作者：

* International Business Machines Corporation (IBM)
* Eurotech

翻译：明歌

协议原文： [http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html](http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#fixed-header)

**摘要**

MQ遥测传输（MQ Telemetry Transport，MQTT）是一个轻量级的基于代理的发布/订阅式消息传输协议，它的设计目标是开放、简单、轻量和易于实现。这些特征使它适用于各种受限环境，比如，但不限于：

* 网络代价昂贵，低带宽或不可靠。
* 在嵌入设备中运行，处理器和内存资源有限。

该协议的特性包括：

* 使用发布/订阅消息模式，提供一对多的消息分发，解除应用程序耦合。
* 消息传输对有效载荷内容不可知。
* 使用TCP/IP提供基础网络连接。
* 有3个消息发布服务质量级别：
  + “至多一次”，消息发布完全依赖于底层TCP/IP网络。消息有可能丢失或重复。这一级别可应用于如下情景，如环境传感器数据，丢失一次读记录无所谓，因为很快下一次读记录就会产生。
  + “至少一次”，确保消息到达，但消息重复有可能发生。
  + “只有一次”，确保消息到达且只到达一次。这一级别可用于如计费系统等场景，在计费系统中，消息丢失或重复可能会导致生成错误的费用。
* 轻量传输，开销很小（固定头部的长度只有2字节），协议交换最小化，以降低网络流量。
* 提供一种机制，当客户端异常中断时，利用 Last Will 和 Testament 特性来通知有关各方。

**版权声明**

© 1999-2010 Eurotech, International Business Machines Corporation (IBM). All rights reserved.

Permission to copy and display the MQ Telemetry Transport specification (the "Specification"), in any medium without fee or royalty is hereby granted by Eurotech and International Business Machines Corporation (IBM) (collectively, the "Authors"), provided that you include the following on ALL copies of the Specification, or portions thereof, that you make:

A link or URL to the Specification at one of the Authors' websites. The copyright notice as shown in the Specification. The Authors each agree to grant you a royalty-free license, under reasonable, non-discriminatory terms and conditions to their respective patents that they deem necessary to implement the Specification. THE SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE SPECIFICATION.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without specific, written prior permission. Title to copyright in the Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.

**目录**

1. [简介](http://joyexpr.com/blog/2013/11/27/mqtt/#intro)
   * 1.1 [V3.1与V3之间的变化](http://joyexpr.com/blog/2013/11/27/mqtt/#new)
2. [消息格式](http://joyexpr.com/blog/2013/11/27/mqtt/#msg-format)
   * 2.1 [固定头部](http://joyexpr.com/blog/2013/11/27/mqtt/#fixed-header)
   * 2.2 [可变头部](http://joyexpr.com/blog/2013/11/27/mqtt/#variable-header)
   * 2.3 [有效载荷](http://joyexpr.com/blog/2013/11/27/mqtt/#payload)
   * 2.4 [消息标识符](http://joyexpr.com/blog/2013/11/27/mqtt/#msg-id)
   * 2.5 [MQTT和UTF-8](http://joyexpr.com/blog/2013/11/27/mqtt/#utf-8)
   * 2.6 [未使用的位](http://joyexpr.com/blog/2013/11/27/mqtt/#unused-bits)
3. [命令消息](http://joyexpr.com/blog/2013/11/27/mqtt/#commands)
   * 3.1 [CONNECT](http://joyexpr.com/blog/2013/11/27/mqtt/#connect)
   * 3.2 [CONNACK](http://joyexpr.com/blog/2013/11/27/mqtt/#connack)
   * 3.3 [PUBLISH](http://joyexpr.com/blog/2013/11/27/mqtt/#publish)
   * 3.4 [PUBACK](http://joyexpr.com/blog/2013/11/27/mqtt/#puback)
   * 3.5 [PUBREC](http://joyexpr.com/blog/2013/11/27/mqtt/#pubrec)
   * 3.6 [PUBREL](http://joyexpr.com/blog/2013/11/27/mqtt/#pubrel)
   * 3.7 [PUBCOMP](http://joyexpr.com/blog/2013/11/27/mqtt/#pubcomp)
   * 3.8 [SUBSCRIBE](http://joyexpr.com/blog/2013/11/27/mqtt/#subscribe)
   * 3.9 [SUBACK](http://joyexpr.com/blog/2013/11/27/mqtt/#suback)
   * 3.10 [UNSUBSCRIBE](http://joyexpr.com/blog/2013/11/27/mqtt/#unsubscribe)
   * 3.11 [UNSUBACK](http://joyexpr.com/blog/2013/11/27/mqtt/#unsuback)
   * 3.12 [PINGREQ](http://joyexpr.com/blog/2013/11/27/mqtt/#pingreq)
   * 3.13 [PINGRESP](http://joyexpr.com/blog/2013/11/27/mqtt/#pingresp)
   * 3.14 [DISCONNECT](http://joyexpr.com/blog/2013/11/27/mqtt/#disconnect)
4. [消息流](http://joyexpr.com/blog/2013/11/27/mqtt/#flows)
   * 4.1 [交付质量级别和消息流](http://joyexpr.com/blog/2013/11/27/mqtt/#qos-flows)
   * 4.2 [消息重传](http://joyexpr.com/blog/2013/11/27/mqtt/#retry)
   * 4.3 [消息排序](http://joyexpr.com/blog/2013/11/27/mqtt/#ordering)
5. [附录A 主题通配符](http://joyexpr.com/blog/2013/11/27/mqtt/#appendix-a)

**1. 简介**

该协议规范主要分为3个主要部分：

* 对所有类型的数据包都通用的消息格式
* 每种特定数据包的具体细节
* 数据包如何在服务器和客户端之间传输

在附录中将介绍如何主题通配符（topic wildcards）的使用方法。

**1.1 V3.1与V3之间的变化**

MQTT V3.1中与MQTT V3之间的不同点如下所示：

* 添加用户验证，用户名和密码现在可以在 CONNECT 数据包中一并发送。
* 为解决一些安全问题，在 CONNACK 数据包中添加了新的返回码。
* 当客户端发送未授权的 PUBLISH 或 SUBSCRIBE 命令时，客户端不会收到相应的通知，即客户端不知道命令不会被执行。而且即使命令未被执行，该MQTT流也应该正常完成。
* MQTT中的字符串现在支持完整的UTF-8字符集，而不仅仅是US-ASCII子集。

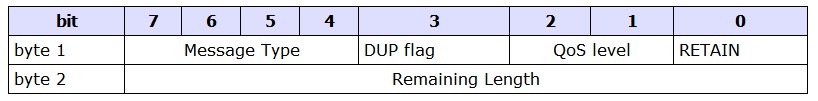
V3.1中，通过 CONNECT 包传送的协议版本号仍然是“3”，与V3相比没有变化。现存的基于MQTT V3的服务器实现须通过正确考虑“剩余长度（Remaining Length）字段”，以及相应地忽略多余的安全信息来接受来自V3.1协议的客户端的连接。

**2. 消息格式**

每个MQTT命令消息的消息头部都包含了一个固定头部。其中一些类型的消息可能还需要一个可变头部和一个有效载荷（可理解为消息体）。消息头部的具体格式将在后面章节中进行详细介绍。

**2.1 固定头部（Fixed header）**

每个MQTT命令消息的消息头部都包含一个固定头部。固定头部的格式如下表如示。



**Byte 1**

包含消息类型和标志（包括DUP，QoS level和RETAIN）字段。

**Byte 2**

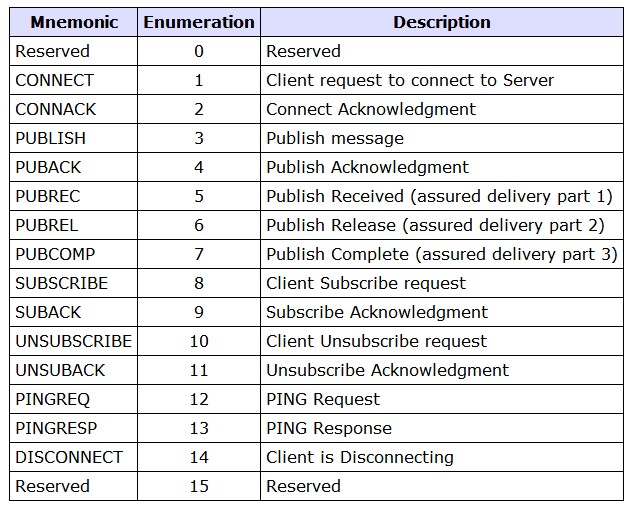
包含剩余长度字段（至少1个字节，最多4个字节）。

以下章节将详细介绍这些字段。所有数据的值都是以big-endian（大端）模式存储：数据的高位字节存放在内存的低地址中，数据的低位字节存放在内存高地址中。一个16位字在内存中的存放顺序是先最高有效位（MSB），然后再最低有效位（LSB）。

**消息类型（Message Type）**

**位置**：byte 1, bits 7-4

该字段为4-bit无符号值。当前协议版本中该字段的具体枚举值如下表所示。



0：保留

1：客户端请求连接服务器

2：连接确认

3：发布消息

4：发布确认

5：发布接收（有保证的交付第1部分）

6：发布释放（有保证的交付第2部分）

7：发布完成（有保证的交付第3部分）

8：客户端订阅请求

9：订阅确认

10：客户端取消订阅请求

11：取消订阅确认

12：PING请求

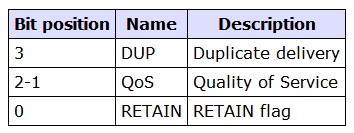
13：PING回复

14：客户端断开连接

15：保留

**标志位（Flags）**

固定头部第1个字节中剩余的部分包含DUP，QoS和RETAIN标志字段。相应bit位置如下表所示。



**DUP**

**位置**：byte 1，bit 3

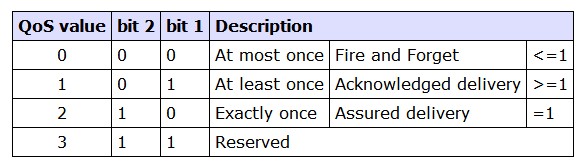
当客户端或服务器试图重发 PUBLISH、PUBREL、SUBSRIBE、UNSUBSCRIBE 消息时，该标志位要被置位（即设为1）。这适用于消息的QoS标志值大于0的情况，此时消息确认是必需的。当DUP位被置位时，可变头部将包含一个消息ID。

消息的接收者应当将该标志视为该消息之前可能已收到的提示消息，而不该依赖于它进行消息重复检测。

**QoS**

**位置**：byte 1，bits 2-1

该标志位标明 PUBLISH 消息的交付质量级别。具体的QoS级别如下表所示。



**RETAIN**

**位置**：byte 1，bit 0

该标志位只用于 PUBLISH 消息。当一个客户端发送一条 PUBLISH 消息给服务器，假设该消息所属的主题（topic）为topicA，如果该标志位被置位（1），服务器在将该条消息发布给当前的所有topicA的订阅者之后，还应当保持这条消息。

当topicA出现了一个新的订阅者，则topicA的最后一条保持消息应当发给该订阅者。当然，如果不存在保持消息，则什么也不用发。

当消息发布者以基于 “report by exception” 的方式发送消息时，这个功能就特别有用，因为这种情况下，消息发送间隔往往较长。这个功能使得新的订阅者可以立刻收到之前保持的或上一个确定有效的消息。

当服务器收到某个主题的 PUBLISH 消息时，对于之前已经订阅该主题的客户端，服务器将给这些客户端发送这一 PUBLISH 消息，发送前，服务器会将该消息的 RETAIN 标志置为0（即不置位），不管服务器之前收到该 PUBLISH 消息时其 RETAIN 标志是否被置位。这样做可以使得区分它接收到的 PUBLISH 消息是服务器之前保持的（RETAIN标志置位）还是即时收到的（RETAIN标志不置位）。

保持消息应当在重启服务器后仍能保留。

如果服务器收到有效载荷长度为0或重复主题的保持消息，服务器可以删除该保持消息。

**剩余长度（Remaining Length）**

**位置**：byte 2（从byte 2，最大可至byte 5）

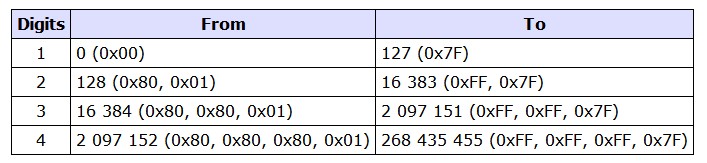
该字段表示当前消息的剩余内容的字节数，包括可变头部和有效载荷的数据。

该字段本身的字节数是根据可变头部和有效载荷的长度不同而变化的。该可变长度编码方案如下：每个字节的低7位（7-0位）编码剩余长度的数据，第8位表示后面是否还有编码剩余长度的字节。即每个字节编码128个值和一个“延续位”。所以只用一个字节时，最大只可表示127字节的长度。

举例如下，十进制数字64只需用1个字节来编码，即0x40。 十进制数字321（=65 + 2x128）则需要用2个字节来编码，其中第1个字节为1100 0001，该字节的低7位表示65，第8位表示后面还有字节；第2个字节为0000 0010，表示2x128。

协议限制该字段最大为4个字节，这允许应用程序发送的最大消息长度为268435455（256MB），即0xFF，0xFF，0xFF，0x7F。

下表给出了增加该字段的字节数时相应可表示的剩余长度值。



该编码方案的算法如下所示，输入为一个十进制数（X），输出为编码后的结果。

do

digit = X MOD 128

X = X DIV 128

// if there are more digits to encode, set the top bit of this digit

if ( X > 0 )

digit = digit OR 0x80

endif

'output' digit

while ( X> 0 )

其中，MOD是模运算符（在C语言中相当于%），DIV表示整数除法（在C语言中相当于/），OR是位或运算符（在C语言中相当于|）。

相应地，剩余长度字段的解码算法如下所示：

multiplier = 1

value = 0

do

digit = 'next digit from stream'

value += (digit AND 127) \* multiplier

multiplier \*= 128

while ((digit AND 128) != 0)

其中，AND是位与运算符（在C语言中相当于&）。

当该解码算法终止，value等于剩余长度的字节数。

值得注意的是，剩余长度编码不是可变头部的一部分。剩余长度的值不包括用于编码剩余长度的字节数。这部分“增加的长度”（1-3字节）是固定头部的一部分，而不属于可变头部。

译者注：可认为剩余长度字段虽然是固定的，但该字段的长度却是变化的（1-4字节）。

**2.2 可变头部（Variable header）**

某些类型的MQTT命令消息还包含了一个可变头部，它位于固定头部和有效载荷之间。

可变的剩余长度字段（1-4字节）不是可变头部的一部分。剩余长度字段的值不包括该字段本身的长度。该值只包括可变头部和有效载荷。更多细节可可见[固定头部](http://joyexpr.com/blog/2013/11/27/mqtt/#fixed-header)。

可变头部中各个字段的格式将在后续章节中进行详细介绍，介绍顺序与该字段在可变头部中的顺序一致。

**协议名称（Protocol name）**

协议名称字段只用于 MQTT CONNECT 消息的可变头部中。该字段以UTF编码方式显示协议名称：MQIsdp，大小写如上所示。

**协议版本（Protocol version）**

协议版本字段只用于 MQTT CONNECT 消息的可变头部中。

该字段用8位无符号值来表示客户端所使用的协议修订级别。当前版本协议中该字段的值为3（0x03），如下表所示。



**连接标志（Connect flags）**

该字段用于 CONNECT 消息的可变头部中，占1字节，包括Clean session、Will、Will QoS和Retain标志。

**Clean session 标志**

**位置**：bit 1（在连接标志字节中，下同）

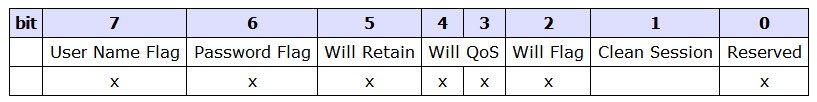
如果没有被置位（即值为0），则当客户端断线时，服务器必须保存该客户端的订阅信息，包括断线期间发布的该客户端订阅的主题中交付质量级别为QoS 1和QoS2的消息，这样当客户端重连时，这部分消息能确保被送达到客户端。同时，服务器还必须保持客户端在断线的那个时刻正在传输中的消息的状态，直到客户端重新连接。

如果被置位（即值为1），则服务器必须丢弃任何之前保持的该客户端的信息，将该连接视为“不存在（Clean）”。同时，当客户端断线时，服务器必须丢弃其所有状态。

通常情况下，客户端会一直在其中一种模式下操作，不会进行切换。该选择取决于具体应用的需求。一个 Clean session 客户端将不会收到过时的信息，且它每次重连时都需要重新订阅主题。而一个 non-clean session客户端则不会漏接任何它在断线时服务器发布的交付质量级别为QoS 1和QoS 2的消息。QoS 0级别的消息由于只是尽可能的交付，所以它永远不会被存储保持。

这个标志以前被称为“Clean start”。因为这个标志其实是作用于整个会话期间的，而不只是在连接的开始阶段，为了明确这个事实，所以将它重命名为“Clean session”。

服务器可以提供一种管理机制，当确定一个客户端将永远不会重新连接时，可以清除该客户端的存储信息。



连接标志中的第0位在目前协议版本中没有使用到，保留为将来使用。。

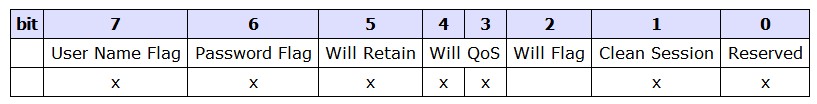
**Will 标志**

**位置**：bit 2

Will消息是指当服务器与客户端通信过程中出现故障或客户端在保活时间内没有与服务器保持正常交流时，服务器特意发给客户端的消息。当客户端通过发送 DISCONNECT 消息正常断开时，Will消息不会发送。

如果Will标志被置位，则Will QoS标志和Will Retain标志的设置将会发生作用，同时，在有效载荷里必须填写Will主题和Will消息内容字段。

Will标志的格式如下表所示。



连接标志中的第0位在目前协议版本中没有使用到，保留为将来使用。

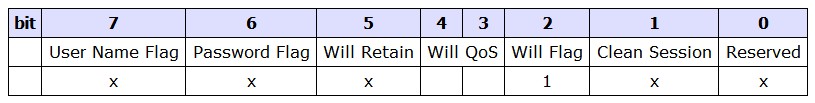
**Will QoS**

**位置**：bit 4-3

Will QoS标志用来设置当客户端异常离线时，服务器发送的Will消息的交付质量级别。Will消息的内容在客户端发送的 CONNECT 消息里的有效载荷里填写。

如果Will标志被置位，则Will QoS字段必须填写，否则该字段的值将被忽略。

Will QoS字段的可选值有0（0x00），1（0x01）和2（0x02），格式如下表所示。



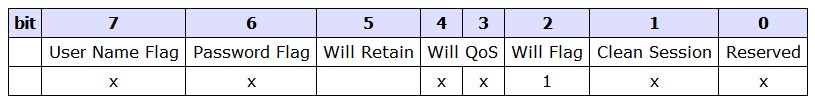
连接标志中的第0位在目前协议版本中没有使用到，保留为将来使用。

**Will Retain 标志**

**位置**：bit 5

Will Retain标志指明服务器是否需要保持客户端异常离线时发送给客户端的Will消息。

如果Will标志被置位，则Will Retain标志必须填写，否则其将被忽略。该标志的格式如下表所示。



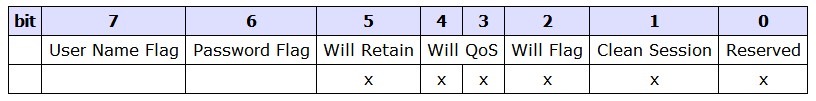
连接标志中的第0位在目前协议版本中没有使用到，保留为将来使用。

**User name and password 标志**

**位置**：bit 6和bit 7

客户端在连接服务器时可以指定用户名和密码，通过将用户名标志和密码标志（可选）置位表明在 CONNECT 消息的有效载荷里包含有用户名和密码。

如果将用户名标志置位，则必须在有效载荷里填写用户名字段，否则用户名字段将被忽略。同样地，如果密码标志被置位，则必须在有效载荷里填写密码字段，否则密码字段将被忽略。只提供密码而不提供用户名是不合法的。



连接标志中的第0位在目前协议版本中没有使用到，保留为将来使用。

**保活计时器（Keep Alive timer）**

保活计时器用于MQTT CONNECT 消息的可变头部中。

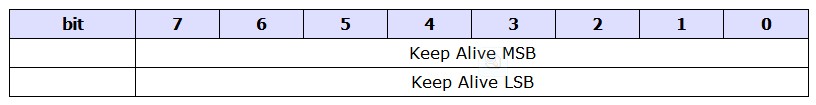
保活计时器定义了服务器收到客户端消息的最大时间间隔，它以秒为单位。它使得服务器不需要等待漫长的TCP/IP超时就可以检测与客户端的网络连接是否断开。客户端有义务在每个保活时间间隔内至少发送一条消息给服务器。如果这期间没有业务相关的消息要发送，客户端则发送一个 PINGREQ 消息给服务器，相应地服务器返回一个 PINGRESQ 消息给客户端。

如果服务器在1.5个保活时间（可宽容0.5个保活时间）内都没有收到客户端的消息，则服务器将其视为客户端发送了一个 DISCONNECT 消息，并断开与客户端的连接。这个动作不影响客户端的订阅。具体细节参见 [DISCONNECT](http://joyexpr.com/blog/2013/11/27/mqtt/#disconnect)。

如果客户端在发送 PINGQ 后的一个保活时间内没有收到服务器发来的 PINGRESP 消息，则客户端可以关闭TCP/IP套接字连接。

保活计时器用2个字节来表示，时间单位为秒。实际设定的值由特定应用决定，不过通常它的值都设为数分钟，最大值接近18个小时。如果设为0，则表示客户端不断线。

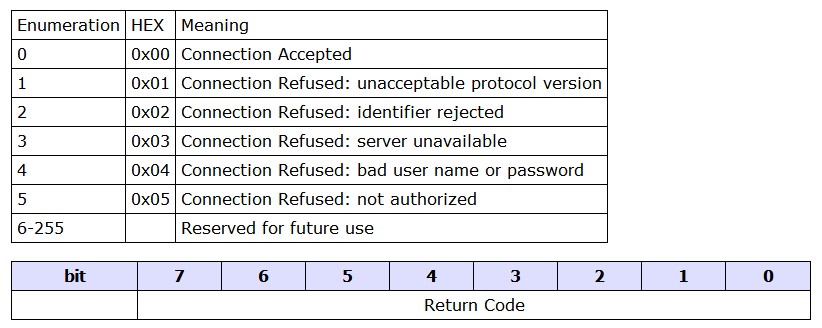
保活计时器的格式如下表所示。2个字节的顺序为先 MSB，再 LSB(大端模式)。



**连接返回码（Connect return code）**

连接返回码用于 CONNACK 消息的可变头部中。

这个字段用一个无符号字节来表示返回码。这些值的含义如下表所示。值为0的返回码通常表示连接成功。



**主题名（Topic name）**

主题名用于 PUBLISH 消息的可变头部中。

主题名决定消息要发送到哪个信息通道。订阅者使用主题名来决定他们要从哪些信息通道接收消息。

主题名是一个UTF编码字符串。更多信息参见[MQTT和UTF-8](http://joyexpr.com/blog/2013/11/27/mqtt/#utf-8)。主题名支持的最大字符度为32767。

**2.3 有效载荷（Payload）**

以下类型的MQTT命令消息拥有一个有效载荷：

**CONNECT**

该有效载荷包含了一个或多个UTF-8编码字符串。它们包括标识客户端的唯一标识符、Will主题和消息、要使用的用户名和密码。其中只有第一项是必选的，其余的取决于可变消息头部中的标志置位情况。

**SUBSCRIBE**

该有效载荷包含一系列要订阅的主题名，以及每个主题的QoS级别。这些字符串都是UTF编码的。

**SUBACK**

该有效载荷包含一系列授权过的QoS级别。它们是服务器管理员允许授权给客户端订阅的各个主题的QoS级别。授权的QoS级别顺序与相应订阅的主题的顺序保持一致。

**PUBLISH**

该有效载荷只包含应用特定的数据。协议不对数据的属性和内容作任何假设，协议把消息的这部分内容视为一个BLOB。

如果你想要对有效载荷数据进行压缩，你必须自己在有效载荷里定义合适的标志来处理压缩事宜。你不能在固定状况或可变头部里定义与特定应用相关的标志。

**2.4 消息标识符（Message Identifiers）**

消息标识符用于以下MQTT消息的可变头部中：PUBLISH，PUBACK，PUBREC，PUBREL，PUBCOMP，SUBSCRIBE，SUBACK，UNSUBSCRIBE，UNSUBACK。

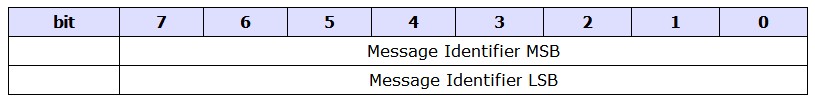
消息标识符（消息ID）字段只存在于固定头部中QoS标志值为1或2的消息中。更多信息可参见[交付质量级别和消息流](http://joyexpr.com/blog/2013/11/27/mqtt/#qos-flows)。

消息ID用16位无符号整数来表示，在同一个方向上的在传消息ID必须是唯一的。它通常是逐个消息递增的，但不强制如此。

客户端与它所连接的服务器一样，都需要维护自己的消息ID列表，二者的消息ID列表互不影响。客户端在发送一个消息ID为1的 PUBLISH 消息的同时也有可能收到来自服务器的消息ID为1的 PUBLISH 消息。

表示消息ID的2个字节的顺序为先 MSB，再 LSB（大端模式）。

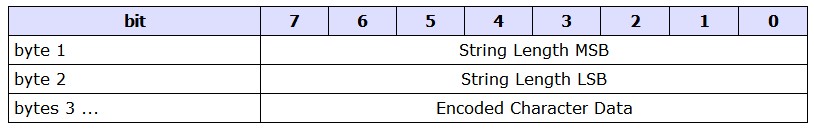
不要使用值为0的消息ID。它是作为无效消息ID保留的。



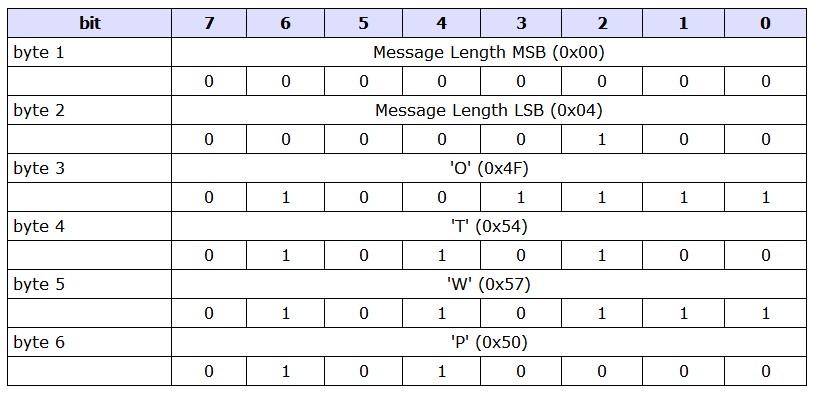
**2.5 MQTT和UTF-8（MQTT and UTF-8）**

UTF-8是一种针对Unicode的可变长度字符编码，它优化了ASCII字符集的编码，以支持基于文本的通信。

在MQTT中，字符串编码的头2个字节用来记录字符串的长度，如下表所示。



字符串长度表示所有字符经过UTF-8编码后的字节数，而不是字符串中字符的个数。例如，经过UTF-8编码后的字符串 OTWP 如下表所示。



Java中的 writeUTF() 和 readUTF() 数据流方法也使用这种格式。

**2.6 未使用的位（Unused bits）**

任何标明为未使用的位都应当置为0。

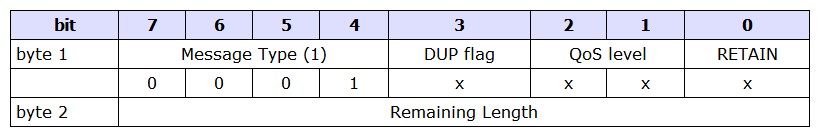
**3. 命令消息（Command messages）**

**3.1 CONNECT - 客户端请求连接服务器**

当客户端与服务器的TCP/IP套接字连接建立起来之后，必须发送一个 CONNECT 消息流来建立一个协议级别的会话。

**固定头部（Fixed header）**

固定头部的格式如下表所示。

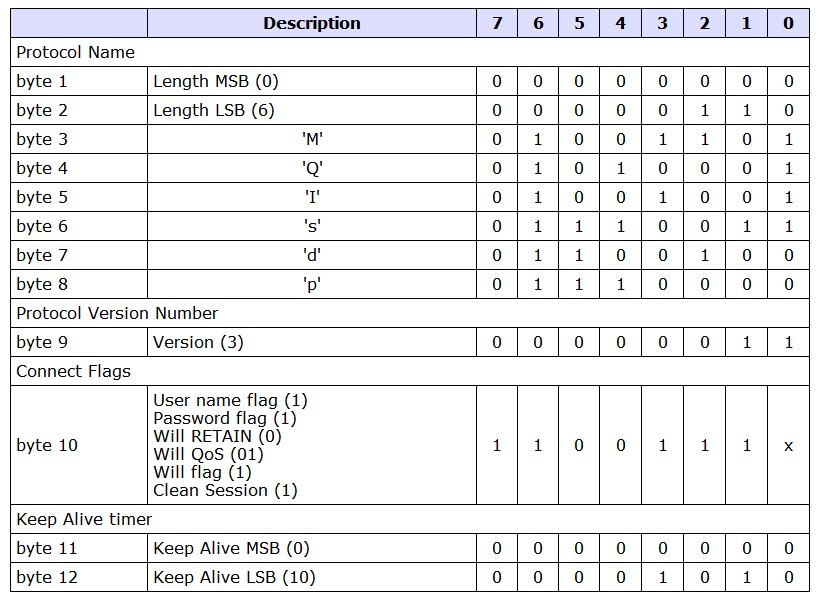


DUP、QoS以及RETAIN标志在 CONNECT 消息中没有被使用到。

剩余长度字段的值为可变头部（12字节）和有效载荷的字节数总和。剩余长度字段自身的长度可能大于1个字节。

**可变头部（Variable header）**

一个可变头部例子的格式如下表所示。



其中，

**用户名标志**

置位（1）。

**密码标志**

置位（1）。

**Clean session标志**

置位（1）。

**保活计时器**

设置为10秒（0x000A）。

**Will 消息**

* Will标志置位（1）
* Will QoS字段值为1
* Will RETAIN标志不置位（0）

**有效载荷（Payload）**

CONNECT 消息的有效载荷根据可变头部中各标志位的置位情况，包含一个或多个UTF-8编码字符串。这些字符串如果出现的话，必须符合以下顺序：

**客户端ID**

这是第1个UTF编码字符串。客户端ID(Client ID)的长度为1至23个字符，服务器根据客户端ID可以指定到唯一的客户端。对于连接到某个服务器的所有客户端，它们的客户端ID必须都是唯一的，客户端ID还是处理QoS级别1和2消息的关键。如果发送的CONNECT 消息中客户端ID的长度大于23个字符，则服务器会回复一个返回码值为2（标识符被拒绝）的 CONNACK 消息。

**Will主题**

如果Will标志被置位，则Will主题将是有效载荷中的下一个字符串。Will消息将会发送给Will主题。消息的QoS级别和RETAIN状态在可变头部的Will QoS和Will RETAIN标志中设置。

**Will消息（内容）**

如果Will标志被置位，则Will消息将是有效载荷中的下一个字符串。Will消息定义了客户端异常离线时服务器发送给Will主题的消息内容。当然，消息内容可以为空（消息长度为0，但该字符串仍包含2个字节以记录其长度为0）。

尽管Will消息内容在 CONNECT 消息中是以UTF-8编码的，但当它最后被发送到Will主题时，只有消息的实际内容被发送，而不包括开头记录长度的2个字节。因而，消息必须只包含7-bits ASCII字符。

**用户名**

如果用户名标志被置位，则用户名将是有效载荷中的下一个字符串。用户名字段用于认证，标明了连接的用户的名字。建议用户名不超过12个字符，但不强制如此。

值得注意的是，为了与MQTT V3版本协议兼容（V3中不支持用户名密码）， 固定头部中的剩余长度字段的优先级应该高于用户名标志。服务器的实现必须允许用户名标志被置位，但不存在用户名字符串的情况。这是合法的，连接应该继续进行。

**密码**

如果密码标志被置位，则密码将是有效载荷中的下一个字符串。密码字段用于认证，对应于连接的用户名。建议密码不超过12个字符，但不强制如此。

值得注意的是，为了与MQTT V3版本协议兼容（V3中不支持用户名密码）， 固定头部中的剩余长度字段的优先级应该高于密码标志。服务器的实现必须允许密码标志被置位，但不存在密码字符串的情况。这是合法的，连接应该继续进行。

**回复（Response）**

服务器收到客户端发送的 CONNECT 消息后会回复一个 CONNACK 消息。

如果在TCP/IP连接建立后的一段合理时间内服务器没有收到客户端发送的 CONNECT 消息，则服务器应该关闭这个连接。

如果客户端在发送 CONNECT 消息后的一段合理时间内客户端没有收到服务器回复的 CONNACK 消息，则客户端应该关闭原先的TCP/IP连接。然后建立新的TCP/IP连接，并发送 CONNECT 消息以重起会话。

以上两种场景中，一段合理时间的设置依赖于特定应用的需求和通信架构。

如果尝试连接的客户端ID在服务器中已经存在，则服务器会断开“旧”的客户端并与新的客户端完成连接操作。

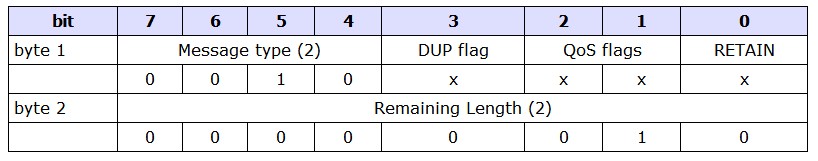
如果客户端发送了一个不合法的 CONNECT 消息，包括提供了不合法的协议名和协议版本号，服务器应该断开连接。如果服务器可以从 CONNECT 消息中识别并且明确客户端使用了一个不合法的协议，服务器可以尝试在断开连接前回复一个 CONNACK 消息，告知客户端“Connection Refused: unacceptable protocol version”。

**3.2 CONNACK - 连接确认**

当客户端向服务器发起 CONNECT 请求，服务器会回复其 CONNACK 消息。

**固定头部（Fixed header）**

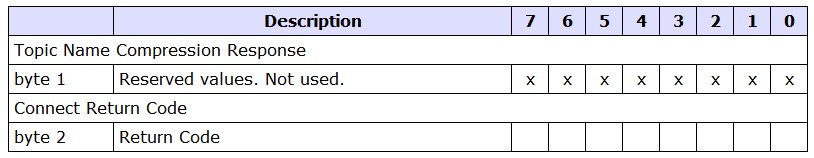
固定头部的格式如下表所示。



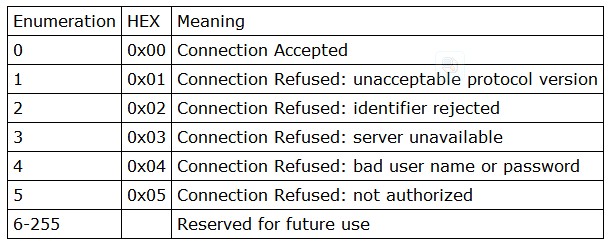
DUP、QoS以及RETAIN标志在 CONNACK 消息中没有被使用到。

**可变头部（Variable header）**

可变头部的格式如下表所示。



其中，连接返回码用1个无符合字节表示，具体含义如下表所示。



如果客户端ID的长度不在1-23字节之间，服务器则会发送返回码2（标识符被拒绝）。

**有效载荷（Payload）**

该类型消息没有有效载荷。

**3.3 PUBLISH - 发布消息**

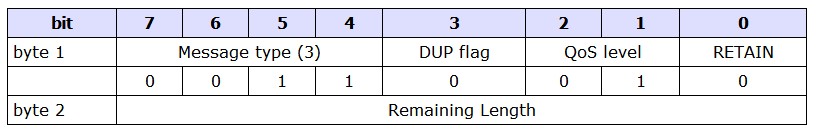
当客户端想发布消息给感兴趣的订阅者时，客户端将发送一条 PUBLISH 消息给服务器。每个 PUBLISH 消息都关联一个主题名（也可称为话题或频道）。主题名是一个层次性的空间，它将订阅者感兴趣的信息资源进行分类。一个主题的消息将会发送给连接时订阅了该主题的客户端。

当客户端订阅了一个或多个主题，属于这些主题的所有消息都将通过服务器发送相应 PUBLISH 消息给客户端。

译者注：其实流程简单来说就是，发布者客户端发送某topic的 PUBLISH 消息给服务器，然后服务器再发 PUBLISH 消息给所有该topic的订阅者客户端。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

设为1。详见[交付质量级别和消息流](http://joyexpr.com/blog/2013/11/27/mqtt/#qos-flows)。

**DUP标志**

设为0。表示该消息是第一次发送。详见[DUP](http://joyexpr.com/blog/2013/11/27/mqtt/#dup)。

**RETAIN标志**

设为0。表示不保持。详见[RETAIN](http://joyexpr.com/blog/2013/11/27/mqtt/#retain)。

**剩余长度字段**

长度包括可变头部和有效载荷。字段自身长度为1-4字节。

**可变头部（Variable header）**

可变头部包含以下字段：

**主题名**

一个UTF编码字符串。

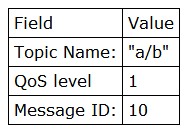
不允许出现主题通配符字符。

客户端订阅主题时主题名可以使用通配符，但服务器最终给订阅客户端发布 PUBLISH 消息时，消息里的主题名是和发布者客户端发布的主题名一致的，也就是肯定不含有通配符。

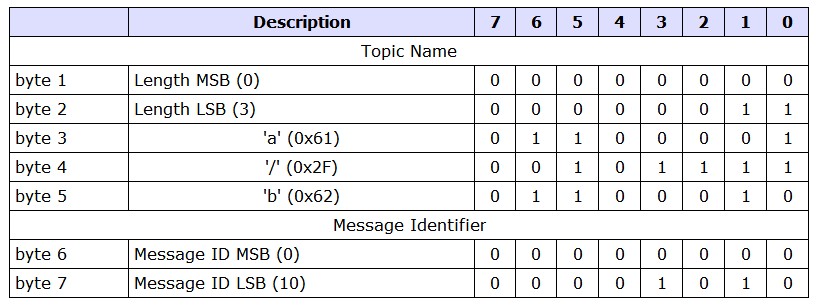
**消息ID**

当消息的QoS级别为1或2时使用。详见[消息标识符](http://joyexpr.com/blog/2013/11/27/mqtt/#msg-id)。

下表是一个 PUBLISH 消息例子的可变头部。



该头部的具体格式如下表所示。

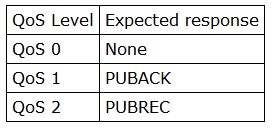


**有效载荷（Payload）**

包含要发的消息的数据。数据的内容和格式由应用决定。固定头部中的剩余长度字段的值包括可变头部和有效载荷。当然，有限载荷长度为0的 PUBLISH 消息也是有效的。

**回复（Response）**

对PUBLISH 消息的回复取决于QoS级别。具体见下表。



**动作（Action）**

PUBLISH 消息可以由发布者客户端发给服务器，也可以由服务器发给订阅者客户端。PUBLISH 消息的接收者（包括服务器或客户端）根据消息的QoS级别做出不同的反应。

**QoS 0**

仅仅将消息发送给所有相关的部分。

**QoS 1**

将消息持久化存储，将消息发送给所有相关的部分，回复 PUBACK 消息给发送者。

**QoS 2**

将消息持久化存储，先不将消息发送给所有相关的部分，而是先回复 PUBREC 消息给发送者。

如果是服务器收到 PUBLISH 消息，则此时所有相关部分指的是订阅了该条 PUBLISH 消息主题的订阅者客户端。如果是订阅者客户端收到 PUBLISH 消息，则此时所有相关部分就是指客户端上等待服务器消息的各个应用。

更多信息请参见[交付质量级别和消息流](http://joyexpr.com/blog/2013/11/27/mqtt/#qos-flows)。

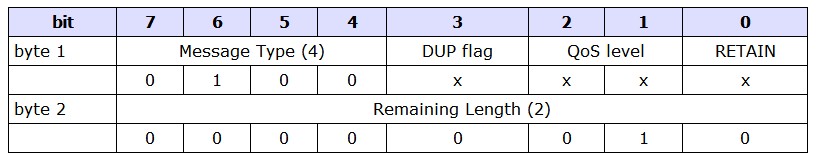
值得注意的是，如果一个服务器实现对某个客户端发送的一个 PUBLISH 消息不允许授权，它没有办法通知客户端。因此，当授权通过时，它需要做出一个正面的确认。根据正常的QoS规则，如果某个客户端没有发布 PUBLISH 消息的授权，这个客户端不会被告知。

**3.4 PUBACK - 发布确认**

PUBACK 消息是对QoS级别1的 PUBLISH 消息的回应。当发布者客户端发送 PUBLISH 消息给服务器，服务器有义务回复 PUBACK 消息。同样地，当服务器发送 PUBLISH 消息给订阅者客户端，客户端也有义务回复 PUBACK 消息。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

未使用。

**DUP标志**

未使用。

**RETAIN标志**

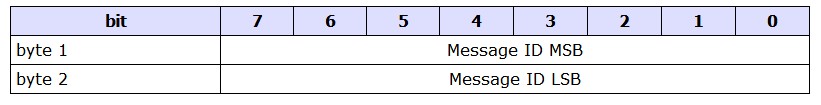
未使用。

**剩余长度字段**

因为没有有效载荷，所有长度为可变头部的长度（2字节）。

**可变头部（Variable header）**

包含被确认的 PUBLISH 消息的消息标识符（消息ID）。格式如下表所示。



**有效载荷（Payload）**

没有有效载荷。

**动作（Action）**

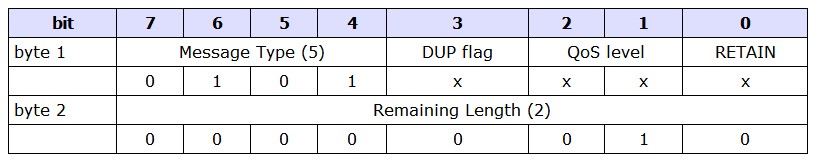
当发布者客户端收到 PUBACK 消息，则它丢弃自己原先发送给服务器的 PUBLISH 消息，因为该消息已经被服务器接收并存储了。

**3.5 PUBREC - 发布接收（有保证的交付第1部分）**

PUBREC 消息是对QoS级别2的 PUBLISH 消息的回应。它是QoS级别2协议流中的第2个消息。PUBREC 消息可以是服务器对发布者客户端发送的 PUBLISH 消息的回应，也可以是订阅者客户端对服务器发送的 PUBLISH 消息的回应。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

未使用。

**DUP标志**

未使用。

**RETAIN标志**

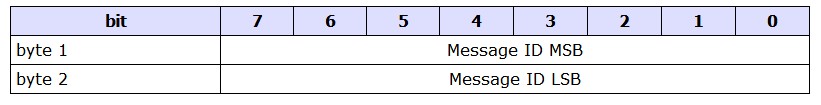
未使用。

**剩余长度字段**

因为没有有效载荷，所有长度为可变头部的长度（2字节）。

**可变头部（Variable header）**

包含被确认的 PUBLISH 消息的消息标识符（消息ID）。格式如下表所示。



**有效载荷（Payload）**

没有有效载荷。

**动作（Action）**

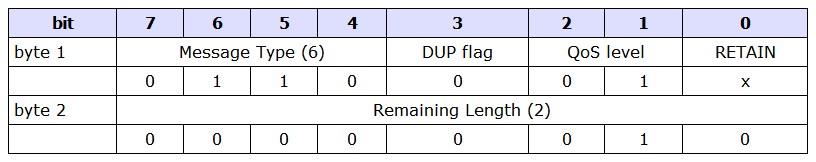
当接收者收到 PUBREC 消息，则它将回复一个 PUBREL 消息，消息中携带的消息ID与收到的 PUBREC 消息中的相同。

**3.6 PUBREL - 发布释放（有保证的交付第2部分）**

PUBREL 消息可以是发布者客户端对服务器发送给它的 PUBREC 消息的回应，也可是服务器对订阅者客户端发送给它的 PUBREC 消息的回应。它是QoS级别2协议流中的第3个消息。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

设为1。PUBREL 消息使用QoS级别1，期望得到 PUBCOMP 消息回应。它的重传方式与 PUBLISH 消息一样。

**DUP标志**

设为0。表示该消息是第一次发送。详见[DUP](http://joyexpr.com/blog/2013/11/27/mqtt/#dup)。

**RETAIN标志**

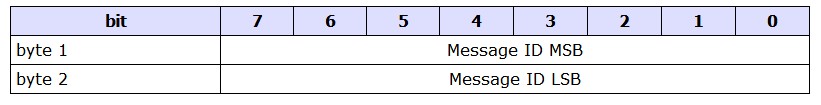
未使用。

**剩余长度字段**

因为没有有效载荷，所有长度为可变头部的长度（2字节）。

**可变头部（Variable header）**

包含被确认的 PUBREC 消息的消息标识符（消息ID）。格式如下表所示。



**有效载荷（Payload）**

没有有效载荷。

**动作（Action）**

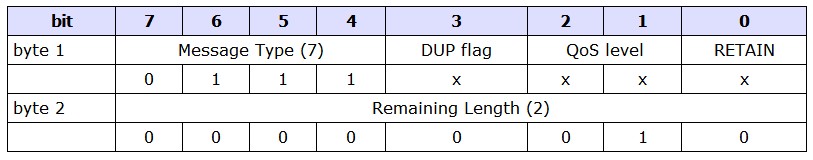
当服务器收到发布者客户端发送的 PUBREL 消息，服务器会将之前收到 PUBLISH 消息发送给所有相关的订阅者，然后发送 PUBCOMP 消息给发布者客户端，其中携带的消息ID与客户端发送的 PUBLISH 消息ID相同。当订阅者客户端收到服务器发送的 PUBREL 消息后，客户端将消息发送给订阅的应用，然后发送 PUBCOMP 消息给服务器。

**3.7 PUBCOMP - 发布完成（有保证的交付第3部分）**

PUBCOMP 消息可以是服务器对发布者客户端发送给它的 PUBREL 消息的回应，也可以是订阅者客户端对服务器发送给它的 PUBREL 消息的回应。它是QoS级别2协议流中的第4个消息。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

未使用。

**DUP标志**

未使用。

**RETAIN标志**

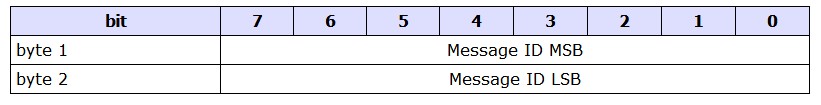
未使用。

**剩余长度字段**

因为没有有效载荷，所有长度为可变头部的长度（2字节）。

**可变头部（Variable header）**

包含被确认的 PUBREL 消息的消息标识符（消息ID）。格式如下表所示。



**有效载荷（Payload）**

没有有效载荷。

**动作（Action）**

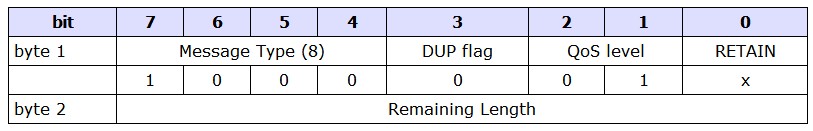
当发布者客户端收到 PUBCOMP 消息，它将丢弃自己原发发送的 PUBLISH 消息，因为该消息已经被正确地发送一次到服务器。

**3.8 SUBSCRIBE - 客户端订阅主题**

SUBSCRIBE 消息允许客户端向服务器订阅一个或多个感兴趣的主题。这些主题的发布消息都会通过PUBLISH 消息从服务器发送给客户端。SUBSCRIBE 消息还可以分别对各个主题指定客户端期望得到的QoS级别。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

SUBSCRIBE 消息使用QoS级别1来进行订阅请求。相应的 SUBACK 消息携带与之相同的消息ID。重传方式与 PUBLISH 消息相同。

**DUP标志**

设为0。表示该消息第一次被发送。详见[DUP](http://joyexpr.com/blog/2013/11/27/mqtt/#dup)。

**RETAIN标志**

未使用。

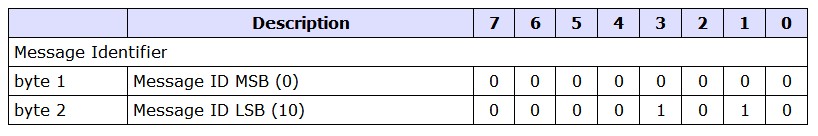
**剩余长度字段**

长度包括可变头部和有效载荷（原文：The length of the payload）。字段自身长度为1-4字节。

**可变头部（Variable header）**

因为 SUBSCRIBE 消息使用QoS级别1，所以可变头部里包含有一个消息ID。详见[消息标识符](http://joyexpr.com/blog/2013/11/27/mqtt/#msg-id)。

下表是一个可变头部例子的格式。



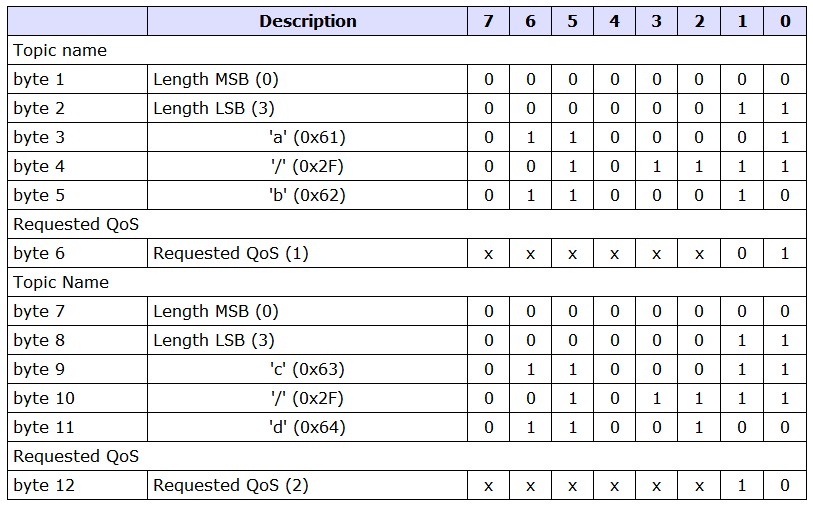
**有效载荷（Payload）**

SUBSCRIBE 消息的有效载荷包括了一系列客户端想要订阅的主题，以及对各个主题，客户端期望接收消息的QoS级别。字符串经过UTF编码，指定的QoS级别虽然只用2bits表示，但仍要占用一个字节。主题字符串可以包含主题通配符来订阅一系列主题集。这些主题/QoS对的形式如下表中的例子所示。



SUBSCRIBE消息里的主题名没有被压缩。

上个例子的具体格式如下表所示。



假设客户端订阅请求的QoS级别通过授权，该客户端之后收到的 PUBLISH 消息的QoS级别将等于或小于请求的QoS级别，这取决于消息发布者设定的消息QoS级别。例如，如果客户端申请订阅某主题的QoS级别为1，对于QoS级别为0的该主题的发布消息，该客户端只能以QoS级别0来接收。而对于QoS级别为2的该主题的发布消息，该客户端将降格以QoS级别1来接收。

如果客户端以QoS级别2来订阅某主题，客户端相当于说：“我愿意以发布消息时设置的QoS级别来接收这个主题的消息。”。（译者注：因为级别2为最高级，发布消息不可能被降格。）

这意味着，发布者有责任检测能被接收的最高QoS级别。不过，订阅者可以根据自己的需求来降格QoS级别。消息的QoS级别不可能被升格。

请求的QoS字段跟随着每个UTF编码的主题名字符串之后，其格式如下表所示。



当前版本协议中该字节的前6位未被使用，保留作将来使用。

如果最后2位都被置位（即值为3），则应当将其视为非法包并关闭连接。

**回复（Response）**

当服务器收到来自客户端的 SUBSCRIBE 消息，应当回复一个 SUBACK 消息。

在客户端收到服务器发送的 SUBACK 消息之前，服务器就可以向客户端的订阅发送 PUBLISH 消息。

值得注意的是，如果一个服务器的实现不允许授权客户端的 SUBSCRIBE 请求，它没有办法告知客户端。因此，当授权通过时，它必须发送 SUBACK 消息来进行正面的回应。如果订阅没有通过授权，客户端不会被告知。

服务器可以以低于客户端申请的QoS级别进行授权。当服务器没有办法提供高QoS级别时，这种情况就可能发生。例如，如果服务器不提供可靠的持久化存储，它可以选择只能QoS级别0来授权订阅。

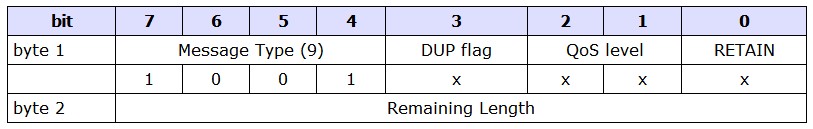
**3.9 SUBACK - 订阅确认**

服务器通过发送 SUBACK 消息来确认其已收到 SUBSCRIBE 消息。

SUBACK 消息包含一系列通过授权的QoS级别。它们的顺序与 SUBSCRIBE 消息里订阅的主题的顺序一致。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

未使用。

**DUP标志**

未使用。

**RETAIN标志**

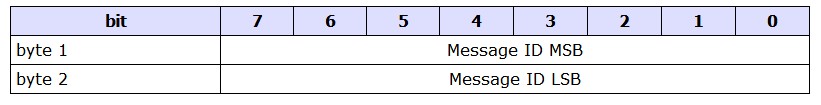
未使用。

**剩余长度字段**

长度包括可变头部和有效载荷（原文：The length of the payload）。字段自身长度为1-4字节。

**可变头部（Variable header）**

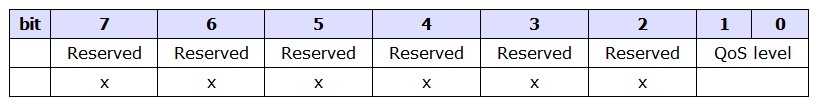
可变头部包含了确认的 SUBSCRIBE 消息的消息ID，格式如下表所示。



**有效载荷（Payload）**

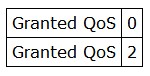
有效载荷包含了一组通过授权的QoS级别。各个QoS级别对应 SUBSCRIBE 消息里的主题名。SUBACK 消息里QoS级别的顺序与 SUBSCRIBE 消息里申请的主题名/QoS对相匹配。可变头部里的消息ID可以让你匹配到对应的 SUBSCRIBE 消息。

每个通过授权的QoS字段用一个字节来编码，如下表所示。

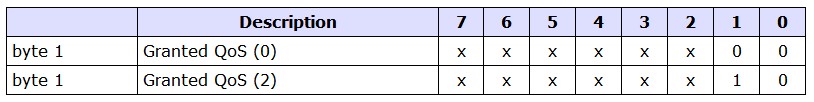


当前版本协议中该字节的前6位未被使用，保留作将来使用。

一个有效载荷的例子如下表所示。



它的具体格式如下表所示。

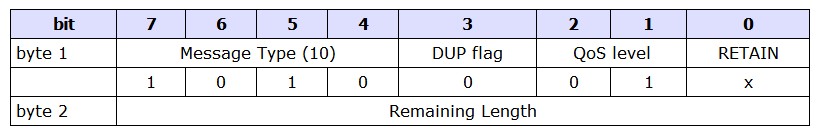


**3.10 UNSUBSCRIBE - 客户端取消订阅主题**

客户端可以通过发送 UNSUBSCRIBE 消息给服务器来退订相应的主题。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

UNSUBSCRIBE 消息使用QoS级别1来进行退订的申请。对应的 SUBACK 消息将用消息ID来识别。重传方式与 PUBLISH 消息相同。

**DUP标志**

设为0。表示该消息是第一次发送。详见[DUP](http://joyexpr.com/blog/2013/11/27/mqtt/#dup)。

**RETAIN标志**

未使用。

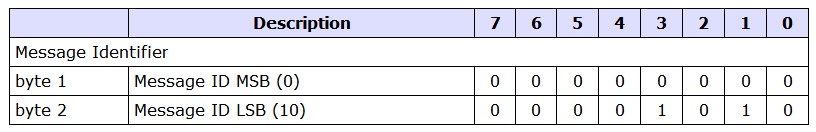
**剩余长度字段**

长度包括可变头部和有效载荷（原文：This the length of the payload）。字段自身长度为1-4字节。

**可变头部（Variable header）**

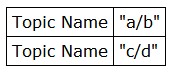
因为 UNSUBSCRIBE 消息使用QoS级别1，所以可变头部里包含一个消息ID。详见[消息标识符](http://joyexpr.com/blog/2013/11/27/mqtt/#msg-id)。

下表是一个消息ID为10的可变头部例子的格式。

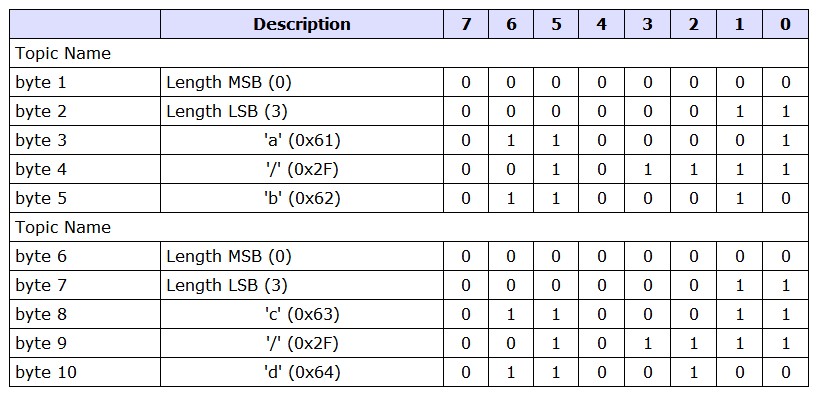


**有效载荷（Payload）**

有效载荷里包含客户端要退订的一系列主题名。这些字符串是UTF编码的，并且是连接的。UNSUBSCRIBE 消息里的主题名没有被压缩。一个有效载荷的例子如下表所示。



它的具体格式如下表所示。



**回复（Response）**

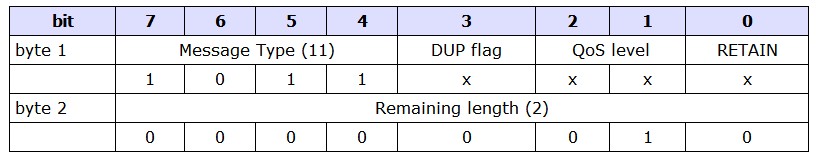
服务器在收到客户端的 UNSUBSCRIBE 消息后，将回复 UNSUBACK 消息。

**3.11 UNSUBACK - 取消订阅确认**

服务器通过发送 UNSUBACK 消息来确认其已收到 UNSUBSCRIBE 消息。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，

**QoS级别**

未使用。

**DUP标志**

未使用。

**RETAIN标志**

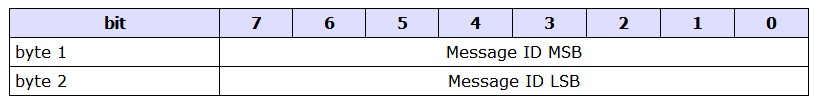
未使用。

**剩余长度字段**

因为没有有效载荷，所有长度为可变头部的长度（2字节）。

**可变头部（Variable header）**

可变头部包含要确认的 UNSUBSCRIBE 消息的消息ID，它的格式如下表所示。



**有效载荷（Payload）**

没有有效载荷。

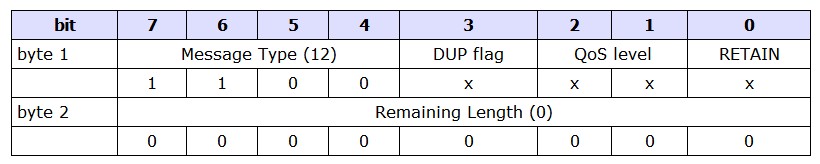
**3.12 PINGREQ - PING请求**

PINGREQ 消息从连接中的客户端发送给服务器，表示询问服务器：“你还活着吗”？

更多信息参见[保活计时器](http://joyexpr.com/blog/2013/11/27/mqtt/#keep-alive-timer)。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，DUP，QoS以及RETAIN标志未使用。

**可变头部（Variable header）**

没有可变头部。

**有效载荷（Payload）**

没有有效载荷。

**回复（Response）**

PINGREQ 消息的期望回复是 PINGRESP 消息。

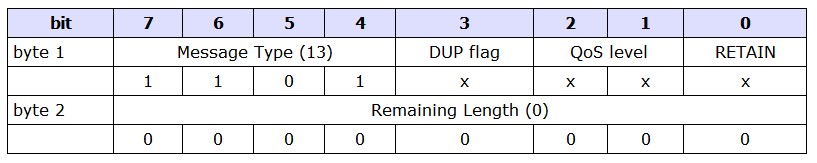
**3.13 PINGRESP - PING回复**

PINGRESP 消息是服务器对客户端发送给它的 PINGREQ 消息的回应，表示回答客户端：“对，哥还活着”。

更多信息参见[保活计时器](http://joyexpr.com/blog/2013/11/27/mqtt/#keep-alive-timer)。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，DUP，QoS以及RETAIN标志未使用。

**可变头部（Variable header）**

没有可变头部。

**有效载荷（Payload）**

没有有效载荷。

**3.14 DISCONNECT - 断开连接通知**

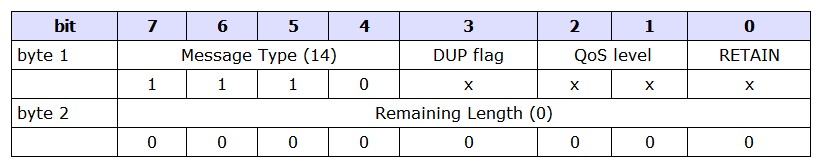
当客户端想要断开它的TCP/IP连接时，通过向服务器发送 DISCONNECT 消息告知服务器。这是为了干净地断开连接，而不仅仅是断开连线。

如果客户端在连接时将clean session标志置位，则之前保持的客户端的信息将会被丢弃。

服务器不应该仅仅依赖客户端发送的 DISCONNECT 消息来关闭TCP/IP连接。

**固定头部（Fixed header）**

固定头部的格式如下表所示。



其中，DUP，QoS以及RETAIN标志未使用。

**可变头部（Variable header）**

没有可变头部。

**有效载荷（Payload）**

没有有效载荷。

**4. 消息流（Flows）**

**4.1 交付质量级别和消息流（Quality of Service levels and flows）**

MQTT 根据交付质量级别（QoS）来发送消息。所有级别描述如下所示：

**QoS 级别0: 至多交付一次**

消息依赖底层的TCP/IP网络做尽可能的发送。不要求响应且协议中也没有定义重传的语义。该消息要么只到达服务器1次，要么干脆一次也没有。

QoS级别0的协议流如下表所示。

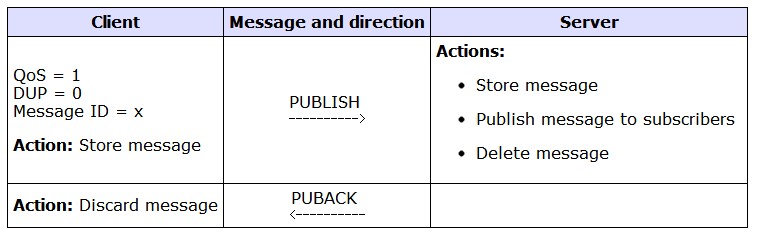


**QoS 级别1：至少交付一次**

服务器确认收到 PUBLISH 消息后要回复客户端一个 PUBACK 消息。不管是通信链路还是发送设备出现了故障，或是确认消息没有在规定时间内收到，则发送者（即之前发送 PUBLISH 消息的客户端）需要重发 PUBLISH 消息，且该消息头部中的DUP标志要置位。该消息将至少到达服务器一次。SUBSCRIBE 和 UNSUBSCRIBE 消息都需要使用QoS级别1。

QoS级别1的消息头部里都包含有一个消息ID。

QoS级别1的协议流如下表所示。



如果客户端没有收到 PUBACK 消息（不管是超过应用设定的时间还是检测到故障然后通信会话重起），客户端将重发DUP标志被置位的 PUBLISH 消息。

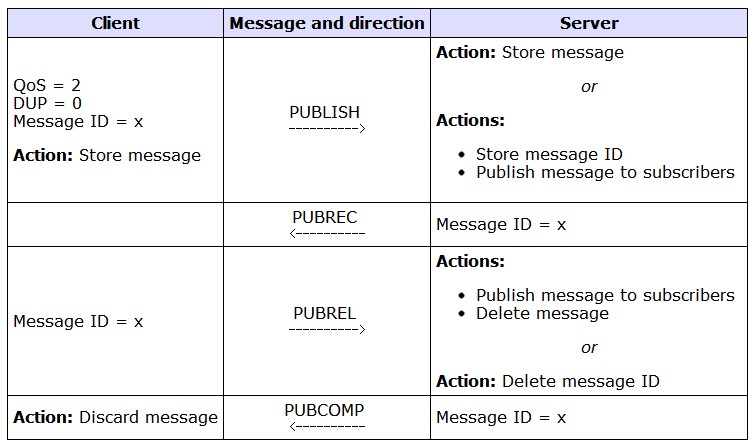
当服务器接收到一条客户端发来的重复消息，服务器仍然会将该重复消息发送给订阅者，然后再发送一条 PUBACK 消息给客户端。

**QoS 级别2：正好交付一次**

QoS级别2是在QoS级别1上更进一步的协议流，可以保证重复消息不会发送给接收端应用。这是最高级别的交付质量，用在重复消息不可接受的情况下。虽然使用该级别将带来网络流量的增长，但对于一些重要的消息内容这通常是可以接受的。

QoS级别2的消息头部里都包含有一个消息ID。

QoS级别2的协议流如下表所示。对于接收方有2种方式（语义）用来处理该 PUBLISH 流。这2种不同语义影响的是协议流中订阅者可以接收消息的时间点。这2种语义都可以保证QoS级别2的协议流，如何选择取决于具体的实现。



如果检测到故障，或发生超时，协议流将重发最后一个未确认的协议消息，包括 PUBLISH 或 PUBREL。具体细节见[消息重传](http://joyexpr.com/blog/2013/11/27/mqtt/#retry)。该级别中附加的协议流可以保证消息只交付给订阅者一次。

**QoS级别1和2的假设**

在任何网络中，设备或通信链路都有可能发生故障。发生故障时，链路的一端可能并不知道另一端发生了什么事，这被称为怀疑窗口（in doubt windows）。在这种背景下，对于消息发送，必须对设备和网络的可靠性作一个假设。

MQTT假设客户端和服务器在一般情况下是可靠的，而通信通道更可能是不可靠的。如果客户端发生故障，则通常是灾难性的故障，而不大可能是暂时性的。从设备中恢复数据的可能性是很低的。有些设备具有非易失性存储，比如闪存ROM。为了应对各种可能的故障，需要提供更多的持久化存储来保护最重要的数据。

除了最基本的通信链路的故障，更多超出MQTT处理能力的场景将使得故障模式矩阵显得更加复杂。

**4.2 消息重传**

尽管TCP通常可以保证数据包的送达，然而MQTT消息没有被收到的情况无疑是存在的。对于需要收到回应的MQTT消息（QoS>0，PUBLISH、PUBREL、SUBSCRIBE、UNSUBSCRIBE），如果在规定时间内消息响应没收到，发送方将重发该消息，且该消息的DUP标志将被置位。

重发超时应当是可配置的。超时的设置最好可以保证消息的正常传输不会引发超时。例如，通过一条拥挤的网络发送一条大消息的时间通常要长于通过一条快速的网络发送一条小消息的时间。一直重复地重传一条超时的消息将使情况变得更糟糕。因此，随着重传次数增加而增大超时的策略通常会被采用。

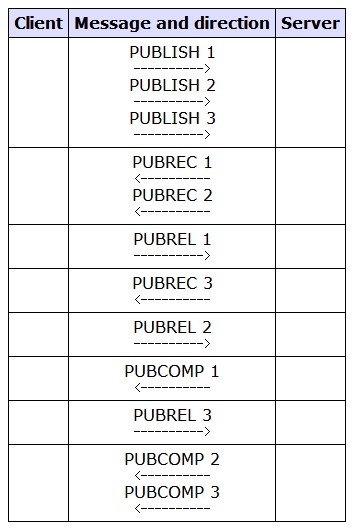
当一个客户端重新连接时，如果Clean session标志没有被置位，则客户端和服务器双方都应当重发断线前正在传输中的消息。

不同于上述“重连”重传行为，客户端不需要重发消息。然而，Brokers需要重发所有未确认的消息。

**4.3 消息排序**

消息的顺序会被一系列因素所影响，包括客户端允许同时在传的 PUBLISH 流的数量和客户端是单线程还是多线程的。为了讨论的简单，我们假设客户端在对网络中的数据包进行读写时是单线程的。

对于一个提供有序消息保证的实现，它必须保证消息流的每个阶段的完成顺序都应该与它们的启动顺序保持一致。例如，对于一系列QoS级别2的消息流，它们发送 PUBREL 消息的顺序与它们发送 PUBISH 消息的顺序保持一致：



客户端允许同时在传的消息数量（称为在传窗口）对保证交付的类型也有一定的影响：

* 对于在传窗口为1的情况，每个消息流的开始都在上个消息流完成之后。这保证了消息的传送顺序与提交它们的顺序保持一致。
* 当在传窗口大于1时，消息的有序化只能通过QoS级别来保证。

**附录 A - 主题通配符（Topic wildcards）**

订阅的主题可以包含特殊字符，从而可以使你一次订阅多个主题。

主题层次分隔符是用来引入主题的结构，因而可以用来区分主题内的不同目的。多层次通配符和单层次通配符可以用于订阅，但它们不能用于在一个主题内发布消息。

**主题层次分隔符（Topic level separator）**

斜线（/）是用来分隔一棵主题树里的各个层次，它提供了一个层次结构的主题空间。主题层次分隔符的引入为解决主题冲突有重要的意义。

**多层次通配符**

数字符号（#）即多层次通配符，用来匹配一个主题内的任意层次。例如，如果你想要订阅了 finance/stock/ibm/#，则你可以收到如下主题的消息：

finance/stock/ibm

finance/stock/ibm/closingprice

finance/stock/ibm/currentprice

多层次通配符可以表示0个或任意多个层次。因而， finance/# 也可以匹配单单 finance 主题，这时， # 代表0层次。当然，在这种情况下，多层次通配符的使用是没有意义的，因为根本没有层次需要分隔。

多层次通配符只有2种用法，一是只有它自己本身，二是放在主题层次分隔符之后。因此， # 和 finance/# 都是合法的，而 finance#则是不合法的。多层次通配符在主题树使用时必须作为最后一个字符。例如， finance/# 是合法的，而 finance/#/closingprice 是不合法的。

**单层次通配符**

加号（+）是用来匹配一个主题层次的通配符。例如， finance/stock/+ 可以匹配 finance/stock/ibm 和 finance/stock/xyz ，但不能匹配 finance/stock/ibm/closingprice。同时，因为单层次通配符必须匹配一个层次，所以 finance/+ 不能匹配 finance 。

单层次通配符可以用于主题树中的任意层次，而且可以与多层次通配符配合使用。与多层次通配符的用法一样，单层次通配符要么只有它自己，要么要放在主题层次分隔符之后。因而， + 和 finance/+ 都是合法的，而 finance+ 是不合法的。单层次分隔符即可以放在主题树的最后，也可以放在主题树的中间。例如， finance/+ 和 finance/+/ibm 都是合法的。

**主题语义和惯用法（Topic semantics and usage）**

当你在构建一个应用时，主题树的设计可以参考以下关于主题名语法和语义的设计原则：

* 一个主题名必须不小于1个字符。
* 主题名是大小写敏感的。例如， ACCOUNTS 和 Accounts 是两个不同的主题。
* 主题名可以包含空格。例如， Accounts payable 应当是一个合法的主题名。
* 以“/”开头的主题名和不以“/”开头的相同主题名是有区别的。例如， /finance 和 finance 是不同的， /finance 匹配于 +/+ 和 /+ ， 但不匹配于 + 。
* 不要在任何主题名中包含 ｀null｀ 字符（Unicode \x0000）。

以下原则适用于主题树的构建和内容：

* 长度被限制为64K，但主题树中的层次数量没有限制。
* 可以有任意数量的根节点，也就是说，允许有任意数量的主题树。