mqtt_send_payload,
48         strlen(mqtt_send_payload), 0, 1, &m
49         essage_id);
50         break;
51     }
52     case MQTT_MSG_PUBLISH : { //收到消息的标志
53         const uint8_t* ptr;
54         uint16_t topic_len =
55             mqtt_parse_pub_topic_ptr(mqttQueueRecv.luamqtt_ctrl
56             ->
57             mqtt_packet_buffer, &ptr
58             );
59         LUAT_DEBUG_PRINT("pub_topic: %.*s",topic_len,ptr);
60         uint16_t payload_len =
61             mqtt_parse_pub_msg_ptr(mqttQueueRecv.luamqtt_ctrl->
62             mqtt_packet_buffer, &ptr);
63         LUAT_DEBUG_PRINT("pub_msg: %.*s",payload_len,ptr);
64         break;
65     }
66     case MQTT_MSG_PUBACK :
67     case MQTT_MSG_PUBCOMP : {
68         LUAT_DEBUG_PRINT("msg_id: %d",mqtt_parse_msg_id
69         (mqttQueueRecv.luamqtt_ctrl->mqtt_pack
70         et_buffer));
71         break;
72     }
73     case MQTT_MSG_RELEASE : {
74         LUAT_DEBUG_PRINT("luamqtt_cb mqtt release");
75         break;
76     }
77     case MQTT_MSG_DISCONNECT : { // mqtt 断开(只要有断开就会上报,无论
78     是否重连)
79         LUAT_DEBUG_PRINT("luamqtt_cb mqtt disconnect");
80         break;
81     }
82     case MQTT_MSG_CLOSE : {
83         //mqtt 关闭(不会再重连)注意: 一定注意和MQTT_MSG_DISCONNECT区
84         别,
85         /*如果要手动重连处理推荐在这里 */
86         LUAT_DEBUG_PRINT("luamqtt_cb mqtt close");
87         if (MQTT_DEMO_AUTOCON == 0)
88         {
89             ret = luamqtt_connect(mqttQueueRecv.luamqtt_ctrl
90             );
91             if (ret) {
92                 LUAT_DEBUG_PRINT("mqtt connect ret=%d\n", ret);

```



```

85         luat_mqtt_close_socket(mqttQueueRecv.luamqtt_ctrl);
86     rl);
87         return;
88     }
89     break;
90 }
91 default:
92     break;
93 }
94 }
95 else
96 {
97     if (luat_mqtt_state_get(luat_mqtt_ctrl) == MQTT_STATE_READY)
98     {
99         uint16_t message_id = 0;
100         mqtt_publish_with_qos(&(luat_mqtt_ctrl->broker),
101                               mqtt_pub_topic,
102                               mqtt_send_payload,
103                               strlen(mqtt_send_payload),
104                               0, 1, &message_id);
105     }
106 }
107 }
108 }
109

```