**MQTT协议分析**

date: 2020-5-9

qq: 2391387647

Name: 邹卓恩

# 第一章 mqtt协议介绍

## 1.1 概述

MQTT是一个客户端服务端架构的发布/订阅模式的消息传输协议。它的设计思想是轻巧、开放、简单、规范，易于实现。这些特点使得它对很多场景来说都是很好的选择，特别是对于受限的环境如机器与机器的通信（M2M）以及物联网环境（IoT）。

## 1.2 特点

1. 开放消息协议，简单易实现
2. 发布订阅模式，一对多消息发布
3. 基于TCP/IP网络连接
4. 1字节固定报头，2字节心跳报文，报文结构紧凑
5. 消息QoS支持，可靠传输保证

## 1.3 应用

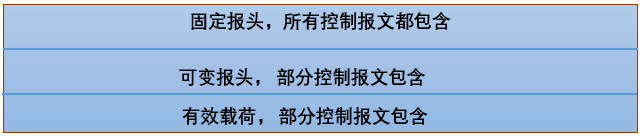
MQTT协议广泛应用于物联网、移动互联网、智能硬件、车联网、电力能源等领域。

1. 物联网M2M通信，物联网大数据采集
2. Android消息推送，WEB消息推送
3. 移动即时消息，例如Facebook Messenger
4. 智能硬件、智能家具、智能电器
5. 车联网通信，电动车站桩采集
6. 智慧城市、远程医疗、远程教育
7. 电力、石油与能源等行业市场

# 第二章 mqtt协议报文格式组成

## 2.1 mqtt控制报文结构

MQTT 协议通过交换预定义的 MQTT 控制报文来通信。 这一节描述这些报文的格式。MQTT 控制报文由三部分组成，如下图:



**图2.1 mqtt报文组成**

## 2.2 mqtt固定报头

每个 MQTT 控制报文都包含一个固定报头， 固定报头的格式如下图:



**图2.2 mqtt固定报头**

## 2.3 mqtt控制报文类型

位置： 第 1 个字节， 二进制位 7-4,表示为 4 位无符号值。

MQTT 控制报文的类型：如下表:

|  |  |  |  |
| --- | --- | --- | --- |
| 名字 | 值 | 报文流动方向 | 描述 |
| Reserved | 0 | 禁止 | 保留 |
| CONNECT | 1 | 客户端到服务端 | 客户端请求连接服务端 |
| CONNACK | 2 | 服务端到客户端 | 连接报文确认 |
| PUBLISH | 3 | 两个方向都允许 | 发布消息 |
| PUBACK | 4 | 两个方向都允许 | QoS 1消息发布收到确认 |
| PUBREC | 5 | 两个方向都允许 | 发布收到（保证交付第一步） |
| PUBREL | 6 | 两个方向都允许 | 发布释放（保证交付第二步） |
| PUBCOMP | 7 | 两个方向都允许 | QoS 2消息发布完成（保证交互第三步） |
| SUBSCRIBE | 8 | 客户端到服务端 | 客户端订阅请求 |
| SUBACK | 9 | 服务端到客户端 | 订阅请求报文确认 |
| UNSUBSCRIBE | 10 | 客户端到服务端 | 客户端取消订阅请求 |
| UNSUBACK | 11 | 服务端到客户端 | 取消订阅报文确认 |
| PINGREQ | 12 | 客户端到服务端 | 心跳请求 |
| PINGRESP | 13 | 服务端到客户端 | 心跳响应 |
| DISCONNECT | 14 | 客户端到服务端 | 客户端断开连接 |
| Reserved | 15 | 禁止 | 保留 |

## 2.4 标记

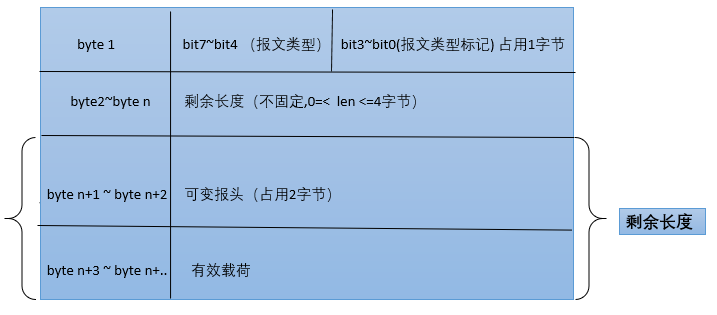
固定报头第 1 个字节的剩余的 4 位 [3-0]包含每个 MQTT 控制报文类型特定的标志 。标记位说明如下表所示:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 控制报文 | 固定报头标志 | Bit 3 | Bit 2 | | | Bit 1 | Bit 0 |
| CONNECT | Reserved | 0 | 0 | 0 | 0 | | |
| CONNACK | Reserved | 0 | 0 | 0 | 0 | | |
| PUBLISH | Used in MQTT 3.1.1 | DUP1 | QoS2 | QoS2 | RETAIN3 | | |
| PUBACK | Reserved | 0 | 0 | 0 | 0 | | |
| PUBREC | Reserved | 0 | 0 | 0 | 0 | | |
| PUBREL | Reserved | 0 | 0 | 1 | 0 | | |
| PUBCOMP | Reserved | 0 | 0 | 0 | 0 | | |
| SUBSCRIBE | Reserved | 0 | 0 | 1 | 0 | | |
| SUBACK | Reserved | 0 | 0 | 0 | 0 | | |
| UNSUBSCRIBE | Reserved | 0 | 0 | 1 | 0 | | |
| UNSUBACK | Reserved | 0 | 0 | 0 | 0 | | |
| PINGREQ | Reserved | 0 | 0 | 0 | 0 | | |
| PINGRESP | Reserved | 0 | 0 | 0 | 0 | | |
| DISCONNECT | Reserved | 0 | 0 | 0 | 0 | | |

DUP1 =控制报文的重复分发标志  
QoS2 = PUBLISH 报文的服务质量等级  
RETAIN3 = PUBLISH 报文的保留标志

## 2.5 剩余长度

位置：从第二个字节开始。剩余长度（ Remaining Length） 表示当前报文剩余部分的字节数， 包括可变报头和负载的数据。 剩余长度不包括用于编码剩余长度字段本身的字节数。



**图2.3 剩余长度包含的报文范围**

剩余长度字段使用一个变长度编码方案， 对小于 128 的值它使用单字节编码。 更大的值按下面的方式处理。低 7 位有效位用于编码数据，最高有效位用于指示是否有更多的字节。 因此每个字节可以编码 128 个数值和一个延续位（ continuation bit） 。 剩余长度字段最大 4 个字节。

例如， 十进制数 64 会被编码为一个字节， 数值是 64， 十六进制表示为 0x40,。十进制数字321(=65+2\*128)被编码为两个字节， 最低有效位在前。 第一个字节是 65+128=193。 注意最高位为  
1 表示后面至少还有一个字节。 第二个字节是 2。

### 2.5.1 示例

123456 = 964 x 128 + 64

964 = 7x128 + 68

7 < 128

也就是123456 = (7 x 128 + 68)x128 + 64

展开:64 + 68 x128 + 7x128x128

第一字节:64 | 0x80 = x (0x80=0x1000 0000或上最高位表示是否还有更多的字节)

第二字节:68 | 0x80 = y (0x80=0x1000 0000或上最高位表示是否还有更多的字节)

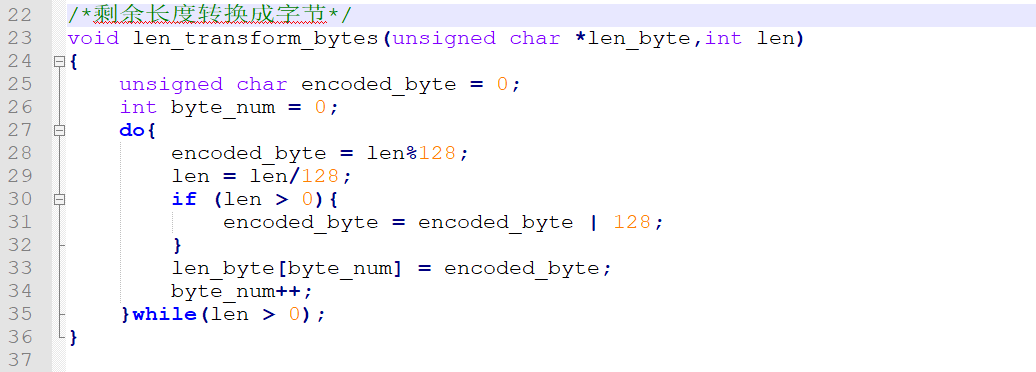
第三字节:7=z

c语言表示:unsigned char len\_byte[4] = { 64 | 128 , 68 | 128, 7 , 0 }

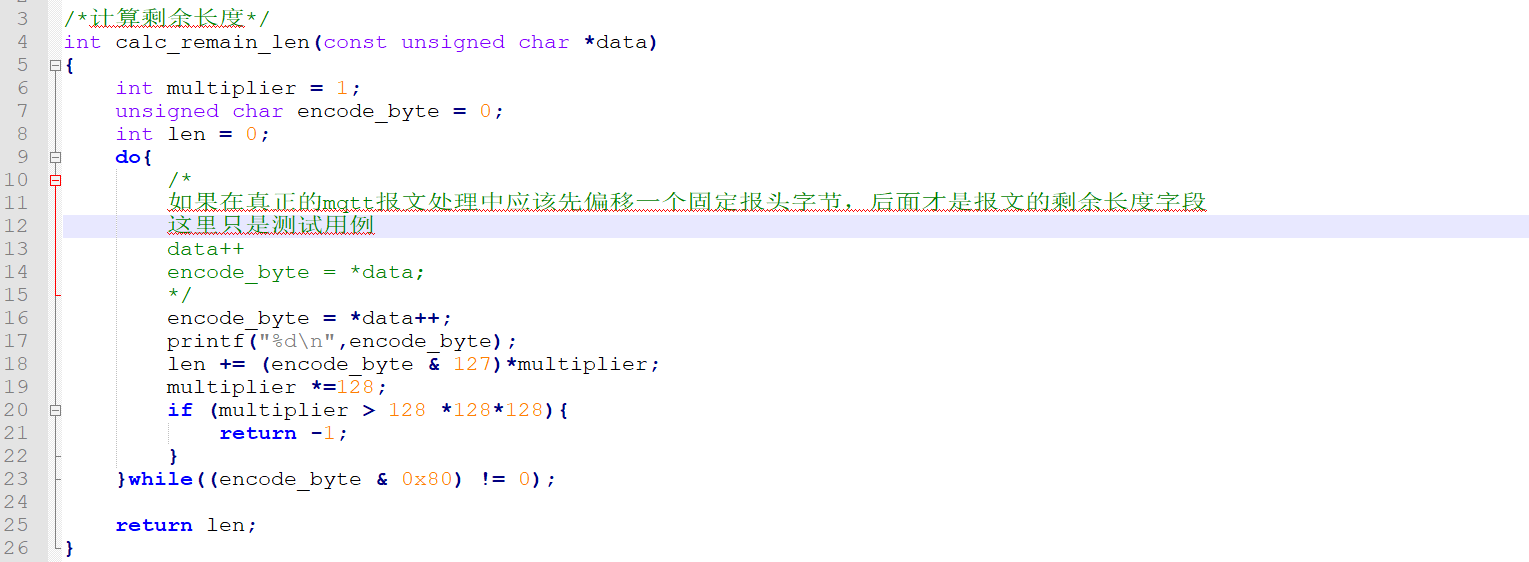
反过来，如果要算出123456

x-128 + (y-128)\*128 + z x 128 x 128

把剩余长度转换成字节表示：



把字节转换成剩余长度表示：



## 2.6 可变报头

某些 MQTT 控制报文包含一个可变报头部分。 它在固定报头和负载之间。可变报头的内容根据报文类型的不同而不同。报文标识符是可变报头一种，可变报头的报文标识符（ Packet Identifier） 字段存在于在多个类型的报文里。

报文标识符类型如下图:



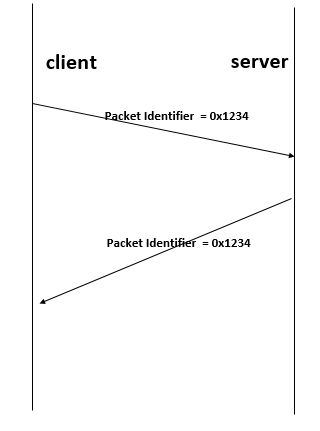
**图2.4 报文标识符**

很多控制报文的可变报头部分包含一个两字节的报文标识符字段。 这些报文是 PUBLISH（ QoS>0 时） ，PUBACK， PUBREC， PUBREL， PUBCOMP， SUBSCRIBE, SUBACK， UNSUBSCIBE，UNSUBACK，如下表所示：

|  |  |
| --- | --- |
| 控制报文 | 报文标识符字段 |
| CONNECT | 不需要 |
| CONNACK | 不需要 |
| PUBLISH | 需要（如果QoS > 0） |
| PUBACK | 需要 |
| PUBREC | 需要 |
| PUBREL | 需要 |
| PUBCOMP | 需要 |
| SUBSCRIBE | 需要 |
| SUBACK | 需要 |
| UNSUBSCRIBE | 需要 |
| UNSUBACK | 需要 |
| PINGREQ | 不需要 |
| PINGRESP | 不需要 |
| DISCONNECT | 不需要 |

客户端和服务端彼此独立地分配报文标识符。 因此，客户端服务端组合使用相同的报文标识符可以实现并发的消息交换。

例如，当client发送一个packet Identifier =0x1234的报文给server时，server的回复报文packet identifier 必须是0x1234,Packet identifier 从1开始递增，到达65535时，又从1开始计算。



**图 2.5 需要 Packet Identifier 的报文类型交互示意图**

## 2.7 有效载荷

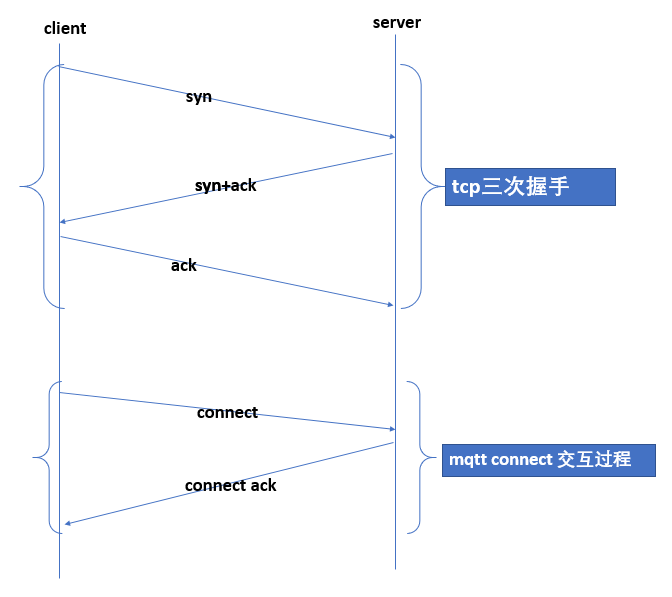
某些 MQTT 控制报文在报文的最后部分包含一个有效载荷，带有有效载荷报文类型如下表所示：

|  |  |
| --- | --- |
| 控制报文 | 有效载荷 |
| CONNECT | 需要 |
| CONNACK | 不需要 |
| PUBLISH | 可选 |
| PUBACK | 不需要 |
| PUBREC | 不需要 |
| PUBREL | 不需要 |
| PUBCOMP | 不需要 |
| SUBSCRIBE | 需要 |
| SUBACK | 需要 |
| UNSUBSCRIBE | 需要 |
| UNSUBACK | 不需要 |
| PINGREQ | 不需要 |
| PINGRESP | 不需要 |
| DISCONNECT | 不需要 |

# 第三章 报文分析

## 3.1 CONNECT-连接服务端

客户端到服务端的网络连接建立（完成三次握手）后，客户端发送给服务端的第一个报文必须是 CONNECT 报文。



**图3.1 三次握手与mqtt connect交互过程**

在一个网络连接上，客户端只能发送一次 CONNECT 报文。服务端必须将客户端发送的第二个 CONNECT报文当作协议违规处理并断开客户端的连接。

有效载荷包含一个或多个编码的字段。 包括客户端的唯一标识符， Will 主题， Will 消息， 用户名和密码。 除了客户端标识之外， 其它的字段都是可选的， 基于标志位来决定可变报头中是否需要包含这些字段。



**图3.2 connect报文组成**

### 3.1.1 connect固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | Mqtt报文类型(1) | | | | Reserved(保留位) | | | |
|  | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Byte2~n | 剩余长度 | | | | | | | |

**表格3.1**

### 3.1.2 协议名字节组成

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 说明 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 协议名 | | | | | | | | | |
| Byte1 | 协议名长度MSB(0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Byte2 | 协议名长度LSB(4) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Byte3 | ‘M’ | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| Byte4 | ‘Q’ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| Byte5 | ‘T’ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| Byte6 | ‘T’ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

数据包检测工具， 例如防火墙， 可以使用协议名来识别 MQTT 流量。

### 3.1.3 协议级别

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 说明 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 协议级别 | | | | | | | | | |
| Byte7 | Level(4) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

客户端用 8 位的无符号值表示协议的修订版本。对于 3.1.1 版协议，协议级别字段的值是 4(0x04)。如果发现不支持的协议级别，服务端必须给发送一个返回码为 0x01（不支持的协议级别）的CONNACK 报文响应CONNECT 报文， 然后断开客户端的连接。

### 3.1.4 连接标记

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | 用户名标记 | 用户密码标记 | Will  retain | Will qos | | Will flag | 清除会话 | reserved |
| Byte 8 | x | x | x | x | | x | x | 0 |

**bit1清除会话**

一般来说， 客户端连接时总是将清理会话标志设置为 0 或 1， 并且不交替使用两种值。 这个选择取决于具体的应用。 清理会话标志设置为 1 的客户端不会收到旧的应用消息， 而且在每次连接成功后都需要重新订阅任何相关的主题。清理会话标志设置为 0 的客户端会收到所有在它连接断开期间发布的 QoS 1 和 QoS 2 级别的消息。因此， 要确保不丢失连接断开期间的消息， 需要使用 QoS 1 或QoS 2 级别，同时将清理会话标志设置为 0。

**Bit2遗嘱标志**

遗嘱标志（Will Flag） 被设置为 1，表示如果连接请求被接受了， 遗嘱（Will Message） 消息必须被存储在服务端并且与这个网络连接关联。之后网络连接关闭时，服务端必须发布这个遗嘱消息， 除非服务端收到DISCONNECT 报文时删除了这个遗嘱消息。

**Bit3和 bit4遗嘱 QoS**

这两位用于指定发布遗嘱消息时使用的服务质量等级, 如果遗嘱标志被设置为 0， 遗嘱 QoS 也必须设置为 0(0x00)，如果遗嘱标志被设置为 1， 遗嘱 QoS 的值可以等于 0(0x00)， 1(0x01)， 2(0x02)， 它的值不能等于 3。

**Bit5遗嘱保留**

如果遗嘱消息被发布时需要保留，需要指定这一位的值, 如果遗嘱标志被设置为 0， 遗嘱保留（Will Retain） 标志也必须设置为 0 。  
如果遗嘱标志被设置为 1：  
• 如果遗嘱保留被设置为 0， 服务端必须将遗嘱消息当作非保留消息发布 。  
• 如果遗嘱保留被设置为 1， 服务端必须将遗嘱消息当作保留消息发布。

**Bit7 用户名标志**

如果用户名（User Name） 标志被设置为 0， 有效载荷中不能包含用户名字段。

如果用户名（User Name） 标志被设置为 1， 有效载荷中必须包含用户名字段。

**Bit6 用户名密码标记**

如果密码（Password） 标志被设置为 0， 有效载荷中不能包含密码字段 。  
如果密码（Password） 标志被设置为 1， 有效载荷中必须包含密码字段 。  
如果用户名标志被设置为 0， 密码标志也必须设置为 0 。

### 3.1.5 保持连接

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte9 | 保持连接 Keep Alive MSB | | | | | | | |
| Byte10 | 保持连接 Keep Alive LSB | | | | | | | |

1. 保持连接（Keep Alive） 是一个以秒为单位的时间间隔，表示为一个 16 位的字，它是指在客户端传输完成。
2. 一个控制报文的时刻到发送下一个报文的时刻， 两者之间允许空闲的最大时间间隔。 客户端负责保证控制。
3. 报文发送的时间间隔不超过保持连接的值。 如果没有任何其它的控制报文可以发送， 客户端必须发送一个PINGREQ 报文。
4. 不管保持连接的值是多少，客户端任何时候都可以发送 PINGREQ 报文，并且使用 PINGRESP 报文判断网络和服务端的活动状态。
5. 如果保持连接的值非零，并且服务端在一点五倍的保持连接时间内没有收到客户端的控制报文， 它必须断开客户端的网络连接， 认为网络连接已断开。
6. 客户端发送了 PINGREQ 报文之后， 如果在合理的时间内仍没有收到 PINGRESP 报文， 它应该关闭到服务端的网络连接。
7. 保持连接的值为零表示关闭保持连接功能。 这意味着，服务端不需要因为客户端不活跃而断开连接。 注意：不管保持连接的值是多少， 任何时候，只要服务端认为客户端是不活跃或无响应的， 可以断开客户端的连接。

### 3.1.6 客户端标识符

服务端使用客户端标识符 (ClientId) 识别客户端。 连接服务端的每个客户端都有唯一的客户端标识符（ClientId） 。客户端和服务端都必须使用 ClientId 识别两者之间的 MQTT 会话相关的状态,客户端标识符 (ClientId) 必须存在而且必须是 CONNECT 报文有效载荷的第一个字段,客户端标识符必须是UTF-8 编码字符串。

### 3.1.7 遗嘱主题

如果遗嘱标志被设置为 1， 有效载荷的下一个字段是遗嘱主题（Will Topic） 。 遗嘱主题必须是 UTF-8 编码字符串。

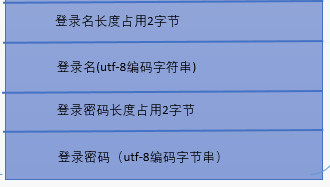
### 3.1.8 遗嘱消息

如果遗嘱标志被设置为 1， 有效载荷的下一个字段是遗嘱消息。 遗嘱消息定义了将被发布到遗嘱主题的应用消息。

### 3.1.9 用户名和密码

如果用户名（ User Name） 标志被设置为 1， 有效载荷的下一个字段就是它。 用户名必须是定义的UTF-8 编码字符串。服务端可以将它用于身份验证和授权。

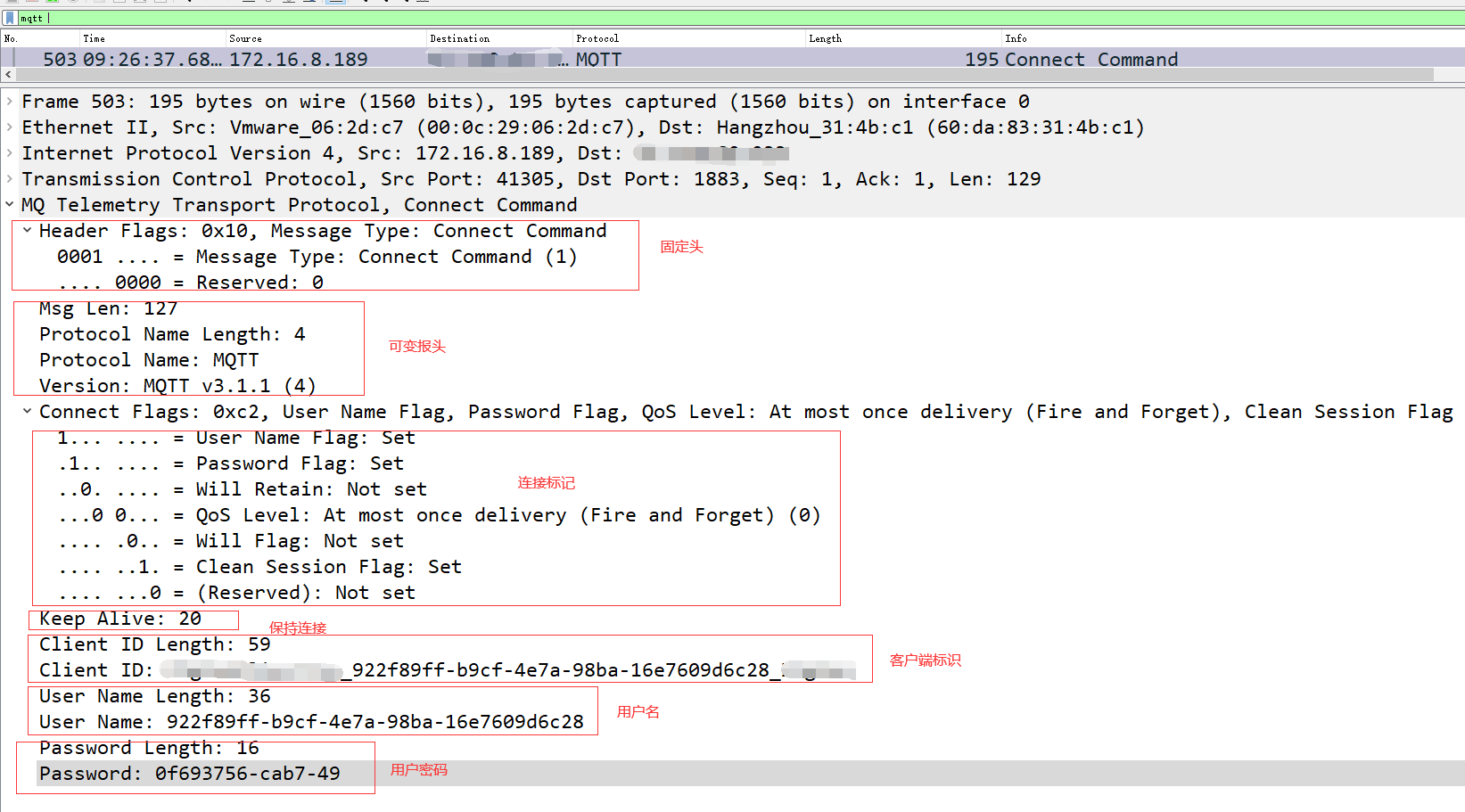
如果密码（ Password） 标志被设置为 1， 有效载荷的下一个字段就是它。密码字段包含一个两字节的长度字段， 长度表示二进制数据的字节数（ 不包含长度字段本身占用的两个字节），后面跟着 0 到 65535 字节的二进制数据。



**图3.2 用户名和密码在connect报文中的组成**

### 3.10.1 wirshark抓包分析connect报文

从抓包可知，从上到下分别是固定报头，可变报头，连接标记，保持连接，用户名，用名密码，其中没有遗嘱相关消息字段，与3.1.1节分析的固定报头组成分析一致。



**图 3.3使用wireshark抓包分析connect报文组成格式**

### 3.10.2 c语言构造mqtt connect报文

static uint8\_t client\_id[512] = {"mqtt\_client"};

static uint8\_t user\_name[512] = {"mqtt"};

static uint8\_t passwd[512] = {"12345678"};

#define KEEP\_ALIVE 20

int mqtt\_connect(int sockfd)

{

uint8 flags = 0x00;

uint8 \*packet = NULL;

uint16 packet\_length = 0;

uint16 clientidlen = strlen(client\_id);

uint16 usernamelen = strlen(user\_name);

uint16 passwordlen = strlen(passwd);

uint16 payload\_len = clientidlen + 2;

// Variable header

uint8 var\_header[10] = {

0x00,0x04,/\*len\*/

0x4d,0x51,0x54,0x54,/\*mqtt\*/

0x04,/\*协议版本\*/};

uint8 fixedHeaderSize = 2; // Default size = one byte Message Type + one byte Remaining Length

uint8 remainLen = 0;

uint8 \*fixed\_header = NULL;

uint16 offset = 0;

// Preparing the flags

if(usernamelen) { /\*用户名长度(可选)\*/

payload\_len += usernamelen + 2;

flags |= MQTT\_USERNAME\_FLAG;/\*或上用户名标记\*/

}

if(passwordlen) { /\*用户密码(可选)\*/

payload\_len += passwordlen + 2;

flags |= MQTT\_PASSWORD\_FLAG;/\*用户密码标记位\*/

}

flags |= MQTT\_CLEAN\_SESSION;

var\_header[7] = flags;/\*连接标记\*/

var\_header[8] = KEEP\_ALIVE>>8;/\*保持连接字段，占用两个字节\*/

var\_header[9] = KEEP\_ALIVE&0xFF;

remainLen = sizeof(var\_header)+payload\_len; /\*剩余长度，也就是可变报头加上负载的长度\*/

if (remainLen > 127) {

fixedHeaderSize++;// add an additional byte for Remaining Length

}

fixed\_header = (uint8 \*)malloc(fixedHeaderSize); /\*固定报头\*/

// Message Type

\*fixed\_header = MQTT\_MSG\_CONNECT;/\*报文类型,connect\*/

if (remainLen <= 127) {// Remaining Length,剩余长度计算，可变长编码

\*(fixed\_header+1) = remainLen;

} else {

// first byte is remainder (mod) of 128, then set the MSB to indicate more bytes

\*(fixed\_header+1) = remainLen % 128;

\*(fixed\_header+1) = \*(fixed\_header+1) | 0x80;

// second byte is number of 128s

\*(fixed\_header+2) = remainLen / 128;

}

packet\_length = fixedHeaderSize+sizeof(var\_header)+payload\_len;/\*固定报头+可变报头+负载长度\*/

packet = (uint8 \*)malloc(packet\_length);/\*分配内存\*/

memset(packet, 0, packet\_length);

memcpy(packet, fixed\_header, fixedHeaderSize);/\*填充固定报头\*/

free(fixed\_header);

offset += fixedHeaderSize;

memcpy(packet+offset, var\_header, sizeof(var\_header));/\*填充可变报头\*/

offset += sizeof(var\_header);

packet[offset++] = clientidlen>>8;// Client ID - UTF encoded,填充clientid长度+clientid

packet[offset++] = clientidlen&0xFF;

memcpy(packet+offset, client\_id, clientidlen);

offset += clientidlen;

if(usernamelen) {// Username - UTF encoded,填充用户名+用户名长度

packet[offset++] = usernamelen>>8;

packet[offset++] = usernamelen&0xFF;

memcpy(packet+offset, user\_name, usernamelen);

offset += usernamelen;

}

if(passwordlen) {// Password - UTF encoded,填充用户密码+用户名密码长度

packet[offset++] = passwordlen>>8;

packet[offset++] = passwordlen&0xFF;

memcpy(packet+offset, passwd, passwordlen);

offset += passwordlen;

}

// Send the packet

if (client\_send(sockfd,packet, packet\_length) < 0){

free(packet);

return -1;

}

free(packet);

return 1;

}

## 3.2 CONNACK-确认连接请求

服务端发送 CONNACK 报文响应从客户端收到的 CONNECT 报文。服务端发送给客户端的第一个报文必须是 CONNACK。

### 3.2.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | MQTT 控制报文类型 (2) | | | | Reserved 保留位 | | | |
|  | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度 | | | | | | | |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

剩余长度字段表示可变报头的长度。 对于 CONNACK 报文这个值等于 2。

### 3.2.2 可变报头

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 描述 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 连接确认标记 | | 保留位 | | | | | | | SP1 |
| Byte1 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X |
| 连接返回码 | | | | | | | | | |
| Btte2 |  | x | x | x | x | x | x | x | x |

**Byte1,Bit0连接确认标志**

位 7-1 是保留位且必须设置为 0,

对于bit0,如果服务端收到一个 CleanSession 为 0 的连接， 当前会话标志的值取决于服务端是否已经保存了 ClientId对应客户端的会话状态。 如果服务端已经保存了会话状态， 它必须将 CONNACK 报文中的当前会话标志设置为 1 。 如果服务端没有已保存的会话状态， 它必须将 CONNACK 报文中的当前会话设置为 0。 还需要将 CONNACK 报文中的返回码设置为 0。

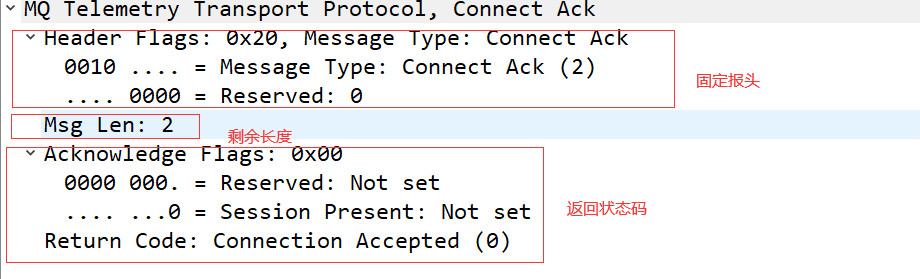
**连接返回码**

如果服务端发送了一个包含非零返回码的 CONNACK 报文， 那么它必须关  
闭网络连接。

|  |  |  |
| --- | --- | --- |
| 值 | 返回码响应 | 描述 |
| 0 | 0x00 | 连接已被服务端接受 |
| 1 | 0x01 | 服务端不支持客户端请求的协议版本 |
| 2 | 0x02 | 客户端标识符是正确的 UTF-8 编码， 但服务 端不允许使用 |
| 3 | 0x03 | 网络连接已建立， 但 MQTT 服务不可用 |
| 4 | 0x04 | 用户名或密码的数据格式无效 |
| 5 | 0x05 | 客户端未被授权连接到此服务器 |
| 6-255 |  | 保留 |

CONNACK没有有效载荷。

### 3.2.3 CONNACK报文wireshark抓包分析



**图3.4 CONNACK 抓包报文**

### 3.2.4 c语言构造connect ack报文

void mqtt\_connect\_ack(int sockfd)

{

uint8\_t cmd[]={ 0x20/\*报文类型\*/, 0x02/\*剩余长度\*/ ,0x00,0x00/\*最后两个字节可变报头表示返回状态码\*/ };

send\_msg(sockfd,cmd,sizeof(cmd));

socket\_record\_t \*socket\_record = look\_up\_by\_sokfd(sockfd);

if(socket\_record==NULL){

return;

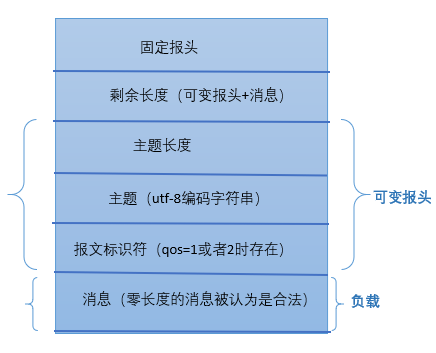
}

socket\_record->is\_connect=0x01;

}

## 3.3 PUBLISH-发布消息

PUBLISH 控制报文是指从客户端向服务端或者服务端向客户端传输一个应用消息。



**图 3.5 publish报文组成格式**

### 3.3.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（3） | | | | dup | Qos等级 | | RETAIN |
|  | 0 | 0 | 1 | 1 | x | x | x | x |
| Byte2 | 剩余长度 | | | | | | | |

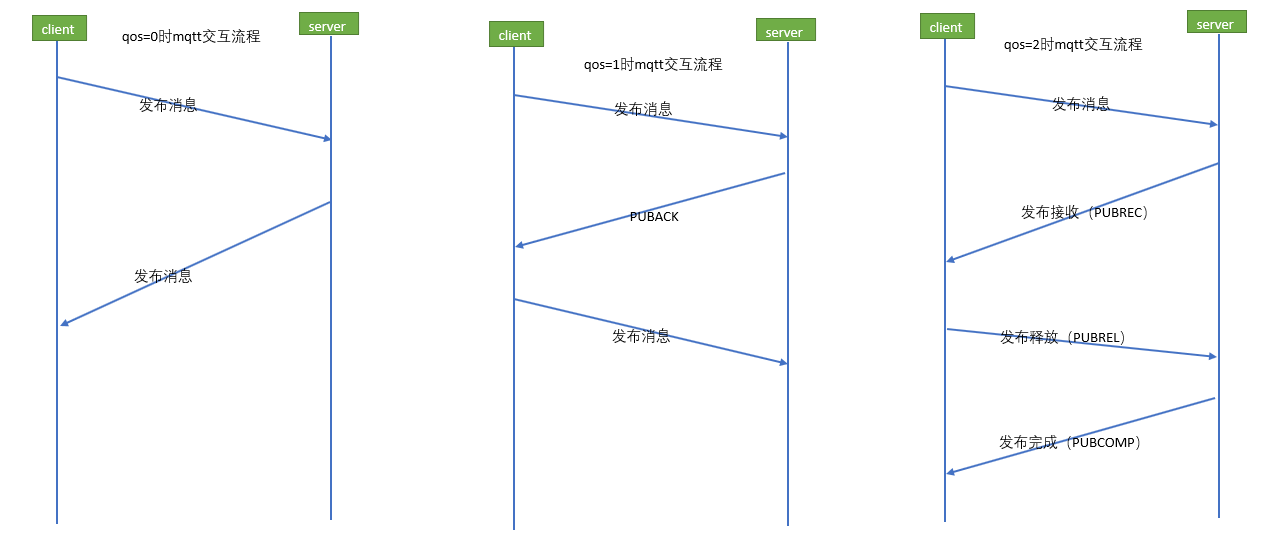
**Bit3 dup**

如果 DUP 标志被设置为 0， 表示这是客户端或服务端第一次请求发送这个 PUBLISH 报文。 如果 DUP 标志被设置为 1，表示这可能是一个早前报文请求的重发。客户端或服务端请求重发一个 PUBLISH 报文时， 必须将 DUP 标志设置为 1.。 对于 QoS0 的消息， DUP 标志必须设置为 0。

**Bit1和bit2 qos等级**

|  |  |  |  |
| --- | --- | --- | --- |
| Qos值 | bit2 | bit1 | 描述 |
| 0 | **0** | **0** | **最多分发一次** |
| 1 | **0** | **1** | **至少分发一次** |
| 2 | **1** | **0** | **只分发一次** |
| - | **1** | **1** | **保留不使用** |

qos由发送端决定，发送端发送什么qos的消息，接收端就回复什么qos的消息。



**不同qos等级mqtt报文交互流程**

**Bit0 保留标记位**

一般设置为0。

**剩余长度**

等于可变报头的长度加上有效载荷的长度。

**可变报头**

可变报头按顺序包含主题名和标识符。主题，用于识别有效载荷数据应该被发布到哪一个信息通道，标识符，只有当 QoS 等级是 1 或 2 时，报文标识符（ Packet Identifier） 字段才能出现在 PUBLISH 报文中。

### 3.3.2 抓包分析PUBLISH报文



**图 3.6 PUBLISH 抓包报文**

### 3.3.3 构造publish 报文

int mqtt\_publish\_with\_qos(int sockfd,const char\* topic, const char\* msg, uint16 msgl, uint8 retain, uint8 qos, uint16\* message\_id)

{

socket\_record\_t \*socket\_record = look\_up\_by\_sokfd(sockfd);

if(NULL == socket\_record){

return -1;

}

DEBUG\_INFO("sockfd:%d",socket\_record->sockfd);

uint16 topiclen = strlen(topic);

uint16 msglen = msgl;

uint8 \*var\_header = NULL; // Topic size (2 bytes), utf-encoded topic

uint8 \*fixed\_header = NULL;

uint8 fixedHeaderSize = 0,var\_headerSize = 0; // Default size = one byte Message Type + one byte Remaining Length

uint16 remainLen = 0;

uint8 \*packet = NULL;

uint16 packet\_length = 0;

uint8 qos\_flag = MQTT\_QOS0\_FLAG; /\*qos标记\*/

uint8 qos\_size = 0; // No QoS included

if(qos == 1) {

qos\_size = 2; // 2 bytes for QoS

qos\_flag = MQTT\_QOS1\_FLAG;

}

else if(qos == 2) {

qos\_size = 2; // 2 bytes for QoS

qos\_flag = MQTT\_QOS2\_FLAG;

}

// Variable header

var\_headerSize = topiclen/\*主题内容\*/+2/\*主题长度占用两字节\*/+qos\_size/\*标识符\*/;

var\_header = (uint8 \*)malloc(var\_headerSize);

memset(var\_header, 0, var\_headerSize);

\*var\_header = topiclen>>8;

\*(var\_header+1) = topiclen&0xFF;

memcpy(var\_header+2, topic, topiclen);

if(qos\_size) {//qos1和qos2的报文需要填充标识符,有点像tcp的seq

socket\_record->publish\_seq++;

if(socket\_record->publish\_seq == 0){

//unsigned short 表示范围0~65535,标识符必须是非零整数

socket\_record->publish\_seq = 1;

}

var\_header[topiclen+2] = (socket\_record->publish\_seq & 0xff00)>>8;

var\_header[topiclen+3] = socket\_record->publish\_seq & 0x00ff;

if(message\_id) {

\*message\_id = socket\_record->publish\_seq;

}

}

fixedHeaderSize = 2; // Default size = one byte Message Type + one byte Remaining Length

remainLen = var\_headerSize+msglen;

if (remainLen > 127) {/\*剩余长度\*/

fixedHeaderSize++; // add an additional byte for Remaining Length

}

fixed\_header = (uint8 \*)malloc(fixedHeaderSize);/\*固定报头+剩余长度\*/

// Message Type, DUP flag, QoS level, Retain

\*fixed\_header = MQTT\_MSG\_PUBLISH | qos\_flag;/\*报文类型和qos标记\*/

if(retain) {

\*fixed\_header |= MQTT\_RETAIN\_FLAG;/\*是否保留\*/

}

// Remaining Length，剩余长度

if (remainLen <= 127) {

\*(fixed\_header+1) = remainLen;

} else {

// first byte is remainder (mod) of 128, then set the MSB to indicate more bytes

\*(fixed\_header+1) = remainLen % 128;

\*(fixed\_header+1) = \*(fixed\_header+1) | 0x80;

// second byte is number of 128s

\*(fixed\_header+2) = remainLen / 128;

}

packet\_length = fixedHeaderSize+var\_headerSize+msglen;/\*固定报头+可变报头+负载长度\*/

packet = (uint8 \*)malloc(packet\_length);

memset(packet, 0, packet\_length);

memcpy(packet, fixed\_header, fixedHeaderSize);/\*填充固定报头\*/

memcpy(packet+fixedHeaderSize, var\_header, var\_headerSize);/\*填充可变报头\*/

memcpy(packet+fixedHeaderSize+var\_headerSize, msg, msglen);/\*负载\*/

free(var\_header);

free(fixed\_header);

send\_msg(sockfd,packet , packet\_length);

free(packet);

return 1;

}

## 3.4 PUBREC-发布收到

PUBREC 报文是对 QoS 等级 2 的 PUBLISH 报文的响应。它是 QoS 2 等级协议交换的第二个报文。

### 4.3.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（5） | | | | 保留位 | | | |
|  | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

**剩余长度**

表示可变报头的长度。 对 PUBREC 报文它的值等于 2。

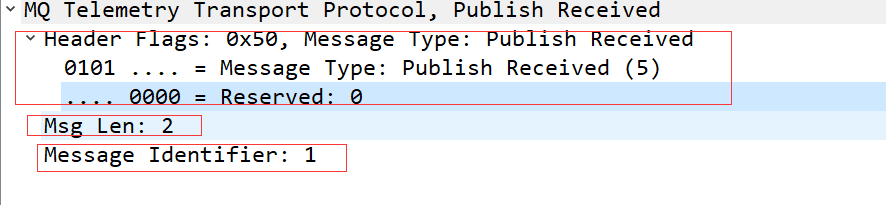
**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

**有效载荷**

PUBREC 报文没有有效载荷。

### 4.3.2 PUBREC抓包报文



**图 3.7 PUBREC抓包报文图示**

### 4.3.3 c语言构造pubrec报文

//如果是PUBREC报文，head\_type=0x50

void mqtt\_qos2\_pubrec(int sockfd , unsigned char \*data,unsigned char head\_type)

{

uint16 msg\_id = mqtt\_parse\_msg\_id(data);/\*报文标识符,回复报文和接受报文的标识符必须一样\*/

unsigned char qos2\_pubrec\_respon[]={head\_type/\*固定报头\*/,0x02/\*剩余长度\*/, (msg\_id&0xff00)>>8 , msg\_id&0x00ff/\*最后两个字节是报文标识符\*/};

send\_msg(sockfd,qos2\_pubrec\_respon,sizeof(qos2\_pubrec\_respon));

}

## 3.5 PUBREL-发布释放

PUBREL 报文是对 PUBREC 报文的响应。 它是 QoS 2 等级协议交换的第三个报文。

### 3.5.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（6） | | | | 保留位 | | | |
|  | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

**剩余长度**

表示可变报头的长度。 对 PUBREL 报文它的值等于 2。

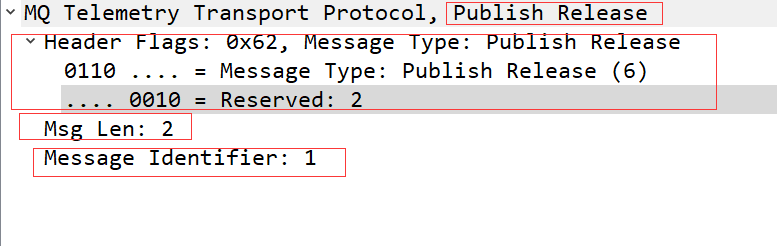
**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

**有效载荷**

PUBREL 报文没有有效载荷。

### 3.5.2 PUBREL抓包报文

****

**图 3.8 PUBREL抓包报文图示**

### 3.5.3 c语言构造pubrel报文

//head\_type=0x62

void mqtt\_qos2\_pubrel(int sockfd , unsigned char \*data,unsigned char head\_type)

{

uint16 msg\_id = mqtt\_parse\_msg\_id(data);

unsigned char qos2\_pubrel\_respon[]={head\_type/\*报文类型\*/,0x02/\*剩余长度\*/, (msg\_id & 0xff00)>>8 , msg\_id & 0x00ff/\*最后两个字节是报文标识符\*/};

send\_msg(sockfd,qos2\_pubrel\_respon,sizeof(qos2\_pubrel\_respon));

}

## 3.6 PUBCOMP-发布完成

PUBCOMP 报文是对 PUBREL 报文的响应。 它是 QoS 2 等级协议交换的第四个也是最后一个报文。

### 3.6.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（7） | | | | 保留位 | | | |
|  | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

**剩余长度**

表示可变报头的长度。 对 PUBCOMP 报文它的值等于 2。

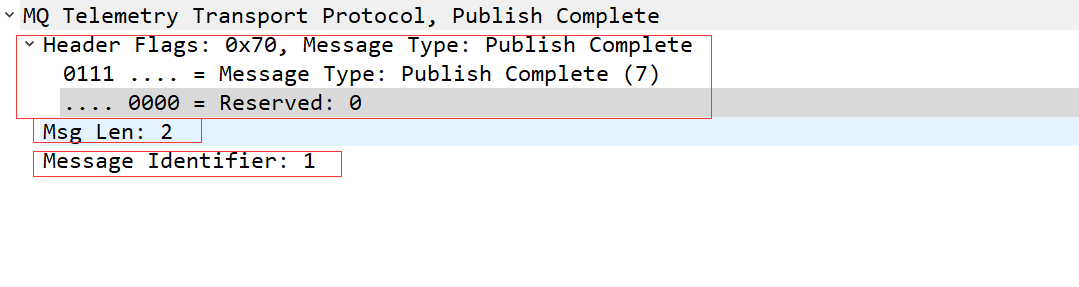
**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

**有效载荷**

PUBCOMP 报文没有有效载荷。

### 3.6.2 PUBCOMP抓包报文



**图 3.9 PUBCOMP抓包报文图示**

### 3.6.3 c语言构造pubcom报文

//head\_type=0x70

void mqtt\_qos2\_pubcomp(int sockfd , unsigned char \*data,unsigned char head\_type)

{

uint16 msg\_id = mqtt\_parse\_msg\_id(data);/\*报文标识符\*/

unsigned char qos2\_pubcomp\_respon[]={head\_type/\*报文类型\*/,0x02/\*剩余长度\*/, (msg\_id & 0xff00)>>8 , msg\_id & 0x00ff/\*最后两个字节报文标识符\*/};

send\_msg(sockfd,qos2\_pubcomp\_respon,sizeof(qos2\_pubcomp\_respon));

}

## 3.7 PINGREQ-心跳请求

客户端发送 PINGREQ 报文给服务端的。用于：

1. 在没有任何其它控制报文从客户端发给服务的时，告知服务端客户端还活着。
2. 请求服务端发送 响应确认它还活着。
3. 使用网络以确认网络连接没有断开。

### 3.7.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（12） | | | | 保留位 | | | |
|  | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度0 | | | | | | | |

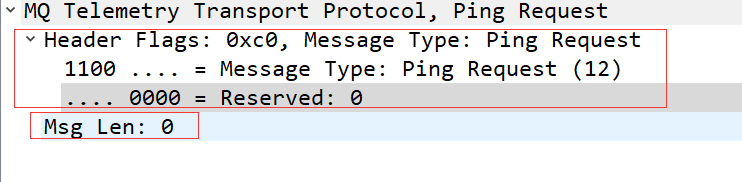
**可变报头**

报文没有可变报头。

**有效载荷**

PINGREQ 报文没有有效载荷。

### 3.7.2 PINGREQ 抓包报文



**图 3.10 PINGREQ抓包报文图示**

### 3.7.3 c语言构造pingreq报文

int mqtt\_ping(int sockfd)

{

uint8 packet[] = {MQTT\_MSG\_PINGREQ/\*报文类型\*/,0x00/\*剩余长度\*/};

int ret = send\_msg(sockfd,packet, sizeof(packet));

return ret;

}

## 3.8 PINGRESP – 心跳响应

服务端发送 PINGRESP 报文响应客户端的 PINGREQ 报文。 表示服务端还活着。

### 3.8.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（13） | | | | 保留位 | | | |
|  | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度0 | | | | | | | |

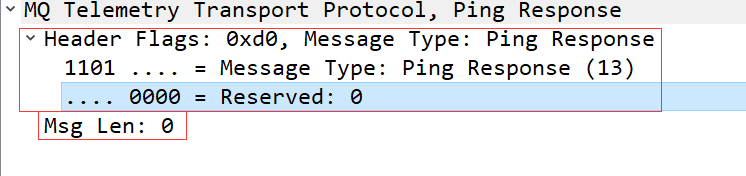
**可变报头**

报文没有可变报头。

**有效载荷**

PINGRESP 报文没有有效载荷。

### 3.8.2 PINGRESP 抓包报文



**图 3.11** PINGRESP**抓包报文图示**

## 3.8.3 c语言构造pinpresp报文

void mqtt\_ping\_req\_reply(int sockfd)

{

uint8\_t cmd[]={0xd0/\*报文类型\*/, 0x00/\*剩余长度\*/};

send\_msg(sockfd,cmd,sizeof(cmd));

}

## 3.9 DISCONNECT –断开连接

DISCONNECT 报文是客户端发给服务端的最后一个控制报文。表示客户端正常断开连接。

### 3.9.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（14） | | | | 保留位 | | | |
|  | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度0 | | | | | | | |

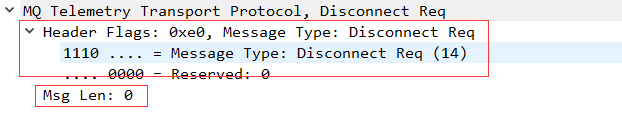
**可变报头**

DISCONNECT报文没有可变报头。

**有效载荷**

DISCONNECT 报文没有有效载荷。

### 3.9.2 DISCONNECT 抓包报文



**图 3.12** DISCONNECT**抓包报文图示**

### 3.9.3 c语言构造disconnect报文

int mqtt\_disconnect(int sockfd)

{

uint8 packet[] = {MQTT\_MSG\_DISCONNECT/\*报文类型\*/,0x00/\*剩余长度\*/};

int ret = client\_send(sockfd,packet, sizeof(packet));

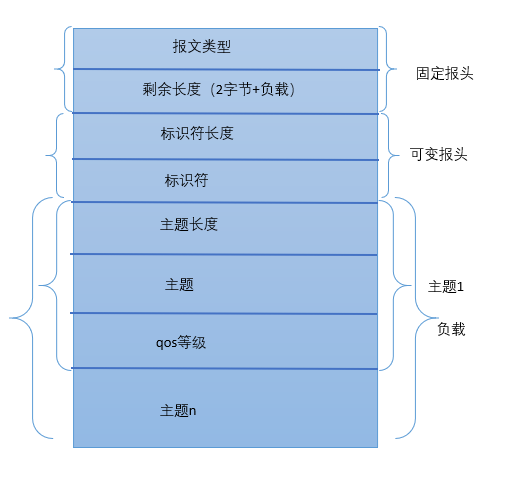
DEBUG\_INFO("ret=%d",ret);

return ret;

}

## 3.10 SUBSCRIBE-订阅主题

客户端向服务端发送 SUBSCRIBE 报文用于创建一个或多个订阅。 每个订阅注册客户端关心的一个或多个主题。 为了将应用消息转发给与那些订阅匹配的主题， 服务端发送 PUBLISH 报文给客户端。 SUBSCRIBE报文也（为每个订阅）指定了最大的 QoS 等级， 服务端根据这个发送应用消息给客户端。



**图 3.13 订阅主题报文组成格式**

### 3.10.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（8） | | | | 保留位 | | | |
|  | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

SUBSCRIBE 控制报固定报头的第 3,2,1,0 位是保留位， 必须分别设置为 0,0,1,0,服务端必须将其它的任何值都当做是不合法的并关闭网络连接。

**剩余长度字段**  
等于可变报头的长度（ 2 字节） 加上有效载荷的长度。

**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

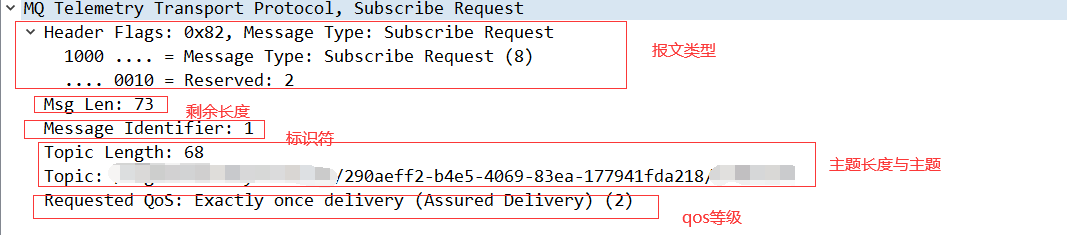
服务端收到客户端发送的一个 SUBSCRIBE 报文时， 必须使用 SUBACK 报文响应。SUBACK 报文必须和等待确认的 SUBSCRIBE 报文有相同的报文标识符。

**有效载荷**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 主题 | | | | | | | | |
| Byte1 | **主题长度MSB** | | | | | | | |
| Byte2 | **主题长度LSB** | | | | | | | |
| Byte3~n | **主题** | | | | | | | |
| 服务质量 | | | | | | | | |
|  | **保留** | | | | | | **qos等级** | |
| ByteN+1 | **0** | **0** | **0** | **0** | **0** | **0** | **x** | **x** |

QoS 不等于 0,1 或 2， 服务端必须认为 SUBSCRIBE 报文是不合法的并关闭网络连接。

### 3.10.2 SUBSCRIBE报文抓包



**图 3.14 订阅主题抓包报文**

### 3.10.3 c语言构造subscribe报文

static uint16 su\_seq = 1;

int mqtt\_subscribe\_theme(int sockfd,char \*Theme , uint8\_t Qos)

{

su\_seq++;//报文标识符

if(su\_seq == 0){

su\_seq = 1;

}

uint16\_t MessageId = su\_seq;

uint8\_t cmd[1024]={0};

//报文标示符长度2 + 主题长度位占用2字节+主题内容+qos标识

int data\_length = 2+2+strlen(Theme)+1;

int playload\_len = strlen(Theme);

uint8\_t len\_byte[4] ={0x00 , 0x00 ,0x00 ,0x00};

uint8\_t byte\_num = length\_trans\_byte\_form(data\_length , len\_byte);/\*把剩余长度转换成变长编码\*/

cmd[0] = 0x82;

memcpy(&cmd[1] , len\_byte , byte\_num);

cmd[1+byte\_num]=(MessageId & 0xff00) >> 8 ;

cmd[1+byte\_num+1] = MessageId & 0x00ff;

cmd[1+byte\_num+1+1] = (playload\_len & 0xff00) >> 8;

cmd[1+byte\_num+1+1+1] = playload\_len & 0x00ff;

memcpy(&cmd[1+byte\_num+1+1+1+1] , Theme , playload\_len);

cmd[1+byte\_num+1+1+1+1+playload\_len] = Qos;

client\_send(sockfd,cmd, 1+byte\_num+1+1+1+1+playload\_len+1);

}

## 3.11 SUBACK –订阅确认

服务端发送 SUBACK 报文给客户端，用于确认它已收到并且正在处理 SUBSCRIBE 报文。

### 3.11.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（9） | | | | 保留位 | | | |
|  | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

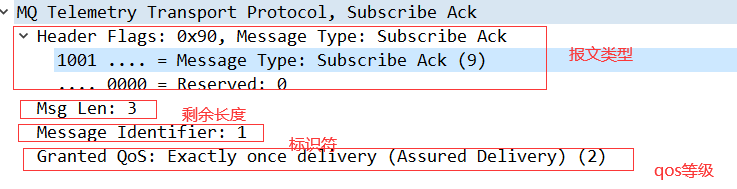
**剩余长度字段**  
等于可变报头的长度加上有效载荷的长度。

**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

可变报头包含等待确认的 SUBSCRIBE 报文的报文标识符。

### 3.11.2 SUBACK抓包报文



**图 3.15订阅主题ack抓包报文**

### 3.11.3 c语言构造suback报文

void mqtt\_subscribe\_ack(int sockfd,const uint8 \*buf)

{

uint16 msg\_id = mqtt\_parse\_msg\_id(buf);/\*提取报文标识符\*/

uint8 qos = MQTTParseMessageQos(buf);/\*提取报文qos\*/

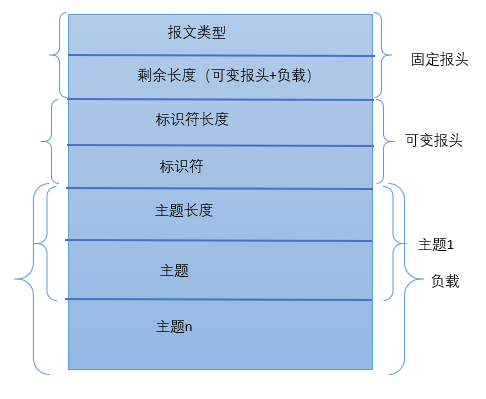
uint8 cmd[]={0x90,0x03/\*剩余长度\*/, (msg\_id & 0xff00) >> 8, msg\_id & 0x00ff,qos};

send\_msg(sockfd,cmd,sizeof(cmd));

}

## 3.12 UNSUBSCRIBE –取消订阅

客户端发送 UNSUBSCRIBE 报文给服务端， 用于取消订阅主题。



**图 3.16取消订阅主题报文结构**

### 3.12.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（10） | | | | 保留位 | | | |
|  | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

UNSUBSCRIBE 报文固定报头的第 3,2,1,0 位是保留位且必须分别设置为 0,0,1,0。 服务端必须认为任何其它的值都是不合法的并关闭网络连接。

**剩余长度字段**  
等于可变报头的长度，加上有效载荷的长度。

**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

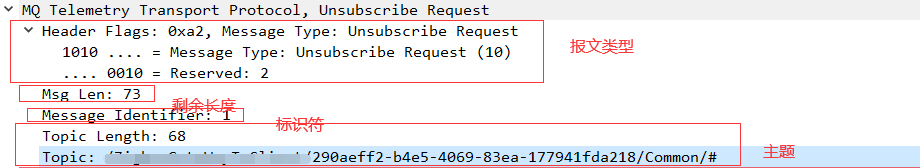
可变报头包含一个报文标识符。

**有效载荷**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 主题1 | | | | | | | | |
| Byte1 | **主题长度MSB** | | | | | | | |
| Byte2 | **主题长度LSB** | | | | | | | |
| Byte3~n | **主题** | | | | | | | |
| 主题2 | | | | | | | | |

UNSUBSCRIBE 报文的有效载荷必须至少包含一个消息过滤器。 没有有效载荷的 UNSUBSCRIBE 报文是违反协议的。

### 3.12.2 UNSUBSCRIBE抓包报文



**图 3.17取消订阅主题抓包报文**

### 3.12.3 c语言构造unsubscribe报文

static uint16 un\_seq = 1;

int mqtt\_unsubscribe\_theme(int sockfd,const char\* topic)

{

un\_seq++;

if(un\_seq == 0){

un\_seq = 1;

}

uint16\_t MessageId = un\_seq;

uint8\_t cmd[1024]={0};

//报文标示符长度2 + 主题长度位占用2字节+主题内容+qos标识

int data\_length = 2+2+strlen(topic)+1;

int playload\_len = strlen(topic);

uint8\_t len\_byte[4] ={0x00 , 0x00 ,0x00 ,0x00};

uint8\_t byte\_num = length\_trans\_byte\_form(data\_length , len\_byte);/\*剩余长度转换成变长编码\*/

cmd[0] = 0xa2;

memcpy(&cmd[1] , len\_byte , byte\_num);

cmd[1+byte\_num]=(MessageId & 0xff00) >> 8 ;

cmd[1+byte\_num+1] = MessageId & 0x00ff;

cmd[1+byte\_num+1+1] = (playload\_len & 0xff00) >> 8;

cmd[1+byte\_num+1+1+1] = playload\_len & 0x00ff;

memcpy(&cmd[1+byte\_num+1+1+1+1] , topic , playload\_len);

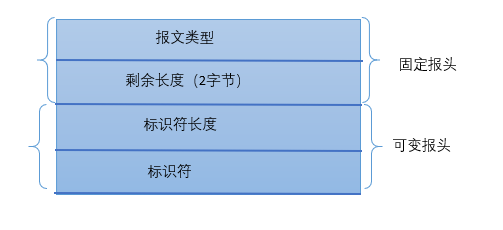
client\_send(sockfd,cmd,1+byte\_num+1+1+1+1+playload\_len+1);

return 1;

}

## 3.13 UNSUBACK –取消订阅确认

服务端发送 UNSUBACK 报文给客户端用于确认收到 UNSUBSCRIBE 报文。



**图 3.18取消订阅主题ack报文组成**

### 3.13.1 固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte 1 | MQTT报文类型（11） | | | | 保留位 | | | |
|  | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Byte2 | 剩余长度 | | | | | | | |

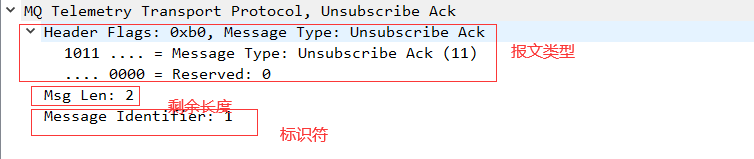
**剩余长度字段**  
表示可变报头的长度， 对 UNSUBACK 报文这个值等于 2。

**可变报头**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Byte1 | **报文标识符MSB** | | | | | | | |
| Byte2 | **报文标识符LSB** | | | | | | | |

可变报头包含等待确认的 UNSUBSCRIBE 报文的报文标识符。

### 3.12.2 UNSUBSCRIBE ACK抓包报文



**图 3.19取消订阅主题ACK抓包报文**

### 3.12.3 c语言构造unsubscribe报文

void mqtt\_unsubscribe\_ack(int sockfd,const uint8 \*buf)

{

uint16 msg\_id = mqtt\_parse\_msg\_id(buf);

uint8 cmd[]={0xb0,0x02/\*剩余长度\*/,(msg\_id & 0xff00) >> 8, msg\_id & 0x00ff/\*最后两个字节是报文标识符\*/};

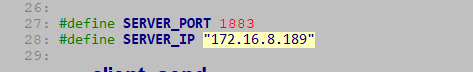
send\_msg(sockfd,cmd,sizeof(cmd));

}

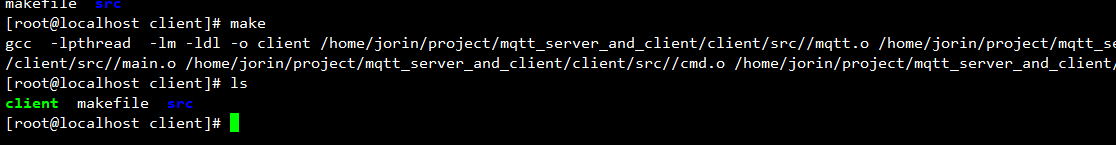
## 3.14 服务端与客户端交互操作过程

### 3.14.1 编译

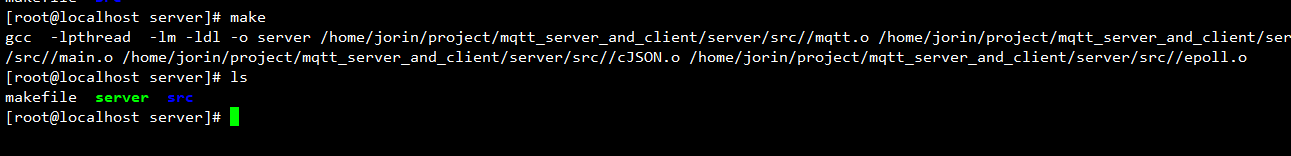
编译client之前先在代码中指定server ip



进入client目录，直接make即可

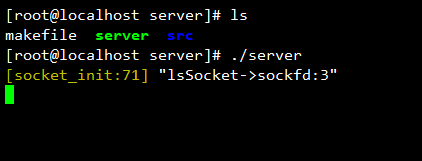


进入server目录，直接make即可

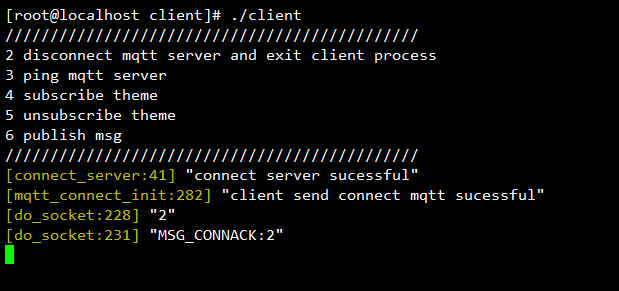


### 3.14.2 执行

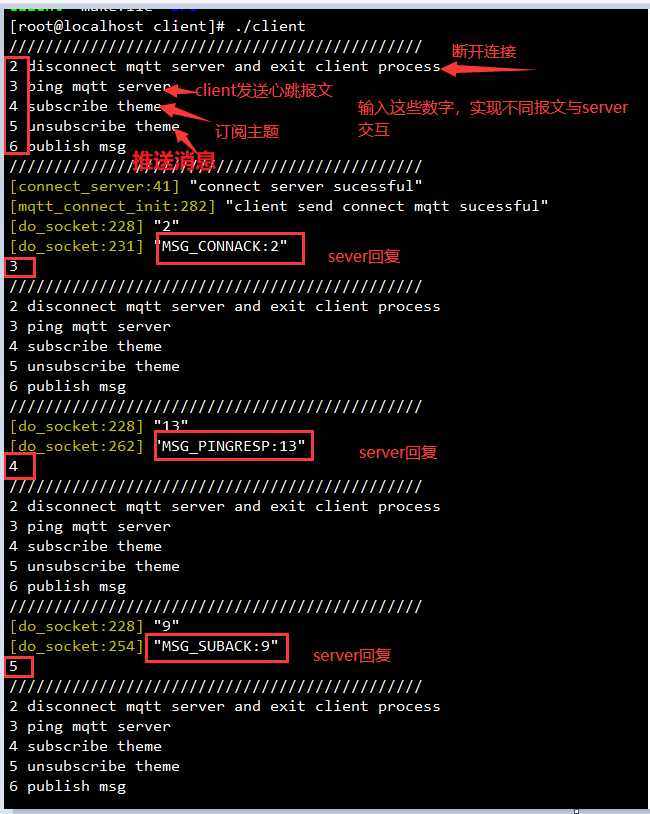
先运行server



再运行client



Client操作流程



在server端查看

