STM32 MCU Development

STM32 单片机开发

-- STM32 基于 MQTT 协议实时监控

目录

前面的示例中我们主要实现了：

1. Led 流水灯及按键中断控制 Led 灯亮灭；
2. 串口实时监测温湿度，远程控制 Led 灯亮灭；
3. WiFi(TCP/IP)实时监测温湿度，远程控制 Led 灯亮灭；

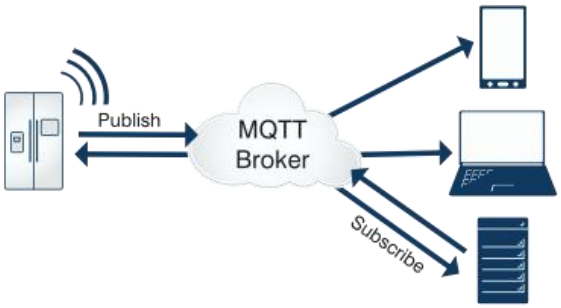
接下来我们将采用标准的物联网协议 MQTT 实现一个完整的项目实战，通过手机 App 实时订阅获取开发板上报的温湿度，并能远程控制 Led、继电器灯泡的亮灭。

1. MQTT 协议介绍

1.1. MQTT 简介

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输协议），是一种基于发布/订阅（publish/subscribe）模式的"轻量级"通讯协议，该协议构建于 TCP/IP 协议上，由 IBM 在 1999 年发布。MQTT 最大优点在于，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。作为一种低开销、低带宽占用的即时通讯协议，使其在物联网、小型设备、移动应用等方面有较广泛的应用。

MQTT 是一个基于客户端-服务器的消息发布/订阅传输协议。MQTT 协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。



由于物联网的环境是非常特别的，所以 MQTT 遵循以下设计原则：

（1）精简，不添加可有可无的功能；
（2）发布/订阅（Pub/Sub）模式，方便消息在传感器之间传递；
（3）允许用户动态创建主题，零运维成本；
（4）把传输量降到最低以提高传输效率；
（5）把低带宽、高延迟、不稳定的网络等因素考虑在内；
（6）支持连续的会话控制；
（7）理解客户端计算能力可能很低；
（8）提供服务质量管理；
（9）假设数据不可知，不强求传输数据的类型与格式，保持灵活性。

1.2. MQTT 协议

MQTT 协议实现方式

实现 MQTT 协议需要客户端和服务器端通讯完成，在通讯过程中，MQTT 协议中有三种身份：发布者（Publisher）、代理（Broker）（服务器）、订阅者（Subscriber）。其中，消息的发布者和订阅者都是客户端，消息代理是服务器，消息发布者可以同时是订阅者。MQTT 传输的消息分为：主题（Topic）和负载（payload）两部分：

（1）Topic，可以理解为消息的类型，订阅者订阅（Subscribe）后，就会收到该主题的消息内容（payload）；
（2）payload，可以理解为消息的内容，是指订阅者具体要使用的内容。

网络传输与应用消息

MQTT 会构建底层网络传输：它将建立客户端到服务器的连接，提供两者之间的一个有序的、无损的、基于字节流的双向传输。当应用数据通过 MQTT 网络发送时，MQTT 会把与之相关的服务质量（QoS）和主题名（Topic）相关连。

MQTT 客户端

一个使用 MQTT 协议的应用程序或者设备，它总是建立到服务器的网络连接。客户端可以：
（1）发布其他客户端可能会订阅的信息；
（2）订阅其它客户端发布的消息；
（3）退订或删除应用程序的消息；
（4）断开与服务器连接。

MQTT 服务器

MQTT 服务器以称为"消息代理"（Broker），可以是一个应用程序或一台设备。它是位于消息发布者和订阅者之间，它可以：
（1）接受来自客户的网络连接；
（2）接受客户发布的应用信息；
（3）处理来自客户端的订阅和退订请求；
（4）向订阅的客户转发应用程序消息。

MQTT 协议中的订阅、主题、会话

一、订阅（Subscription）
订阅包含主题筛选器（Topic Filter）和最大服务质量（QoS）。订阅会与一个会话（Session）关联。一个会话可以包含多个订阅。每一个会话中的每个订阅都有一个不同的主题筛选器。

二、会话（Session）
每个客户端与服务器建立连接后就是一个会话，客户端和服务器之间有状态交互。会话存在于一个网络之间，也可能在客户端和服务器之间跨越多个连续的网络连接。

三、主题名（Topic Name）
连接到一个应用程序消息的标签，该标签与服务器的订阅相匹配。服务器会将消息发送给订阅所匹配标签的每个客户端。

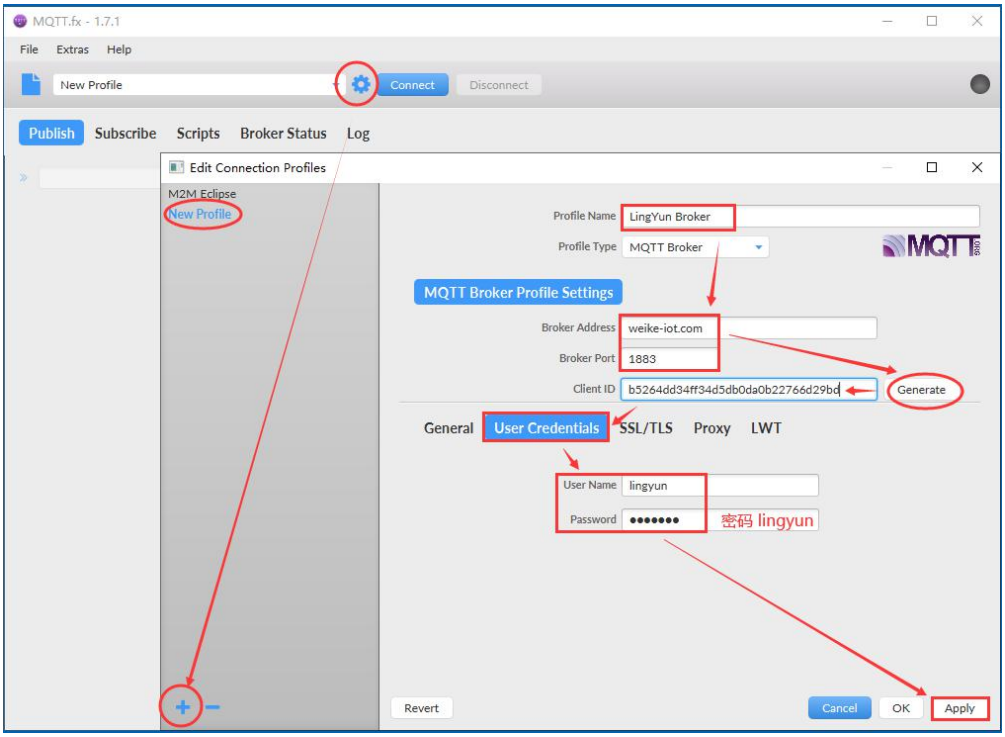
四、主题筛选器（Topic Filter）
一个对主题名通配符筛选器，在订阅表达式中使用，表示订阅所匹配到的多个主题。

五、负载（Payload）
消息订阅者所具体接收的内容。

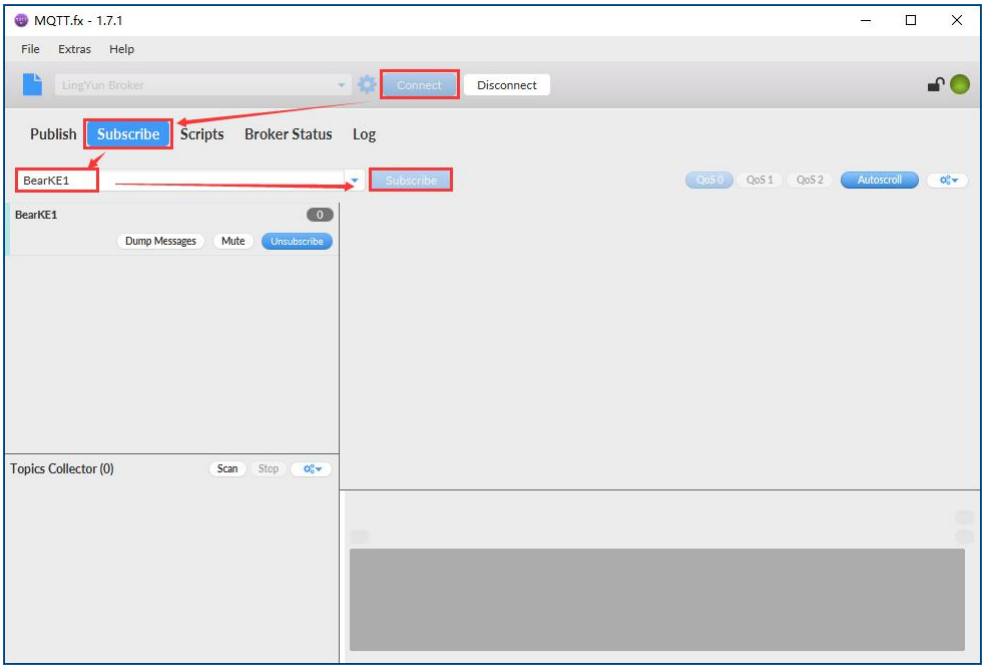
1.3. MQTT 使用

MQTT.fx 是目前主流的 MQTT 桌面客户端，它支持 Windows、Mac、Linux 操作系统，可以快速验证是否可与 MQTT Broker 连接并发布或订阅消息，这样我们可以使用它来理解 MQTT 协议的学习和调试。 需要注意的是，从 2021 年开始 MQTT.fx 交给新成立的一家公司 Softblade 运作，最新版本需要注册码才能工作，但对于我们当前的应用来说，v1.7.1 就足够了，其下载地址为：http://www.jensd.de/apps/mqttfx/1.7.1/ 。

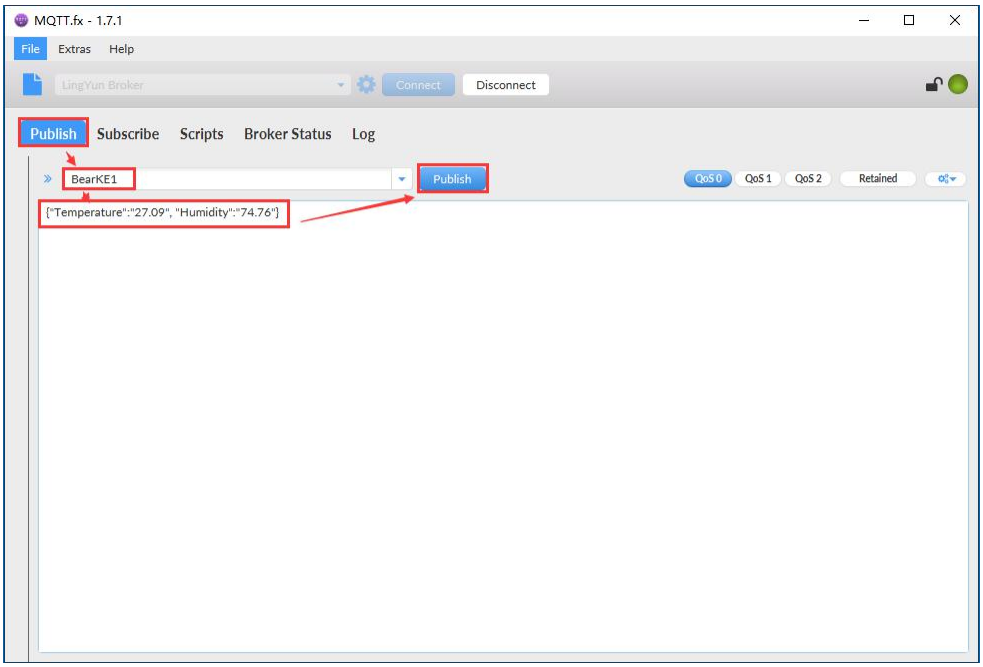
下载完成后点击安装即可，然后打开并运行 MQTT.fx 程序，现在我们将模拟实现 MQTT 的订阅和发布端。首先，如下图所示创建 MQTT Broker 的连接，这里直接使用凌云实验室的 Broker 服务器。其 Broker 地址为 weike-iot.com ，端口号为 MQTT 默认的端口号 1883、用户名和密码均为 lingyun。



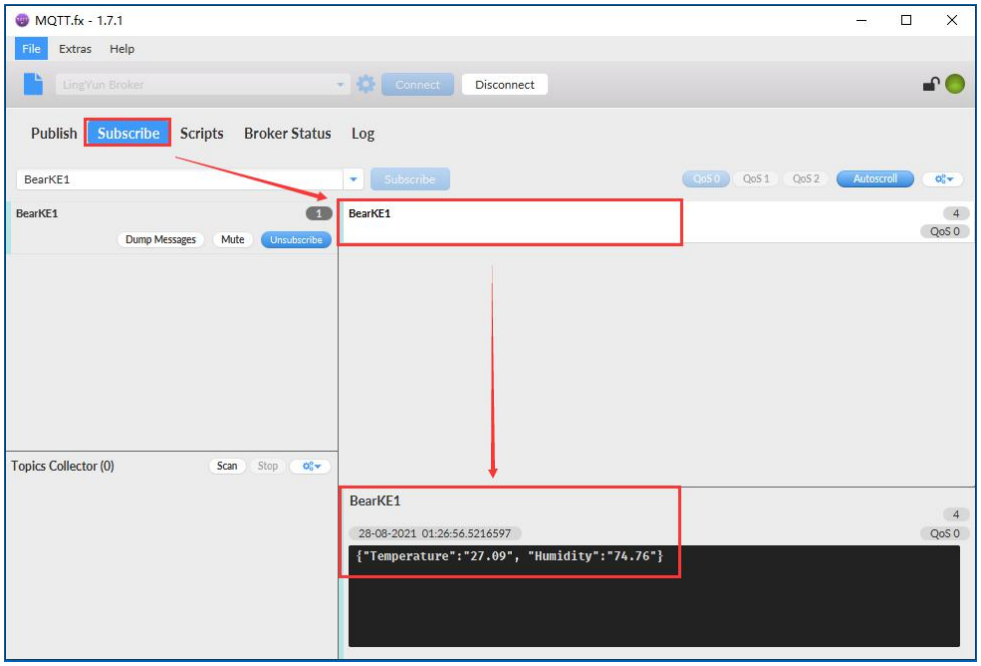
接下来点击“Connect”按钮连接凌云实验室的 MQTT Broker 服务器，然后选择 “Subscribe”菜单并设置订阅主题为 BearKE1 ，再点击右侧的 “Subscribe”开始订阅该主题。



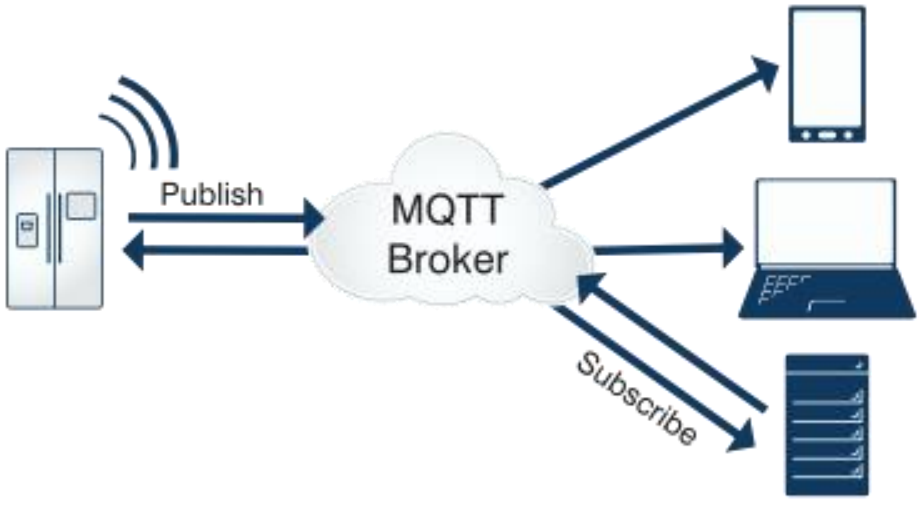
接下来再选择“Publish”菜单，设置发布的主题也同样为 BearKE1（订阅的主题和发布的主题一致，订阅端才可以收到）， 然后这里设置发布的消息为 JSON 格式的温湿度值（事实上随便一个什么消息都可以）。然后点击“Publish”按钮开始发布消息。



接下来切换到 “Subscribe”菜单下，我们就可以订阅者可以收到 Publish 发布的消息了。



需要注意的是，这里测试的时候发布者和订阅者虽说都是在同一个 PC 上的同一个程序，但事实上他们运行在不同的 PC 上，使用不同的 MQTT 工具他们之间也可以互相通信。具体工作流程如下图所示：



1.4. MQTT 源码库

MQTT 是基于 TCP/IP 协议、与编程语言无关的标准物联网通信协议，正因为其在物联网系统中有非常广泛的应用，所以各种编程语言都有其开源实现，具体请参考：

- 服务端列表: https://github.com/mqtt/mqtt.github.io/wiki/servers
- 客户端列表: https://github.com/mqtt/mqtt.github.io/wiki/libraries

由于单片机的 Flash、内存容量有限，所以我们不能在上面跑很多其它开源的库，而应该采用适合单片机使用的 MQTT 库，这里我们就选择 Paho Embedded C 库，Paho 是 MQTT 的官方开源库，其有很多版本，各版本之间的特性比较如下：

Client	MQTT 3.1	MQTT 3.1.1	MQTT 5.0	LWT	SSL / TLS	Automatic Reconnect	Offline Buffering	Message Persistence	WebSocket Support	Standard MQTT Support	Blocking API	Non-Blocking API	High Availability
Java	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Python	✔	✔	✖	✔	✔	✔	✔	✖	✔	✔	✔	✔	✖
JavaScript	✔	✔	✖	✔	✔	✔	✔	✔	✔	✖	✖	✔	✔
GoLang	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
C	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
C++	✔	✔	✖	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔
Rust	✔	✔	✖	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔
.Net (C#)	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✖	✔	✖
Android Service	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
Embedded C/C++	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✔	✔	✖

最新版 Paho MQTT 库项目地址: https://github.com/eclipse/paho.mqtt.embedded-c 。因为众所周知的原因，github 访问经常很慢，大家也可以从凌云实验室的文件服务器上下载，下载地址为：

http://master.iot-yun.club:2211/src/paho.mqtt.embedded-c-master.zip

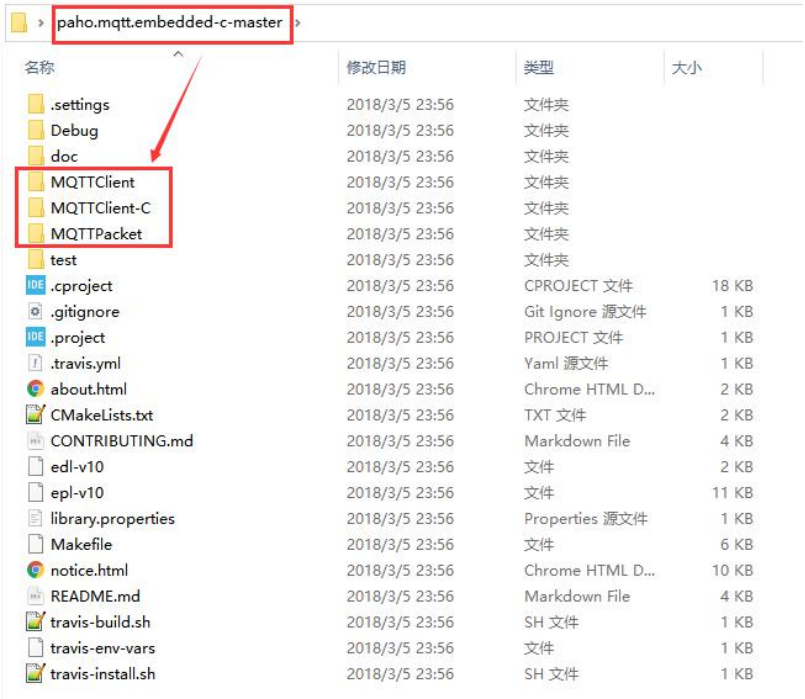
需要注意的是，这个官方的源码拿来并不能直接使用，我们需要将其针对 ESP8266 WiFi 模块做移植。

2. Paho MQTT 移植

2.1. 源码分析

将下载好的 Paho 源码解压缩并打开文件夹，我们可以看到这里主要有三个源码相关文件夹：

- MQTTClient，这里有 arduino、linux、mbed 等系统移植相关的 c++参考代码；
- MQTTClient-C，这里有 cc3200、linux、FreeRTOS 等系统移植相关的 c 参考代码；
- MQTTPacket，这里面提供了 MQTT 协议数据报文打包、解析的相关代码。

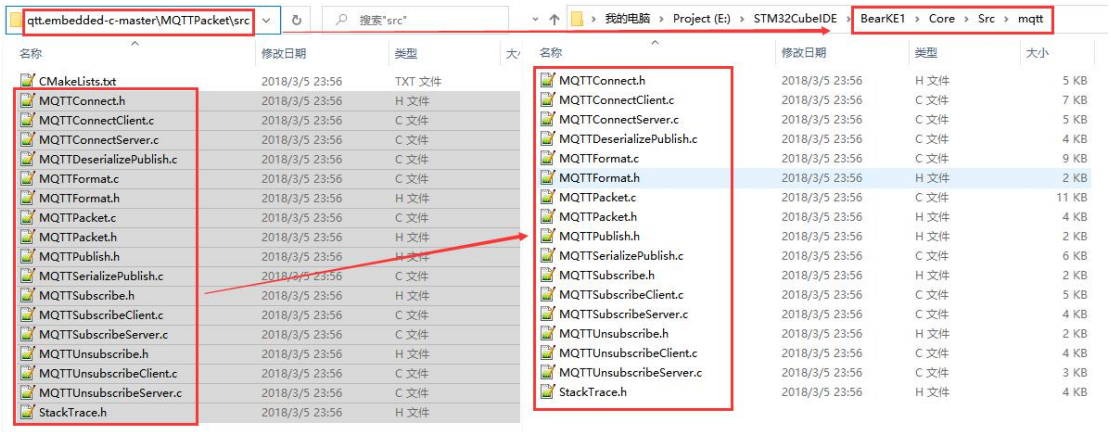


这里我们在移植它到 STM32 单片机的时候，我们只需要 MQTTPacket\src 文件夹下的一些源码文件即可。

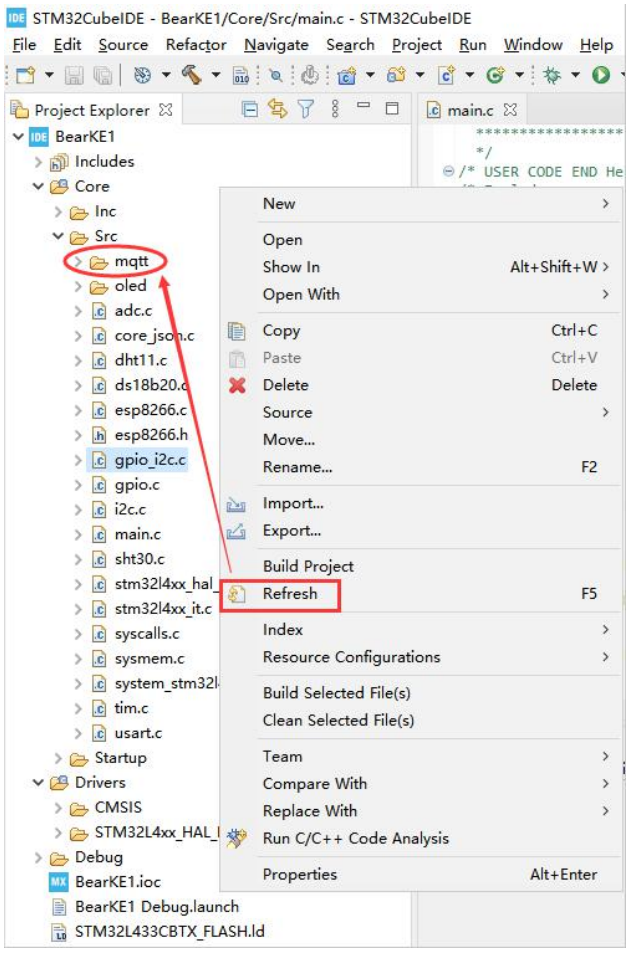
2.2. 源码移植

在 STM32CubeIDE 的项目路径 BearKE1\Core\Src 下，新建一个 mqtt 的文件夹：

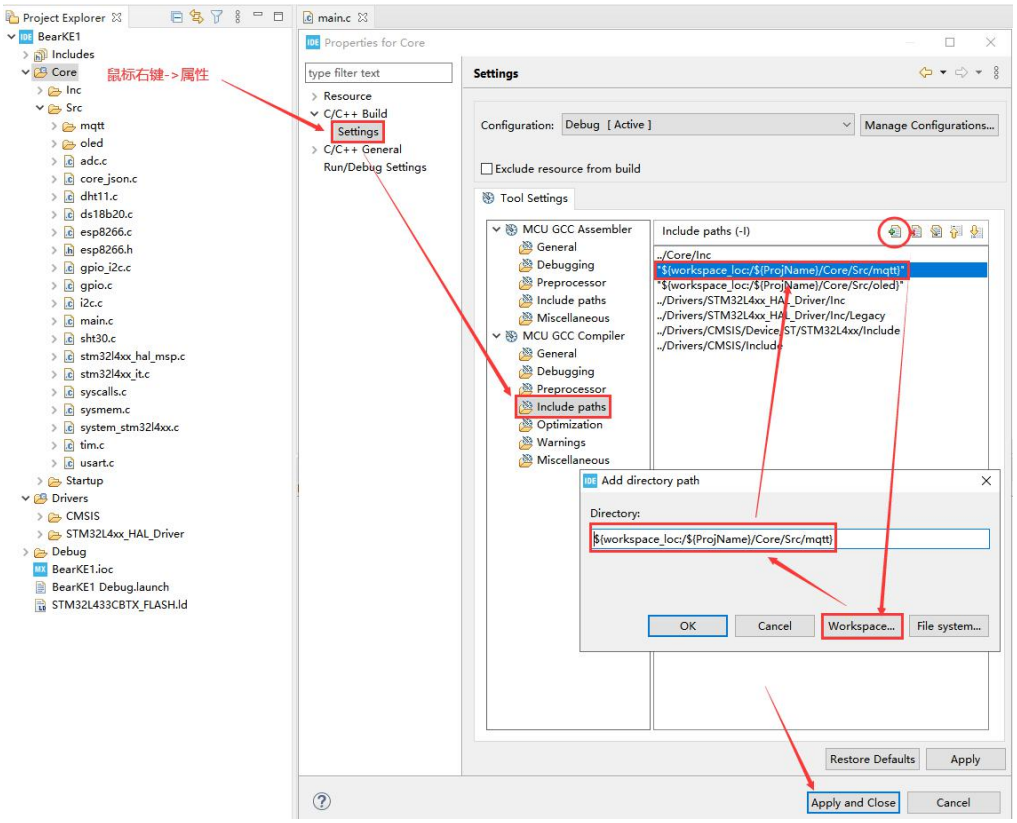
接下来把 Paho 源码里 MQTTPacket\src 文件夹下的所有.c/.h 文件拷贝到 STM32CubeIDE 开发板项目 Core\Src\mqtt 路径下。



接下来使用 STM32CubeIDE 打开项目，鼠标右键点在左侧项目栏上选择“Refresh”，这样就会自动添加 mqtt 文件夹相关文件到项目文件夹下去。



再次鼠标右键点在左侧项目栏 Core 上选择“Properties”，然后添加 mqtt 的路径到 include 包含头文件的路径下。解决将来代码编译时找不到头文件的 bug。



2.3. MQTT 发送接口

前面我们提到，Paho MQTT 库主要是提供了 MQTT 协议数据报文打包和报文解析的相关函数。因为不同的硬件和通信方式不一样，其底层的收发接口也不一样，这样底层的 MQTT 消息发送的代码和上层 MQTT 订阅、发布功能代码都需要用户在移植的时候自己实现。

现在我们创建 STM32 单片机上使用 ESP8266 发送 MQTT 消息的底层源文件 transport.c/h。首先在 mqtt 下创建并编写头文件 transport.h 如下：

```
/*****************************************************************************
*   Copyright: (C)2021 GuoWenxue<guowenxue@gmail.com>
*
* Description: Paho MQTT library portable lowlevel API for ESP8266
*
*   ChangeLog:
*       Version   Date       Author        Description
*       V1.0.0  2021.08.25  GuoWenxue   Release initial version.
*
*****************************************************************************/
```

```
#ifndef __TRANSPORT_H_
#define __TRANSPORT_H_

/* 使用 ESP8266 创建 socket 连接到 MQTT 服务器函数 */
extern int transport_open(char* host, int port);

/* ESP8266 关闭 socket 连接函数 */
extern int transport_close(void);

/* 使用 ESP8266 发送一个 MQTT 数据报文的函数 */
extern int transport_sendPacketBuffer(unsigned char* buf, int buflen);

/* 使用 ESP8266 接收 MQTT 数据报文函数 */
extern int transport_getdata(unsigned char* buf, int count);

/* 清除 ESP8266 接收 MQTT 数据的 socket buffer 函数 */
extern void transport_clearBuf(void);

#endif /* endof __TRANSPORT_H_ */
```

接着在 mqtt 文件夹下创建并编写源文件 transport.c 如下：

```
/*******************************************************************************
*   Copyright: (C)2021 GuoWenxue<guowenxue@gmail.com>
*
* Description: Paho MQTT library portable lowlevel API for ESP8266
*
*   ChangeLog:
*       Version   Date       Author        Description
*       V1.0.0  2021.09.26  GuoWenxue   Release initial version.
*
*******************************************************************************/
#include <string.h>
#include "transport.h"
#include "esp8266.h"

static unsigned char        s_sock_buf[256];
static int                 s_rx_bytes;


int transport_open(char* addr, int port)
{
        return esp8266_sock_connect(addr, port) ? -1 : 0 ;
}

int transport_close(void)
{
        return esp8266_sock_disconnect();
}

int transport_sendPacketBuffer( unsigned char* buf, int buflen)
{
        return esp8266_sock_send(buf, buflen);
}
```

```
int transport_getdata(unsigned char* buf, int count)
{
        int                         rv = 0;

        if( !s_rx_bytes )
        {
         rv = esp8266_sock_recv(s_sock_buf, sizeof(s_sock_buf));
         if( !rv )
         {
              return 0;
         }

         s_rx_bytes = rv;
        }

        rv = count>s_rx_bytes ? s_rx_bytes : count ;

        memcpy(buf, s_sock_buf, rv);
        s_rx_bytes -= rv;

        if( s_rx_bytes > 0)
         memmove(s_sock_buf, &s_sock_buf[rv], s_rx_bytes);

        return rv;
}

void transport_clearBuf(void)
{
        memset(s_sock_buf, 0, sizeof(s_sock_buf));
        s_rx_bytes = 0;
}
```

2.4. MQTT 上层接口

接下来我们再编写 MQTT 订阅、发布等上层调用功能函数源文件 core_mqtt.c/h。首先在 mqtt 下创建并编写头文件 core_mqtt.h 如下：

```
/*****************************************************************************
 *     Copyright:  (C) Guo Wenxue <guowenxue@gmail.com>
 *                 All rights reserved.
 *
 *      Filename:  core_mqtt.h
 *    Description:  This head file is MQTT client API based on Paho MQTT Embedded
 *
 *   ChangeLog:
 *       Version   Date       Author        Description
 *       V1.0.0  2021.08.26  GuoWenxue   Release initial version.
 *
 *****************************************************************************/
```

```
#ifndef  _CORE_MQTT_H_
#define  _CORE_MQTT_H_

#include "MQTTPacket.h"
#include "transport.h"

#define MQTT_KEEP_ALIVE_TIMEOUT_SECONDS       ( 60U )

enum
{
    Qos0=0,
    Qos1,
    Qos2,
};

/* MQTT 连接 Broker 函数 */
extern int mqtt_connect(char *host, int port, char *clentid, char *username, char *passwd);

/* MQTT 断开 Broker 连接函数 */
extern int mqtt_disconnect(void);

/* MQTT 订阅主体函数 */
extern int mqtt_subscribe_topic(char *topic, int qos, int msgid);

/* MQTT 取消主体订阅函数 */
extern int mqtt_unsubscribe_topic(char *topic, int msgid);

/* MQTT 发布消息函数 */
extern int mqtt_publish(char *topic, int qos, char *payload);

/* MQTT 保持连接心跳包函数 */
extern int mqtt_pingreq(void);

#endif   /* ----- #ifndef _CORE_MQTT_H_  ----- */
```

接着在 mqtt 文件夹下创建并编写源文件 core_mqtt.c 如下：

```
/*****************************************************************************
 *     Copyright:  (C) Guo Wenxue <guowenxue@gmail.com>
 *                 All rights reserved.
 *
 *      Filename:  core_mqtt.h
 *    Description:  This file is MQTT client API based on Paho MQTT Embedded
 *
 *   ChangeLog:
 *       Version   Date       Author        Description
 *       V1.0.0  2021.08.26  GuoWenxue   Release initial version.
 *
 *****************************************************************************/
```

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

#include "stm32l4xx_hal.h"
#include "core_mqtt.h"

int mqtt_connect(char *host, int port, char *clientid, char *username, char *passwd)
{
    MQTTPacket_connectData         data = MQTTPacket_connectData_initializer;
    int                           rv;
    unsigned char                 buf[256];
    unsigned char                 sessionPresent;
    unsigned char                 connack_rc;

    if( !host || port<=0 || !clientid )
    {
         printf("ERROR: Invalid input arguments\r\n");
         return -1;
    }

    if( (rv=transport_open(host, port)) < 0 )
    {
          printf("socket connect [%s:%d] failure, rv=%d\r\n", host, port, rv);
          return rv;
    }
    printf("socket connect [%s:%d] ok\r\n", host, port);

    data.clientID.cstring = clientid;
    data.keepAliveInterval = MQTT_KEEP_ALIVE_TIMEOUT_SECONDS;
    data.cleansession = 1;

    if( username && passwd )
    {
         data.username.cstring = username;
         data.password.cstring = passwd;
    }

    rv=MQTTSerialize_connect(buf, sizeof(buf), &data);
    if( rv < 0 )
    {
        printf("MQTTSerialize_connect failure, rv=%d\n", rv);
        return -1;
    }

    if ( rv != transport_sendPacketBuffer(buf,rv) )
    {
        printf("transport_sendPacketBuffer for mqtt_connect failure, rv=%d\n", rv);
        return -2;
    }

    HAL_Delay(800);
```

```
    memset(buf, 0, sizeof(buf));
    rv = MQTTPacket_read(buf, sizeof(buf), transport_getdata);
    if( CONNACK != rv )
    {
        printf("MQTTPacket_read for MQTT CONNACK failure, rv=%d\n", rv);
        return -3;
    }

    if( (rv=MQTTDeserialize_connack(&sessionPresent, &connack_rc, buf, sizeof(buf))) != 1 ||
connack_rc!=0)
    {
        printf("MQTTDeserialize_connack failure, rv=%d\n", rv);
        return -4;
    }

    return 0;
}


int mqtt_disconnect(void)
{
    int                             rv;
    unsigned char                    buf[256];

    rv = MQTTSerialize_disconnect(buf, sizeof(buf));
    if( rv < 0 )
    {
        printf("MQTTSerialize_disconnect failure, rv=%d\n", rv);
        return -1;
    }

    if ( rv != transport_sendPacketBuffer(buf,rv) )
    {
        printf("transport_sendPacketBuffer for mqtt_disconnect failure, rv=%d\n", rv);
        return -2;
    }

    return 0;
}

int mqtt_subscribe_topic(char *topic, int qos, int msgid)
{
    MQTTString                      topicString = MQTTString_initializer;
    unsigned short                   submsgid;
    int                             subcount, granted_qos;
    int                             rv;
    unsigned char                    buf[256];

    topicString.cstring = topic;

    rv = MQTTSerialize_subscribe(buf, sizeof(buf), 0, msgid, 1, &topicString, &qos);
    if( rv < 0 )
    {
        printf("MQTTSerialize_subscribe failure, rv=%d\n", rv);
        return -1;
    }
```

```
    if ( rv != transport_sendPacketBuffer(buf,rv) )
    {
        printf("transport_sendPacketBuffer for mqtt_subscribe_topic failure, rv=%d\n", rv);
        return -2;
    }

    HAL_Delay(800);

    memset(buf, 0, sizeof(buf));
    rv = MQTTPacket_read(buf, sizeof(buf), transport_getdata);
    if( SUBACK != rv)
    {
        printf("MQTTPacket_read for MQTT SUBACK failure, rv=%d\n", rv);
        return -3;
    }

    rv = MQTTDeserialize_suback(&submsgid, 1, &subcount, &granted_qos, buf, sizeof(buf));
    if( !rv || submsgid!=msgid || granted_qos==0x80)
    {
        printf("MQTTDeserialize_suback failure, rv=%d\n", rv);
        return -4;
    }

    return 0;
}

int mqtt_unsubscribe_topic(char *topic, int msgid)
{
    MQTTString                  topicString = MQTTString_initializer;
    unsigned short               submsgid;
    int                         rv;
    unsigned char                buf[256];

    topicString.cstring = topic;
    rv = MQTTSerialize_unsubscribe(buf, sizeof(buf), 0, msgid, 1, &topicString);
    if( rv < 0 )
    {
        printf("MQTTSerialize_subscribe failure, rv=%d\n", rv);
        return -1;
    }

    if ( rv != transport_sendPacketBuffer(buf,rv) )
    {
        printf("transport_sendPacketBuffer for mqtt_unsubscribe_topic failure, rv=%d\n", rv);
        return -2;
    }

    HAL_Delay(800);

    memset(buf, 0, sizeof(buf));
    rv = MQTTPacket_read(buf, sizeof(buf), transport_getdata);
    if( UNSUBACK != rv)
    {
        printf("MQTTPacket_read for MQTT UNSUBACK failure, rv=%d\n", rv);
        return -3;
    }
```

```
    rv = MQTTDeserialize_unsuback(&submsgid, buf, sizeof(buf));
    if( !rv || submsgid!=msgid )
    {
        printf("MQTTDeserialize_unsuback failure, rv=%d\n", rv);
        return -4;
    }

    return 0;
}


int mqtt_publish(char *topic, int qos, char *payload)
{
    MQTTString                    topicString = MQTTString_initializer;
    int                           rv;
    unsigned char                 buf[256];
    unsigned char                 dup = 0;
    unsigned char                 retained = 0;
    unsigned short                packetid = 0;

    topicString.cstring = topic;

    rv = MQTTSerialize_publish(buf, sizeof(buf), dup, qos, retained, packetid, topicString,
(unsigned char*)payload, strlen(payload));
    if( rv < 0 )
    {
        printf("MQTTSerialize_publish failure, rv=%d\n", rv);
        return -1;
    }

    if ( rv != transport_sendPacketBuffer(buf,rv) )
    {
        printf("transport_sendPacketBuffer for MQTTSerialize_publish failure, rv=%d\n", rv);
        return -2;
    }

    HAL_Delay(800);

    memset(buf, 0, sizeof(buf));
    rv = MQTTPacket_read(buf, sizeof(buf), transport_getdata);
    if( PUBLISH!=rv && -1!=rv )
    {
        printf("MQTTPacket_read for MQTT PUBLISH failure, rv=%d\n", rv);
        return -3;
    }

    return 0;
}


int mqtt_pingreq(void)
{
    int                           rv;
    unsigned char                 buf[256];
```

```
    rv = MQTTSerialize_pingreq(buf, sizeof(buf));
    if( rv < 0 )
    {
        printf("MQTTSerialize_pingreq failure, rv=%d\n", rv);
        return -1;
    }

    if ( rv != transport_sendPacketBuffer(buf,rv) )
    {
        printf("transport_sendPacketBuffer for MQTTSerialize_pingreq failure, rv=%d\n", rv);
        return -2;
    }

    HAL_Delay(800);

    memset(buf, 0, sizeof(buf));
    rv = MQTTPacket_read(buf, sizeof(buf), transport_getdata);
    if( PINGRESP != rv)
    {
        printf("MQTTPacket_read for MQTT PINGRESP failure, rv=%d\n", rv);
        return -3;
    }

    return 0;
}
```

至此，Paho 库的相关代码移植工作完成，接下来我们再在 main.c 中实现 MQTT 的订阅和发布。

3. MQTT 发布实现温湿度实时上报

3.1. 源码修改

此前我们已经做好了使用 ESP8266 WiFi 模块自动连接无线路由器并上报到 PC 端 socket 服务器的功能，接下来我们将在该代码的基础上做些小的修改，实现 MQTT 上报到物联网云平台。在之前代码的基础上，修改 main.c 如下：

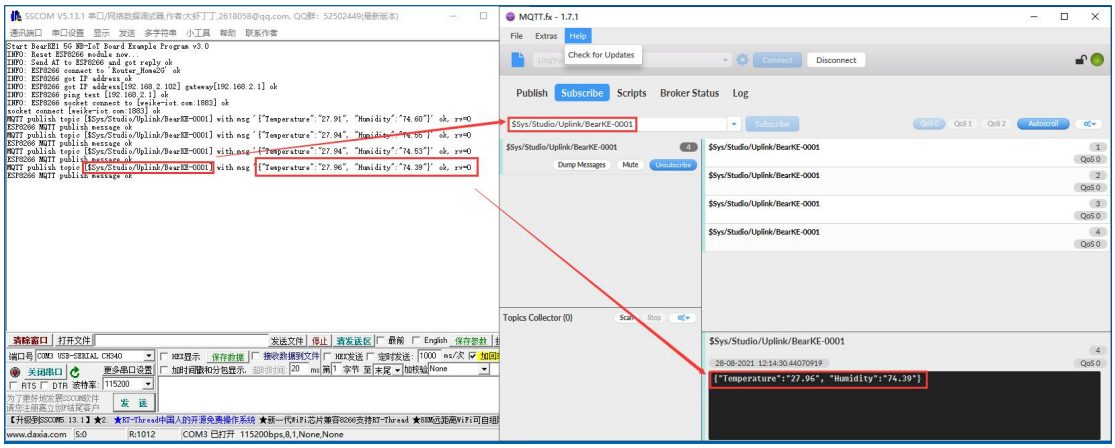
```
... ...

在文件开始处添加头文件 esp8266.h ：
/* USER CODE BEGIN Includes */
#include <string.h>
#include "dht11.h"
#include "sht30.h"
#include "core_json.h"
#include "oled.h"
#include "esp8266.h"
#include "core_mqtt.h"
/* USER CODE END Includes */
... ...

/* USER CODE BEGIN PV */
#define FLAG_WIFI_CONNECTED     (1<<0)
#define FLAG_SOCK_CONNECTED     (1<<1)

#define DEF_ROUTER_SSID         "Router_Home2G"
#define DEF_ROUTER_PWD          "password"

#define MQTT_BROKER_HOSTNAME     "weike-iot.com"  //设置连接凌云实验室 MQTT Broker 服务器
#define MQTT_BROKER_PORT         1883     // 设置 MQTT Broker 监听的端口号
#define MQTT_BROKER_USERNAME     "lingyun"
#define MQTT_BROKER_PASSWORD     "lingyun"

#define MQTT_CLIENT_ID          "BearKE-0001"  //根据自己开发板的编号设置一个 ID
#define MQTT_PUB_TOPIC          "$Sys/Studio/Uplink/"MQTT_CLIENT_ID
/* USER CODE END PV */

... ...

int main(void)
{

  ... ...

  /* USER CODE BEGIN WHILE */
  sysled_hearbeat();
  beep_start(2, 300);
  printf("Start BearKE1 5G NB-IoT Board Example Program v3.0\r\n");

  OLED_Init();
  OLED_ShowBanner(TIME_1S*2);
```

```
esp8266_module_init();

while (1)
{
    /* WiFi 没有连接上无线路由器的话，开始连接无线路由器并 ping 测试 */
    if( ! (wifi_flag&FLAG_WIFI_CONNECTED) )
    {
          if( esp8266_join_network(DEF_ROUTER_SSID, DEF_ROUTER_PWD) )
          {
               esp8266_module_init();
               HAL_Delay(2000);
               continue;
          }

          if( esp8266_get_ipaddr(ipaddr, gateway, sizeof(ipaddr) ) )
          {
               HAL_Delay(1000);
               continue;
          }

          if( esp8266_ping_test(gateway) )
          {
               HAL_Delay(1000);
               continue;
          }
          wifi_flag |= FLAG_WIFI_CONNECTED; /* set wifi connected flag */
    }

    /* 没有连上 MQTT Broker 的话就开始连接服务器 */
    if( ! (wifi_flag&FLAG_SOCK_CONNECTED) )
    {
          rv = mqtt_connect(MQTT_BROKER_HOSTNAME, MQTT_BROKER_PORT, MQTT_CLIENT_ID,
             MQTT_BROKER_USERNAME, MQTT_BROKER_PASSWORD);

          if( rv )
          {
               HAL_Delay(1000);
               continue;
          }

          wifi_flag |= FLAG_SOCK_CONNECTED; /* set socket connected flag */
    }

    /* 定时发布采样温度值到 MQTT Broker */
    if( time_after(HAL_GetTick(), last_time+3000) )
    {
          rv = report_tempRH_json();
          if( 0==rv )
          {
               printf("ESP8266 MQTT publish message ok\n");
          }
          else
          {
               printf("ESP8266 MQTT publish message failure, rv=%d\n", rv);
               wifi_flag &= ~FLAG_SOCK_CONNECTED; /* clear socket connected flag */
```

```
                if( esp8266_ping_test(gateway) )
                {
                     wifi_flag &= ~FLAG_WIFI_CONNECTED; /* clear wifi connected flag */
                }
           }

           last_time = HAL_GetTick(); /*update last report time */
     }
     /* USER CODE END WHILE */
  }
 ... ...
}

... ...

/* USER CODE BEGIN 4 */
int report_tempRH_json(void)
{
    char         buf[128];
    float        temperature, humidity=0.0;
    uint32_t     temp, humd;
    int          rv;

    if ( SHT30_SampleData(&temperature, &humidity) < 0 )
    {
         printf("ERROR: SHT30 Sample data failure\n");
         return -1;
    }

    memset(buf, 0, sizeof(buf));
    snprintf(buf, sizeof(buf), "{\"Temperature\":\"%.2f\", \"Humidity\":\"%.2f\"}", temperature,
humidity);

    temp = (int)(temperature*100);
    humd = (int)(humidity*100);
    OLED_ShowTempHumdity(temp, humd, TIME_1S*2);

    rv = mqtt_publish(MQTT_PUB_TOPIC, Qos0, buf);
    printf("MQTT publish topic [%s] with msg '%s' %s, rv=%d\r\n", MQTT_PUB_TOPIC, buf, rv ? "failure":
"ok", rv);

    return rv;
}

... ...
```

3.2. 运行测试

编译并重新烧录运行单片机程序，打开 MQTT.fx 连接凌云实验室 MQTT Broker，然后订阅 STM32 单片机的上报主题 “$Sys/Studio/Uplink/BearKE-0001”,这时候就可以每隔 3 秒收到单片机上报的数据。



如果我们想接收所有的小熊座 NB-IoT 开发板的数据，这时候可以订阅主题“$Sys/Studio/Uplink/#”，这里的 # 表示通配符，它将监听“$Sys/Studio/Uplink/”下的所有主题。

使用凌云实验室的智能家居 App 程序订阅该主题，就可以实时获取 STM32 单片机上报的温湿度值。

4. MQTT 订阅实现 Led 灯远程控制

4.1. 源码修改

前面我们实现了 MQTT 实时发布并上报温湿度值到物联网云平台的功能，接下来我们再实现订阅 Led 灯控制命令的主题，并实现远程 Led 灯的控制。在之前代码的基础上，修改 main.c 如下：

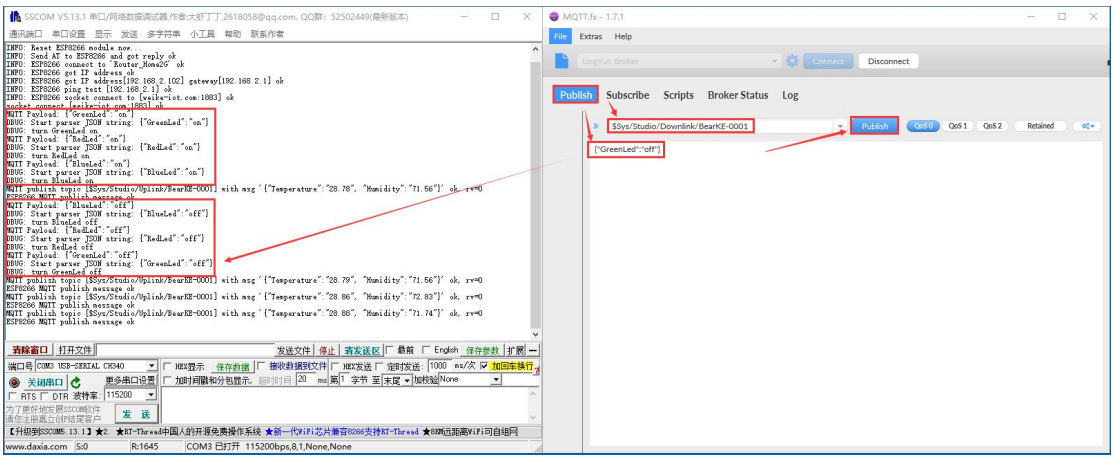
```
... ...

在这里添加 MQTT 订阅的主题相关宏
/* USER CODE BEGIN PV */
#define FLAG_WIFI_CONNECTED     (1<<0)
#define FLAG_SOCK_CONNECTED     (1<<1)

#define DEF_ROUTER_SSID         "Router_Home2G"
#define DEF_ROUTER_PWD          "password"

#define MQTT_BROKER_HOSTNAME     "weike-iot.com"  //设置连接凌云实验室 MQTT Broker 服务器
#define MQTT_BROKER_PORT         1883     // 设置 MQTT Broker 监听的端口号
#define MQTT_BROKER_USERNAME     "lingyun"
#define MQTT_BROKER_PASSWORD     "lingyun"

#define MQTT_CLIENT_ID          "BearKE-0001"  //根据自己开发板的编号设置一个 ID
#define MQTT_PUB_TOPIC          "$Sys/Studio/Uplink/"MQTT_CLIENT_ID
#define MQTT_SUB_TOPIC          "$Sys/Studio/Downlink/"MQTT_CLIENT_ID
/* USER CODE END PV */

... ...

在这里添加 MQTT 订阅消息处理的函数声明，在后面我们将会定义该函数
/* USER CODE BEGIN PFP */
int report_tempRH_json(void);
int parser_led_json(char *json_string, int bytes);
void mqtt_subscribe_proc(void);
/* USER CODE END PFP */

int main(void)
{

  while (1)
  {
      ... ...

      /* 没有连上 MQTT Broker 的话就开始连接服务器 */
      if( ! (wifi_flag&FLAG_SOCK_CONNECTED) )
      {
            rv = mqtt_connect(MQTT_BROKER_HOSTNAME, MQTT_BROKER_PORT, MQTT_CLIENT_ID,
               MQTT_BROKER_USERNAME, MQTT_BROKER_PASSWORD);
            if( rv )
            {
                HAL_Delay(1000);
                continue;
            }
```

```
            /* 连上 MQTT Broker 后订阅下发三色 Led 控制命令的主题 */
            mqtt_subscribe_topic(MQTT_SUB_TOPIC, Qos0, 1);

            wifi_flag |= FLAG_SOCK_CONNECTED; /* set socket connected flag */
        }

        /* 处理 MQTT 订阅收到的消息 */
        mqtt_subscribe_proc();

        /* 定时发布采样温度值到 MQTT Broker，这里修改为 30s 上报一次 */
        if( time_after(HAL_GetTick(), last_time+30000) )
        {
            ... ...
        }
        /* USER CODE END WHILE */
    }
   ... ...
}

... ...

/* USER CODE BEGIN 4 */
void mqtt_subscribe_proc(void)
{
    unsigned char                   buf[256];;
    int                             packet_type;
    MQTTString                      topicName;
    unsigned char                   dup;
    int                             qos;
    unsigned char                   retained;
    unsigned short                  msgid;
    unsigned char                  *payload = NULL;
    int                             payloadlen = 0;
    int                             rv;

    memset(buf, 0, sizeof(buf));

    packet_type = MQTTPacket_read(buf, sizeof(buf), transport_getdata);
    if( packet_type < 0 )
    {
        return ;
    }

    switch( packet_type )
    {
        case 0:
         break;

         case PUBLISH:
         {
              rv = MQTTDeserialize_publish(&dup, &qos, &retained, &msgid, &topicName, &payload,
&payloadlen, buf, sizeof(buf));
              if ( rv == 1)
              {
                   printf("MQTT Payload: %s\n", payload);
                   parser_led_json((char *)payload, payloadlen);
              }
```

```
                else
                {
                    printf("ERROR: MQTTDeserialize_publish() failure, rv=%d\r\n", rv);
                }
                break;
            }

            case CONNACK:
            case PUBACK:
            case SUBACK:
            case UNSUBACK:
            case PUBREC:
            case PUBREL:
            case PUBCOMP:
                break;
    }
}
... ...
```

4.2. 运行测试

编译并重新烧录运行单片机程序，打开 MQTT.fx 连接凌云实验室 MQTT Broker，然后设置发布的主题为单片机的订阅主题 “$Sys/Studio/Downlink/BearKE-0001”,这时候发布 JSON 格式的 Led 控制命令，就可以控制 Led 灯的亮灭。



使用凌云实验室的智能家居 App 程序连接凌云实验室的 MQTT Borker 之后，就可以通过手机 App 控制三色 Led 灯的亮灭了。