腾讯课堂 零声教育 C/C++Linux服务器开发/高级架构师【零声教育】-学习视频教程-腾讯课堂 (qq.com)

课程源码地址：https://gitlab.0voice.com/2304_vip/mqtt_0voice
git clone git@gitlab.0voice.com:2304_vip/mqtt_0voice.git

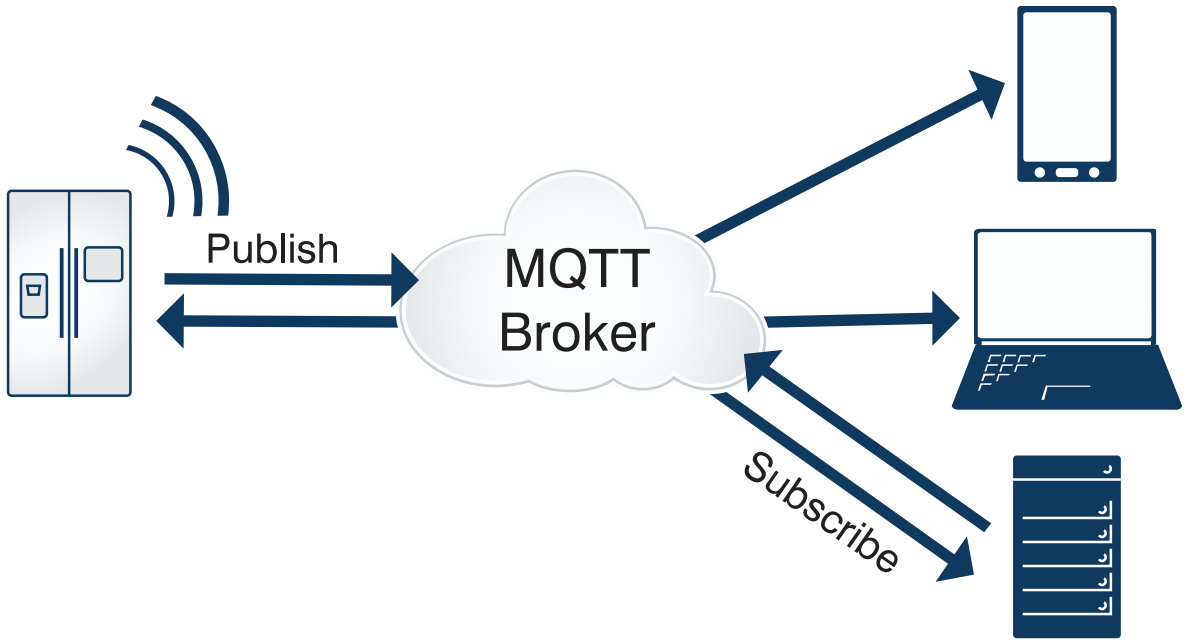
重点内容：

- 掌握mqtt开发环境搭建
- 理解mqtt发布订阅模式
- 理解QoS不同传输级别的含义
- 掌握客户端发布、订阅api实现
- 理解主题和通配符的关系
- 掌握如何设计主题

目前最成熟的mqtt方案供应商：https://www.emqx.com/

1 MQTT发布订阅模式框架

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输协议），是一种基于发布/订阅（publish/subscribe）模式的"轻量级"通讯协议，该协议构建于TCP/IP协议上，由IBM在1999年发布。MQTT最大优点在于，可以以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。作为一种低开销、低带宽占用的即时通讯协议，使其在物联网、小型设备、移动应用等方面有较广泛的应用。MQTT是一个基于客户端-服务器的消息发布/订阅传输协议。MQTT协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。



2 MQTT设计原则和主要特性

2.1 MQTT设计原则

由于物联网的环境是非常特别的，所以MQTT遵循以下设计原则：

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

2.2 MQTT主要特性

MQTT协议工作在低带宽、不可靠的网络的远程传感器和控制设备通讯而设计的协议，它具有以下主要的几项特性：

1. 使用发布/订阅消息模式，提供一对多的消息发布，解除应用程序耦合。这一点很类似于XMPP，但是MQTT的信息冗余远小于XMPP，,因为XMPP使用XML格式文本来传递数据。
2. 对负载内容屏蔽的消息传输。
3. 使用TCP/IP提供网络连接。主流的MQTT是基于TCP连接进行数据推送的，但是同样有基于UDP的版本，叫做MQTT-SN。这两种版本由于基于不同的连接方式，优缺点自然也就各有不同了。
4. 有三种消息发布服务质量：
    1. "至多一次"，消息发布完全依赖底层TCP/IP网络。会发生消息丢失或重复。这一级别可用于如下情况，环境传感器数据，丢失一次读记录无所谓，因为不久后还会有第二次发送。这一种方式主要普通APP的推送，倘若你的智能设备在消息推送时未联网，推送过去没收到，再次联网也就收不到了。
    2. "至少一次"，确保消息到达，但消息重复可能会发生。
    3. "只有一次"，确保消息到达一次。在一些要求比较严格的计费系统中，可以使用此级别。在计费系统中，消息重复或丢失会导致不正确的结果。这种最高质量的消息发布服务还可以用于即时通讯类的APP的推送，确保用户收到且只会收到一次。
5. 小型传输，开销很小（固定长度的头部是2字节），协议交换最小化，以降低网络流量。这就是为什么在介绍里说它非常适合"在物联网领域，传感器与服务器的通信，信息的收集"，要知道嵌入式设备的运算能力和带宽都相对薄弱，使用这种协议来传递消息再适合不过了。
6. 使用Last Will和Testament特性通知有关各方客户端异常中断的机制。Last Will：即遗言机制，用于通知同一主题下的其他设备发送遗言的设备已经断开了连接。Testament：遗嘱机制，功能类似于Last Will。

3 MQTT协议原理

3.1 MQTT协议实现方式 header + body

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

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

3.2 网络传输与应用消息

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

3.3 MQTT客户端

一个使用MQTT协议的应用程序或者设备，它总是建立到服务器的网络连接。客户端可以：

1. 发布其他客户端可能会订阅的信息；
2. 订阅其它客户端发布的消息；
3. 退订或删除应用程序的消息；
4. 断开与服务器连接。

3.4 MQTT服务器

MQTT服务器以称为"消息代理"（Broker），可以是一个应用程序或一台设备。它是位于消息发布者和订阅者之间，它可以：

1. 接受来自客户的网络连接；
2. 接受客户发布的应用信息；
3. 处理来自客户端的订阅和退订请求；
4. 向订阅的客户转发应用程序消息。

3.5 MQTT协议中的订阅、主题、会话

一、订阅（Subscription）
订阅包含主题筛选器（Topic Filter）和最大服务质量（QoS）。订阅会与一个会话（Session）关联。一个会话可以包含多个订阅。每一个会话中的每个订阅都有一个不同的主题筛选器。
二、会话（Session）
每个客户端与服务器建立连接后就是一个会话，客户端和服务器之间有状态交互。会话存在于一个网络之间，也可能在客户端和服务器之间跨越多个连续的网络连接。
三、主题名（Topic Name）

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

四、主题筛选器（Topic Filter）

一个对主题名**通配符筛选器**，在订阅表达式中使用，表示订阅所匹配到的多个主题。

五、负载（Payload） 和body一个意思

消息订阅者所具体接收的内容。

3.6 MQTT协议中的方法

MQTT协议中定义了一些方法（也被称为动作），来于表示对确定资源所进行操作。这个资源可以代表预先存在的数据或动态生成数据，这取决于服务器的实现。通常来说，资源指服务器上的文件或输出。主要方法有：

1. Connect。等待与服务器建立连接。
2. Disconnect。等待MQTT客户端完成所做的工作，并与服务器断开TCP/IP会话。
3. Subscribe。等待完成订阅。
4. UnSubscribe。等待服务器取消客户端的一个或多个topics订阅。
5. Publish。MQTT客户端发送消息请求，发送完成后返回应用程序线程。

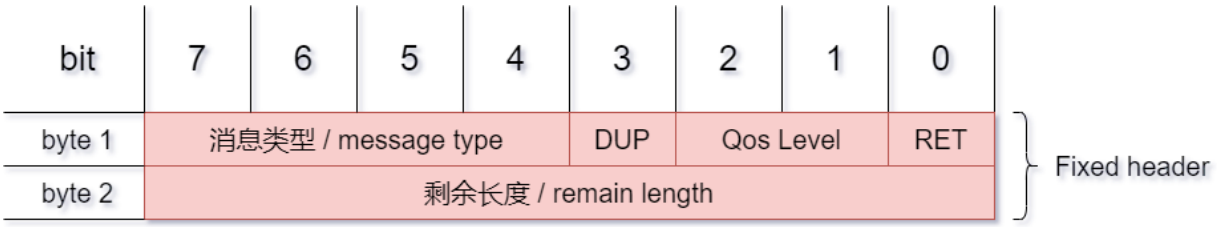
4 MQTT协议数据包结构

在MQTT协议中，一个MQTT数据包由：固定头（Fixed header）、可变头（Variable header）、消息体（payload）三部分构成。MQTT数据包结构如下：

- 固定头（Fixed header）。存在于所有MQTT数据包中，表示数据包类型及数据包的分组类标识。
- 可变头（Variable header）。存在于部分MQTT数据包中，数据包类型决定了可变头是否存在及其具体内容。
- 消息体（Payload）。存在于部分MQTT数据包中，表示客户端收到的具体内容。

4.1 MQTT固定头

固定头存在于所有MQTT数据包中，其结构如下：



4.1.1 MQTT数据包类型

位置：Byte 1中bits 7-4。
相于一个4位的无符号值，类型、取值及描述如下：

名称	值	流方向	描述
Reserved	0	不可用	保留位
CONNECT	1	客户端到服务器	客户端请求连接到服务器
CONNACK	2	服务器到客户端	连接确认
PUBLISH	3	双向	发布消息
PUBACK	4	双向	发布确认
PUBREC	5	双向	发布收到（保证第1部分到达）
PUBREL	6	双向	发布释放（保证第2部分到达）
PUBCOMP	7	双向	发布完成（保证第3部分到达）
SUBSCRIBE	8	客户端到服务器	客户端请求订阅
SUBACK	9	服务器到客户端	订阅确认
UNSUBSCRIBE	10	客户端到服务器	请求取消订阅
UNSUBACK	11	服务器到客户端	取消订阅确认
PINGREQ	12	客户端到服务器	PING请求
PINGRESP	13	服务器到客户端	PING应答
DISCONNECT	14	客户端到服务器	中断连接
Reserved	15	不可用	保留位

4.1.2 标识位

位置：Byte 1中bits 3-0。
在不使用标识位的消息类型中，标识位被作为保留位。如果收到无效的标志时，接收端必须关闭网络连接：

数据包	标识位	Bit 3	Bit 2	Bit 1	Bit 0

数据包	标识位	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	保留位	0	0	0	0
CONNACK	保留位	0	0	0	0
PUBLISH	MQTT 3.1.1使用	DUP1	QoS2	QoS2	RETAIN3
PUBACK	保留位	0	0	0	0
PUBREC	保留位	0	0	0	0
PUBREL	保留位	0	0	0	0
PUBCOMP	保留位	0	0	0	0
SUBSCRIBE	保留位	0	0	0	0
SUBACK	保留位	0	0	0	0
UNSUBSCRIBE	保留位	0	0	0	0
UNSUBACK	保留位	0	0	0	0
PINGREQ	保留位	0	0	0	0
PINGRESP	保留位	0	0	0	0
DISCONNECT	保留位	0	0	0	0

（1）DUP：发布消息的副本。用来在保证消息的可靠传输，如果设置为1，**则在下面的变长中增加MessageId**，并且需要回复确认，以保证消息传输完成，但不能用于检测消息重复发送。
（2）QoS：发布消息的服务质量，即：保证消息传递的次数

- 00：最多一次，即：<=1
- 01：至少一次，即：>=1
- 10：一次，即：=1
- 11：预留

（3）**RETAIN**： 发布保留标识，表示服务器要保留这次推送的信息，如果有新的订阅者出现，就把这消息推送给它，如果设有那么推送至当前订阅者后释放。

4.1.3 剩余长度（Remaining Length）-probuff 长度表示

地址：Byte 2。

- 位置：**固定报头中，从第2个字节开始。**
- 剩余长度等于**可变报头的长度（10字节）加上有效载荷的长度。**
- 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。

剩余长度字段 的帧格式：
帧格式 - 剩余长度字段

第1个字节		**第2个字节**		**...**
Bit 7	Bit 6:0	Bit 7	Bit 6:0	...
进位标志位	长度低字节	进位标志位	长度高字节	...

- **剩余长度字段 的字节长度**：最少1个字节，最多4个字节。
- **剩余长度字段 可以表示的长度**：1个字节时，可以表示剩余 0~127 长度。4个字节时，最大表示长度为 2^(7*4) - 1 = **2^28 - 1** = 268435455 长度

之所以1个字节不能表示 2^8 - 1 = 255长度，是因为：**每个字节的最高位 Bit7**，并不表示数据，是**进位标志位。**（base128编码）

- **示例1**：

设本帧剩余字节为 200，计算剩余长度字段。

1. 使用电脑计算器，将 200 转换为二进制 1100 1000（MSB高位在前）
2. 从**右侧低位每7Bit**进行一次拆分，依次拆分出：
    1. 第1个字节为 100 1000，有进位，高位加上进位1为 1100 1000 = 0xC8 (16进制)。
    2. 第2个字节为 1，无进位，为 1 = 0x01 (16进制)。

	第1个字节		**第2个字节**	
	Bit 7	Bit 6:0	Bit 7	Bit 6:0
	进位标志位		进位标志位	
2进制	1	**100 1000**	0	**000 0001**
2进制	1100 1000		0000 0001	
16进制	0xC8		0x01	

4.2 MQTT可变头

MQTT数据包中包含一个可变头，它驻位于固定的头和负载之间。可变头的内容因数据包类型而不同，较常的应用是作为包的标识：

Bit	**7 - 0**
byte 1	包标签符（MSB）
byte 2...	包标签符（LSB）

很多类型数据包中都包括一个2字节的数据包标识字段，这些类型的包有：PUBLISH (QoS > 0)、PUBACK、PUBREC、PUBREL、PUBCOMP、SUBSCRIBE、SUBACK、UNSUBSCRIBE、UNSUBACK。

MQTT笔记：常用的控制报文 https://blog.csdn.net/weixin_43952192/article/details/108447143

4.3 Payload消息体

Payload消息体位MQTT数据包的第三部分，包含CONNECT、SUBSCRIBE、SUBACK、UNSUBSCRIBE四种类型的消息：

1. CONNECT，消息体内容主要是：客户端的ClientID、订阅的Topic、Message以及用户名和密码。
2. SUBSCRIBE，消息体内容是一系列的要订阅的主题以及QoS。
3. SUBACK，消息体内容是服务器对于SUBSCRIBE所申请的主题及QoS进行确认和回复。
4. UNSUBSCRIBE，消息体内容是要订阅的主题。

5 QoS等级与会话

服务质量(QoS)级别是消息发送方和消息接收方之间的协议，它定义了特定消息的交付保证。MQTT中有3个QoS级别:

- QoS 0 - 最多一次
- QoS 1 - 至少一次
- QoS 2 - 恰好一次

在MQTT中讨论QoS时，需要考虑消息传递的两个方面:

1. 消息从发布客户端传递到代理。
2. 从代理到订阅客户端的消息传递

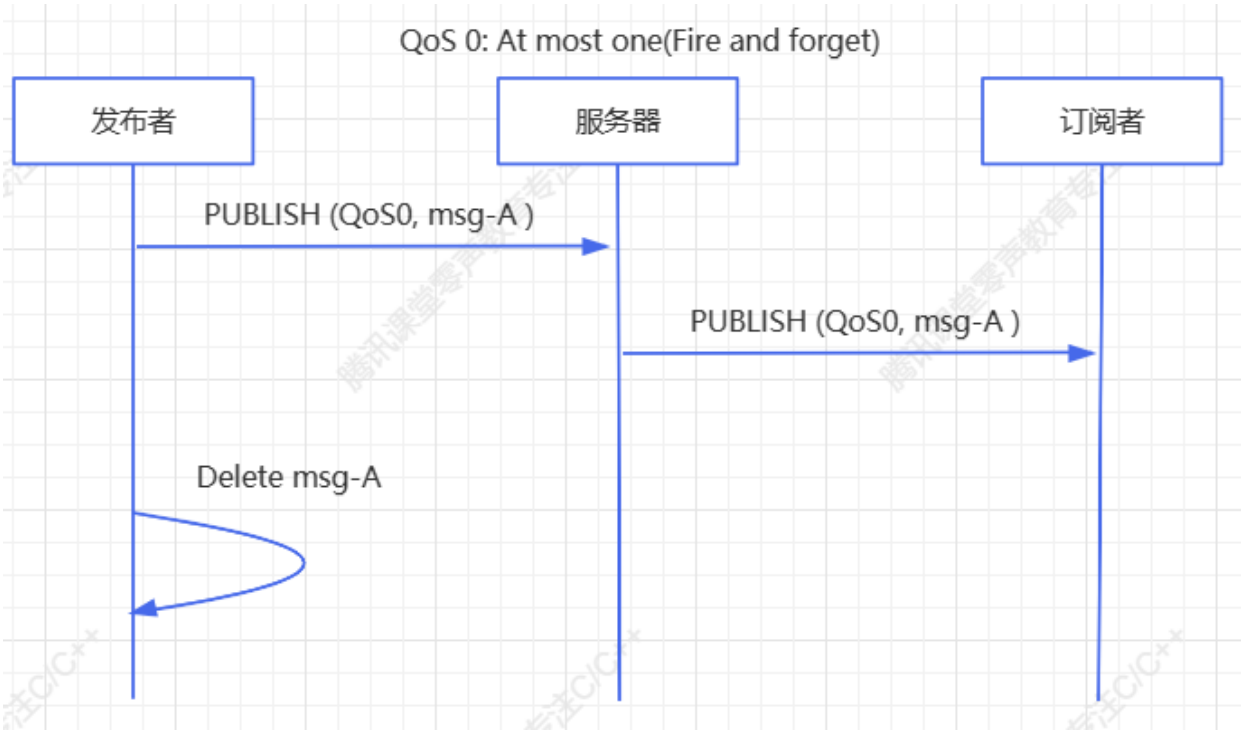
5.1 Qos不同级别的工作逻辑

QoS 0 - 最多一次

最小QoS级别为零0。这个服务级别保证了一次投递。没有交付成功的保证。接收方不确认收到消息，发送方也不存储和重新传输消息。

通信时序图

整个过程中的 `Sender` 不关心 `Receiver` 是否收到消息，它"尽力"发完消息，至于是否有人收到，它不在乎



最省资源的

两阶段：

一：发布端 -》 服务端 broker

二：服务端 broker-》订阅端

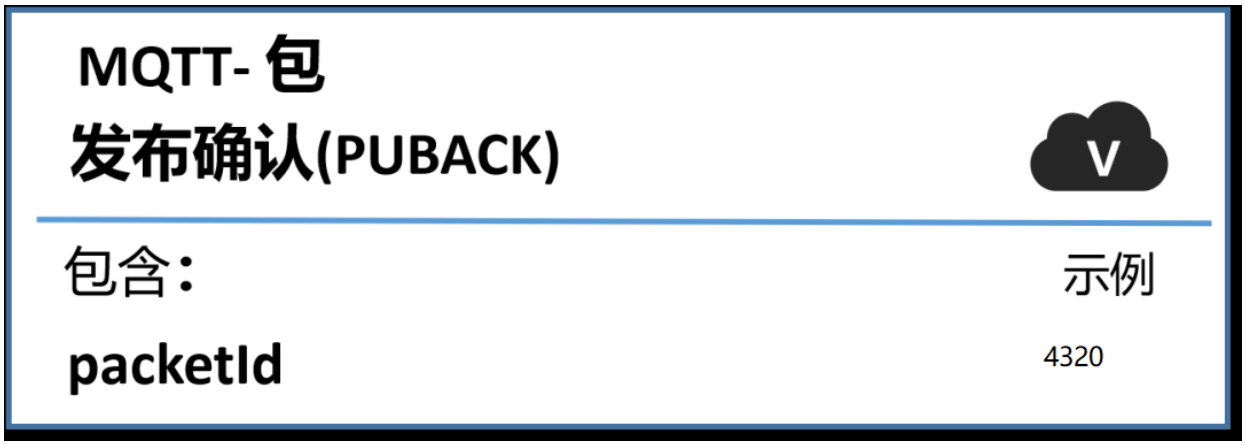
业务，tcp是没法保证

QoS 1 - 至少一次

QoS级别1保证消息至少被接收者收到一次。**发送方存储消息**，直到从接收方获得确认收到消息的发布确认(PUBACK)包为止。一条消息可能被多次发送或传递。

发送方使用每个包中的包标识符将发布(PUBLISH)包与相应的发布确认(PUBACK)包进行匹配。如果发送方在合理的时间内**没有收到发布确认(PUBACK)包**，则重新发送发布(PUBLISH)包。当接收端收到QoS为1的消息时，可以立即处理。例如，如果接收方是一个代理，则代理将消息发送到所有订阅客户端，**然后回答一个发**

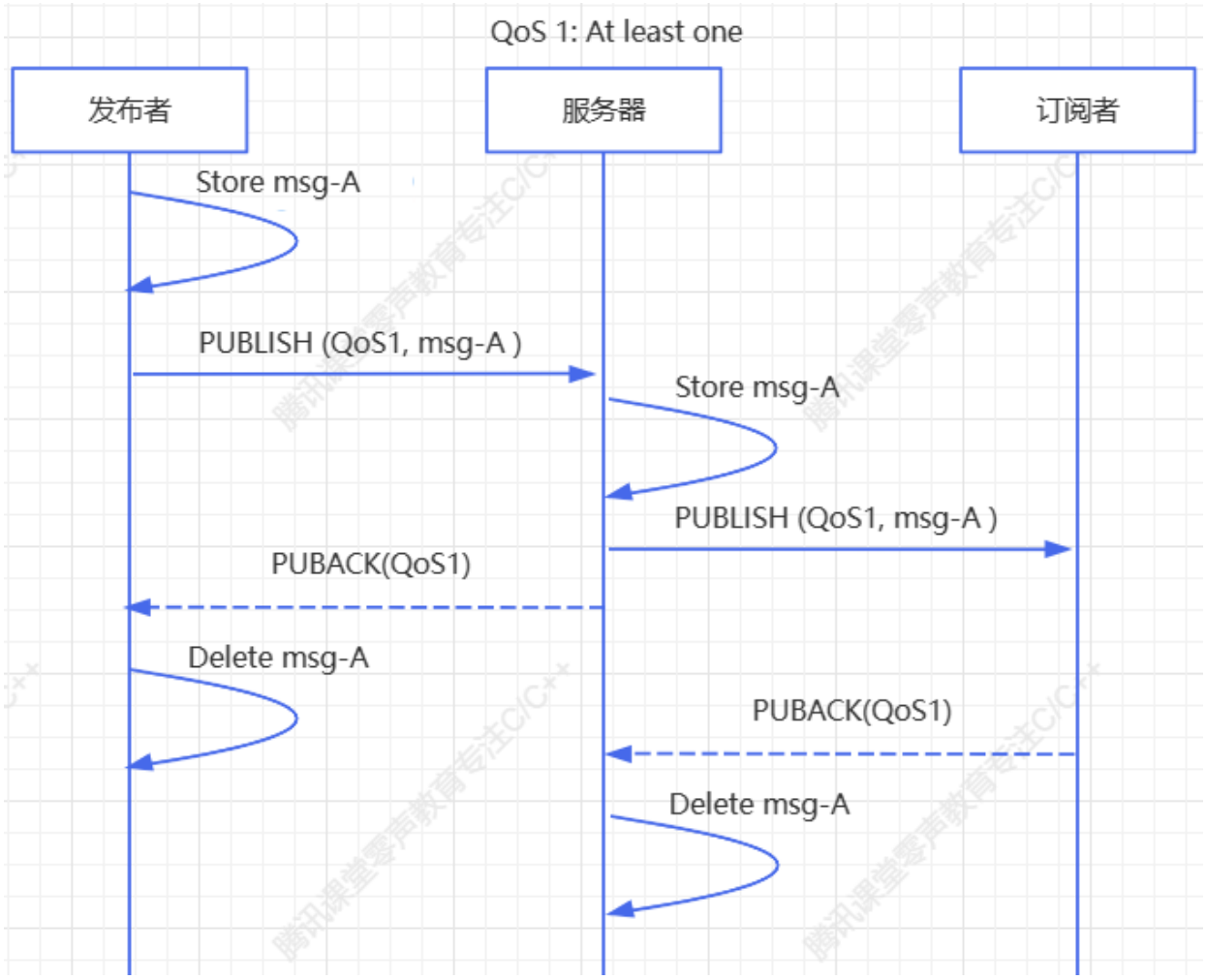
布确认(PUBACK)包。



如果发布客户端再次发送消息，它将设置一个**重复(DUP)标志。**在QoS 1中，这个DUP标志只用于内部目的，**不被代理或客户端处理。**消息的接收者发送发布确认(PUBACK)，而不考虑DUP标志。

通信时序图

此时，`Sender` 发送的一条消息，`Receiver` 至少能收到一次，也就是说 `Sender` 向 `Receiver` 发送消息，如果发送失败，会继续重试，直到 `Receiver` 收到消息为止，**但是因为重传的原因，`Receiver` 有可能会收到重复的消息**；



1）Sender 向 Receiver 发送一个带有消息数据的 PUBLISH 包， 并在本地保存这个 PUBLISH 包。

2）Receiver 收到 PUBLISH 包以后，向 Sender 发送一个 PUBACK 数据包，PUBACK 数据包没有消息体（Payload），在可变头中（Variable header）中有一个包标识（Packet Identifier），和它收到的 PUBLISH 包中的 Packet Identifier 一致。

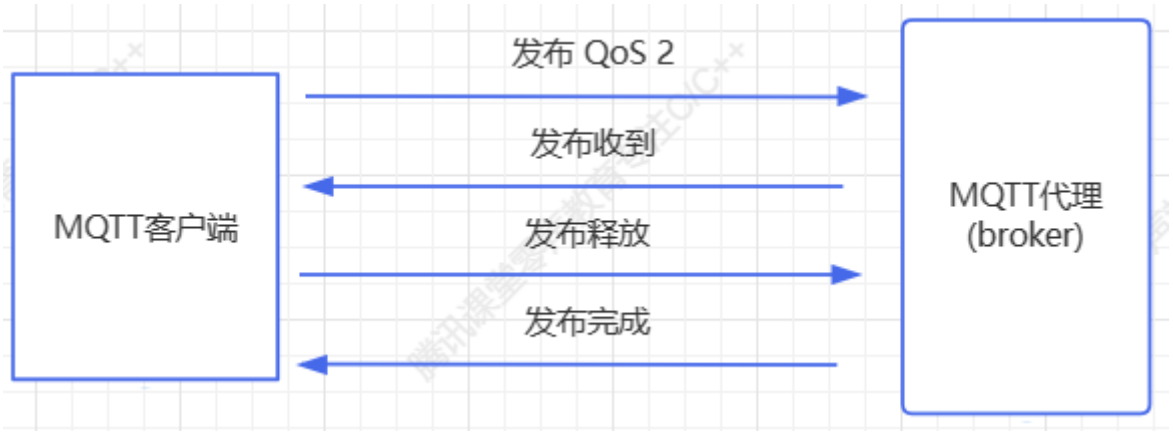
3）Sender 收到 PUBACK 之后，根据 PUBACK 包中的 Packet Identifier 找到本地保存的 PUBLISH 包，然后丢弃掉，一次消息的发送完成。

4）如果 Sender 在一段时间内没有收到 PUBLISH 包对应的 PUBACK，它将该 PUBLISH 包的 DUP 标识设为 1（代表是重新发送的 PUBLISH 包），然后重新发送该 PUBLISH 包。重复这个流程，直到收到 PUBACK，然后执行第 3 步。

QoS 2 - 恰好一次

QoS 2是MQTT中最高级别的服务。此级别保证每个消息仅被预期的收件者**接收一次**。QoS 2是服务级别中**最安全、最慢的质量**。这种保证由**发送方和接收方之间至少两个请求/响应流(四部分握手)提供**。发送方和接收方使用原始发布(PUBLISH)消息的包标识符来协调消息的传递。

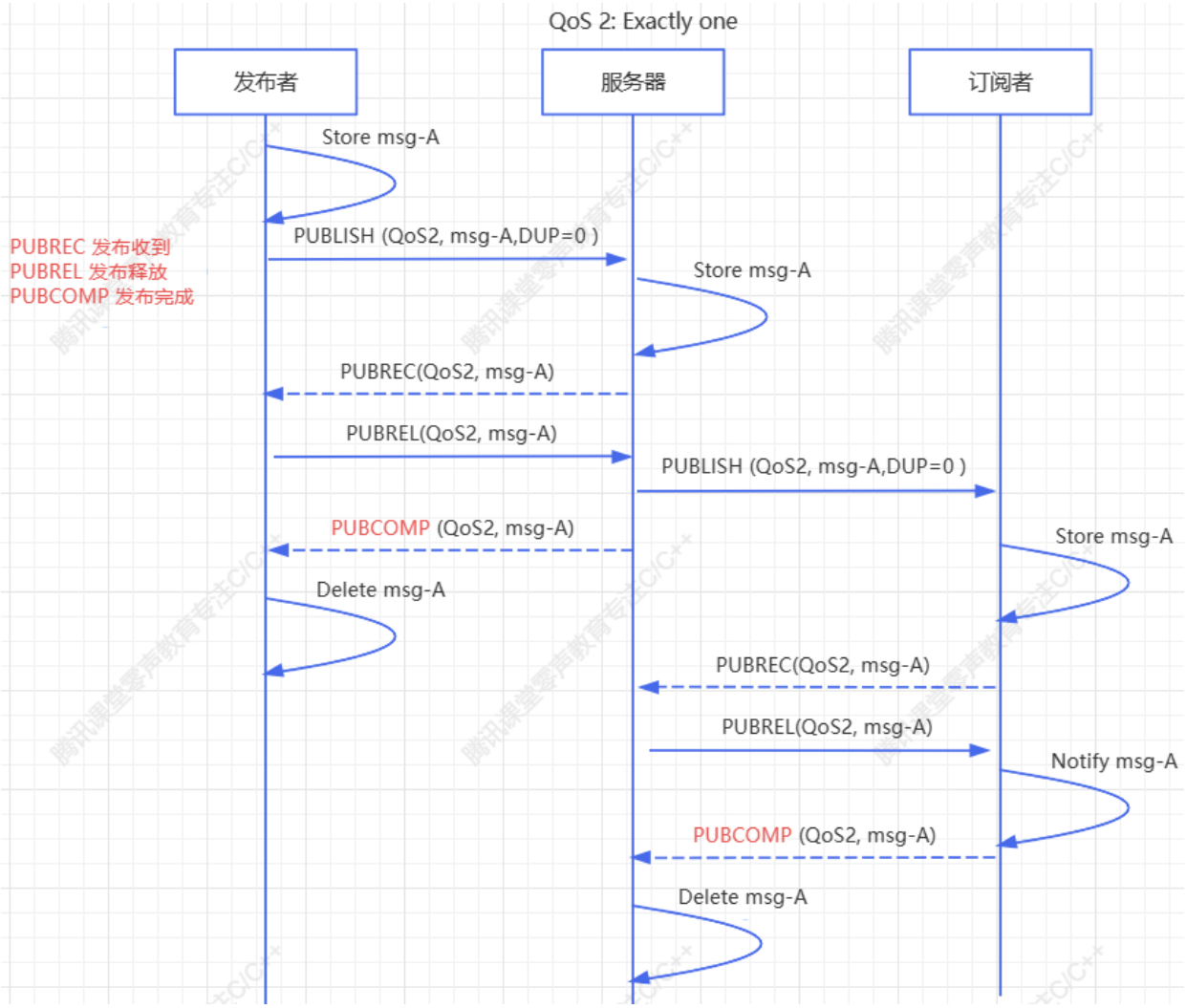
QoS2 不仅要确保 Receiver 能收到 Sender 发送的消息，还要保证消息不重复。它的重传和应答机制就要复杂一些，同时开销也是最大的。



当接收端从发送端收到QoS 2 发布(PUBLISH)包时，将对发布(PUBLISH)消息进行相应的处理，并返回一个承认该发布(PUBLISH)包的发布收到(PUBREC)包。如果发送方没有从接收方得到发布收到(PUBREC)包，它将再次发送带有重复(DUP)标志的发布(PUBLISH)包，直到它收到确认。

通信时序图

Sender 发送的一条消息，Receiver 确保能收到而且只收到一次，也就是说 Sender 尽力向 Receiver 发送消息，如果发送失败，会继续重试，直到 Receiver 收到消息为止，同时保证 Receiver 不会因为消息重传而收到重复的消息。



QoS 使用 2 套请求/应答流程（一个 4 段的握手）来确保 Receiver 收到来自 Sender 的消息，且不重复：

1）Sender 发送 QoS 为 2 的 PUBLISH 数据包，数据包 Packet Identifier 为 P，并在本地保存该 PUBLISH 包；

2）Receiver 收到 PUBLISH 数据包以后，在本地保存 PUBLISH 包的 Packet Identifier P，并回复 Sender 一个 PUBREC 数据包，PUBREC 数据包可变头中的 Packet Identifier 为 P，没有消息体（Payload）；

3）当 Sender 收到 PUBREC，它就可以安全地丢弃掉初始的 Packet Identifier 为 P 的 PUBLISH 数据包，同时保存该 PUBREC 数据包，同时回复 Receiver 一个 PUBREL 数据包，PUBREL 数据包可变头中的 Packet Identifier 为 P，没有消息体；如果 Sender 在一定时间内没有收到 PUBREC，它会把 PUBLISH 包的 DUP 标识设为 1，重新发送该 PUBLISH 数据包（Payload）；

4）当 Receiver 收到 PUBREL 数据包，它可以丢弃掉保存的 PUBLISH 包的 Packet Identifier P，并回复 Sender 一个 PUBCOMP 数据包，PUBCOMP 数据包可变头中的 Packet Identifier 为 P，没有消息体（Payload）；

5）当 Sender 收到 PUBCOMP 包，那么它认为数据包传输已完成，它会丢弃掉对应的 PUBREC 包。如果 Sender 在一定时间内没有收到 PUBCOMP 包，它会重新发送 PUBREL 数据包。

我们可以看到在 QoS2 中，完成一次消息的传递，**Sender 和 Reciever 之间至少要发送四个数据包**，QoS2 是最安全也是最慢的一种 QoS 等级了。

扣费用，算钱

5.2 QoS 和会话（Session）

客户端的会话状态包括：

- 已经发送给服务端，但是还没有完成确认的QoS 1和QoS 2级别的消息
- 已从服务端接收，但是还没有完成确认的QoS 2级别的消息。

服务端的会话状态包括：

- 会话是否存在，即使会话状态的其它部分都是空。
- 客户端的订阅信息。
- 已经发送给客户端，但是还没有完成确认的QoS 1和QoS 2级别的消息。
- 即将传输给客户端的QoS 1和QoS 2级别的消息。
- 已从客户端接收，但是还没有完成确认的QoS 2级别的消息。
- 可选，准备发送给客户端的QoS 0级别的消息。

保留消息不是服务端会话状态的一部分，会话终止时**不能**删除保留消息。
如果 Client 想接收离线消息，**必须使用持久化的会话**（CONNECT报文中可变头（byte8[1]）Clean Session = 0）连接到 Broker，这样 Broker 才会存储 Client 在离线期间没有确认**接收的 QoS 大于 1 的消息。**

5.3 QoS 等级的选择

在以下情况下你可以选择 QoS0：

- Client 和 Broker 之间的网络连接非常稳定，例如一个通过有线网络连接到 Broker 的测试用 Client；
- 可以接受丢失部分消息，比如你有一个传感器以非常短的间隔发布状态数据，所以丢一些也可以接受；
- 你不需要离线消息。

在以下情况下你应该选择 QoS1：

- 你需要接收所有的消息，而且你的应用可以接受并处理重复的消息；
- 你无法接受 QoS2 带来的额外开销，QoS1 发送消息的速度比 QoS2 快很多。

在以下情况下你应该选择 QoS2：

- 你的应用必须接收到所有的消息，而且你的应用在重复的消息下无法正常工作，同时你也能接受 QoS2 带来的额外开销。

实际上，QoS1 是应用最广泛的 QoS 等级，QoS1 发送消息的速度很快，而且能够保证消息的可靠性。虽然使用 QoS1 可能会收到重复的消息，但是在应用程序里面处理重复消息，通常并不是件难事。

6 mosquitto库中常见的函数应用总结

更多的API学习：请参考：https://mosquitto.org/api/files/mosquitto-h.html

1、int mosquitto_lib_init(void)

功能：使用mosquitto库函数前，要先初始化，使用之后就要清除。清除函数；int mosquitto_lib_cleanup();

返回值：MOSQ_ERR_SUCCESS 总是

2、int mosquitto_lib_cleanup(void)

功能：使用完mosquitto函数之后，要做清除工作。

返回值： MOSQ_ERR_SUCCESS 总是

3、struct mosquitto *mosquitto_new(const char * id, bool clean_session, void * obj)

功能：创建一个新的mosquitto客户端实例，新建客户端。

参数：

①id ： 用作客户端ID的字符串。如果为NULL，将生成一个随机客户端ID。如果id为NULL，clean_session必须为true。

②clean_session：设置为true以指示代理在断开连接时清除所有消息和订阅，设置为false以指示其保留它们，客户端将永远不会在断开连接时丢弃自己的传出消息。调用mosquitto_connect或mosquitto_reconnect将导致重新发送消息。使mosquitto_reinitialise将客户端重置为其原始状态。如果id参数为NULL，则必须将其设置为true。

简言之：就是断开后是否保留订阅信息true/false

③obj： 用户指针，将作为参数传递给指定的任何回调，（回调参数）

返回：成功时返回结构mosquitto的指针，失败时返回NULL，询问errno以确定失败的原因：

ENOMEM内存不足。

EINVAL输入参数无效。

4、void mosquitto_destroy(struct mosquitto * mosq)

功能：释放客户端

参数：mosq： struct mosquitto指针

5、void mosquitto_connect_callback_set(struct mosquitto * mosq, void (*on_connect)(struct mosquitto *mosq, void *obj, int rc))

功能：连接确认回调函数，当代理发送CONNACK消息以响应连接时，将调用此方法。

参数：

obj：mosquitto_new中提供的用户数据

rc 连接响应的返回码，其中有：

0-成功

1-连接被拒绝（协议版本不可接受）

2-连接被拒绝（标识符被拒绝）

3-连接被拒绝（经纪人不可用）

4-255-保留供将来使用

6、void mosquitto_disconnect_callback_set(struct mosquitto *mosq,void (*on_disconnect)(struct mosquitto *mosq,void *obj, int rc));

功能：断开连接回调函数，当代理收到DISCONNECT命令并断开与客户端的连接，将调用此方法。

参数：

rc：0表示客户端已经调用mosquitto_disconnect，任何其他值，表示断开连接时意外的。

7、int mosquitto_connect(struct mosquitto * mosq, const char * host, int port, int keepalive)

功能: 连接到MQTT代理/服务器（主题订阅要在连接服务器之后进行）

参数：

①mosq： 有效的mosquitto实例，mosquitto_new（）返回的mosq.

②host : 服务器ip地址

③port：服务器的端口号

④keepalive：保持连接的时间间隔， 单位秒。如果在这段时间内没有其他消息交换，则代理应该将PING消息发送到客户端的秒数。

返回：MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_ERRNO 如果系统调用返回错误。变量errno包含错误代码

8、int mosquitto_disconnect(struct mosquitto * mosq)

功能：断开与代理/服务器的连接。

返回：

MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_NO_CONN 如果客户端未连接到代理。

9、int mosquitto_publish(struct mosquitto * mosq, int * mid, const char * topic, int payloadlen, const void * payload, int qos, bool retain)

功能：主题发布的函数

参数：①mosq：有效的mosquitto实例，客户端

②mid：指向int的指针。如果不为NULL，则函数会将其设置为该特定消息的消息ID。然后可以将其与发布回调一起使用，以确定何时发送消息。请注意，尽管MQTT协议不对QoS = 0的消息使用消息ID，但libmosquitto为其分配了消息ID，以便可以使用此参数对其进行跟踪。

③topic：要发布的主题，以null结尾的字符串

④payloadlen：有效负载的大小（字节），有效值在0到268，435，455之间；主题消息的内容长度

⑤payload： 主题消息的内容，指向要发送的数据的指针，如果payloadlen >0，则它必须时有效的存储位置。

⑥qos：整数值0、1、2指示要用于消息的服务质量。

⑦retain：设置为true以保留消息。

返回:

MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_NOMEM 如果发生内存不足的情况。

MOSQ_ERR_NO_CONN 如果客户端未连接到代理。

MOSQ_ERR_PROTOCOL 与代理进行通信时是否存在协议错误。

MOSQ_ERR_PAYLOAD_SIZE 如果payloadlen太大。

MOSQ_ERR_MALFORMED_UTF8 如果主题无效，则为UTF-8

MOSQ_ERR_QOS_NOT_SUPPORTED 如果QoS大于代理支持的QoS。

MOSQ_ERR_OVERSIZE_PACKET 如果结果包大于代理支持的包。

10、int mosquitto_subscribe(struct mosquitto * mosq, int * mid, const char * sub, int qos)

功能：订阅主题函数

参数：①mosq：有效的mosquitto实例，客户端

②mid： 指向int的指针。如果不是 NULL，则该函数会将此设置为此特定消息的消息 ID。然后，这可以与发布回调一起使用，以确定消息的发送时间。请注意，尽管 MQTT 协议不对 QoS=0 的消息使用消息 ID，但 libmosquitto 会为它们分配消息 ID，以便可以使用此参数跟踪它们。

③sub： 主题名称，订阅模式。

④qos : 此订阅请求的服务质量。

返回值：

MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_NOMEM 如果发生内存不足的情况。

MOSQ_ERR_NO_CONN 如果客户端未连接到代理。

MOSQ_ERR_MALFORMED_UTF8 如果主题无效，则为UTF-8

MOSQ_ERR_OVERSIZE_PACKET 如果结果包大于代理支持的包。

11、void mosquitto_message_callback_set(struct mosquitto * mosq, void (*on_message)(struct mosquitto *, void *, const struct mosquitto_message *))

功能：消息回调函数，收到订阅的消息后调用。

参数：①mosq： 有效的mosquitto实例，客户端。

②on_message 回调函数，格式如下：void callback（struct mosquitto * mosq，void * obj，const struct mosquitto_message * message）

回调的参数：

①mosq：进行回调的mosquitto实例

②obj： mosquitto_new中提供的用户数据

③message: 消息数据，回调完成后，库将释放此变量和关联的内存，客户应复制其所需要的任何数据。

struct mosquitto_message{

int mid;//消息序号ID

char *topic; //主题

void *payload; //主题内容 ， MQTT 中有效载荷

int payloadlen; //消息的长度，单位是字节

int qos; //服务质量

bool retain; //是否保留消息

};

12、mosquitto_subscribe_callback_set()

void mosquitto_subscribe_callback_set(struct mosquitto *mosq, void (*on_subscribe)(struct mosquitto *mosq, void *obj, int mid, int qos_count, const int *granted_qos))

功能描述：设置订阅回调。当代理响应订阅请求时，将调用此函数

参数解析：

①mosq：结构体mosquitto的指针

②on_subscribe：一个回调函数

void (*on_subscribe)(struct mosquitto *mosq, void *obj, int mid, int qos_count, const int *granted_qos)

①mosq：结构体mosquitto的指针

②obj：创建客户端的回调参数，是mosquitto_new中提供的用户数据

③mid：订阅消息的消息 id

④qos_count：授予的订阅数（granted_qos的大小）

⑤granted_qos：一个整数数组，指示为每个订阅授予的 QoS

13、int mosquitto_loop_forever(struct mosquitto * mosq, int timeout, int max_packets)

功能：此函数在无限阻塞循环中为你调用loop（），对于只想在程序中运行MQTT客户端循环的情况，这很有用，如果服务器连接丢失，它将处理重新连接，如果在回调中调用mosqitto_disconnect（）它将返回。

参数：①mosq: 有效的mosquitto实例，客户端

②timeout： 超时之前，在select（）调用中等待网络活动的最大毫秒数，设置为0以立即返回，设置为负可使用默认值为1000ms。

③max_packets： 该参数当前未使用，应设为为1，以备来兼容

返回值：

MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_NOMEM 如果发生内存不足的情况。

MOSQ_ERR_NO_CONN 如果客户端未连接到代理。

MOSQ_ERR_CONN_LOST 如果与代理的连接丢失。

MOSQ_ERR_PROTOCOL 与代理进行通信时是否存在协议错误。

MOSQ_ERR_ERRNO 如果系统调用返回错误。变量errno包含错误代码

14、int mosquitto_loop_stop(struct mosquitto * mosq, bool force)

功能:网络事件阻塞回收结束处理函数，这是线程客户端接口的一部分。调用一次可停止先前使用mosquitto_loop_start创建的网络线程。该调用将一直阻塞，直到网络线程结束。为了使网络线程结束，您必须事先调用mosquitto_disconnect或将force参数设置为true。

参数：①mosq :有效的mosquitto实例

②force：设置为true强制取消线程。如果为false，则必须已经调用mosquitto_disconnect。

返回：

MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_NOT_SUPPORTED 如果没有线程支持。

15、nt mosquitto_loop_start(struct mosquitto * mosq)

功能：网络事件循环处理函数，通过创建新的线程不断调用mosquitto_loop() 函数处理网络事件，不阻塞

返回：

MOSQ_ERR_SUCCESS 成功。

MOSQ_ERR_INVAL 如果输入参数无效。

MOSQ_ERR_NOT_SUPPORTED 如果没有线程支持。

7 理解 MQTT 主题与通配符

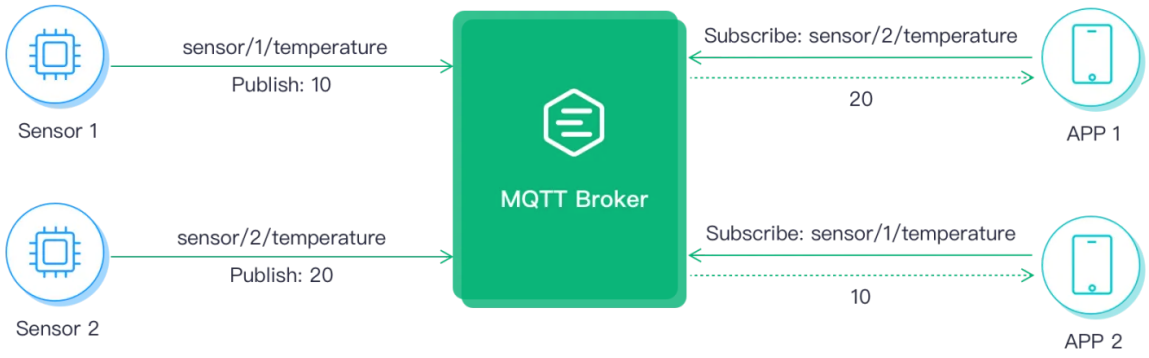
MQTT 主题本质上是一个 UTF-8 编码的字符串，是 MQTT 协议进行消息路由的基础。MQTT 主题类似 URL 路径，使用斜杠 `/` 进行分层：

```
chat/room/1
sensor/10/temperature
sensor/+/temperature
sensor/#
```

为了避免歧义且易于理解，通常不建议主题以 `/` 开头或结尾，例如 `/chat` 或 `chat/`。

不同于消息队列中的主题（比如 Kafka 和 Pulsar），**MQTT 主题不需要提前创建。**MQTT 客户端在订阅或发布时即自动的创建了主题，开发者无需再关心主题的创建，并且也不需要手动删除主题。

下图是一个简单的 MQTT 订阅与发布流程，`APP 1` 订阅了`sensor/2/temperature` 主题后，将能接收到`Sensor 2` 发布到该主题的消息。



7.1 MQTT 主题通配符

MQTT 主题通配符包含单层通配符 `+` 及多层通配符 `#`，主要用于客户端一次订阅多个主题。

> **注意**：通配符只能用于订阅，不能用于发布。

单层通配符

加号 (“+” U+002B) 是用于单个主题层级匹配的通配符。在使用单层通配符时，单层通配符必须占据整个层级，例如：

```
+ 有效
sensor/+ 有效
sensor/+/temperature 有效
sensor+ 无效（没有占据整个层级）
```

如果客户端订阅了主题 `sensor/+/temperature`，将会收到以下主题的消息：

```
sensor/1/temperature
sensor/2/temperature
...
sensor/n/temperature
```

但是不会匹配以下主题：

```
sensor/temperature
sensor/bedroom/1/temperature
```

多层通配符

井字符号（“#” U+0023）是用于匹配主题中任意层级的通配符。多层通配符表示它的父级和任意数量的子层级，在使用多层通配符时，它必须占据整个层级并且必须是主题的最后一个字符，例如：

```
# 有效，匹配所有主题
sensor/# 有效
sensor/bedroom# 无效（没有占据整个层级）
sensor/#/temperature 无效（不是主题最后一个字符）
```

如果客户端订阅主题 `senser/#`，它将会收到以下主题的消息：

```
sensor
sensor/temperature
sensor/1/temperature
```

7.2 不同场景中的主题设计

智能家居

比如我们用传感器监测卧室、客厅以及厨房的温度、湿度和空气质量，可以设计以下几个主题：

- `myhome/bedroom/temperature`
- `myhome/bedroom/humidity`
- `myhome/bedroom/airquality`
- `myhome/livingroom/temperature`
- `myhome/livingroom/humidity`
- `myhome/livingroom/airquality`
- `myhome/kitchen/temperature`
- `myhome/kitchen/humidity`
- `myhome/kitchen/airquality`

接下来，可以通过订阅 `myhome/bedroom/+` 主题获取卧室的温度、湿度及空气质量数据，订阅 `myhome/+/temperature` 主题获取三个房间的温度数据，订阅 `myhome/#` 获取所有的数据。

充电桩

充电桩的上行主题格式为 `ocpp/cp/${cid}/notify/${action}`，下行主题格式为 `ocpp/cp/${cid}/reply/${action}`。

- `ocpp/cp/cp001/notify/bootNotification`

  充电桩上线时向该主题发布上线请求。

- `ocpp/cp/cp001/notify/startTransaction`

  向该主题发布充电请求。

- `ocpp/cp/cp001/reply/bootNotification`

  充电桩上线前需订阅该主题接收上线应答。

- `ocpp/cp/cp001/reply/startTransaction`

  充电桩发起充电请求前需订阅该主题接收充电请求应答。

即时消息

- `chat/user/${user_id}/inbox`

  一对一聊天：用户上线后订阅该收件箱主题，将能接收到好友发送给自己的消息。给好友回复消息时，只需要将该主题的 `user_id` 换为好友的的 id 即可。

- `chat/group/${group_id}/inbox`

  群聊：用户加群成功后，可订阅该主题获取对应群组的消息，回复群聊时直接给该主题发布消息即可。

- `req/user/${user_id}/add`

  添加好友：可向该主题发布添加好友的申请（`user_id` 为对方的 id）。

  接收好友请求：用户可订阅该主题（`user_id` 为自己的 id）接收其他用户发起的好友请求。

- `resp/user/${user_id}/add`

  接收好友请求的回复：用户添加好友前，需订阅该主题接收请求结果（`user_id` 为自己的 id）。

  回复好友申请：用户向该主题发送消息表明是否同意好友申请（`user_id` 为对方的 id）。

- `user/${user_id}/state`

  用户在线状态：用户可以订阅该主题获取好友的在线状态

7.3 更多主题通配符问题

- 以 `$SYS/` 开头的主题为系统主题，系统主题主要用于获取 MQTT 服务器自身运行状态、消息统计、客户端上下线事件等数据。
- 共享订阅是 MQTT 5.0 引入的新特性，用于在多个订阅者之间实现订阅的负载均衡，MQTT 5.0 规定的共享订阅主题以 `$share` 开头。

具体参考：通过案例理解 MQTT 主题与通配符 | EMQ (emqx.com)

8 其他安全认证机制

通过用户名密码认证保障 MQTT 接入安全 | EMQ (emqx.com)

MQTT 5.0 中的安全认证机制：增强认证介绍 | EMQ (emqx.com)

深入解析 MQTT 中基于 Token 的认证和 OAuth 2.0 | EMQ (emqx.com)

参考

MQTT笔记：常用的控制报文 https://blog.csdn.net/weixin_43952192/article/details/108447143

MQTT 协议学习： QoS等级 与 会话 MQTT 协议学习： QoS等级 与 会话 (qq.com)

通过案例理解 MQTT 主题与通配符 | EMQ (emqx.com)

MQTT通信协议 基础(5) - 服务质量(QoS)级别 (qq.com)