优秀不够，你是否无可替代

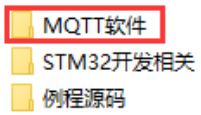
导航

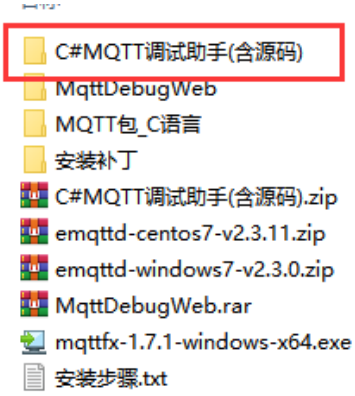
博客园
首页
新随笔
联系
订阅 XML
管理

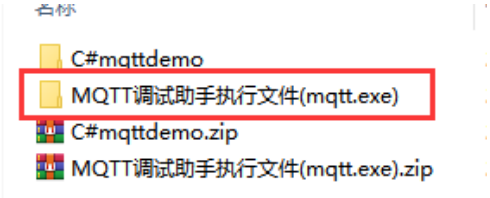
MQTT协议

先来体验一下MQTT通信

1.打开调试助手





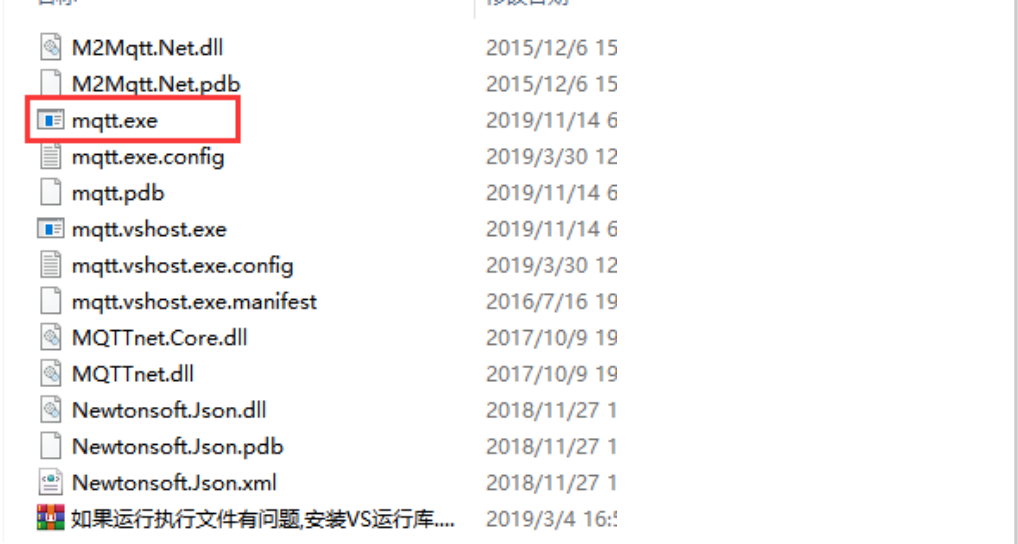


最新评论

1. Re:C#开发: 通信篇-TCP客户端
感谢分享，直接就用上了
--Zfen
2. Re:03-STM32+Air724UG远程升级篇OTA(阿里云物联网平台)-STM32+Air724UG使用阿里云物联网平台OTA远程更新STM32程序
楼主，单片机和Air724模块之间是通过AT指令通讯的吗？
--a314825348

阅读排行榜

1. ESP8266使用详解(AT,LUA,SDK)(172431)
2. 1-安装MQTT服务器(Windows),并连接测试(97685)



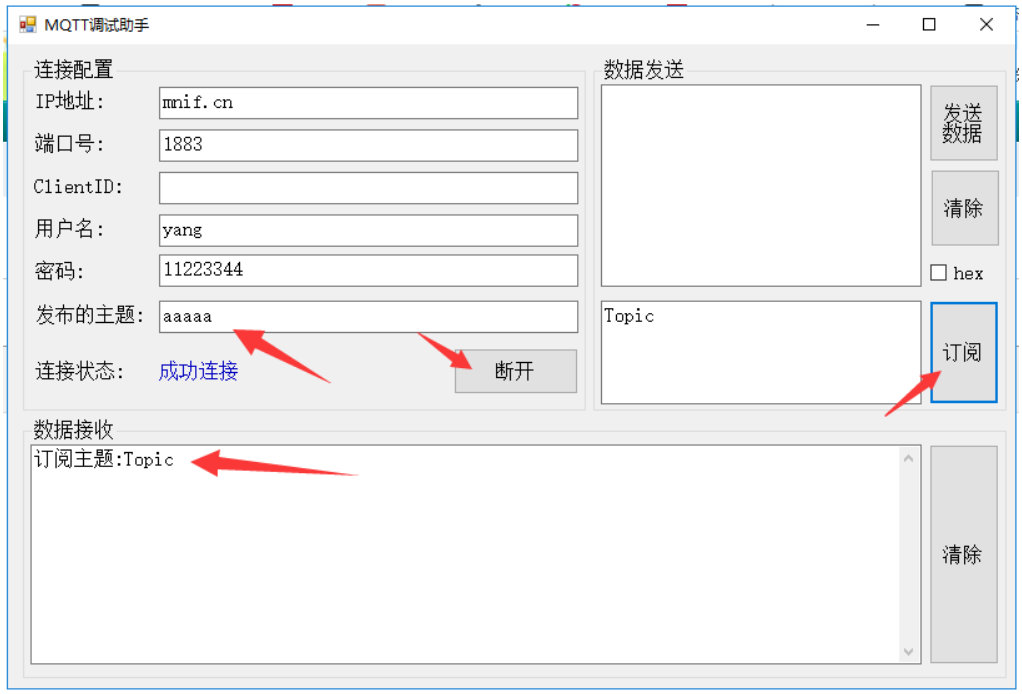
2.需要打开两个,默认连接提供的服务器测试.

第一个配置如下:

发布的主题:aaaaa

订阅的主题:Topic

点击连接,然后点击订阅



第二个配置如下:

发布的主题:Topic

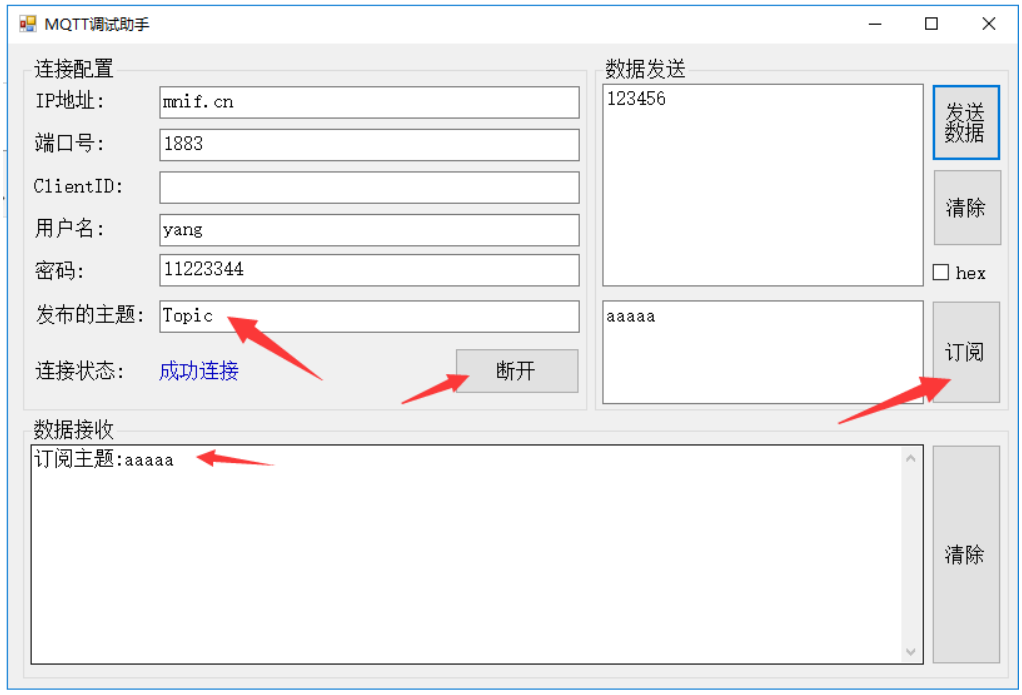
订阅的主题:aaaaa

点击连接,然后点击订阅

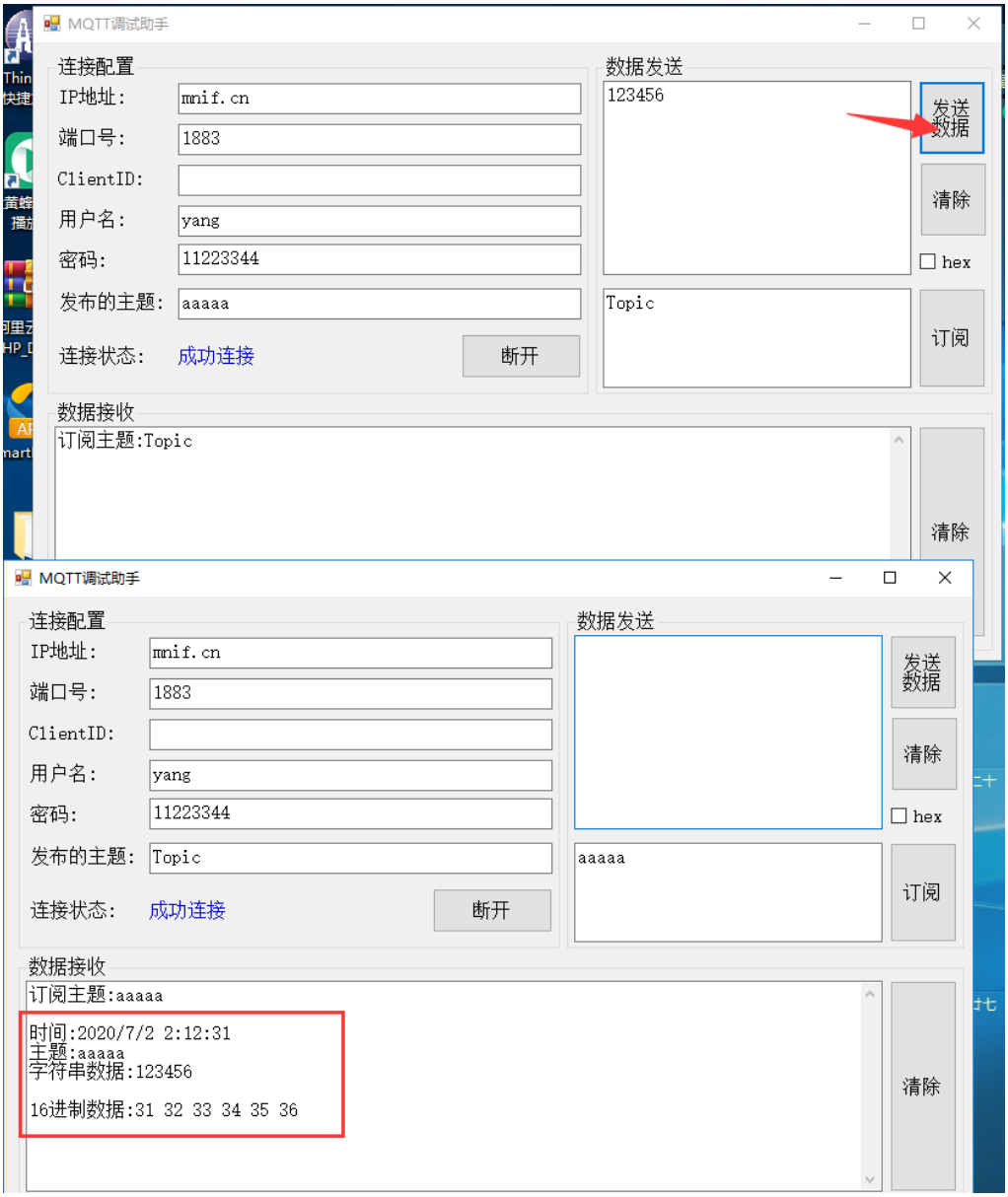
3. ESP8266刷AT固件与nodemcu固件(64173)
4. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(63374)
5. 有人WIFI模块使用详解(38307)
6. (一)基于阿里云的MQTT远程控制(Android 连接MQTT服务器,ESP8266连接MQTT服务器实现远程通信控制----简单的连接通信)(35675)
7. 关于TCP和MQTT之间的转换(32739)
8. C#中public与private与static(31683)
9. android 之TCP客户端编程(31612)
10. android客服端+eps8266+单片机+路由器之远程控制系统(31237)

推荐排行榜

1. C#委托+回调详解(9)
2. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(8)
3. 用ESP8266+android,制作自己的WIFI小车(Android 软件)(6)
4. ESP8266使用详解(AT,LUA,SDK)(6)
5. 关于TCP和MQTT之间的转换(5)



4.第一个软件发消息:发送的消息123456,然后点击发送



用户会看到第二个软件收到消息
提示:这个软件是自己开发的,里面的显示都是自己规定的.

其实过程是这样的:
两个客户端都连接了一个MQTT服务器(这个只是个软件,后面章节会告诉大家怎么安装)
第一个客户端发布的主题那一栏填写的是 aaaaa 然后发送的消息是 123456
点击发送的时候实际上该消息就发给了MQTT服务器
整个消息格式呢大概是这样的 XXXXaaaaaXXXX123456
XXXX呢代表其它信息,方便服务器区分出来整个消息中的
发布的主题(aaaaa)和发布的消息(123456)
其实 aaaaa 就充当了这个消息的标识

第二个客户端的订阅那一项填写的是 aaaaa
其实就是在告诉服务器,我需要数据标识是 aaaaa的消息
既然你告诉了服务器了,那么服务器只要接收到数据标识是 aaaaa 的消息,那么就会主动把消息发给你

5.同理,让下面的客户端把消息发给上面的客户端

6.简要说明:

连接上MQTT服务器以后,只要是两个设备之间订阅和发布的主题对上了那么这两个设备就可以通信了

对于初学者可能疑惑,你软件点点点到底内部是怎么做到的

如果你想知道更多就接着看,我先说明一下过程.

其实MQTT就是一个TCP服务器,它是在TCP通信的时候封装了一套协议.

咱们就叫它MQTT协议,注意本质上就是TCP传输数据,这个数据有格式而已!

首先是使用TCP连接,然后发送MQTT连接协议,然后发送MQTT订阅主题的协议.

这样的话,服务器就知道你需要哪种标识的数据了.

当服务器收到这种标识的数据的时候,服务器就会主动转发给你.

其实MQTT服务器主要工作就是做数据转发,但是你需要告诉它你需要什么样的数据.

思考

1.其实理解一个东西最好的方式就是:你要设想如果让你自己做一个这样的服务器,你会怎么做.

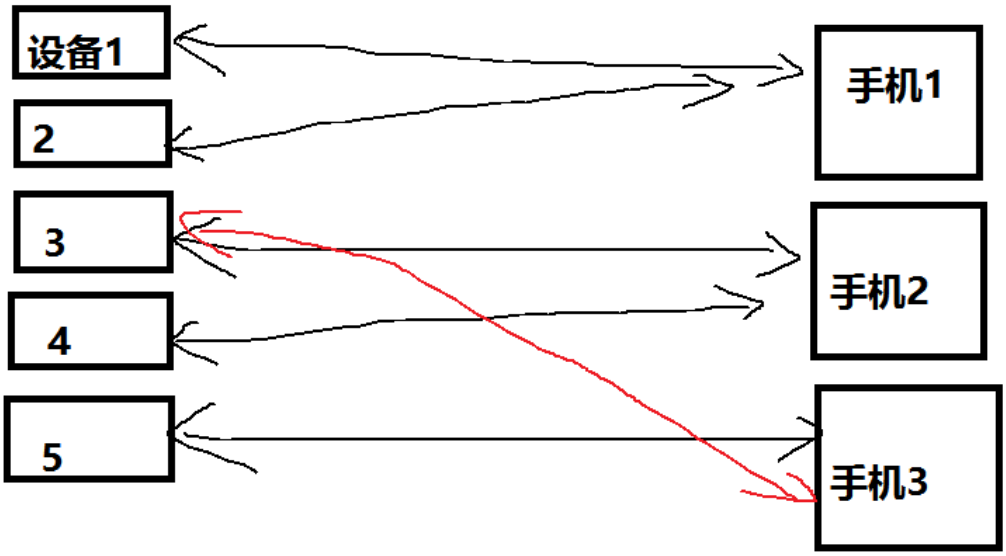
2.现在需求是做一个负责数据转发的软件

首先,平时的时候咱做的TCP服务器都是,一个或者多个客户端连接咱做的TCP服务器,然后TCP服务器处理客户端的数据.

现在呢!需求变了!

假设我有5个网络设备,3个手机.我现在想让网络设备把数据远程传给手机.而且我还需要记录网络设备上传的数据.

假设通信是这样的(而且后期还会不停的增加设备和手机)



3.咋办???

1. 需要记录所有设备的数据

2. 设备和手机之间存在多对一和一对多,所以,必须需要个公共的服务器进行数据的中转.

假设你就把这个服务器做成TCP服务器,有人问,你咋不做成UDP呢?

UDP是无连接状态,发送数据不好判断是不是发送成功,我还是少找些麻烦!

还有就是要实现远程,有个公网IP就可以,可以自己买个服务器,上网络公司拉一根专网

或者用自己电脑,用花生壳映射

还是用云服务器吧!就是运行在别人的服务器上的一台电脑(就是一台电脑),IP地址直接是公网.方便.

4.怎么设计这个TCP服务器???
1.为了应对这种通信,首先设备发送的数据决不能是单单的数据,必须加点东西
2.如果把发送的数据带上标识呢? 假设设备1发送的数据是: aaaaa数据 (aaaaa是数据标识,后面是真实数据)
3.然后呢!假设手机1就接收数据标识是aaaaa的数据,怎么让服务器转发给它呢???
4.如果手机1在连接上TCP服务器的时候 告诉TCP服务器我接收数据标识是 aaaaa的数据
5.通过上面的方式是不是有点眉头了????
咱呢姑且把 "告诉TCP服务器我接收数据标识是 aaaaa的数据" 这个事情呢,起个名字 订阅的主题是 aaaaa
把 "假设设备1发送的数据是: aaaaa数据 " 消息前面的 aaaaa 叫做 发布的主题是aaaaa

5.总结上面的就是
手机1先连接TCP服务器,然后呢,规定个协议,告诉TCP服务器我订阅的主题是aaaaa
这样呢服务器就记住了,当出现消息前面的主题是aaaaa的消息的时候,他就把这个消息发给手机1

当然咱假设,设备1连接上TCP服务器,然后,告诉TCP服务器我订阅的主题是wwww
这样呢服务器就记住了,当出现消息前面的主题是wwww的消息的时候,他就把这个消息发给设备1

然后设备1连接上TCP服务器以后呢,这样发送信息(假设发送的消息是123456): aaaaa123456
服务器一接收到客户端的消息,就取出来这个消息的标识是什么,取出来的是 aaaaa
然后呢,看下记录的谁需要消息标识是aaaaa的消息,然后找到了手机1
最后把这个消息发送给手机1这个客户端,然后手机1就接收到了1123456这个消息

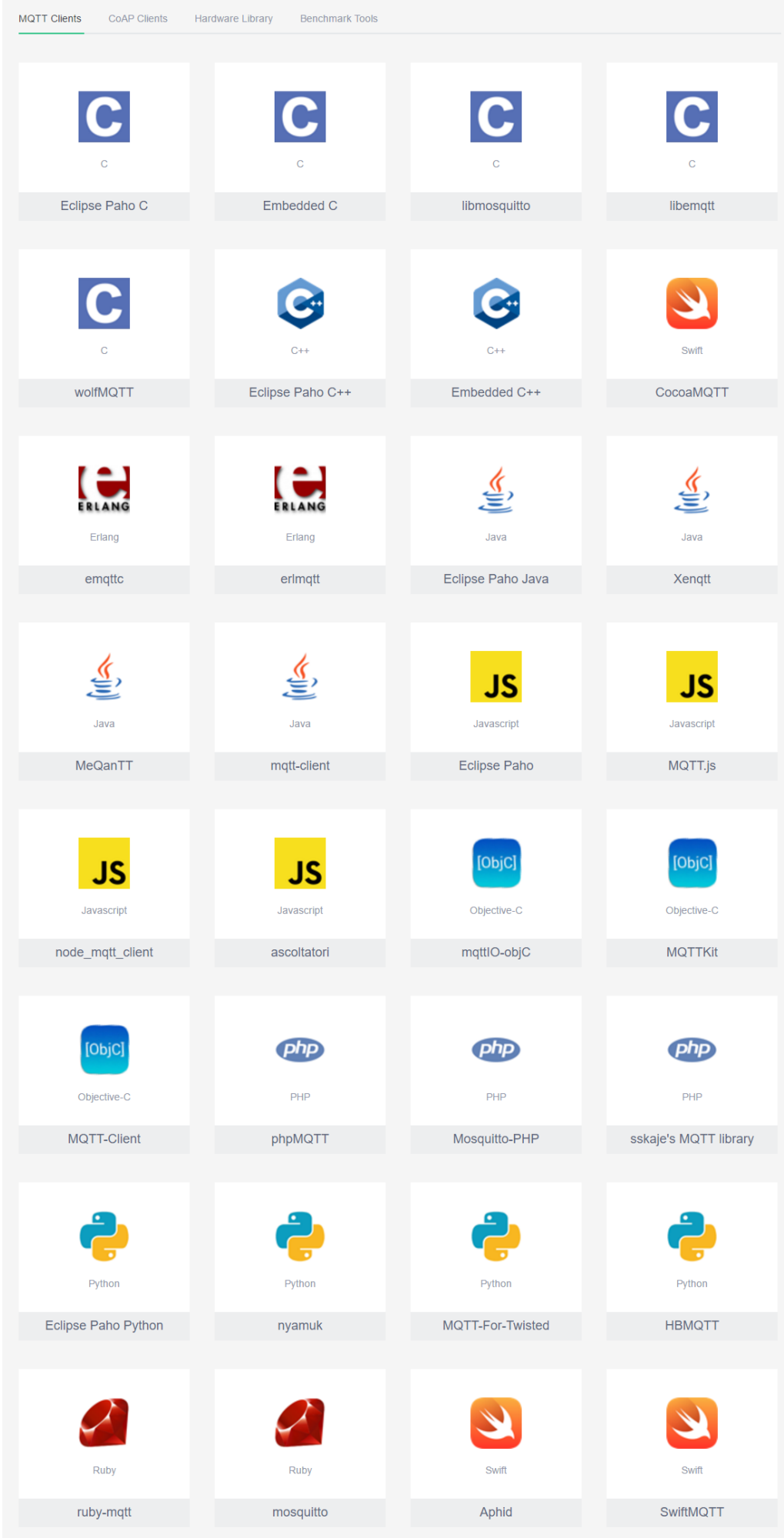
同理:手机1发送 wwww998877 然后这个消息就会发给设备1 ,设备1就会收到 998877

6.总结

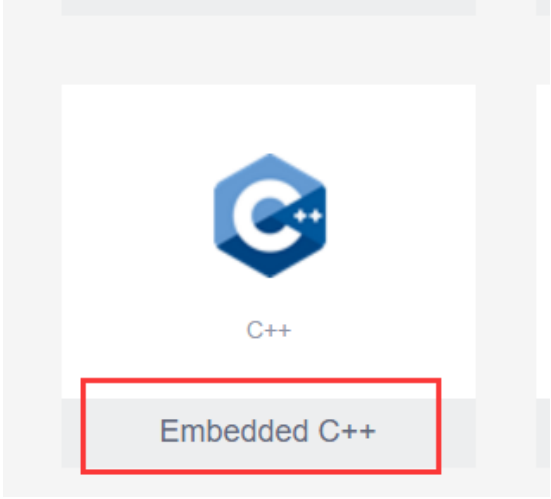
这个服务器道理上是这样,服务器记录各个设备的信息,各个设备订阅的主题,然后呢,判断这个消息然后进行转发
但是...咱做个简单的完全可以做出来,但是要想做的完善,而且要支持庞大消息数量的设备(来个百万级).....不是一朝一夕就可以的.
其实很长时间以前,人们就有这种需求了.多对一和一对多通信
所以呢,一些组织和单位就开始解决这种问题,开始做这种软件,所以MQTT就诞生了.
之所以叫MQTT是因为是外国人做的这种TCP服务器,外国人呢,为实现这种功能的TCP服务器取了个名字叫
Message Queuing Telemetry Transport
然后取每个首字母 就叫 MQTT了
其实有很多家做MQTT软件,但是呢,我比较喜欢用emqtt

来说一下具体的MQTT协议

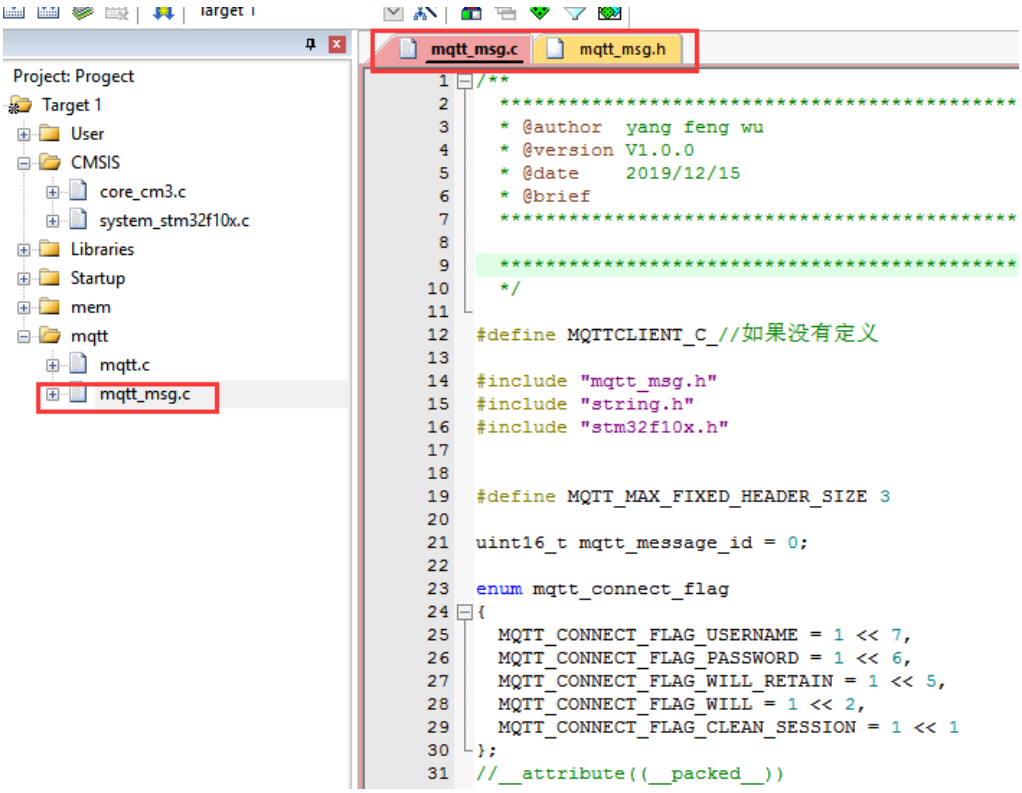
1,首先咱知道就是个TCP服务器,所以呢,需要先用TCP连接上他们的服务器.
2,咱用Android ,C#,QT,网页等等连接MQTT服务器的时候有现成的封装好的库可以用,其实说白了就是调用函数而已.....
3,但是对于单片机而言要想实现MQTT通信,那么就需要借助网络模块
大部分的网络模块都可以实现TCP通信,咱呢,就需要在TCP的基础上按照MQTT协议封装下咱的数据
注:其实官方给了现成的MQTT的封装数据和解析数据的程序)
https://docs.emqx.io/sdk_tools?category=MQTT_Clients (官方提供的各个开发的库)



单片机用下面这个,不过我以前用的这个,库功能很全,但是因为占用内存有点大,所以后期使用的是自己重新封装的.



下面是自己当前使用的mqtt最底层




```
/**
  ******************************************************************************
  * @author  yang feng wu
  * @version V1.0.0
  * @date    2019/12/15
  * @brief
  ******************************************************************************

  ******************************************************************************
  */

#define MQTTCLIENT_C_//如果没有定义
```

```
#include "mqtt_msg.h"
#include "string.h"
#include "stm32f10x.h"


#define MQTT_MAX_FIXED_HEADER_SIZE 3

uint16_t mqtt_message_id = 0;

enum mqtt_connect_flag
{
  MQTT_CONNECT_FLAG_USERNAME = 1 << 7,
  MQTT_CONNECT_FLAG_PASSWORD = 1 << 6,
  MQTT_CONNECT_FLAG_WILL_RETAIN = 1 << 5,
  MQTT_CONNECT_FLAG_WILL = 1 << 2,
  MQTT_CONNECT_FLAG_CLEAN_SESSION = 1 << 1
};
//__attribute((__packed__))
struct  mqtt_connect_variable_header
{
  uint8_t lengthMsb;
  uint8_t lengthLsb;
  uint8_t magic[4];
  uint8_t version;
  uint8_t flags;
  uint8_t keepaliveMsb;
  uint8_t keepaliveLsb;
};


int mqtt_get_type(unsigned char* buffer) { return (buffer[0] & 0xf0) >> 4; }
int mqtt_get_connect_ret_code(unsigned char* buffer) { return (buffer[3]); }
int mqtt_get_qos(unsigned char* buffer) { return (buffer[0] & 0x06) >> 1; }


int append_string(int *length,unsigned  char* buffer,int buffer_length,unsign
{
  if((*length) + len + 2 > buffer_length)//加上 ClientID 和 记录 ClientID个数(两
    return -1;

  buffer[(*length)++] = len >> 8;
  buffer[(*length)++] = len & 0xff;
  c_memcpy(buffer + (*length), string, len);
  (*length) += len;
  return len + 2;
}



uint16_t append_message_id(int *length,unsigned  char* buffer,int buffer_leng
{
  // If message_id is zero then we should assign one, otherwise
  // we'll use the one supplied by the caller
  while(message_id == 0)
    message_id = ++mqtt_message_id;

  if((*length) + 2 > buffer_length)
    return 0;

  buffer[(*length)++] = message_id >> 8;
  buffer[(*length)++] = message_id & 0xff;
```

```
  return message_id;
}


int fini_message(unsigned char **data_ptr,int    length,unsigned char* buffer
{
  int remaining_length = length - MQTT_MAX_FIXED_HEADER_SIZE;

  if(remaining_length > 127)
  {
    buffer[0] = ((type & 0x0f) << 4) | ((dup & 1) << 3) | ((qos & 3) << 1) |
    buffer[1] = 0x80 | (remaining_length % 128);
    buffer[2] = remaining_length / 128;
    length = remaining_length + 3;
    *data_ptr = buffer;
  }
  else
  {
    buffer[1] = ((type & 0x0f) << 4) | ((dup & 1) << 3) | ((qos & 3) << 1) |
    buffer[2] = remaining_length;
    length = remaining_length + 2;
    *data_ptr = buffer + 1;
  }

  return length;
}



uint16_t mqtt_get_id(unsigned char* buffer, uint16_t length)
{
  if(length < 1)
    return 0;

  switch(mqtt_get_type(buffer))
  {
    case MQTT_MSG_TYPE_PUBLISH:
    {
      int i;
      int topiclen;

      for(i = 1; i < length; ++i)
      {
        if((buffer[i] & 0x80) == 0)
        {
          ++i;
          break;
        }
      }

      if(i + 2 >= length)
        return 0;
      topiclen = buffer[i++] << 8;
      topiclen |= buffer[i++];

      if(i + topiclen >= length)
        return 0;
      i += topiclen;

      if(mqtt_get_qos(buffer) > 0)
      {
        if(i + 2 >= length)
```

```
          return 0;
        //i += 2;
      } else {
          return 0;
      }
      return (buffer[i] << 8) | buffer[i + 1];
    }
    case MQTT_MSG_TYPE_PUBACK:
    case MQTT_MSG_TYPE_PUBREC:
    case MQTT_MSG_TYPE_PUBREL:
    case MQTT_MSG_TYPE_PUBCOMP:
    case MQTT_MSG_TYPE_SUBACK:
    case MQTT_MSG_TYPE_UNSUBACK:
    case MQTT_MSG_TYPE_SUBSCRIBE:
    {
      // This requires the remaining length to be encoded in 1 byte,
      // which it should be.
      if(length >= 4 && (buffer[1] & 0x80) == 0)
        return (buffer[2] << 8) | buffer[3];
      else
        return 0;
    }

    default:
      return 0;
  }
}



/**
* @brief   获取MQTT返回的数据长度(去掉1和2字节后面数据的长度)
* @param   buffer   MQTT返回的数据首地址
* @param   length   返回的数据个数
* @retval  数据长度
* @warning None
* @example
**/
int mqtt_get_total_length(unsigned char* buffer, uint16_t length)
{
  int i;
  int totlen = 0;

  for(i = 1; i < length; ++i)
  {
    totlen += (buffer[i] & 0x7f) << (7 * (i - 1));
    if((buffer[i] & 0x80) == 0)
    {
      ++i;
      break;
    }
  }
  totlen += i;

  return totlen;
}




/**
* @brief   打包连接MQTT指令
```

```
* @param   info     MQTT信息
* @param   data_ptr 打包的数据首地址
* @param   buffer   打包进的数组
* @param   buffer_length 数组长度
* @retval  数据长度
* @warning None
* @example
**/
int mqtt_msg_connect(mqtt_connect_info_t* info,unsigned char **data_ptr,unsig
{
    int length;
  struct mqtt_connect_variable_header* variable_header;

    mqtt_message_id = 0;

    length = MQTT_MAX_FIXED_HEADER_SIZE;//头.连接类型1位,数据个数2位(如果大于127就

  if(length + sizeof(*variable_header) > buffer_length)//数组不够存储的
    return 0;

  variable_header = (void*)(buffer + length);//把数组分给这个结构体里面的变量
  length += sizeof(*variable_header);//存储完 连接类型,整个数据个数,版本号个数,版本号

  variable_header->lengthMsb = 0;//版本名称个数高位
  variable_header->lengthLsb = 4;//版本名称个数低位
  c_memcpy(variable_header->magic, "MQTT", 4);//版本名称MQTT
  variable_header->version = 4;//版本号
  variable_header->flags = 0;//先清零
  variable_header->keepaliveMsb = info->keepalive >> 8;//心跳包时间
  variable_header->keepaliveLsb = info->keepalive & 0xff;//心跳包时间

  if(info->clean_session)//清除连接信息
    variable_header->flags |= MQTT_CONNECT_FLAG_CLEAN_SESSION;

  if(info->client_id != NULL && info->client_id[0] != '\0')//client_id
  {
    if(append_string(&length,buffer,buffer_length, info->client_id, c_strlen(
      return -1;//数组不够用呀...
  }
  else
    return -2;//没有设置client_id

  if(info->will_topic != NULL && info->will_topic[0] != '\0')//遗嘱
  {
    if(append_string(&length,buffer,buffer_length , info->will_topic, c_strle
      return -3;

    if(append_string(&length,buffer,buffer_length , info->will_message, c_str
      return -4;

    variable_header->flags |= MQTT_CONNECT_FLAG_WILL;//需要遗嘱
    if(info->will_retain)//遗嘱是够需要服务器保留
      variable_header->flags |= MQTT_CONNECT_FLAG_WILL_RETAIN;//保留遗嘱
    variable_header->flags |= (info->will_qos & 3) << 3;//遗嘱消息等级
  }

  if(info->username != NULL && info->username[0] != '\0')//username
  {
    if(append_string(&length,buffer,buffer_length, info->username, c_strlen(i
      return -5;

    variable_header->flags |= MQTT_CONNECT_FLAG_USERNAME;//有用户名
```

```
  }

  if(info->password != NULL && info->password[0] != '\0')//password
  {
    if(append_string(&length,buffer,buffer_length, info->password, c_strlen(i
      return -6;

    variable_header->flags |= MQTT_CONNECT_FLAG_PASSWORD;//有密码
  }

  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_CONNECT, 0, 0, 0
}


/**
* @brief  判断是否连接上MQTT
* @param  服务器返回的数据
* @param
* @retval 0 连接成功
* @example
**/
int  mqtt_msg_connect_ack(unsigned char *buff)
{
    if(mqtt_get_type(buff) == MQTT_MSG_TYPE_CONNACK)
    {
        return mqtt_get_connect_ret_code(buff);
    }
    return -1;
}


/**
* @brief   断开连接
* @param   data_ptr 打包的数据首地址
* @param   buffer   打包进的数组
* @param   buffer_length 数组长度
* @retval  数据长度
* @warning None
* @example
**/
int mqtt_msg_disconnect(unsigned char **data_ptr,unsigned char* buffer,int bu
{
    int length;
    length = MQTT_MAX_FIXED_HEADER_SIZE;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_DISCONNECT, 0, 0
}



/**
* @brief   订阅主题
* @param   topic   订阅的主题
* @param   qos     消息等级
* @param   data_ptr 打包的数据首地址
* @param   buffer   打包进的数组
* @param   buffer_length 数组长度
* @retval  数据长度
* @warning None
* @example
**/
int mqtt_msg_subscribe_topic(unsigned char* topic, int qos,unsigned char **da
{
```

```
    int length;
    length = MQTT_MAX_FIXED_HEADER_SIZE;

    if(topic == NULL || topic[0] == '\0')
        return -1;

    if((mqtt_message_id = append_message_id(&length, buffer, buffer_length, 0
        return -2;

    if(append_string(&length, buffer, buffer_length, topic, c_strlen(topic))
        return -3;

    if(length + 1 > buffer_length)
    return -4;
  buffer[length++] = qos;

    return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_SUBSCRIBE, 0,
}



/**
* @brief  判断是否成功订阅
* @param  buffer  服务器返回的数据
* @param  length  服务器返回的数据长度
* @retval 0:成功  1:失败
* @example
**/
int mqtt_msg_subscribe_ack(unsigned char* buffer, uint16_t length)
{
    if(mqtt_get_type(buffer) == MQTT_MSG_TYPE_SUBACK)
    {
        if(mqtt_get_id(buffer,length) == mqtt_message_id)
        {
            return 0;
        }
        else
        {
            return 1;
        }
    }
    else
    {
        return 1;
    }
}


/**
* @brief   发布消息
* @param   topic    主题
* @param   data     消息
* @param   data_length 消息长度
* @param   qos      消息等级
* @param   retain   是否需要保留消息
* @param   data_ptr 打包的数据首地址
* @param   buffer   打包进的数组
* @param   buffer_length 数组长度
* @retval  数据长度
* @warning None
* @example
**/
```

```
int mqtt_msg_publish(unsigned char* topic,unsigned  char* date, int data_leng
{
    int length;
    length = MQTT_MAX_FIXED_HEADER_SIZE;

  if(topic == NULL || topic[0] == '\0')
    return -1;

  if(append_string(&length, buffer, buffer_length, topic, strlen(topic)) < 0)
    return -2;

  if(qos > 0)
  {
    if((mqtt_message_id = append_message_id(&length, buffer, buffer_length,
      return -3;
  }
  else
    mqtt_message_id = 0;

  if(length + data_length > buffer_length)
    return -4;
  memcpy(buffer + length, date, data_length);
  length += data_length;

  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PUBLISH, 0, qos,
}



int mqtt_msg_puback(uint16_t message_id,unsigned char **data_ptr,unsigned cha
{
    int length;
  length = MQTT_MAX_FIXED_HEADER_SIZE;
  if(append_message_id(&length, buffer, buffer_length,message_id) == 0)
    return -1;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PUBACK, 0, 0, 0)
}


int mqtt_msg_pubrec(uint16_t message_id,unsigned char **data_ptr,unsigned cha
{
    int length;
  length = MQTT_MAX_FIXED_HEADER_SIZE;
  if(append_message_id(&length, buffer, buffer_length,message_id) == 0)
    return -1;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PUBREC, 0, 0, 0)
}


int mqtt_msg_pubrel(uint16_t message_id,unsigned char **data_ptr,unsigned cha
{
    int length;
  length = MQTT_MAX_FIXED_HEADER_SIZE;

  if(append_message_id(&length, buffer, buffer_length,message_id) == 0)
    return -1;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PUBREL, 0, 1, 0)
}


int mqtt_msg_pubcomp(uint16_t message_id,unsigned char **data_ptr,unsigned ch
{
```

```
  int length;
  length = MQTT_MAX_FIXED_HEADER_SIZE;
  if(append_message_id(&length, buffer, buffer_length,message_id) == 0)
    return -1;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PUBCOMP, 0, 0, 0
}


const char* mqtt_get_publish_topic(unsigned char* buffer, uint16_t* length)
{
  int i;
  int totlen = 0;
  int topiclen;

  for(i = 1; i < *length; ++i)
  {
    totlen += (buffer[i] & 0x7f) << (7 * (i -1));
    if((buffer[i] & 0x80) == 0)
    {
      ++i;
      break;
    }
  }
  totlen += i;

  if(i + 2 >= *length)
    return NULL;
  topiclen = buffer[i++] << 8;
  topiclen |= buffer[i++];

  if(i + topiclen > *length)
    return NULL;

  *length = topiclen;
  return (const char*)(buffer + i);
}


const char* mqtt_get_publish_data(unsigned char* buffer, uint16_t* length)
{
  int i;
  int totlen = 0;
  int topiclen;
  int blength = *length;
  *length = 0;

  for(i = 1; i < blength; ++i)
  {
    totlen += (buffer[i] & 0x7f) << (7 * (i - 1));
    if((buffer[i] & 0x80) == 0)
    {
      ++i;
      break;
    }
  }
  totlen += i;

  if(i + 2 >= blength)
    return NULL;
  topiclen = buffer[i++] << 8;
  topiclen |= buffer[i++];
```

```
  if(i + topiclen >= blength)
    return NULL;

  i += topiclen;

  if(mqtt_get_qos(buffer) > 0)
  {
    if(i + 2 >= blength)
      return NULL;
    i += 2;
  }

  if(totlen < i)
    return NULL;

  if(totlen <= blength)
    *length = totlen - i;
  else
    *length = blength - i;
  return (const char*)(buffer + i);
}

/**
* @brief   打包服务器返回的心跳包数据(用不到)
* @param   data_ptr 打包的数据首地址
* @param   buffer   打包进的数组
* @param   buffer_length 数组长度
* @retval  数据长度
* @warning None
* @example
**/
int mqtt_msg_pingresp(unsigned char **data_ptr,unsigned char* buffer,int buff
{
    int length;
    length = MQTT_MAX_FIXED_HEADER_SIZE;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PINGRESP, 0, 0,
}

/**
* @brief   获取发送给服务器的心跳包数据
* @param   data_ptr 打包的数据首地址
* @param   buffer   打包进的数组
* @param   buffer_length 数组长度
* @retval  数据长度
* @warning None
* @example
**/
int mqtt_msg_pingreq(unsigned char **data_ptr,unsigned char* buffer,int buffe
{
    int length;
    length = MQTT_MAX_FIXED_HEADER_SIZE;
  return fini_message(data_ptr,length, buffer, MQTT_MSG_TYPE_PINGREQ, 0, 0, 0
}
```

```
#ifndef MQTTCLIENT_H_
#define MQTTCLIENT_H_


#ifndef MQTTCLIENT_C_//如果没有定义
#define MQTTCLIENT_Cx_ extern
#else
#define MQTTCLIENT_Cx_
#endif

#include "string.h"
#include "stm32f10x.h"

#define c_memcpy memcpy
#define c_memset memset
#define c_strlen strlen


enum mqtt_message_type
{
  MQTT_MSG_TYPE_CONNECT = 1,
  MQTT_MSG_TYPE_CONNACK = 2,
  MQTT_MSG_TYPE_PUBLISH = 3,
  MQTT_MSG_TYPE_PUBACK = 4,
  MQTT_MSG_TYPE_PUBREC = 5,
  MQTT_MSG_TYPE_PUBREL = 6,
  MQTT_MSG_TYPE_PUBCOMP = 7,
  MQTT_MSG_TYPE_SUBSCRIBE = 8,
  MQTT_MSG_TYPE_SUBACK = 9,
  MQTT_MSG_TYPE_UNSUBSCRIBE = 10,
  MQTT_MSG_TYPE_UNSUBACK = 11,
  MQTT_MSG_TYPE_PINGREQ = 12,
  MQTT_MSG_TYPE_PINGRESP = 13,
  MQTT_MSG_TYPE_DISCONNECT = 14
};

enum mqtt_connack_return_code
{
    MQTT_CONN_FAIL_SERVER_NOT_FOUND = -5,
    MQTT_CONN_FAIL_NOT_A_CONNACK_MSG = -4,
    MQTT_CONN_FAIL_DNS = -3,
    MQTT_CONN_FAIL_TIMEOUT_RECEIVING = -2,
    MQTT_CONN_FAIL_TIMEOUT_SENDING = -1,
    MQTT_CONNACK_ACCEPTED = 0,
    MQTT_CONNACK_REFUSED_PROTOCOL_VER = 1,
    MQTT_CONNACK_REFUSED_ID_REJECTED = 2,
    MQTT_CONNACK_REFUSED_SERVER_UNAVAILABLE = 3,
    MQTT_CONNACK_REFUSED_BAD_USER_OR_PASS = 4,
    MQTT_CONNACK_REFUSED_NOT_AUTHORIZED = 5
};



//连接MQTT指令
typedef struct mqtt_connect_info
{
  unsigned  char* client_id;
  unsigned  char* username;
  unsigned  char* password;
  unsigned  char* will_topic;
```

```
    unsigned  char* will_message;
    int keepalive;
    int will_qos;
    int will_retain;
    int clean_session;

} mqtt_connect_info_t;


int mqtt_get_type(unsigned char* buffer);
int mqtt_get_connect_ret_code(unsigned char* buffer);
int mqtt_get_qos(unsigned char* buffer);
uint16_t mqtt_get_id(unsigned char* buffer, uint16_t length);

int mqtt_msg_connect(mqtt_connect_info_t* info,unsigned char **data_ptr,unsig
int mqtt_msg_connect_ack(unsigned char *buff);
int mqtt_msg_subscribe_topic(unsigned char* topic, int qos,unsigned char **da
int mqtt_msg_subscribe_ack(unsigned char* buffer, uint16_t length);
int mqtt_msg_publish(unsigned char* topic,unsigned char* date, int data_lengt

int mqtt_get_total_length(unsigned char* buffer, uint16_t length);

int mqtt_msg_puback(uint16_t message_id,unsigned char **data_ptr,unsigned cha
int mqtt_msg_pubrel(uint16_t message_id,unsigned char **data_ptr,unsigned cha
int mqtt_msg_pubrec(uint16_t message_id,unsigned char **data_ptr,unsigned cha
int mqtt_msg_pubcomp(uint16_t message_id,unsigned char **data_ptr,unsigned ch

const char* mqtt_get_publish_topic(unsigned char* buffer, uint16_t* length);
const char* mqtt_get_publish_data(unsigned char* buffer, uint16_t* length);

int mqtt_msg_pingreq(unsigned char **data_ptr,unsigned char* buffer,int buffe

#endif
```

4.咱利用网络模块的TCP连接上以后

然后需要发送第一条消息(注:并不是上来就可以订阅主题的)

MQTT软件规定呢,你发送的第一条信息是连接信息(相当于咱要先登录)

他规定了几个参数!

ClientID: 各个客户端必须设定一个ID,各个客户端必须都不一样

假设是 123456

用户名: 咱安装MQTT软件的时候可以设置MQTT软件的登录的用户名

假设是yang

密码: 咱安装MQTT软件的时候可以设置MQTT软件的登录的密码

假设是 11223344

测试MQTT连接协议

1.以下封装的函数是我为了能够让大家好理解整个MQTT协议,所以再次做了精简(切勿使用下面的作为工程项目)

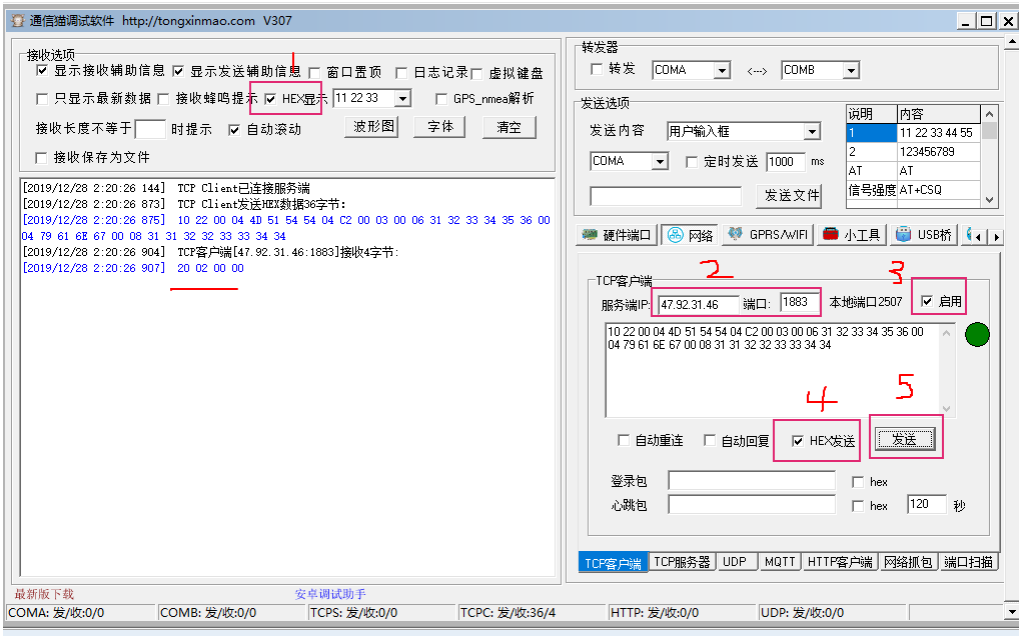
```
/**
* @brief  连接服务器的打包函数
* @param
* @retval
* @example
**/
int ConnectMqtt(char *ClientID,char *Username,char *Password)
{
    int ClientIDLen = strlen(ClientID);
    int UsernameLen    = strlen(Username);
    int PasswordLen = strlen(Password);
    int DataLen = 0;
    int Index = 2;
    int i = 0;
    DataLen = 12 + 2+2+ClientIDLen+UsernameLen+PasswordLen;
    MqttSendData[0] = 0x10;                //MQTT Message Type CONNECT
    MqttSendData[1] = DataLen;    //剩余长度(不包括固定头部)
    MqttSendData[Index++] = 0;         // Protocol Name Length MSB
    MqttSendData[Index++] = 4;         // Protocol Name Length LSB
    MqttSendData[Index++] = 'M';         // ASCII Code for M
    MqttSendData[Index++] = 'Q';         // ASCII Code for Q
    MqttSendData[Index++] = 'T';         // ASCII Code for T
    MqttSendData[Index++] = 'T';         // ASCII Code for T
    MqttSendData[Index++] = 4;         // MQTT Protocol version = 4
    MqttSendData[Index++] = 0xc2;         // conn flags
    MqttSendData[Index++] = 0;         // Keep-alive Time Length MSB
    MqttSendData[Index++] = 60;         // Keep-alive Time Length LSB  60S心跳
    MqttSendData[Index++] = (0xff00&ClientIDLen)>>8;// Client ID length MSB
    MqttSendData[Index++] = 0xff&ClientIDLen;    // Client ID length LSB

    for(i = 0; i < ClientIDLen; i++)
    {
        MqttSendData[Index + i] = ClientID[i];
    }
    Index = Index + ClientIDLen;

    if(UsernameLen > 0)
    {
        MqttSendData[Index++] = (0xff00&UsernameLen)>>8;//username length MSB
        MqttSendData[Index++] = 0xff&UsernameLen;    //username length LSB
        for(i = 0; i < UsernameLen ; i++)
        {
            MqttSendData[Index + i] = Username[i];
        }
        Index = Index + UsernameLen;
    }

    if(PasswordLen > 0)
    {
```

```
            MqttSendData[Index++] = (0xff00&PasswordLen)>>8;//password length MSB
            MqttSendData[Index++] = 0xff&PasswordLen;    //password length LSB
            for(i = 0; i < PasswordLen ; i++)
            {
                MqttSendData[Index + i] = Password[i];
            }
            Index = Index + PasswordLen;
        }
        return Index;
    }
```

假设我ClientID填写的是:123456
UserName填写的是:yang
Password填写的是:11223344

执行以后得到以下数据
10 22 00 04 4D 51 54 54 04 C2 00 78 00 06 31 32 33 34 35 36 00 04 79 61 6E 67 00 08 31 31 32 32 33 33 34 34
然后把这个数据发给TCP 服务器,如果没有错误,服务器就会回 20 02 00 00
咱可以用TCP调试助手试一试

IP地址:47.92.31.46 注意:IP地址可能不能不能使用了
如果IP不可以连接可以填域名 mnif.cn
端口号:1883



先说一件事情 所有的MQTT数据哈 第一个字节是说明整个数据是干什么的数据 第二个字节是说它后面的数据的总个数

10 : 固定,MQTT规定的连接用0x10

22: 是说0x22后面有0x22个数据 34个

00 04: 后面记录MQTT版本号的字节个数

4D 51 54 54: M Q T T 版本号字符 这个是4版本,不同版本不一样

3版本的是MQIsdp 额,了解就可以

04: 版本号是 0x04

C2:这个呢想了解具体呢,需要看协议 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028

C2 的二进制是 1100 0010

bit7 bit6:是否有用户名和密码

bit5 :遗嘱是否需要服务器保留

bit4 bit3:遗嘱的消息等级

bit2:是否设置了遗嘱

bit1:是否清除以前的连接信息

bit0:保留,默认0

上面就是说有用户名和密码,每次连接的时候清除连接信息,没有设置遗嘱(后面会说)

3.1.2.3 Connect Flags

The Connect Flags byte contains a number of parameters specifying the behavior of the MQTT connection. It also indicates the presence or

Figure 3.4 - Connect Flag bits

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero [MC

00 78: 心跳包是120S一次(这个自己连接的时候自己设置),

MQTT规定客户端必须发心跳包,客户端发送的心跳包数据是 0xC0 0x00,这是MQTT规定的

如果心跳包间隔了你设定心跳包的1.5倍时间,你没有发给服务器,服务器就认为你掉线了,这是关于遗嘱的问题,,后面会说

你发给服务器 0xC0 0x00 服务器会回你 0xD0 0x00 这个知道就行了

00 06:客户端的ClientId有6位
后面的 31 32 33 34 35 36 就是ClientId ,这是MQTT服务器规定的,每个客户端必须有各自的ClientId

00 04: MQTT的用户名
79 61 6E 67 我安装MQTT的时候设置的MQTT的用户名是yang

00 08: MQTT的密码
31 31 32 32 33 33 34 34 我安装MQTT的时候设置的MQTT密码

好了,总结下就是连接上TCP服务器 然后发送
10 22 00 04 4D 51 54 54 04 C2 00 03 00 06 31 32 33 34 35 36 00 04 79 61 6E 67 00 08 31 31 32 32 33 33 34 34
服务器呢就会回你 20 02 00 00
20: 固定
02: 后面有两个数据
00 00
注意后面的第一个数据 00 ,如果你设置了 Clean Session为1 :便会回复 01

最后一个数据呢,有几个返回值,0就说明成功,其它就是有各种问题
比如说回的是 20 02 00 04 就说明用户名或者密码有问题.

3.2.2.3 Connect Return code

Byte 2 in the Variable header.

The values for the one byte unsigned Connect Return code field are listed in Table 3.1 – Connect Return code values. If a well formed CONNECT Packe appropriate non-zero Connect return code from this table. If a server sends a CONNACK packet containing a non-zero return code it MUST then close th

Table 3.1 – Connect Return code values

Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect
6-255		Reserved for future use

订阅主题

以下封装的函数是我为了能够让大家好理解整个MQTT协议,所以再次做了精简(切勿使用下面的作为工程项目)

```
/**
* @brief  MQTT订阅/取消订阅数据打包函数
* @param  SendData
* @param  topic                主题
* @param  qos          消息等级
* @param  whether      订阅/取消订阅请求包
* @retval
* @example
**/
int MqttSubscribeTopic(char *topic,u8 qos,u8 whether)
{
    int topiclen = strlen(topic);
    int i=0,index = 0;

    if(whether)
        MqttSendData[index++] = 0x82;                          //0x82 //消息类型和
    else
        MqttSendData[index++] = 0xA2;                          //0xA2 取消订阅
    MqttSendData[index++] = topiclen + 5;                  //剩余长度(不包括固定头部
    MqttSendData[index++] = 0;                             //消息标识符,高位
    MqttSendData[index++] = 0x01;                        //消息标识符,低位
    MqttSendData[index++] = (0xff00&topiclen)>>8;     //主题长度(高位在前,低位在后
    MqttSendData[index++] = 0xff&topiclen;                 //主题长度

    for (i = 0;i < topiclen; i++)
    {
        MqttSendData[index + i] = topic[i];
    }
    index = index + topiclen;

    if(whether)
    {
        MqttSendData[index] = qos;//QoS级别
        index++;
    }
    return index;
}
```

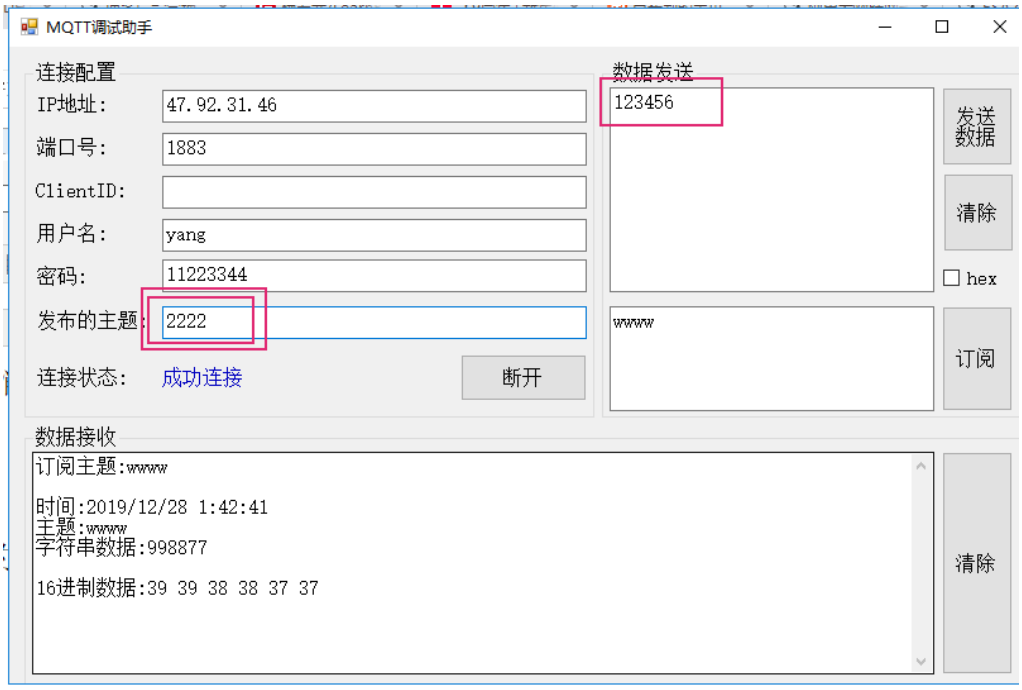
假设告诉服务器我订阅的是2222
假设订阅的时候订阅的主题的消息标识是1,消息等级是0

那么打包以后就是 82 09 00 01 00 04 32 32 32 32 00 然后把这个数据发给TCP服务器

让测试用TCP调试助手订阅,然后用咱的MQTT调试助手发信息给咱的TCP调试助手

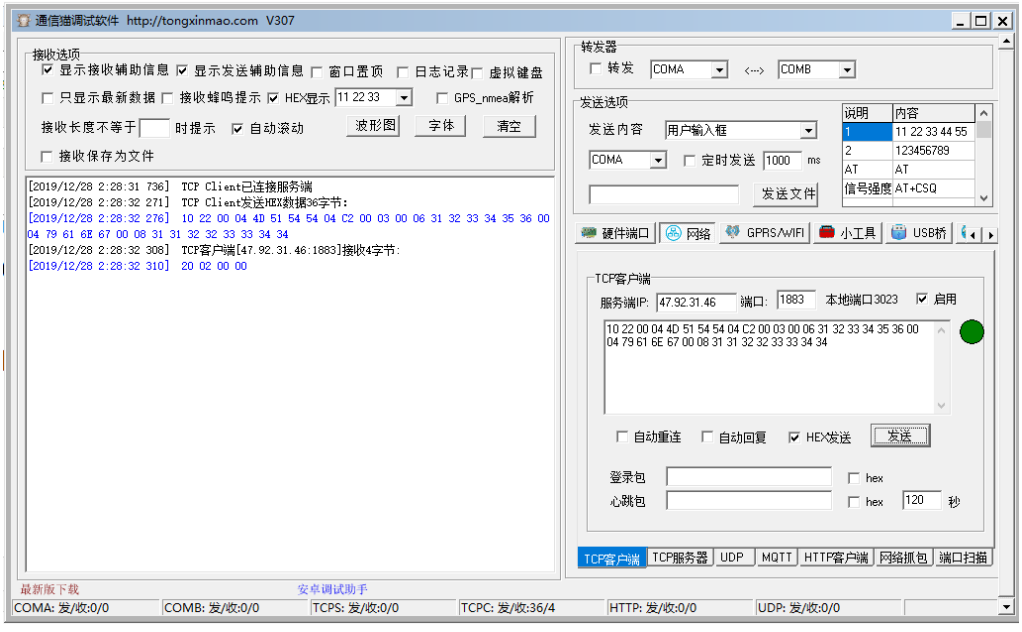
注意:现在咱的TCP可能已经断开了,因为咱的TCP调试助手没有在规定时间内发送心跳包

首先准备好调试助手



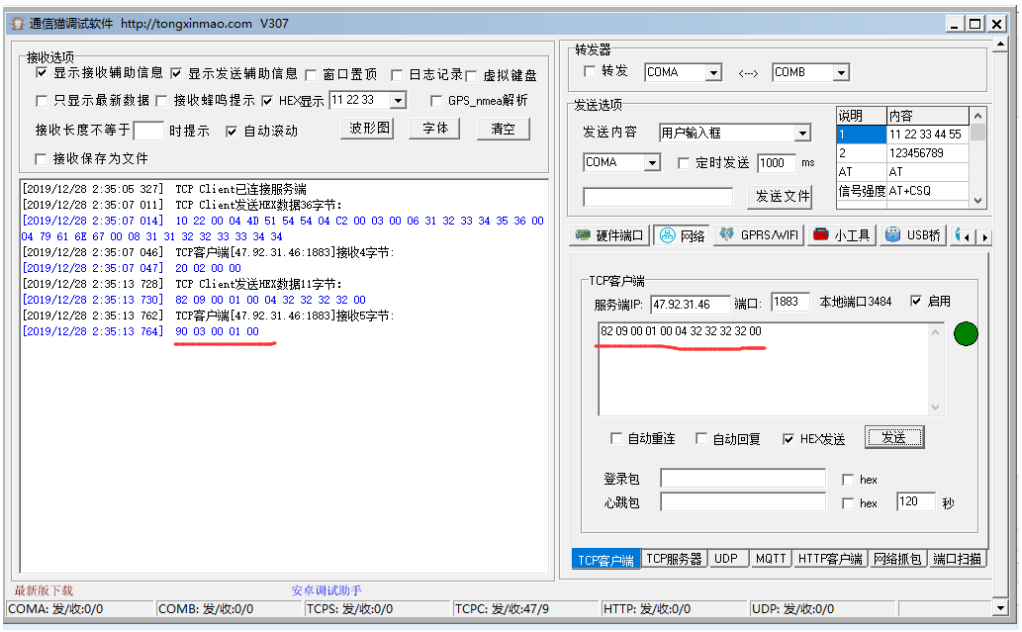
首先连接

10 22 00 04 4D 51 54 54 04 C2 00 03 00 06 31 32 33 34 35 36 00 04 79 61 6E 67 00 08 31 31 32 32 33 33 34 34

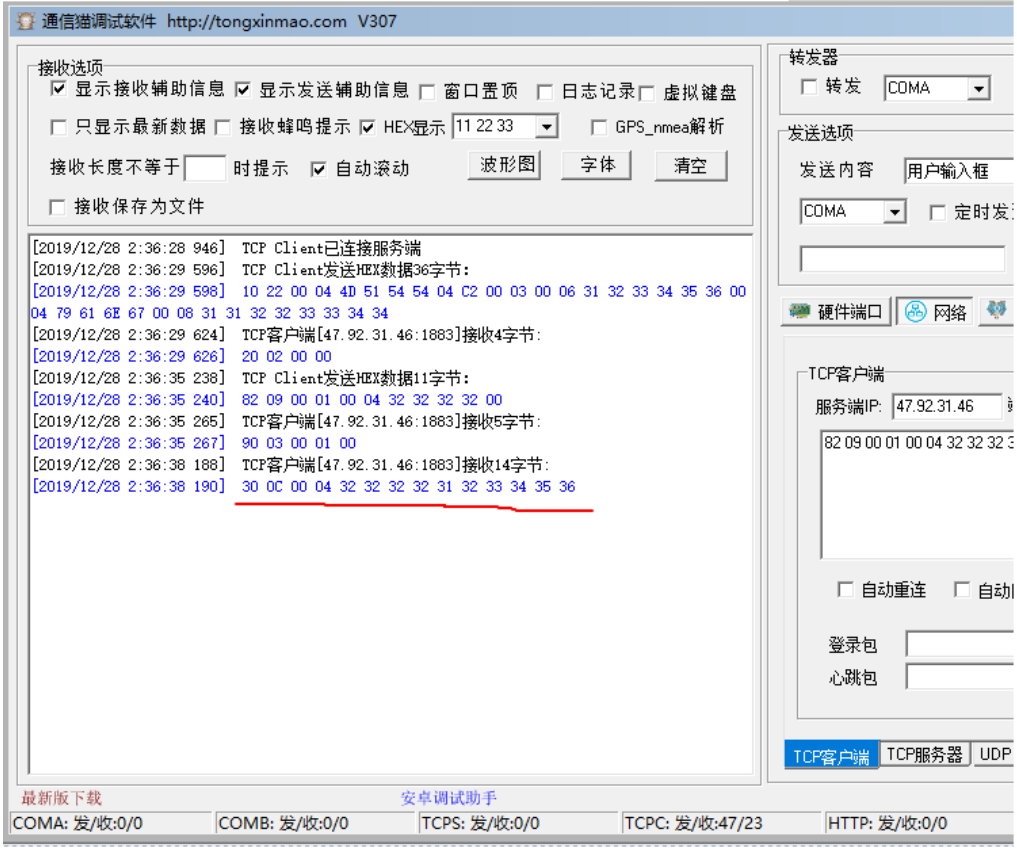


然后订阅

82 09 00 01 00 04 32 32 32 32 00



然后用MQTT调试助手发消息



0x82: 告诉MQTT服务器,我要订阅主题

0x09: 后面的数据个数

0x00 0x01 注意哈,订阅主题的时候可以设置了标识 标识呢 1-65535

之所以有这个家伙:咱订阅的时候怎么判断订阅成功了呢???

订阅成功以后呢!服务器会返回咱订阅成功的回复,回复里面就包含着咱写的这个标识

咱呢可以对比下这个标识,然后呢就知道到底是不是订阅成功了.

0x00 0x04 后面订阅主题的长度

32 32 32 32 订阅的主题是 2222

最后一个 00 是说消息等级,一般呢,订阅设置为0 就可以

那就说一下这个消息等级有什么用吧!

咱发送数据的时候也会携带一个消息等级

假设是0 那么这条消息是不是真的发给MQTT服务器(Broker)了,就不知道了,

假设是1 那么一个客户端发送消息以后呢,服务器一看消息等级是1,那么就会回给那个发送消息的客户端一个应答消息

客户端可以根据有没有回复应答确认发没发送成功

假设是2 这个呢服务器和客户端之间会有双向的应答!后面会详细说.

如果按照上面发呢,服务器会回
90 03 00 01 00
90:固定
03:后面的数据长度
00 01:这条主题的标识
00:消息等级

如果订阅多个主题假设订阅两个主题 消息等级第一个是0 第二个是1
90 04 00 01 00 01
90:固定
03:后面的数据长度
00 01:这条主题的标识
00:消息等级
01:消息等级

假设订阅失败
后面的消息等级就会变为 0x80 (订阅一个主题)
90 03 00 01 00
90:固定
03:后面的数据长度
00 01:这条主题的标识
80:消息等级变为0x80

订阅两个主题,第一个订阅失败
后面的消息等级就会变为 0x80
90 03 00 01 00
90:固定
04:后面的数据长度
00 01:这条主题的标识
80:消息等级
00:消息等级

发布消息

发布的时候呢,信息里面都有以下内容

发布的主题,消息,消息等级,是不是需要服务器保留消息,消息的标识

```
/**
* @brief  MQTT发布数据打包函数
* @param  mqtt_message
* @param  topic                主题
* @param  qos          消息等级
* @retval
* @example
**/
int MqttPublishData(char * topic, char * message, u8 qos)
{
    int topic_length = strlen(topic);
    int message_length = strlen(message);
    int i,index=0;
    static u16 id=0;

    MqttSendData[index++] = 0x30;    // MQTT Message Type PUBLISH  30:消息等级是


    if(qos)
        MqttSendData[index++] = 2 + topic_length + 2 + message_length;//数据长
    else
        MqttSendData[index++] = 2 + topic_length + message_length;   // Remai


    MqttSendData[index++] = (0xff00&topic_length)>>8;//主题长度
    MqttSendData[index++] = 0xff&topic_length;

    for(i = 0; i < topic_length; i++)
    {
        MqttSendData[index + i] = topic[i];//拷贝主题
    }
    index += topic_length;

    if(qos)
    {
        MqttSendData[index++] = (0xff00&id)>>8;
        MqttSendData[index++] = 0xff&id;
        id++;
    }

    for(i = 0; i < message_length; i++)
    {
        MqttSendData[index + i] = message[i];//拷贝数据
    }
    index += message_length;

    return index;
}
```

发布的主题: 谁订阅了这个主题,服务器就会把相应的消息传给谁
消息等级:上面说了
是不是需要服务器保留消息:一会和遗嘱一块说
消息的标识:每条消息加个标识,用来区分消息

遗嘱

还记得上面

Figure 3.4 - Connect Flag bits

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero

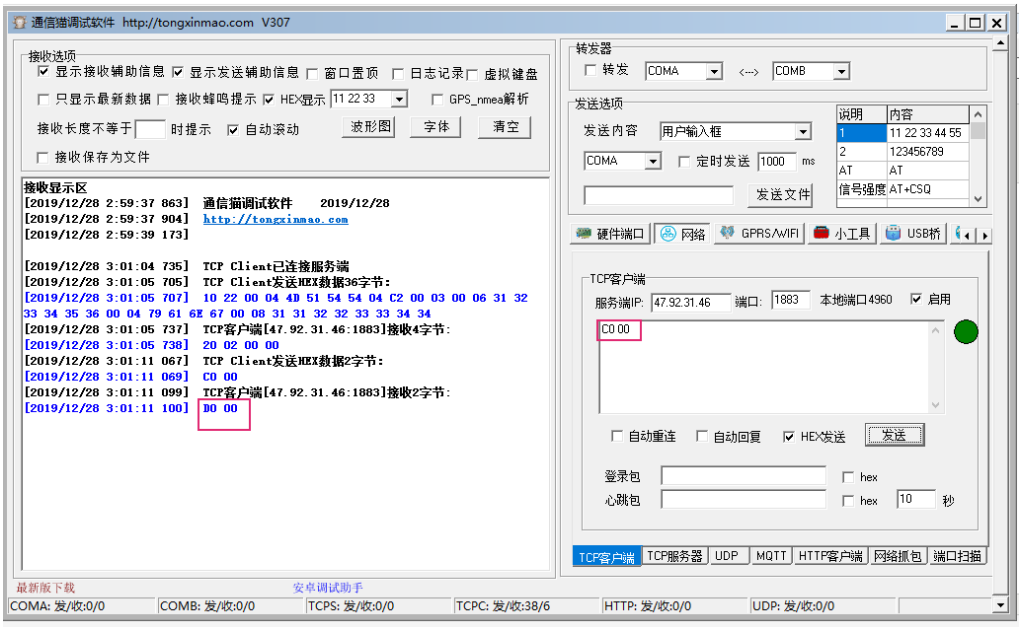
3.1.2.4 Clean Session

我直接说遗嘱是啥意思哈!
假设我手机和一个设备订阅主题和发布主题对应,我就能和这个设备通信了
但是,我怎么知道这个设备掉线了呢?
当然完全可以自己发信息给那个设备,如果不回复,就说明掉线了
但是呢!MQTT服务器提供了一种方式
假设我设置好一个设备的遗嘱消息是 offline 遗嘱发布的主题是 aaaaa
另一个设备订阅的主题是 aaaaa
如果设备掉线,服务器就会给订阅了aaaaa的设备发送 offline

还记得上面说的不 服务器如果在你设置的心跳包时间的1.5倍收不到心跳包就认为你掉线了.
当然订阅系统主题也可以,这个后面再说.

心跳包

MQTT规定的,发送完连接协议之后
发送的心跳包数据是C0 00
发送时间:连接协议里面的心跳包时间(你可以提前发)
然后服务器回复 D0 00



接收所有设备数据

1.有人会问,如果我想监控所有设备的数据应该怎么做

就是说,我有个所有设备都可以管理的后台

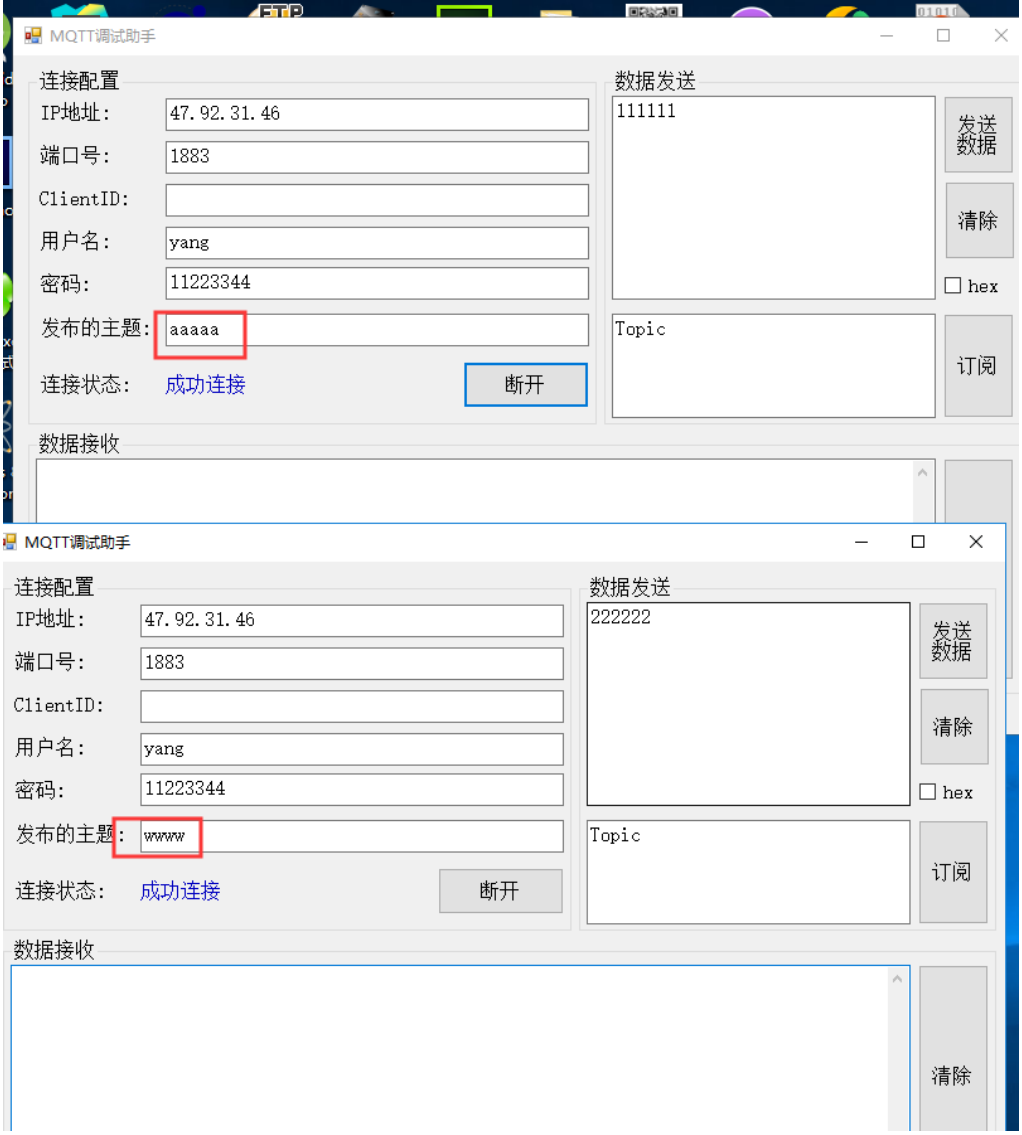
假设我是用C#做了一个MQTT的上位机,监控所有的数据

笨法:

你订阅的时候把所有设备发布的主题全部订阅一遍

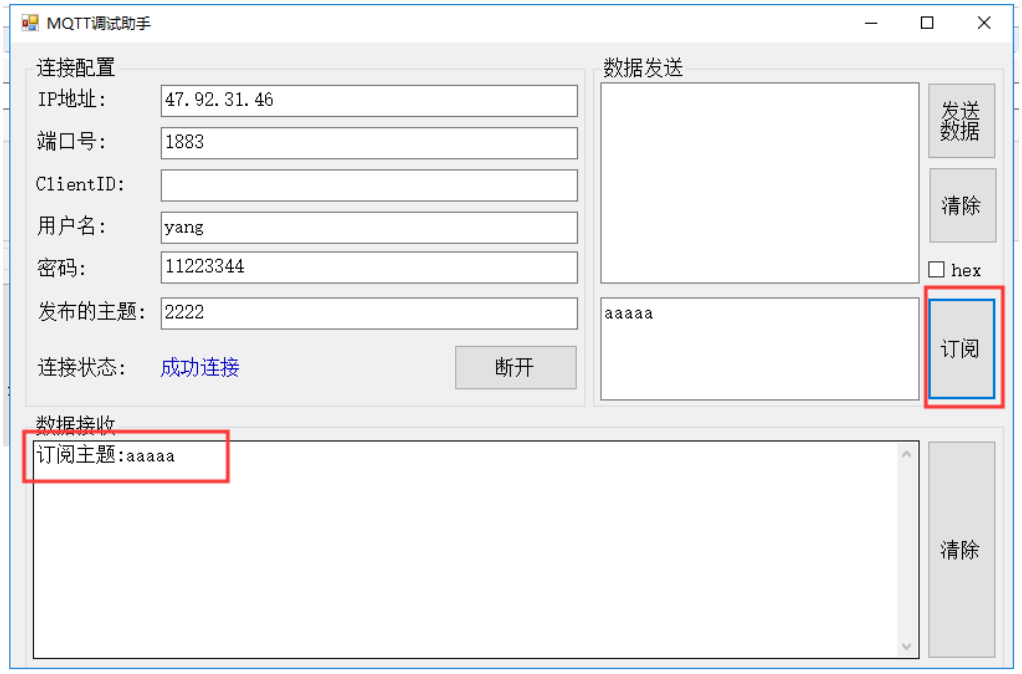
假设现在其中一个设备,想获取其它连个设备的数据

其它两个设备发布的主题如下:



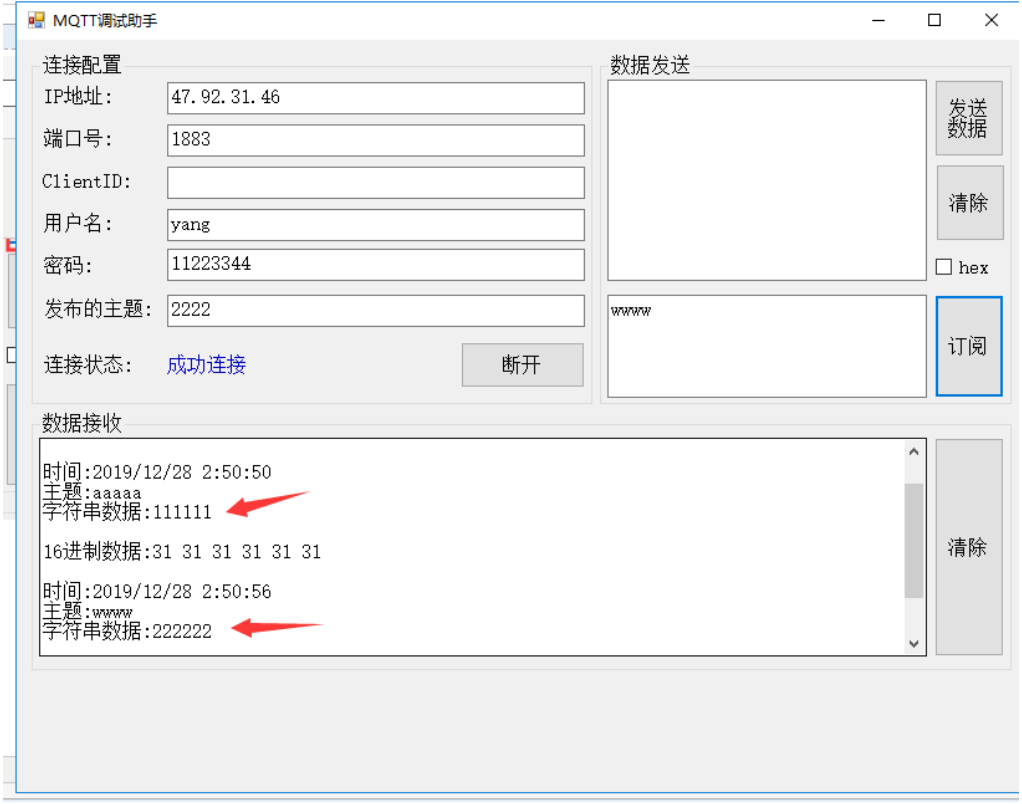
另一个设备

订阅 aaaaa 然后再订阅 wwww



然后就可以了

另一个客户端



MQTT自带的绝招:

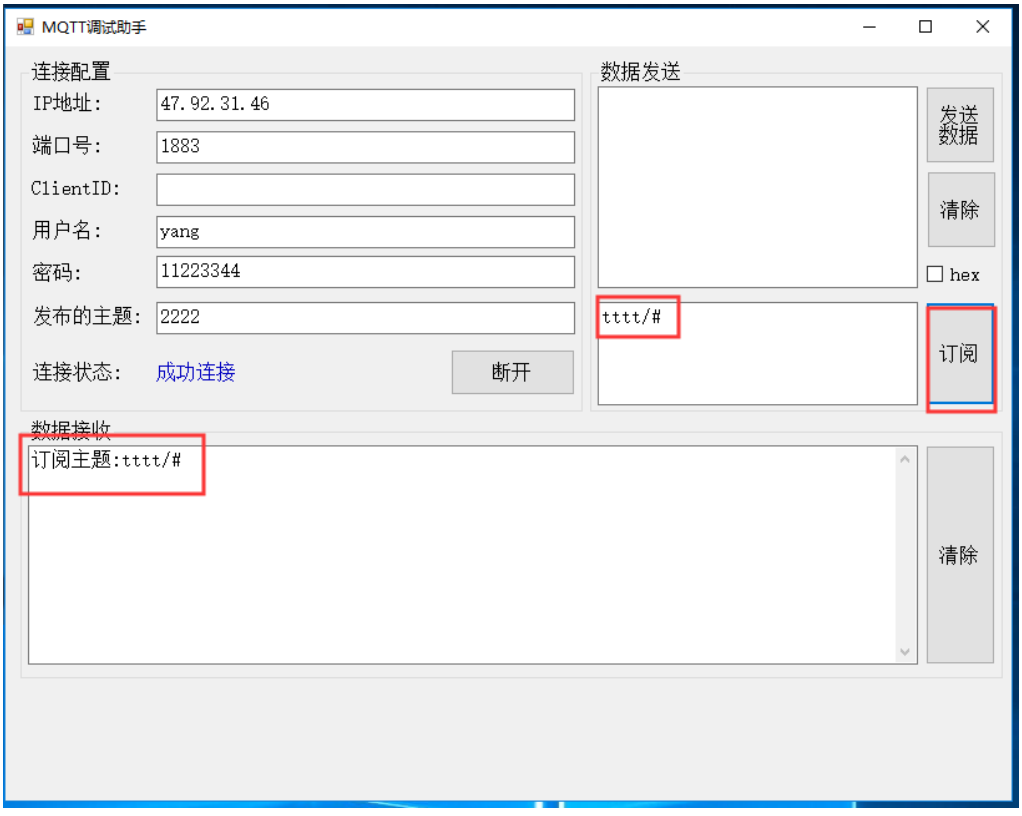
先说一下哈

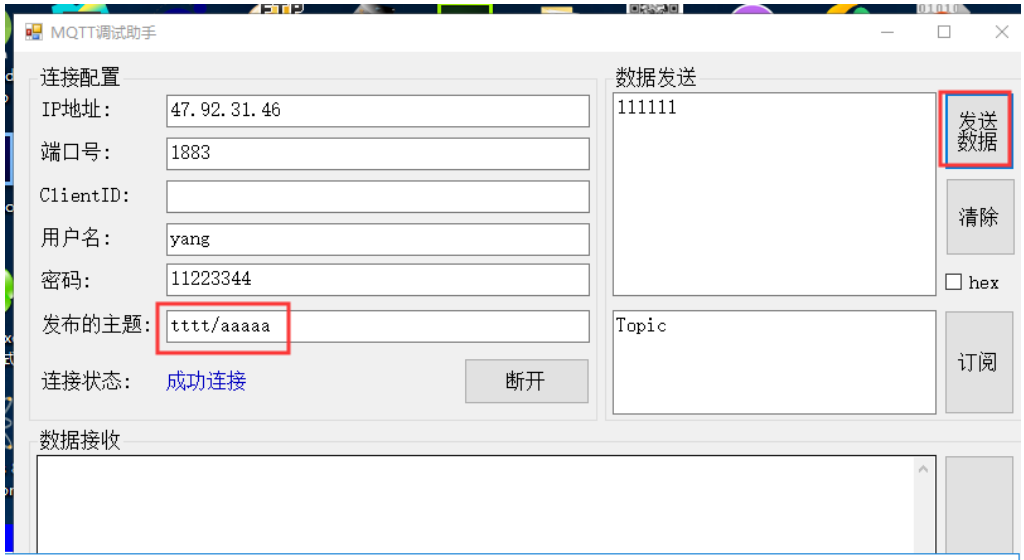
假设一个客户端发布的主题是 tttt/aaaaa

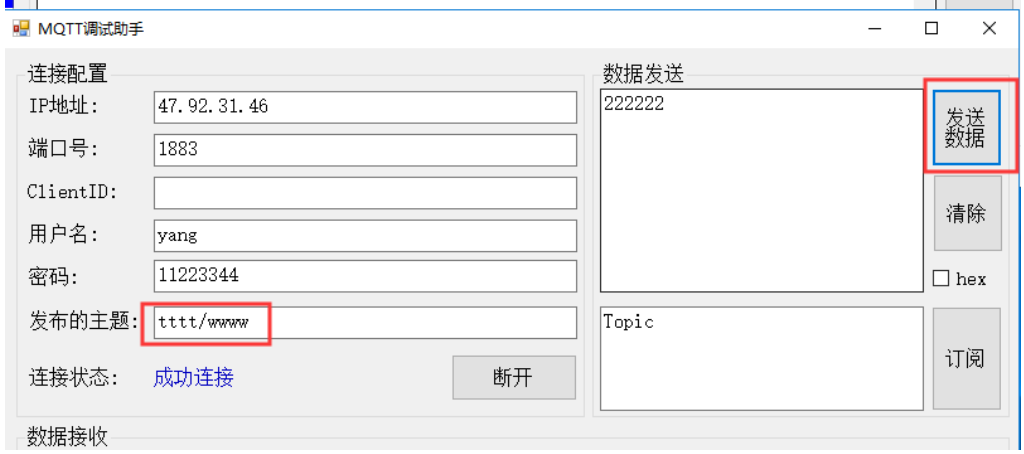
还有一个客户端发布的主题是 tttt/wwww

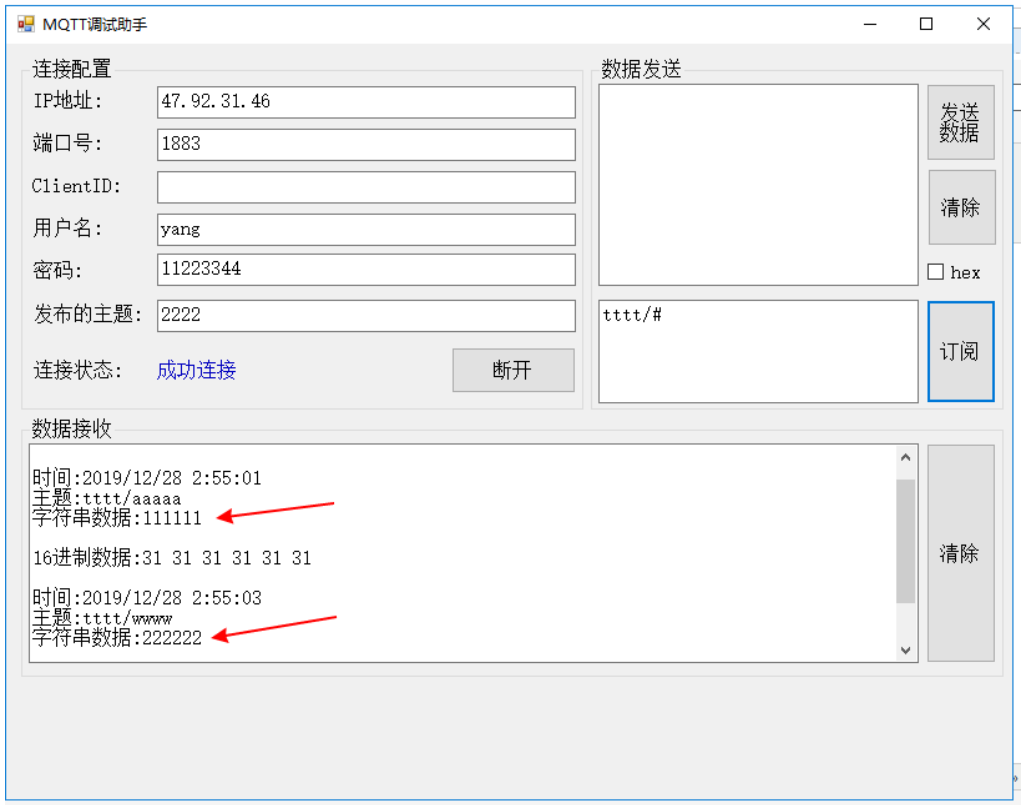
如果想让有一个客户端接收他俩的数据

你只需要订阅 tttt/#



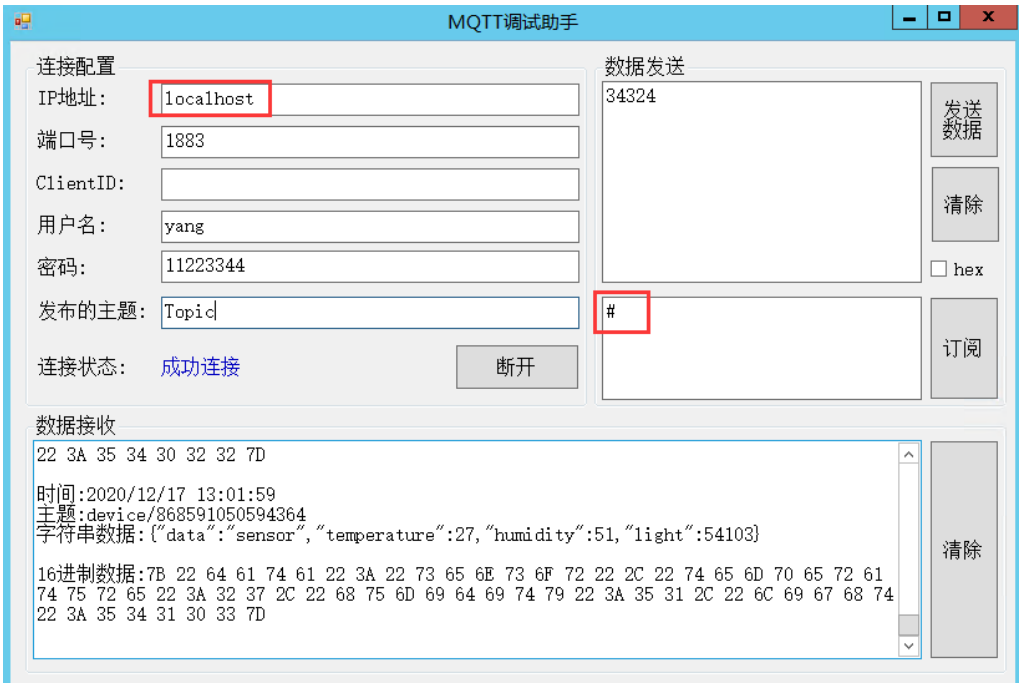

MQTT调试助手
连接配置
IP地址: 47.92.31.46
端口号: 1883
ClientID:
用户名: yang
密码: 11223344
发布的主题: tttt/aaaaa
连接状态: 成功连接
断开
数据发送
111111
发送数据
清除
hex
Topic
订阅
数据接收


MQTT调试助手
连接配置
IP地址: 47.92.31.46
端口号: 1883
ClientID:
用户名: yang
密码: 11223344
发布的主题: tttt/wwww
连接状态: 成功连接
断开
数据发送
222222
发送数据
清除
hex
Topic
订阅
数据接收

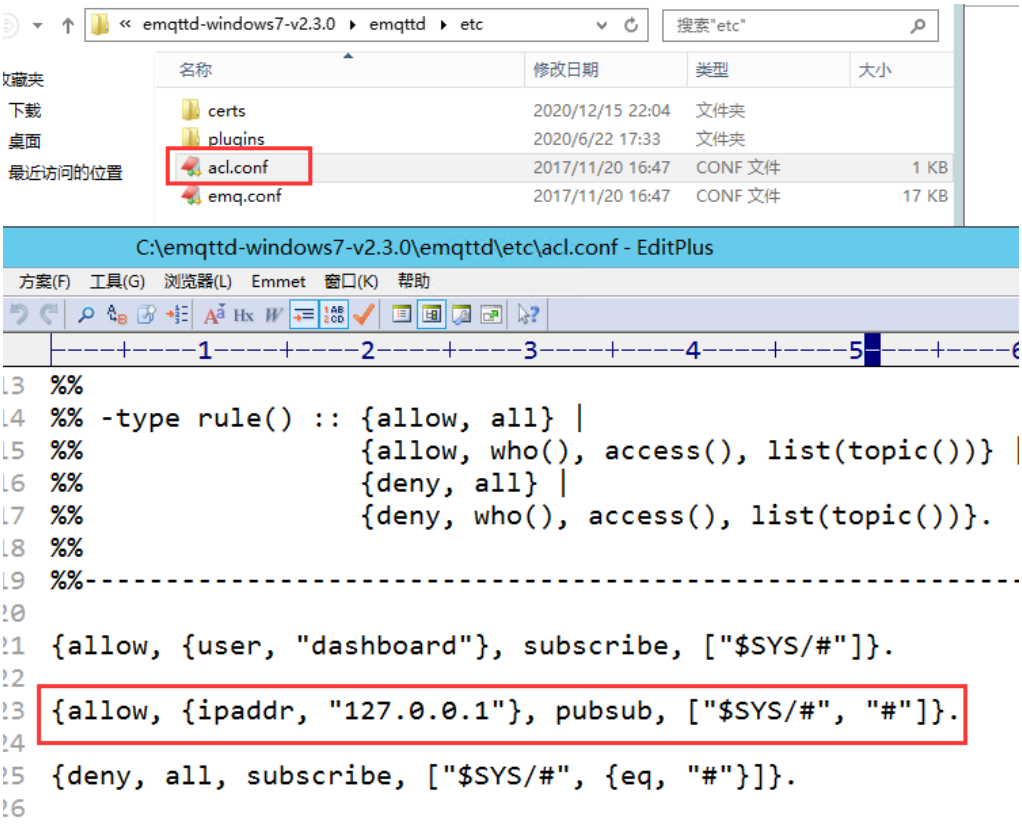

MQTT调试助手
连接配置
IP地址: 47.92.31.46
端口号: 1883
ClientID:
用户名: yang
密码: 11223344
发布的主题: 2222
连接状态: 成功连接
断开
数据发送
发送数据
清除
hex
tttt/#
订阅
数据接收
时间:2019/12/28 2:55:01
主题:tttt/aaaaa
字符串数据:111111
16进制数据:31 31 31 31 31 31
时间:2019/12/28 2:55:03
主题:tttt/wwww
字符串数据:222222
清除

2.上面的仅仅局限于wwww/XXX的设备,真正接收所有设备数据

把客户端放到安装MQTT软件的那台服务器上(后期安装MQTT软件就可以试验了),IP地址填写 localhost　订阅的主题填写 #



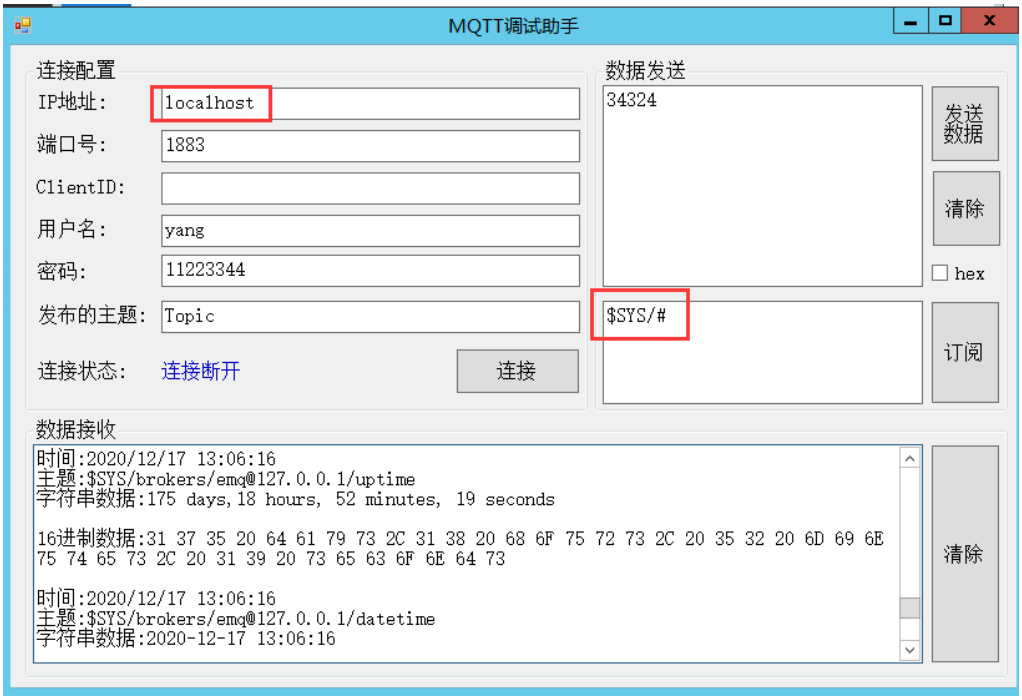
只所以必须在服务器才可以. 是因为这个权限控制(后期安装MQTT软件就看到了)



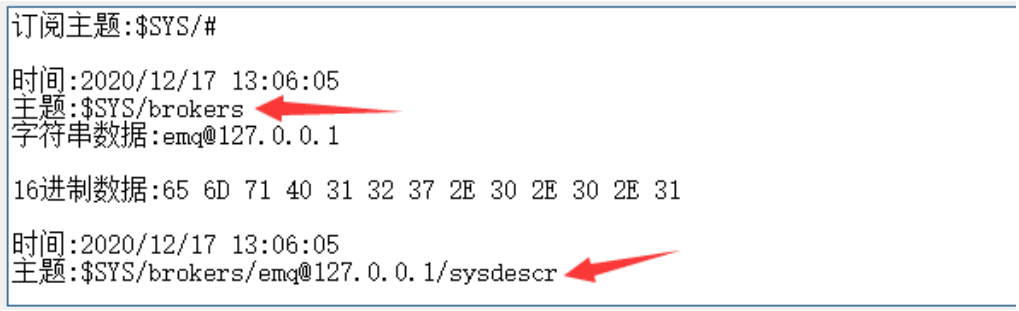
允许 IP地址(127.0.0.1) 订阅和发布的主题 $SYS/# #

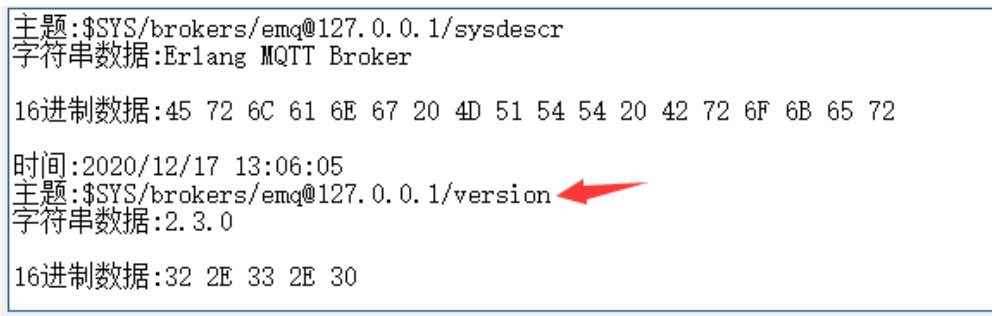
用系统主题监控设备上下线

1.在服务器上订阅 $SYS/#



在刚一执行订阅





$SYS/brokers : 集群节点列表

$SYS/brokers/emq@127.0.0.1/sysdescr 服务器描述

$SYS/brokers/emq@127.0.0.1/version　　服务器版本

咱主要关注的是下面的

会接收到这个主题:　$SYS/brokers/emq@127.0.0.1/datetime

这个主题的消息是:　2020-12-17 13:06:16

这个主题每隔1S发布一次时间,这个是系统自带的

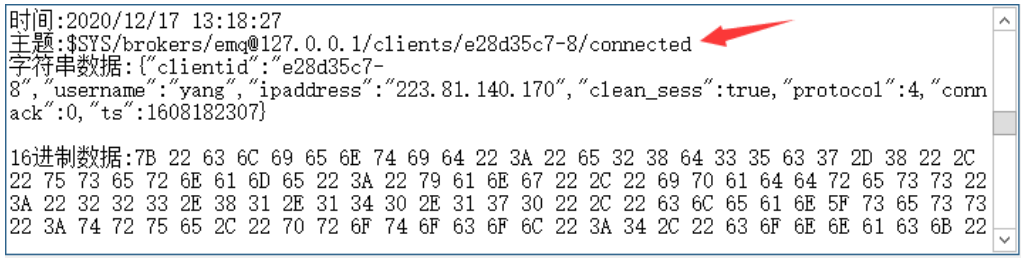
会接收到这个主题:　$SYS/brokers/emq@127.0.0.1/uptime

这个主题的消息是:　175 days,18 hours, 52 minutes, 19 seconds

这个主题每隔1S发布一次时间,这个是MQTT服务器启动运行的时间

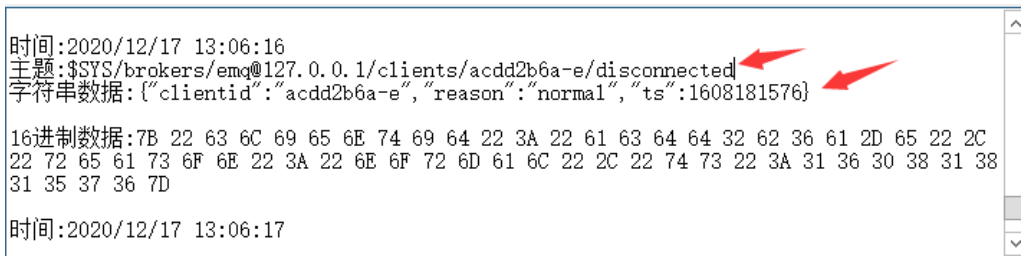
某个设备上线,下面是clientid为 e28d35c7-8 的设备上线了

$SYS/brokers/emq@127.0.0.1/clients/e28d35c7-8/connected



某个设备掉线,下面是clientid为 acdd2b6a-e的设备掉线了

$SYS/brokers/emq@127.0.0.1/clients/acdd2b6a-e/disconnected



2.监控所有设备上下线只需要

监控设备上线,订

阅: $SYS/brokers/emq@127.0.0.1/clients/+/connected

监控设备下线,订
阅: $SYS/brokers/emq@127.0.0.1/clients/+/disconnected

提示:上面的加号代表任意.这也是MQTT的一个招式.

MQTT消息等级和DUP

1.假设客户端1 发布的主题是 1111 ;消息等级是:0 ;发送的消息是999 最终发送的信息如下:

30 0b 00 04 31 31 31 31 39 39 39
消息等级是0是说明该消息发送出去就完事了,服务器不会回复任何应答信息.
至于该消息发没发给服务器,不知道!

假设客户端2 订阅的主题是:1111 消息等级是 0
假设客户端1 确实把消息发给了服务器
客户端2 收到消息以后,不需要做任何操作

2.假设客户端1 发布的主题是 1111 ;消息等级是:1 ;发送的消息是999 最终发送的信息如下:

32 0b 00 04 31 31 31 31 XX XX 39 39 39
XX XX是在发送的时候需要加上的消息标识符:
消息标识符XX XX随意即可:范围1-65535
假设消息标识符是 00 01

发送完以上消息以后,服务器会回复: (PUBACK) 告诉客户端我收到了
40 02 00 01
(00 01就是咱上面发送的消息标识符)
这样就证明消息确实送达给了服务器

如果客户端1 发布完消息以后没有接收到服务器的应答
则可以重新发布消息
32 0b 00 04 31 31 31 31 XX XX 39 39 39

XX XX可以和上次的一样,也可以不一样

3.假设客户端2订阅了主题是 1111 ;消息等级是:1

服务器接收到客户端1发送的消息之后,转发给客户端2

32 0b 00 04 31 31 31 31 XX XX 39 39 39

注意现在的XX XX(消息标识符)是服务器自己随机生成的了

假设标识符是 00 02

客户端2在接收到消息之后需要返回应答(PUBACK) 告诉服务器我收到了

40 02 00 02

如果客户端2不回复:40 02 00 02 (后面咱就叫 PUBACK)

服务器便会一直发送消息给客户端2

3A 0b 00 04 31 31 31 31 00 02 39 39 39

注意开头变为了 3A (服务器自动会把重传标志置一)

3.3 PUBLISH – Publish message

A PUBLISH Control Packet is sent from a Client to a Server or from Server to a Client to transport an Applica

3.3.1 Fixed header

Figure 3.10 – PUBLISH Packet fixed header illustrates the fixed header format:

Figure 3.10 – PUBLISH Packet fixed header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

3.3.1.1 DUP

Position: byte 1, bit 3.

If the DUP flag is set to 0, it indicates that this is the first occasion that the Client or Server has attempted to

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet [M(

The value of the DUP flag from an incoming PUBLISH packet is not propagated when the PUBLISH Packet packet is a retransmission [MQTT-3.3.1-3].

Non normative comment

The recipient of a Control Packet that contains the DUP flag set to 1 cannot assume that it has seen

高4位是 3 固定

后面四位:

第一位:DUP 标记这条消息是不是重传的

第2,3位:消息等级 01 :消息等级1   10:消息等级2

最后一位:RETAIN 是否需要服务器保留这条消息

本来是 32   0011 0010

变为了 3A   0011 1010

其实服务器加上DUP是为了让客户端知道,我这条消息是重传的,
因为服务器第一次发的时候客户端没有返回PUBACK,但是服务器知道我确实是传给了客户端
客户端这边假设真的是没有及时的回复PUBACK,那么有两种方式处理

1.客户端再次接收到消息以后,无论消息有没有DUP标志,直接处理消息
如果判断这条消息是需要返回 PUBACK的,那么直接根据消息里面的消息标识符返回 PUBACK 即可

2.判断下如果有DUP标志,那么再提取下消息标识,看一下我先前是不是处理了有相同消息标识符的消息
如果有就说明我已经处理了,只是没有返回PUBACK,那么我不去处理这条消息
直接根据消息里面的消息标识符返回PUBACK就可以

其实....
但是整体来说,对于消息等级是1的消息统统处理即可,然后根据消息里面的消息标识符返回PUBACK即可
先说一下为什么
其实在客户端1发布消息等级是1的消息的时候,如果客户端1由于某些原因没有接收到服务器的PUBACK
那么客户端1还会再发布先前的消息,其实现在就有两条或者多条相同的消息在服务器里面
这些相同的消息(标识符不一样的消息)就会发给客户端2,如果客户端2一直不应答(PUBACK),
那么服务器便会把所有的没有收到应答的消息的DUP标记置一以后不停的发给客户端2...
直至客户端2应答了所有的消息,或者客户端2断线了,服务器才停止发送
对于单片机而言,这些处理只能自己去实现,为了方便和节省内存,对于消息等级是1的消息
可以直接根据消息里面的消息标识符返回PUBACK,所以对于消息等级是1的消息,其实客户端至少会接收到1次消息

4.假设客户端1 发布的主题是 1111 ;消息等级是:2 ;发送的消息是999 最终发送的信息如下:

34 0b 00 04 31 31 31 31 XX XX 39 39 39
XX XX是在发送的时候需要加上的消息标识符:
消息标识符XX XX随意即可:范围1-65535
假设消息标识符是 00 01
注意:服务器接收到此消息以后并不会立即发送给订阅了主题是1111,消息等级是2的客户端

服务器接收到以后会返回: PUBREC) "告诉客户端我收到了"
50 02 00 01

客户端1需要返回: PUBREL) "好的"
62 02 00 01
注意:返回这个以后,消息才会下发给订阅了主题是1111,消息等级是2的客户端

服务器接着会返回: PUBCOMP)
70 02 00 01

很多人介绍QS2都说保证消息只传输一次,其实实际上是这样保证的
客户端1在发送完消息以后
34 0b 00 04 31 31 31 31 XX XX 39 39 39
服务器会返回(PUBREC)

但是只要客户端1不回复 (PUBREL)
无论客户端1现在发送多少条消息等级是2的消息
服务器都不会理会,服务器只会记录你发送的最后一条消息
客户端1只有回复了(PUBREL)
服务器才会把最后一条消息转发出去
最后返回 (PUBCOMP)

5.假设客户端2订阅了主题是 1111 ;消息等级是:2

服务器接收到客户端1发送的消息，然后确认接收到客户端1的(PUBREL)之后,
转发给客户端2 :

34 0b 00 04 31 31 31 31 XX XX 39 39 39

注意现在的XX XX(消息标识符)是服务器自己随机生成的了

假设标识符是 00 02

客户端2接收到以后需要返回: PUBREC) "告诉服务器我收到了"
50 02 00 02
注意:如果客户端2不回复: PUBREC),那么服务器会不停的发送
34 0b 00 04 31 31 31 31 XX XX 39 39 39
直至客户端2回复: PUBREC)

服务器接收到以后会返回: PUBREL) "好的"
62 02 00 02

客户端2最后需要返回: PUBCOMP)
70 02 00 02
注意:即使客户端2不返回(PUBCOMP),服务器隔一段时间也会默认客户端2回复了(PUBCOMP)

MQTT的retain

3.2.3 Payload

The CONNACK Packet has no payload.

3.3 PUBLISH – Publish message

A PUBLISH Control Packet is sent from a Client to a Server or from Server to a Client to transport an Applica

3.3.1 Fixed header

Figure 3.10 – PUBLISH Packet fixed header illustrates the fixed header format:

Figure 3.10 – PUBLISH Packet fixed header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

3.3.1.1 DUP

Position: byte 1, bit 3.

If the DUP flag is set to 0, it indicates that this is the first occasion that the Client or Server has attempted to

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet [M

The value of the DUP flag from an incoming PUBLISH packet is not propagated when the PUBLISH Packet packet is a retransmission [MQTT-3.3.1-3].

发布消息的时候,第一个字节的最后一位代表 Retain
意思是,是否需要服务器保留这个消息

如果设置了服务器保留了这个消息,那么只要客户端订阅了这个消息的主题
服务器就会立马发送给客户端保留的这个消息
详细说明:
假设我发布的主题是:1111 消息是:999 ,消息等级随意(假设是0) 然后设置了 Retain位置为1
31 09 00 04 31 31 31 31 39 39 39
客户端1把这条消息发给了服务器
然后过了一会,客户端2上线了
订阅了主题 1111
因为上面的消息设置了让服务器保留
所以只要客户端2 一订阅1111,便会收到 999这个消息
这就是Retain的作用

当然,有更巧妙的应用
我一般用这个来发送在线或者掉线,或者开关量数据
1.我设置客户端1的遗嘱发布的主题是 1111 遗嘱消息是 offline Retain为1
2.我设置客户端1连接上服务器以后先发布一条消息
发布的主题也是 1111 消息是: online Retain为1

注意:服务器最终只会保留最后一条需要保留的消息

只要是另一个客户端订阅 1111
如果客户端1是掉线的,那么便会立即收到 offline
如果客户端1是在线的,呢么便会立即收到 online

有些时候咱控制开关,咱打开上位机以后想立即知道开关的状态
最好的方式就是把发送开关亮数据的主题 Retain设置为1
那么只要是上位机一订阅设备发布的开关量主题,
便会立即得到开关量数据!
这样便提高了用户体验.

其它

有人测试MQTT,发现只要设备连接上MQTT,不需要订阅,就能接收到服务器的消息

然后他就有点懵!
我说一下,其实这个功能也是属于MQTT的范畴!
大家看MQTT协议,只知道订阅了某个主题就可以收到某个主题的信息
注意:MQTT协议中并没有说只有订阅才可以收到!

我说一下:其实MQTT就是个TCP服务器,MQTT客户端就是个TCP客户端
其实大家要从大的方向上去考虑,整个的通信就分为两层.
第一层是TCP通信
第二层是解析TCP数据
就是这样而已!!!!

我问一下,我TCP服务器可以给TCP客户端主动发信息吧???
其实就是上面说的,不需要订阅就可以接收到信息!
有些人会想,怎么会?
不订阅为啥可以接收?
其实你只是跟着别人学了个表面东西,然后给自己设定了一个错误的思路:只有订阅才能接收信息!

服务器可以主动推送信息给各个客户端!只要是信息格式是正确的就可以
因为TCP接收到信息以后就是解析一下是不是MQTT协议格式的数据而已!
有些人会想,为啥客户端解析协议里面不判断下是不是自己订阅的呢??
这个是你自己根据自己的需求去做的事情!

你要想让你的协议兼容到各个使用场景,你说你是做成开放形式还是封闭的?
你可能咋一听感觉为啥还能这样!
其实本来就是有的东西,只是你以前不了解而已!

结语

以上说的,对于高级语言已经封装好了包,只需要调用API即可,但是对于单片机而言,需要自己处理.

分类: STM32+Air724UG(4G模组)物联网开发

杨奉武
关注 - 1
粉丝 - 620

0 0

« 上一篇： 3-STM32+Air724UG基本控制篇(自建物联网平台)-整体运行测试-微信小程序扫码绑定Air724,并通过MQTT和模组实现远程通信控制
» 下一篇： 101-STM32+Air724UG基本控制篇(自建物联网平台)-基础搭建-购买云主机,安装MQTT服务器软件(.Windows系统)

posted on 2021-04-03 21:54 杨奉武 阅读(277) 评论(0) 编辑 收藏 举报

· 特斯拉FSD V9正式开始推送 UI界面大升级：模型精细到赞叹
· 提升骑手配送体验！美团将改进骑手差评申诉问题
» 更多新闻...