STM32 FOTA 例程之 Paho MQTTClient embeddedC 使用

前言

MQTT 协议的全称叫“消息队列遥测传输”协议。它是一个轻量级的通信协议。旨在为在低带宽、高延时、不稳定网络中的物联网设备提供消息传输服务。它运行在 TCP/IP 协议之上，采用客户端/服务器，发布/订阅消息模式工作，并提供一对多的消息分发。STM32 FOTA demo 就是通过 MQTT 协议进行 MCU 固件新版本信息的推送。Paho 是一个开源的 MQTT 客户端实现，它提供了多种开发语言下的实现。在此 demo 中，用的是 embeddedC 这个版本。可以在 https://github.com/eclipse/paho.mqtt.embedded-c下载到最新的版本。

Paho MQTTClient EmbeddedC 的代码构成

从 github 上下载下来的源代码，包括三个部分：

MQTTPacket : 该文件夹下包括了底层的 C 代码，提供基本的简单的解析数据，以及将数据串行化的功能。是其他两个上层接口的基础，也可以单独使用。

MQTTClient ： 该文件夹下提供 C++的上层接口，现在提供 Linux, Arduino 和 mbed 的实现。

MQTTClient-C：该文件夹下提供 C 的上层接口，针对那些不支持 C++编程的平台。

Demo 要用到的就是 MQTTPacket 和 MQTTClient-C 这两个文件夹下的源文件。

使用 Paho MQTTClient EmbeddedC

下面来看看如何使用 Paho MQTTClient EmbeddedC 来在 MCU 端实现 MQTT 通信。

配置网络传输接口

MQTT 是一个 TCP 之上的应用层协议，它发送和接收数据都要通过下层的 TCP/IP 协议栈进行。所以 MQTT 与下层的协议之间一定有一个接口。

Paho 对这个数据收发的接口进行了形式上的定义：

```
struct Network
{
  net_sockhnd_t my_socket;
  int (*mqttread) (Network*, unsigned char*,int,int);
  int (*mqttwrite) (Network*,unsigned char*,int,int);
  int (*disconnect) (Network*);
};
```

并通过以下形式在 Paho 中（MQTTClient.c）调用：

```
rc = c->ipstack->mqttread(c->ipstack, &i, 1, timeout);
…
int rc = c->ipstack->mqttread(c->ipstack, c->readbuf, 1, TimerLeftMS(timer));
…
```

```
rc = c->ipstack->mqttwrite(c->ipstack, &c->buf[sent], length, TimerLeftMS(timer));
```

Paho 的实现会通过结构体 Network 中的 mqttread 和 mqttwrite 成员函数作为接口来从底层网络读取数据以及向底层网络发送数据。因此 Paho 的适配工作需要注册这些成员函数，并实现之：

在 STM32 FOTAdemo 中，Network 结构体是在 baidu_iot_network_wrapper.h 中定义的：

```
struct Network
{
  net_sockhnd_t my_socket;
  int (*mqttread) (Network*, unsigned char*,int,int);
  int (*mqttwrite) (Network*,unsigned char*,int,int);
  int (*disconnect) (Network*);
};
```

baidu_iot_network_wrapper.c 文件中，有 MQTT 的网络接口函数的具体实现。

```
int mqtt_network_new(Network *network)
{
  network->my_socket = 0;
  network->mqttread = mqtt_socket_recv;
  network->mqttwrite = mqtt_socket_send;
  network->disconnect = mqtt_socket_disconnect;

  return SUCCESS;
}
…
int mqtt_socket_send(Network *network, unsigned char *buf, int len, int timeout)
{
  uint16_t rc;
  int socket = (int) (network->my_socket);

  rc = net_sock_send((void*)socket,buf,len);

  return rc;
}

int mqtt_socket_recv(Network *network, unsigned char *buf, int len, int timeout)
{
    uint16_t rc;
  int socket = (int) (network->my_socket);

  rc = net_sock_recv((void*)socket,buf,len);
  return rc;
}
```

新建一个 MQTT 客户端

通过 MQTTClientInit 函数可以新建一个 MQTT 客户端。我们来看看在 Paho 中，MQTT 客户端都定义了哪些属性。

MQTTClient 的定义在 MQTTClient.h 文件中，如下：

```
typedef struct MQTTClient
```

```
{
    unsigned int next_packetid, command_timeout_ms;
    size_t buf_size, readbuf_size;
    unsigned char *buf, *readbuf;
    unsigned int keepAliveInterval;
    char ping_outstanding;
    int isconnected;
    int cleansession;

    struct MessageHandlers
    {
        const char* topicFilter;
        void (*fp) (MessageData*);
    } messageHandlers[MAX_MESSAGE_HANDLERS];      /* Message handlers are indexed by
subscription topic */

    void (*defaultMessageHandler) (MessageData*);

    Network* ipstack;
    Timer last_sent, last_received;
#if defined(MQTT_TASK)
    Mutex mutex;
    Thread thread;
#endif
} MQTTClient;
```

MQTTClient 结构体的定义包括：接收/发送数据的缓冲区（readbuffer 和 buf）,保持心跳的时间间隔（keepAliveInterval），当前的连接状态（isconnected），消息句柄（messageData）以及网络接口（ipstack）等内容。

调用 MQTTClientInit 函数时需要输入的参数有：已经初始化好的网络接口（Network*）结构体，COMMAND_TIMEOUT_MS 和接收/发送数据的 buffer。

初始化 MQTTClient 后，就可以通过 MQTTConnect 来和服务器建立连接了。在和服务器建立连接的时候，还需要设定一些和建立连接以及后面通信相关的参数，比如：用户名、密码、心跳包的间隔、遗嘱信息(will)、设备与服务器意外断开后服务器是否要保留后续消息（cleansession）等等。都可以通过对 Connect_para 进行初始化来设置这些参数，再调用 MQTTConnect 函数建立相关连接。

下面就是一段新建 MQTTClient 并与服务器建立连接的代码。

```
int connect2MQTTServer(void)
{
  int ret;
  uint8_t count=0;
  MQTTPacket_connectData Connect_para = MQTTPacket_connectData_initializer;

  ret = 0;

MQTTClientInit(&Client,&sNetwork,COMMAND_TIMEOUT_MS,MQTT_write_buf,sizeof(MQTT_write_buf)
,MQTT_read_buf,sizeof(MQTT_read_buf));
  //Connect_para.MQTTVersion = MQTT_3_1_1;
  Connect_para.clientID.cstring = MQTT_CLIENT_ID;
  Connect_para.username.cstring =  MQTT_USER_NAME;
```

```
  Connect_para.password.cstring =  MQTT_PASSWORD;

  //MQTT connect,will option is set in Connect_para
  do
  {
    count++;
    msg_info("Attempt %d/%d ...\n",count,CONNECT_MAX_ATTEMPT_COUNT);
    if((ret = MQTTConnect(&Client, &Connect_para)) != 0)
    {
       msg_error("Client connection with Baidu MQTT Broker failed with error
code: %d\n",ret);
    }
  }while((ret != 0)&&(count < CONNECT_MAX_ATTEMPT_COUNT));

  return ret;
}
```

发送数据

和服务器的 MQTT 连接建立成功后，就可以发布和订阅消息了。

Paho 中发布消息的函数是

```
int MQTTPublish(MQTTClient* c, const char* topicName, MQTTMessage* message)
```

它有三个输入参数：

MQTTClient* c：就是前面新建的 MQTTClient

const char* topicName：mqtt 中每一条消息都是和某个主题相对应的，所以在发布消息的时候一定要指明这条消息是发往哪个主题

MQTTMessage* message：将要被发送的消息

除了消息的内容，还需要设定好消息传递的 Qos 级别以及服务器是否需要保存这条消息等。

下面是本例程中，向百度天工 IoT 的 MQTT 服务器发布消息的一段代码：

```
int baiduiot_devicestatus_update(void)
{
  //pub_device_status();
  int ret;
  MQTTMessage MQTT_msg;
  ret =0;

  MQTT_msg.qos = QOS1;//QoS1;
  MQTT_msg.dup = 0;//The DUP flag MUST be set to 1 by the Client or Server when it
attempts to re-deliver a PUBLISH Packet
  //The DUP flag MUST be set to 0 for all QoS 0 messages
  MQTT_msg.retained = 1; //the Server MUST store 757 the Application Message and its QoS
  MQTT_msg.payload = payload_buf;


  //TODO:prepare pub payload,@sz
  PrepareMqttPayload(payload_buf, sizeof(payload_buf));
```

```
  MQTT_msg.payloadlen = strlen(payload_buf);

  //Create message to publish
  //

  if((ret=MQTTPublish(&Client, BAIDU_DEVICE_SHADOW_UPDATE_TOPIC,&MQTT_msg)) != 0)
  {
    msg_error("Failed to publish data. %d\n",ret);
  }
  else
  {

    msg_info("publish device status successfully.\n");
  }
  return ret;
}
```

接收数据

如果设备端希望能接收服务器发的某个主题的消息，需要做这么几件事：

- 向服务器订阅这类消息所属的主题
- 注册用来处理接收到的消息的回调函数

通过以下函数就可以实现向服务器订阅某个主题

```
int MQTTSubscribe(MQTTClient* c, const char* topicFilter, enum QoS qos, messageHandler
messageHandler)
```

```
int baiduiot_sub_shadow(void)
{
  //sub_baiduIOT_shadow_topic();
  int ret;
  ret = 0;


if((ret=MQTTSubscribe(&Client,BAIDU_DEVICE_SHADOW_DELTA_TOPIC,QOS0,MQTTcallbackHandler))!
=0)
    printf("fail to subscribe to topic.\n");
  else
    printf("subscribe to topic: %s\n",BAIDU_DEVICE_SHADOW_DELTA_TOPIC);

  return ret;
}
```

并注册消息处理回调函数：MQTTcallbackHandler()。

注意，这里同样需要指定一个 QoS 级别，之后服务器向设备端推送消息的时候，就会按照这个 QoS 级别进行。在 MQTT 应用中，即使是同一个设备端和服务器之间的通信，发布消息和订阅消息也可以采用不同的 QoS 级别。

另外还有一个函数很重要

```
int MQTTYield(MQTTClient* c, int timeout_ms)
```

该函数需要被定期调用，来接收服务器发来的数据。前面注册的消息处理回调函数 MQTTcallbackHandler()就是在 MQTTYield 调用时被执行的。

```
MQTTYield(c, MQTT_YIELD_DELAY) → cycle(c, &timer) → deliverMessage(c, &topicName, &msg)
……
……..
            if (c->messageHandlers[i].fp != NULL)
            {
                MessageData md;
                NewMessageData(&md, topicName, message);
                c->messageHandlers[i].fp(&md);
                rc = SUCCESS;
            }
```

```
MQTTSubscribe(…,MQTTcallbackHandler) → MQTTSubscribeWithResults(…,MQTTcallbackHandler) →
MQTTSetMessageHandler(…, messageHandler)
……
……
            c->messageHandlers[i].topicFilter = topicFilter;
            c->messageHandlers[i].fp = messageHandler;
```

介绍到这里，相信大家已经能够使用 Paho 来实现和 MQTT 服务器的基本通信了。
最后小结一下：对于一个 MQTT 客户端，MQTT 应用先通过 MQTTClientInit()建立连接，发布消息就调用 MQTTPublish()；订阅消息通过 MQTTSubscribe()，但是需要同时注册对收到的消息的处理函数，以函数参数的形式传给 MQTTSubscribe；对于所订阅的消息，会异步地从云端下发过来，MQTT 客户端需要定期调用 MQTTYield()来收取并处理。

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。