WS63V100 MQTT

开发指南

文档版本　02

发布日期　2024-06-27

前言

概述

本文档主要介绍基于 MQTT 功能开发实现示例。

MQTT 基于开源组件 paho.mqtt.c-1.3.12 实现，详细说明请参考官方说明：
https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/index.html

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
WS63	V100

读者对象

本文档主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

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

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。
注意	表示如不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
02	2024-06-27	• 更新“2.2 订阅示例代码”小节内容。 • 更新“2.3 分发示例代码”小节内容。 • 更新“3.1 支持加密通路”小节内容。
01	2024-04-10	第一次正式版本发布。
00B01	2024-03-15	第一次临时版本发布。

目 录

1 API 接口描述

1.1 结构体说明

paho.mqtt.c 详细的结构体说明请参考官方说明文档：
https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/annotated.html

1.2 API 列表

paho.mqtt.c 详细的 API 说明请参考官方说明文档：
https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/globals_func.html

1.3 配置说明

paho.mqtt.c 详细配置说明请参考官方说明文档：
https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/globals_defs.html

2 开发指南

2.1 开发流程

使用 paho.mqtt.c 的应用程序通常使用类似的结构：

- 创建一个客户端对象
- 设置选项以连接到 MQTT 服务器
- 如果正在使用多线程（异步模式）操作，请设置回调函数（请参见官方说明“https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/async.html”）。
- 订阅客户需要接收的任何主题
- 重复直到完成：
  - 发布客户端需要的所有消息
  - 处理任何传入的消息
- 断开客户端
- 释放客户端正在使用的所有内存

具体实现可以参考官方说明中的示例：

- Synchronous publication example：
  https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/pubsync.html
- Asynchronous publication example：
  https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/pubasync.html

- Asynchronous subscription example:
  https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/subasync.html

2.2 订阅示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "MQTTClient.h"
#include "MQTTClientPersistence.h"
#include "osal_debug.h"
#include "MQTTClient.h"
#include "los_memory.h"
#include "los_task.h"

#define CLIENTID_SUB       "ExampleClientSub"
#define QOS                1
#define TIMEOUT            10000L
#define KEEPALIVEINTERVAL 20
#define CLEANSESSION       1

volatile MQTTClient_deliveryToken deliveredtoken;
volatile char g_subEnd = 0;
extern int MQTTClient_init(void);
void delivered(void *context, MQTTClient_deliveryToken dt)
{
    (void)context;
    osal_printk("Message with token value %d delivery confirmed\r\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;
    char *payloadptr = NULL;
    (void)context;
    (void)topicLen;
    osal_printk("Message arrived\r\n");
    osal_printk("        topic: %s\r\n", topicName);
    osal_printk("     message: ");
```

```
    payloadptr = message->payload;
    for (i = 0; i < message->payloadlen; i++) {
        osal_printk("%c", payloadptr[i]);
    }
    osal_printk("\r\n");

    if(memcmp(message->payload, "byebye", message->payloadlen) == 0) {
        g_subEnd = 1;
        osal_printk("g_subEnd = %d\r\n",g_subEnd);
    }
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}

void connlost(void *context, char *cause)
{
    (void)context;
    osal_printk("\nConnection lost\r\n");
    osal_printk("        cause: %s\r\n", cause);
}

int mqtt_002(char *addr, char *topic, char *user_name, char *password)
{
    osal_printk("start mqtt sync subscribe...\r\n");
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc = 0;

    MQTTClient_init();
    MQTTClient_create(&client, addr, CLIENTID_SUB, MQTTCLIENT_PERSISTENCE_NONE,
NULL);
    conn_opts.keepAliveInterval = KEEPALIVEINTERVAL;
    conn_opts.cleansession = CLEANSESSION;
    if (user_name != NULL) {
        conn_opts.username = user_name;
        conn_opts.password = password;
    }

    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        osal_printk("Failed to connect, return code %d\r\n", rc);
```

```
        return rc;
    }
    g_subEnd = 0;
    osal_printk("Subscribing to topic %s\nfor client %s using QoS%d\r\n\n"
            "wait for msg \" byebye\"\r\n\n", topic, CLIENTID_SUB, QOS);
    MQTTClient_subscribe(client, topic, QOS);
    do {
        LOS_TaskDelay(10);
    } while((g_subEnd == 0));
    osal_printk("Subscribing End\r\n", rc);
    MQTTClient_unsubscribe(client, topic);
    MQTTClient_disconnect(client, TIMEOUT);
    MQTTClient_destroy(&client);
    return rc;
}
```

2.3 分发示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "MQTTClient.h"
#include "MQTTClientPersistence.h"
#include "osal_debug.h"
#include "MQTTClient.h"
#include "los_memory.h"
#include "los_task.h"

#define CLIENTID_PUB       "ExampleClientPub"
#define QOS                1
#define TIMEOUT            10000L
#define KEEPALIVEINTERVAL 20
#define CLEANSESSION       1
extern int MQTTClient_init(void);
int mqtt_001(char *addr, char *topic, char *msg, char *user_name, char *password)
{
    osal_printk("start mqtt publish...\r\n");

    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
```

```
    MQTTClient_deliveryToken token;
    int rc = 0;

    MQTTClient_init();
    MQTTClient_create(&client, addr, CLIENTID_PUB, MQTTCLIENT_PERSISTENCE_NONE,
NULL);
    conn_opts.keepAliveInterval = KEEPALIVEINTERVAL;
    conn_opts.cleansession = CLEANSESSION;
    if (user_name != NULL) {
        conn_opts.username = user_name;
        conn_opts.password = password;
    }

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        osal_printk("Failed to connect, return code %d\r\n", rc);
        return -1;
    }

    pubmsg.payload = msg;
    pubmsg.payloadlen = (int)strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, topic, &pubmsg, &token);
    osal_printk("Waiting for up to %d seconds for publication of %s\r\n"
            "on topic %s for client with ClientID: %s\r\n",
            (int)(TIMEOUT/1000), msg, topic, CLIENTID_PUB);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    osal_printk("Message with delivery token %d delivered\r\n", token);
    MQTTClient_disconnect(client, TIMEOUT);
    MQTTClient_destroy(&client);
    return rc;
}
```

3 注意事项

3.1 支持加密通路

- 如果需要实现 MQTT 加密传输，MQTT 配置项中需要设置 SSL 参数。只做单端认证（客户端对服务端进行认证）时，需要提供认证服务端的根 CA 证书；做双端认证（客户端与服务端相互认证）时，除根 CA 证书外，还需要提供客户端证书与私钥。加密分发可参考如下代码示例：

说明

建议使用 TLS 版本为 1.2，证书位数至少为 2048 位。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "MQTTClient.h"
#include "MQTTClientPersistence.h"
#include "osal_debug.h"
#include "MQTTClient.h"
#include "los_memory.h"
#include "los_task.h"

#define CLIENTID_PUB       "ExampleClientPub"
#define QOS                1
#define TIMEOUT            10000L
#define KEEPALIVEINTERVAL 20
#define CLEANSESSION       1
```

```
/* 客户端证书，请自行填充 */
unsigned char client_crt[] = "\
-----BEGIN CERTIFICATE-----\r\n\
*****************************************************************\r\n\
*****************************************************************\r\n\
-----END CERTIFICATE-----\r\n\
";
/* 客户端私钥，请自行填充 */
unsigned char client_key[] = "\
-----BEGIN RSA PRIVATE KEY-----\r\n\
*****************************************************************\r\n\
*****************************************************************\r\n\
-----END RSA PRIVATE KEY-----\r\n\
";
/* 根CA证书，请自行填充 */
unsigned char ca_crt[] = "\
-----BEGIN CERTIFICATE-----\r\n\
*****************************************************************\r\n\
*****************************************************************\r\n\
-----END CERTIFICATE-----\r\n\
";
extern int MQTTClient_init(void);
int mqtt_005(char *addr, char *topic, char *msg, char *user_name, char *password)
{
    osal_printk("start mqtt ssl publish...\r\n");
    MQTTClient_SSLOptions ssl_opts = MQTTClient_SSLOptions_initializer;
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    int rc = 0;

    MQTTClient_init();
    cert_string keyStore = {client_crt, sizeof(client_crt)};
    cert_string trustStore = {ca_crt, sizeof(ca_crt)};
    key_string privateKey = {client_key, sizeof(client_key)};
    ssl_opts.los_keyStore = &keyStore;
    ssl_opts.los_trustStore = &trustStore;
    ssl_opts.los_privateKey = &privateKey;
    ssl_opts.sslVersion = MQTT_SSL_VERSION_TLS_1_2;

    MQTTClient_create(&client, addr, CLIENTID_PUB,
```

```
MQTTCLIENT_PERSISTENCE_NONE, NULL);
    conn_opts.keepAliveInterval = KEEPALIVEINTERVAL;
    conn_opts.cleansession = CLEANSESSION;
    conn_opts.ssl = &ssl_opts;
    if (user_name != NULL) {
        conn_opts.username = user_name;
        conn_opts.password = password;
    }

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        osal_printk("Failed to connect, return code %d\r\n", rc);
        return -1;
    }
    pubmsg.payload = msg;
    pubmsg.payloadlen = (int)strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, topic, &pubmsg, &token);
    osal_printk("Waiting for up to %d seconds for publication of %s\r\n"
            "on topic %s for client with ClientID: %s\r\n",
            (int)(TIMEOUT/1000), msg, topic, CLIENTID_PUB);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    osal_printk("Message with delivery token %d delivered\r\n", token);
    MQTTClient_disconnect(client, TIMEOUT);
    MQTTClient_destroy(&client);
    return rc;
}
```