第一章 MQTT协议与EMQ

学习目标

目标1：能够说出MQTT协议的概念，特点及作用

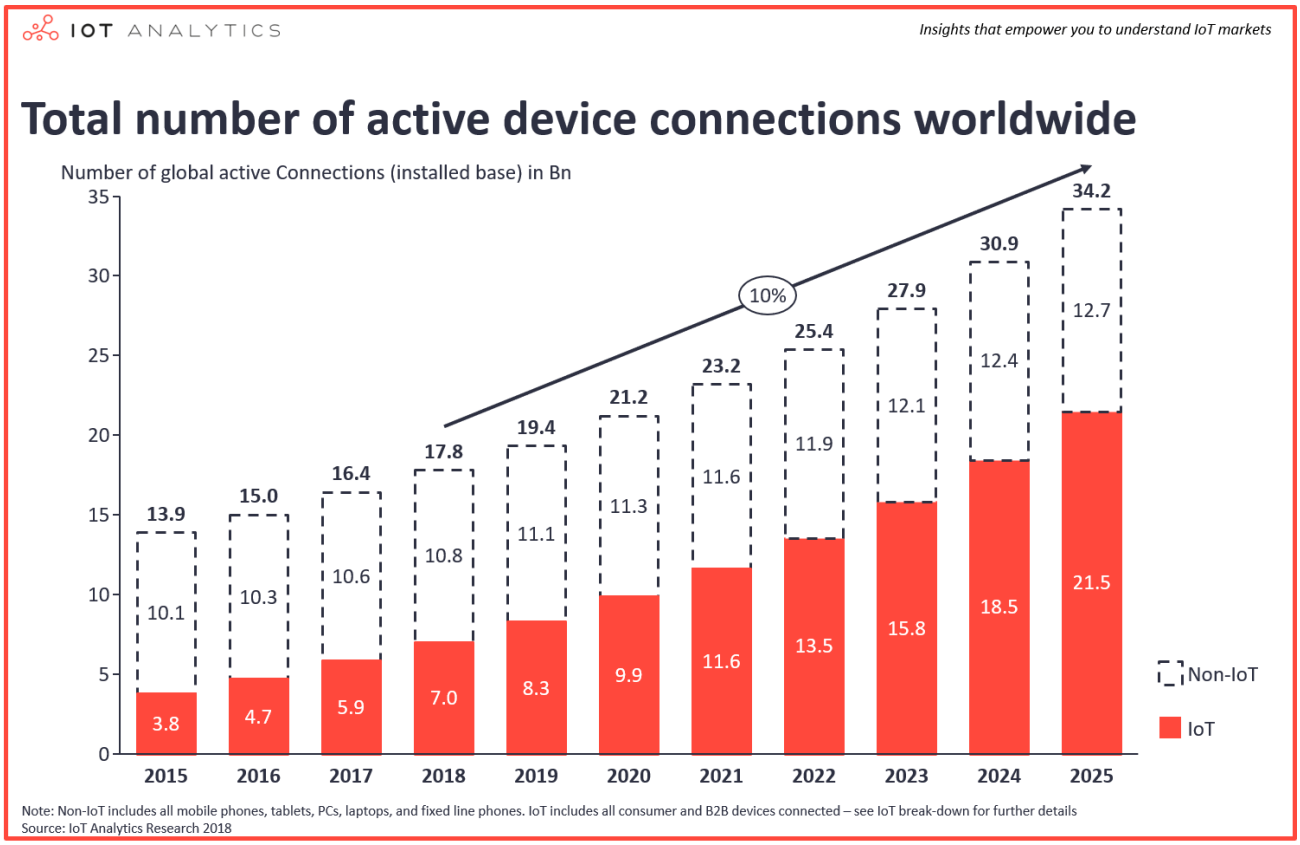
目标2：能够说出发布/订阅，代理，主题，会话，服务质量的相关概念，MQTT协议中的动作

目标3：能够说出MQTT协议的组成，固定头中的组成结构，可变头主要的组成结构

目标4：能够说出EMQ X是什么，完成EMQ X Broker环境搭建以及websocket客户端完成消息数据的收发

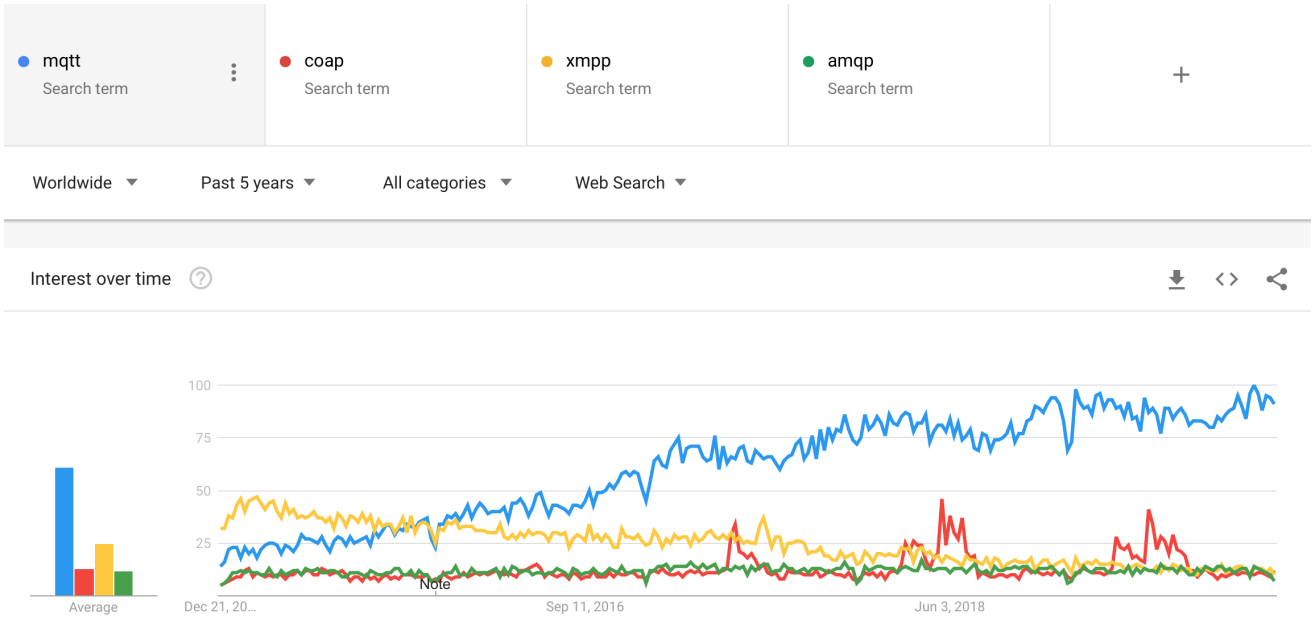
1. MQTT协议介绍

随着 5G 时代的来临，万物物联的伟大构想正在成为现实。联网的物联网设备在 2018 年已经达到了 70 亿，在未来两年，仅智能水电气表就将超过10亿



海量的设备接入和设备管理对网络带宽、通信协议以及平台服务架构都带来了很大挑战。对于物联网协议来说，必须针对性地解决物联网设备通信的几个关键问题：其网络环境复杂而不可靠、其内存和闪存容量小、其处理器能力有限。

MQTT 是基于 Publish/Subscribe 模式的物联网通信协议，凭借简单易实现、支持 QoS、报文小等特点，占据了物联网协议的半壁江山：



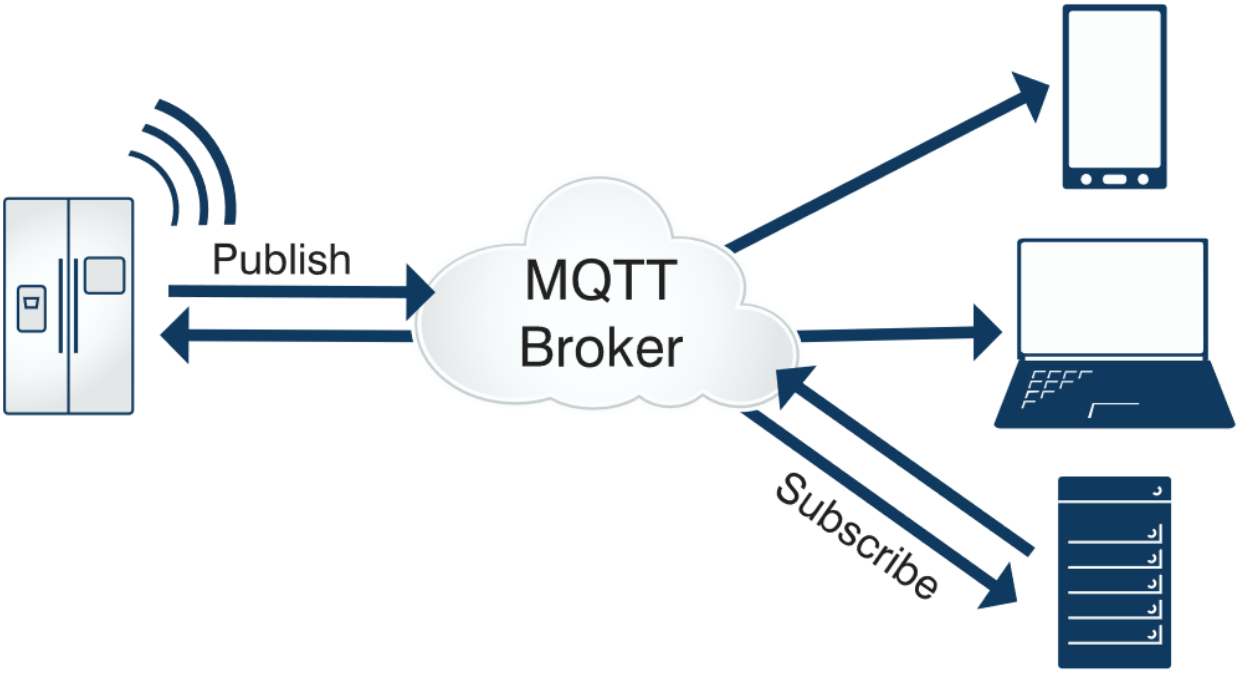
mqtt官网：http://mqtt.org/

mqtt中文网：http://mqtt.p2hp.com/

1.1 MQTT简介

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

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



1.2 MQTT协议设计规范

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，解耦Client/Server模式，带来的好处在于不必预先知道对方的存在（ip/port），不必同时运行；
- （3）允许用户动态创建主题（不需要预先创建主题），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）支持连续的会话保持和控制（心跳）；
- （7）理解客户端计算能力可能很低；
- （8）提供服务质量（*quality of service level*：QoS）管理；
- （9）不强求传输数据的类型与格式，保持灵活性（指的是应用层业务数据）。

1.3 MQTT协议主要特性

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）开放消息协议，简单易实现。
- （2）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。
- （3）对负载（协议携带的应用数据）内容屏蔽的消息传输。
- （4）基于TCP/IP网络连接,提供有序，无损，双向连接。

  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。
- （5）消息服务质量（QoS）支持，可靠传输保证；有三种消息发布服务质量：

QoS0："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。

QoS1："至少一次"，确保消息到达，但消息重复可能会发生。

QoS2："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。

- （6）1字节固定报头，2字节心跳报文，最小化传输开销和协议交换，有效减少网络流量。

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。

- （7）在线状态感知：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。

1.4 MQTT协议应用领域

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。

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

2. MQTT协议原理

2.1 MQTT协议实现方式

实现MQTT协议需要客户端和服务器端通讯完成，在通讯过程中，MQTT协议中有三种身份：发布者（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。

2.2 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。

2.3 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。

2.4 MQTT服务器端

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。

2.5 发布/订阅、主题、会话

MQTT 是基于 **发布(Publish)/订阅(Subscribe)** 模式来进行通信及数据交换的，与 HTTP 的 **请求(Request)/应答(Response)** 的模式有本质的不同。

订阅者(Subscriber) 会向 **消息服务器(Broker)** 订阅一个 **主题(Topic)** 。成功订阅后，消息服务器会将该主题下的消息转发给所有的订阅者。

主题(Topic)以 '/' 为分隔符区分不同的层级。包含通配符 '+' 或 '#' 的主题又称为 **主题过滤器(Topic Filters)**; 不含通配符的称为 **主题名(Topic Names)** 例如:

```
chat/room/1

sensor/10/temperature

sensor/+/temperature

$SYS/broker/metrics/packets/received

$SYS/broker/metrics/#
```

```
'+': 表示通配一个层级，例如a/+，匹配a/x，a/y

'#': 表示通配多个层级，例如a/#，匹配a/x，a/b/c/d

注：‘+’ 通配一个层级，’#’ 通配多个层级(必须在末尾)。
```

发布者(Publisher) 只能向 '主题名' 发布消息，订阅者(Subscriber) 则可以通过订阅 '主题过滤器' 来通配多个主题名称。

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。

2.6 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：客户端连接到服务器
- （2）CONNACK：连接确认
- （3）PUBLISH：发布消息
- （4）PUBACK：发布确认
- （5）PUBREC：发布的消息已接收
- （6）PUBREL：发布的消息已释放
- （7）PUBCOMP：发布完成
- （8）SUBSCRIBE：订阅请求
- （9）SUBACK：订阅确认
- （10）UNSUBSCRIBE：取消订阅
- （11）UNSUBACK：取消订阅确认
- （12）PINGREQ：客户端发送心跳
- （13）PINGRESP：服务端心跳响应
- （14）DISCONNECT：断开连接
- （15）AUTH：认证

3. MQTT协议数据包结构

官方文档中对于MQTT协议包的结构有着具体的说明：http://mqtt.org/documentation

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：

- （1）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，如连接，发布，订阅，心跳等。其中固定头是必须的，所有类型的MQTT协议中，都必须包含固定头。
- （2）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。可变头部不是可选的意思，而是指这部分在有些协议类型中存在，在有些协议中不存在。

- （3）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。 与可变头一样，在有些协议类型中有消息内容，有些协议类型中没有消息内容。

3.1 固定头（Fixed header）

bit / 地址	7	6	5	4	3	2	1	0
Byte 1	MQTT数据包类型				不同类型MQTT数据包的具体标识			
Byte 2…	剩余长度							

固定头存在于所有MQTT数据包中， 固定头包含两部分内容，首字节(字节1)和剩余消息报文长度(从第二个字节开始，长度为1-4字节)，剩余长度是当前包中剩余内容长度的字节数，包括变量头和有效负载中的数据）。剩余长度不包含用来编码剩余长度的字节。

剩余长度使用了一种可变长度的结构来编码，这种结构使用单一字节表示0-127的值。大于127的值如下处理。每个字节的低7位用来编码数据，最高位用来表示是否还有后续字节。因此每个字节可以编码128个值，再加上一个标识位。剩余长度最多可以用四个字节来表示。

数据包类型

位置：第一个字节(Byte 1) 中的7-4个bit位(Bit[7-4])，表示4位无符号值

通过第一个字节的高4位确定消息报文的类型，4个bit位能确定16种类型，其中0000和1111是保留字段。MQTT消息报文类型如下：

报文类型	字段值	数据方向	描述	7-4bit值
保留	0	禁用	保留	0000
CONNECT	1	Client--->Server	客户端连接到服务器	0001
CONNACK	2	Server ---> Client	连接确认	0010
PUBLISH	3	Client <--> Server	发布消息	0011
PUBACK	4	Client <--> Server	发布确认(QoS1)	0100
PUBREC	5	Client <--> Server	消息已接收(Qos2 第一阶段)	0101
PUBREL	6	Client <--> Server	消息释放(QoS2 第二阶段)	0110
PUBCOMP	7	Client <--> Server	发布结束(QoS2 第三阶段)	0111
SUBSCRIBE	8	Client ---> Server	客户端订阅请求	1000
SUBACK	9	Server--->Client	服务端订阅确认	1001
UNSUBACRIBE	10	Client ---> Server	客户端取消订阅	1010
UNSUBACK	11	Server ---> Client	服务端取消订阅确认	1011
PINGREQ	12	Client--->Server	客户端发送心跳	1100
PINGRESP	13	Server ---> Client	服务端回复心跳	1101
DISCONNECT	14	Client--->Server	客户端断开连接请求	1110
AUTH（V5.0使用）	15	Client <--> Server	认证数据交换	1111

标志位

位置：第一个字节中的0-3个bit位(Bit[3-0])。意思是字节位Bit[3-0]用作报文的标识。

首字节的低4位(bit3~bit0)用来表示某些报文类型的控制字段，实际上只有少数报文类型有控制位，如下图：

报文类型	固定头标记	Bit 3	Bit2	BIT1	Bit 0
CONNECT	保留	0	0	0	0
CONNACK	保留	0	0	0	0
PUBLISH	Used in MQTT3.1.1(V5.0可从官网查看)	DUP	Qos	QoS	RETAIN
PUBACK	保留	0	0	0	0
PUBREC	保留	0	0	0	0
PUBREL	保留	0	0	1	0
PUBCOMP	保留	0	0	0	0
SUBSCRIBE	保留	0	0	1	0
SUBACK	保留	0	0	0	0
UNSUBACRIBE	保留	0	0	1	0
UNSUBACK	保留	0	0	0	0
PINGREQ	保留	0	0	0	0
PINGRESP	保留	0	0	0	0
DISCONNECT	保留	0	0	0	0

（1）：其中Bit[3]为DUP字段，如果该值为1，表明这个数据包是一条重复的消息；否则该数据包就是第一次发布的消息。

（2）：Bit[2-1]为Qos字段：

如果Bit 1和Bit 2都为0，表示QoS 0：至多一次；

如果Bit 1为1，表示QoS 1：至少一次；

如果Bit 2为1，表示QoS 2：只有一次；

如果同时将Bit 1和Bit 2都设置成1，那么客户端或服务器认为这是一条非法的消息，会关闭当前连接。

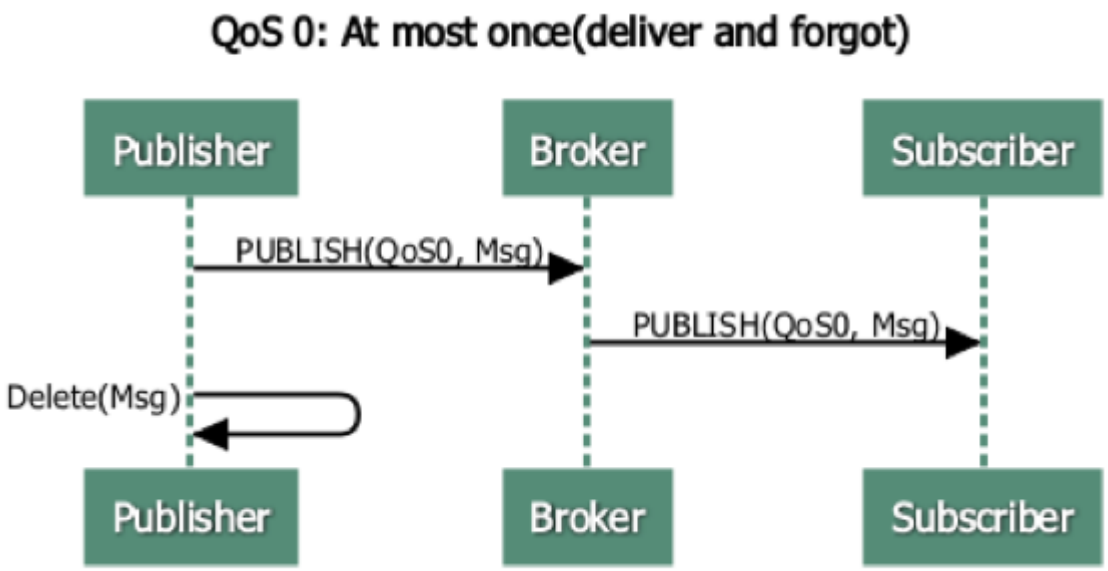
> 目前Bit[3-0]只在PUBLISH协议中使用有效，并且表中指明了是MQTT 3.1.1版本。对于其它MQTT协议版本，内容可能不同。所有固定头标记为"保留"的协议类型，Bit[3-0]必须保持与表中保持一致，如SUBSCRIBE协议，其Bit 1必须为1。如果接收方接收到非法的消息，会强行关闭当前连接。

MQTT消息QoS

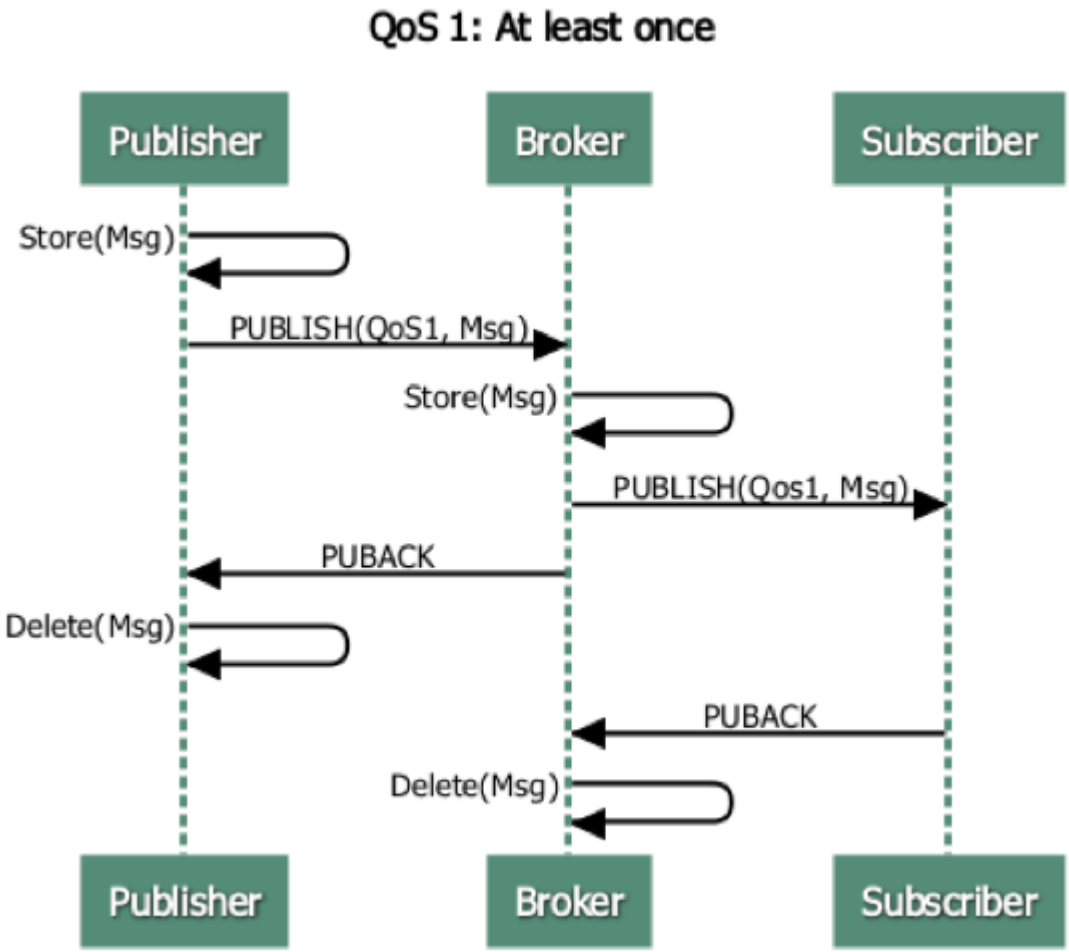
MQTT发布消息服务质量保证（QoS）不是端到端的，是客户端与服务器之间的。订阅者收到MQTT消息的QoS级别，最终取决于发布消息的QoS和主题订阅的QoS。

发布消息的QoS	主题订阅的QoS	接收消息的QoS
0	0	0
0	1	0
0	2	0
1	0	0
1	1	1
1	2	1
2	0	0
2	1	1
2	2	2

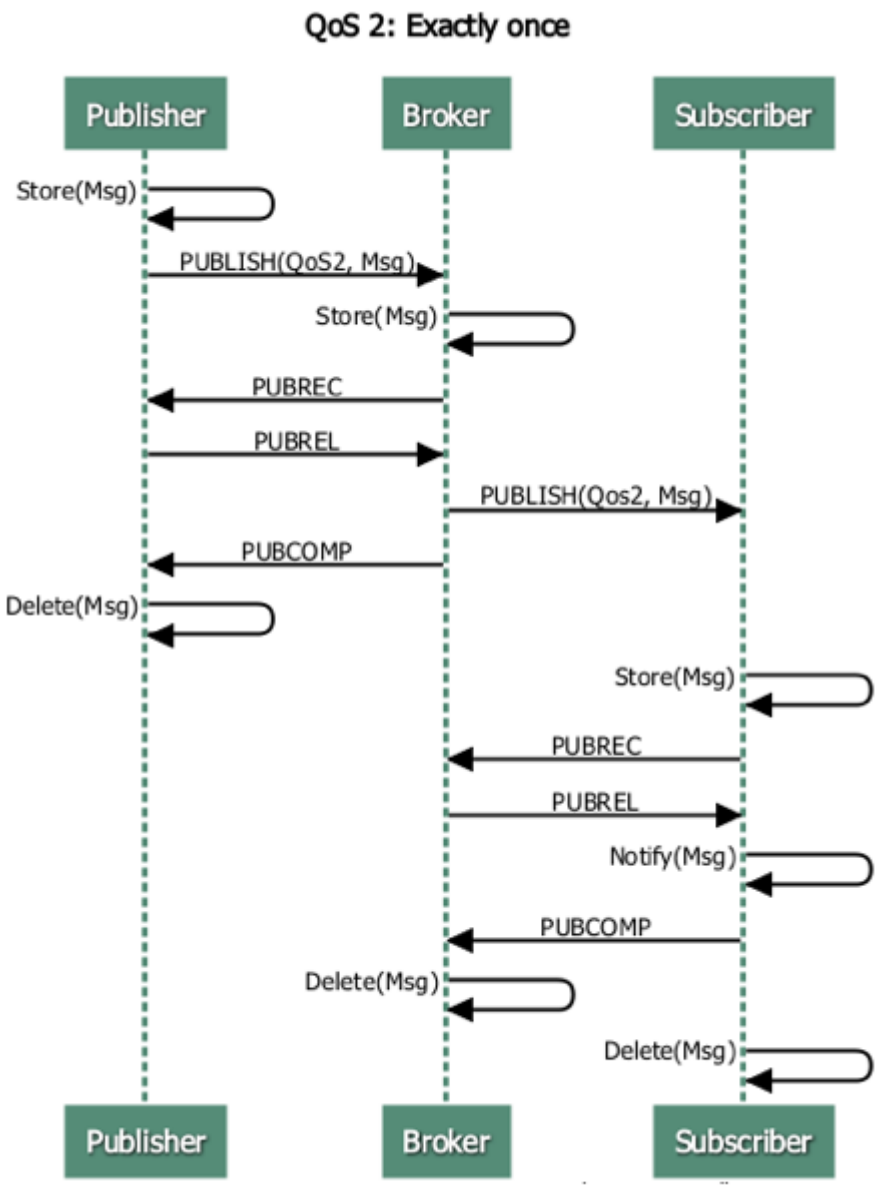
- Qos0消息发布订阅



- Qos1消息发布订阅


QoS 1: At least once
Publisher
Broker
Subscriber
Store(Msg)
PUBLISH(QoS1, Msg)
Store(Msg)
PUBLISH(Qos1, Msg)
PUBACK
Delete(Msg)
PUBACK
Delete(Msg)
Publisher
Broker
Subscriber

- Qos2消息发布订阅



（3）：Bit[0]为RETAIN字段，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。

3.2 可变头(Variable Header)

可变头的意思是可变化的消息头部。有些报文类型包含可变头部有些报文则不包含。可变头部在固定头部和消息内容之间，其内容根据报文类型不同而不同。

bit	7	6	5	4	3	2	1	0	存在的报文类型
6 Byte	Protocol name（协议名） (UTF-8字符串“MQTT”)								CONNECT
1 Byte	Protocol Version（版本号）								CONNECT
1 Byte	Connect flags（连接标记）								CONNECT
	User Name Flag	Password Flag	Will Retain	Will Qos		Will Flag	Clean Session	Reserved	
2 Byte	Keep Alive timer（心跳时长）								CONNECT
1 Byte	连接确认标识 7-1位是保留位必须设置为0，0是session present)								CONNACK
1 Byte	Connect return code（连接返回码）								CONNACK
3~32767 Byte	Topic name（主题名称）								PUBLISH
2 byte	Message ID（消息ID）								PUBLISH(Qos>0), PUBACK, PUBREC,PUBREL, PUBCOMP,SUBSCRIBE, SUBACK,UNSUBSCRIBE ,UNSUBACK

协议名

协议名是表示协议名*MQTT*的UTF-8编码的字符串。MQTT规范的后续版本不会改变这个字符串的偏移和长度。

	说明	7 6 5 4 3 2 1 0
协议名		
byte 1	长度MSB (0)	0 0 0 0 0 0 0 0
byte 2	长度LSB (4)	0 0 0 0 0 1 0 0
byte 3	‘M’	0 1 0 0 1 1 0 1
byte 4	‘Q’	0 1 0 1 0 0 0 1
byte 5	‘T’	0 1 0 1 0 1 0 0
byte 6	‘T’	0 1 0 1 0 1 0 0

支持多种协议的服务端使用协议名字段判断数据是否为MQTT报文。协议名**必须**是UTF-8字符串“MQTT”。如果服务端不愿意接受CONNECT但希望表明其MQTT服务端身份，**可以**发送包含原因码为0x84（不支持的协议版本）的CONNACK报文，然后**必须**关闭网络连接

协议版本

位无符号值表示客户端的版本等级。3.1.1版本的协议等级是4，MQTT v5.0的协议版本字段为5（0x05）

MQTT会话(Clean Session)

MQTT客户端向服务器发起CONNECT请求时，可以通过‘Clean Session’标志设置会话。

‘Clean Session’设置为0，表示创建一个持久会话，在客户端断开连接时，会话仍然保持并保存离线消息，直到会话超时注销。

‘Clean Session’设置为1，表示创建一个新的临时会话，在客户端断开时，会话自动销毁。

Will Flag/Will Qos/Will Retain

如果Will Flag被设置为1，这意味着，如果连接请求被接受，服务端必须存储一个Will Message，并和网络连接关联起来。之后在网络连接断开的时候必须发布Will Message，除非服务端收到DISCONNECT包删掉了Will Message

Will Message会在某些情况下发布，包括但不限于：

- 服务端发现I/O错误或网络失败。
- 客户端在Keep Alive时间内通信失败。
- 客户端没有发送DISCONNECT包就关闭了网络连接。
- 服务端因协议错误关闭了网络连接。

如果Will Flag被设置为1，连接标识中的Will QoS和Will Retain字段将会被服务端用到

Will QoS这两个bit表示发布Will Message时使用QoS的等级

Will Retain这个bit表示Will Message在发布之后是否需要保留。

如果Will Flag设置为0，那么Will Retain必须是0

如果Will Flag设置为1：

- 如果Will Retain设置为0，那么服务端必须发布Will Message，不必保存
- 如果Will Retain设置为1，那么服务端必须发布Will Message，并保存

User Name Flag

如果User Name Flag设置为0，那么用户名不必出现在载荷中

如果User Name Flag设置为1，那么用户名必须出现在载荷中

Password Flag

如果Password Flag设置为0，那么密码不必出现在载荷中

如果Password Flag设置为1，那么密码必须出现在载荷中

如果User Name Flag设置为0，那么Password Flag必须设置为0

MQTT连接保活心跳

心跳的作用：

```
PINGREQ包从客户端发往服务端，可以用来：
1：在没有其他控制包从客户端发送给服务端的时候，告知服务端客户端的存活状态。
2：请求服务端响应，来确认服务端是否存活。
3：确认网络连接的有效性。
PINGRESP包从服务端发送给客户端来响应PINGREQ包。它代表服务端是存活的。
```

MQTT客户端向服务器发起CONNECT请求时，通过KeepAlive参数设置保活周期。

Keep Alive是以秒为单位的时间间隔。用2字节表示，它指的是客户端从发送完成一个控制包到开始发送下一个的最大时间间隔。客户端有责任确保两个控制包发送的间隔不能超过Keep Alive的值。如果没有其他控制包可发，客户端必须发送PINGREQ包

> 客户端可以在任何时间发送PINGREQ包，不用关心Keep Alive的值，用PINGRESP来判断与服务端的网络连接是否正常。
>
> 如果Keep Alive的值非0，而且服务端在一个半Keep Alive的周期内没有收到客户端的控制包，服务端必须作为网络故障断开网络连接
>
> 如果客户端在发送了PINGREQ后，在一个合理的时间都没有收到PINGRESP包，客户端应该关闭和服务端的网络连接。
>
> Keep Alive的值为0，就关闭了维持的机制。这意味着，在这种情况下，服务端不会断开静默的客户端。
>
> **MQTT遗愿消息(Last Will)**
>
> MQTT客户端向服务器端CONNECT请求时，可以设置是否发送遗愿消息(Will Message)标志，和遗愿消息主题(Topic)与内容(Payload)。
>
> MQTT客户端异常下线时(客户端断开前未向服务器发送DISCONNECT消息)，MQTT消息服务器会发布遗愿消息。

常见的一种可变头比如：**Packet Identifier(消息ID)**

一个消息ID包含2字节，高字节在前，低字节在后。包含Packet Identifier的协议类型包括：

PUBLISH(QoS > 0)、 PUBACK 、 PUBREC 、 PUBREL 、 PUBCOMP 、 SUBSCRIBE 、 SUBACK 、 UNSUBSCRIBE 、 UNSUBACK

消息ID默认是从1开始并自增，如果一个消息ID被用完后，这个消息ID可以被重用。对于PUBLISH (QoS 1)来说，如果发送端接收到PUBACK，那么这个消息ID就用完了。对于PUBLISH(QoS 2)，如果接收方收到PUBCOMP，那么这个消息ID就用完了。对于SUBSCRIBE和UNSUBSCRIBE，消息ID使用完成的标记是发送方收到了对应的SUBACK和UNSUBACK。

另外客户端和服务端的消息ID是独立分配的，客户端和服务端可以同时使用同一个消息ID。

3.3 消息体(Payload)

有些报文类型是包含Payload的，Payload意思是消息载体的意思

如PUBLISH的Payload就是指消息内容（应用程序发布的消息内容）。而CONNECT的Payload则包含Client Identifier，Will Topic，Will Message，Username，Password等信息。

包含payload的报文类型如下：

报文类型	是否需要payload
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None
PUBREC	None
PUBREL	None
PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required
UNSUBACK	Required
PINGREQ	None
PINGRESP	None
DISCONNECT	None

总结：我们介绍了MQTT协议的消息格式，MQTT消息格式包含Fixed Header， Variable Header和Payload。因为MQTT消息格式非常精简，所以可以高效的传输数据。

Fixed Header中包含首字节，高4位用来表示报文类型，低4位用于类型控制。目前只有PUBLISH使用了类型控制字段。其它控制字段被保留并且必须与协议定义保持一致。

Fixed Header同时包含Remaining Length，这是剩余消息长度，最大长度为4字节，理论上一条MQTT最大可以传输256MB数据。Remaining Length=Variable Header+Payload长度。

Variable Header是可变头部，有些报文类型中需要包含可变头部，可变头部根据报文类型不同而不同。比如Packet Identifier在发布，订阅/取消订阅等报文中都使用到。

Payload是消息内容，也只在某些报文类型中出现，其内容和格式也根据报文类型不同而不同。

4. EMQX简介

MQTT属于是物联网的通信协议，在MQTT协议中有两大角色：客户端（发布者/订阅者），服务端（Mqtt broker）；针对客户端和服务端需要有遵循该协议的的具体实现，EMQ/EMQ X就是MQTT Broker的一种实现。

EMQ官网：<https://www.emqx.io/cn/

4.1 EMQ X是什么

EMQ X 基于 Erlang/OTP 平台开发的 MQTT 消息服务器，是开源社区中最流行的 MQTT 消息服务器。

EMQ X 是开源百万级分布式 MQTT 消息服务器（MQTT Messaging Broker），用于支持各种接入标准 MQTT 协议的设备，实现从设备端到服务器端的消息传递，以及从服务器端到设备端的设备控制消息转发。从而实现物联网设备的数据采集，和对设备的操作和控制。

4.2 为什么选择EMQ X

到目前为止，比较流行的 MQTT Broker 有几个：

1. Eclipse Mosquitto: https://github.com/eclipse/mosquitto

   使用 C 语言实现的 MQTT Broker。Eclipse 组织还还包含了大量的 MQTT 客户端项目：https://www.eclipse.org/paho/#

2. EMQX: https://github.com/emqx/emqx

   使用 Erlang 语言开发的 MQTT Broker，支持许多其他 IoT 协议比如 CoAP、LwM2M 等

3. Mosca: https://github.com/mcollina/mosca

   使用 Node.JS 开发的 MQTT Broker，简单易用。

4. VerneMQ: https://github.com/vernemq/vernemq

   同样使用 Erlang 开发的 MQTT Broker

从支持 MQTT5.0、稳定性、扩展性、集群能力等方面考虑，EMQX 的表现应该是最好的。

与别的MQTT服务器相比**EMQ X 主要有以下的特点：**

- 经过100+版本的迭代，EMQ X 目前为开源社区中最流行的 MQTT 消息中间件，在各种客户严格的生产环境上经受了严苛的考验；
- EMQ X 支持丰富的物联网协议，包括 MQTT、MQTT-SN、CoAP、 LwM2M、LoRaWAN 和 WebSocket 等；
- 优化的架构设计，支持超大规模的设备连接。企业版单机能支持百万的 MQTT 连接；集群能支持千万级别的 MQTT 连接；
- 易于安装和使用；
- 灵活的扩展性，支持企业的一些定制场景；
- 中国本地的技术支持服务，通过微信、QQ等线上渠道快速响应客户需求；
- 基于 Apache 2.0 协议许可，完全开源。EMQ X 的代码都放在 Github 中，用户可以查看所有源代码。
- EMQ X 3.0 支持 MQTT 5.0 协议，是开源社区中第一个支持 5.0协议规范的消息服务器，并且完全兼容 MQTT V3.1 和 V3.1.1 协议。除了 MQTT 协议之外，EMQ X 还支持别的一些物联网协议
- 单机支持百万连接，集群支持千万级连接；毫秒级消息转发。EMQ X 中应用了多种技术以实现上述功能，
  - 利用 Erlang/OTP 平台的软实时、高并发和容错（电信领域久经考验的语言）
  - 全异步架构
  - 连接、会话、路由、集群的分层设计
  - 消息平面和控制平面的分离等
- 扩展模块和插件，EMQ X 提供了灵活的扩展机制，可以实现私有协议、认证鉴权、数据持久化、桥接转发和管理控制台等的扩展
- 桥接：EMQ X 可以跟别的消息系统进行对接，比如 EMQ X Enterprise 版本中可以支持将消息转发到 Kafka、RabbitMQ 或者别的 EMQ 节点等

- 共享订阅：共享订阅支持通过负载均衡的方式在多个订阅者之间来分发 MQTT 消息。比如针对物联网等数据采集场景，会有比较多的设备在发送数据，通过共享订阅的方式可以在订阅端设置多个订阅者来实现这几个订阅者之间的工作负载均衡

4.3 EMQ X 与物联网平台的关系是什么

典型的物联网平台包括设备硬件、数据采集、数据存储、分析、Web / 移动应用等。EMQ X 位于数据采集这一层，分别与硬件和数据存储、分析进行交互，是物联网平台的核心：前端的硬件通过 MQTT 协议与位于数据采集层的 EMQ X 交互，通过 EMQ X 将数据采集后，通过 EMQ X 提供的数据接口，将数据保存到后台的持久化平台中（各种关系型数据库和 NOSQL 数据库），或者流式数据处理框架等，上层应用通过这些数据分析后得到的结果呈现给最终用户。

4.4 EMQ X 有哪些产品

EMQ X 公司主要提供三个产品，可在官网首页产品导航查看每一种产品；主要体现在支持的连接数量、产品功能和商业服务等方面的区别：

- EMQ X Broker：EMQ X 开源版，完整支持 MQTT V3.1.1/V5.0 协议规范，完整支持 TCP、TLS、WebSocket 连接，支持百万级连接和分布式集群架构；LDAP, MySQL, Redis, MongoDB 等扩展插件集成，支持插件模式扩展服务器功能；支持跨 Linux、Windows、macOS 平台安装，支持公有云、私有云、K8S/容器部署
- EMQ X Enterprise：EMQ X 企业版，在开源版基础上，支持物联网主流协议 MQTT、MQTT-SN、CoAP/LwM2M、HTTP、WebSocket 一站式设备接入；JT-808/GBT-32960 等行业协议支持，基于 TCP/UDP 私有协议的旧网设备接入兼容，多重安全机制与认证鉴权；高并发软实时消息路由；强大灵活的内置规则引擎；企业服务与应用集成；多种数据库持久化支持；消息变换桥接转发 Kafka；管理监控中心
- EMQ X Platform：EMQ X 平台版，EMQ X Platform 是面向千万级超大型 IoT 网络和应用，全球首选电信级物联网终端接入解决方案。千万级大容量；多物联网协议；电信级高可靠；卓越 5G 网络支持；跨云跨 IDC 部署；兼容历史系统；完善的咨询服务（从咨询到运维）

4.5 EMQ X 消息服务器功能列表

- 完整的 MQTT V3.1/V3.1.1 及 V5.0 协议规范支持
  - QoS0, QoS1, QoS2 消息支持
  - 持久会话与离线消息支持
  - Retained 消息支持
  - Last Will 消息支持
- TCP/SSL 连接支持
- MQTT/WebSocket/SSL 支持
- HTTP 消息发布接口支持
- $SYS/# 系统主题支持
- 客户端在线状态查询与订阅支持
- 客户端 ID 或 IP 地址认证支持
- 用户名密码认证支持
- LDAP 认证
- Redis、MySQL、PostgreSQL、MongoDB、HTTP 认证集成
- 浏览器 Cookie 认证

- 基于客户端 ID、IP 地址、用户名的访问控制 (ACL)
- 多服务器节点集群 (Cluster)
- 支持 manual、mcast、dns、etcd、k8s 等多种集群发现方式
- 网络分区自动愈合
- 消息速率限制
- 连接速率限制
- 按分区配置节点
- 多服务器节点桥接 (Bridge)
- MQTT Broker 桥接支持
- Stomp 协议支持
- MQTT-SN 协议支持
- CoAP 协议支持
- Stomp/SockJS 支持
- 延时 Publish ($delay/topic)
- Flapping 检测
- 黑名单支持
- 共享订阅 ($share/:group/topic)
- TLS/PSK 支持
- 规则引擎
    - 空动作 (调试)
    - 消息重新发布
    - 桥接数据到 MQTT Broker
    - 检查 (调试)
    - 发送数据到 Web 服务

4.6 EMQ X服务端环境搭建与配置

4.6.1 安装

EMQ X 目前支持的操作系统:

- Centos6
- Centos7
- OpenSUSE tumbleweed
- Debian 8
- Debian 9
- Debian 10
- Ubuntu 14.04
- Ubuntu 16.04
- Ubuntu 18.04
- macOS 10.13
- macOS 10.14
- macOS 10.15

- Windows Server 2019

产品部署建议 Linux 服务器，不推荐 Windows 服务器。

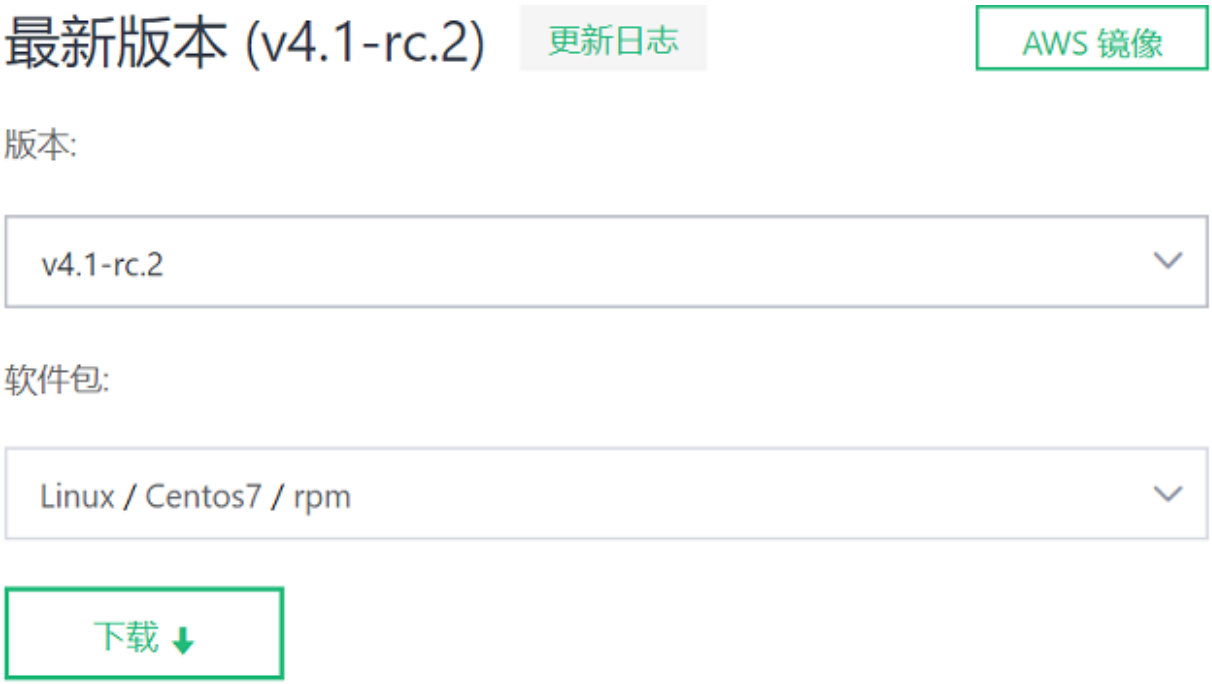
安装的方式有很多种，可供自由选择：

Shell脚本安装、包管理器安装、二进制包安装、ZIP压缩包安装、Homebrew安装、Docker运行安装、Helm安装、源码编译安装

我们在课程资料中所给的CentOS7的虚拟机上进行安装EMQ X broker：

二进制包安装

从该地址下载最新版本：https://www.emqx.io/cn/downloads#broker



已在今日课程资料中提供，然后上传到虚拟机，接着进行安装

1：执行如下命令执行安装

```
[root@docker emqx]# rpm -ivh emqx-centos7-v4.0.5.x86_64.rpm
```

2：安装完成后直接使用如下命令启动emqx

```
[root@docker emqx]# emqx start
EMQ X Broker v4.0.5 is started successfully!
```

3：查看emqx broker的启动状态

```
[root@docker emqx]# emqx_ctl status
Node 'emqx@127.0.0.1' is started
emqx v4.0.5 is running
```

EMQ X broker提供了Dashboard 以方便用户管理设备与监控相关指标，启动后我们通过访问服务端18083端口

地址：http://192.168.200.129:18083

默认用户名：admin，默认密码：public

4：停止emqx broker请使用如下命令

```
[root@docker emqx]# emqx stop
ok
```

5：卸载 EMQ X Broker

```
[root@docker emqx]# rpm -e emqx
```

Docker运行安装

在EMQ X Broker下载页面直接提供的有基于docker的安装命令

1：首先拉取emqx的镜像

```
[root@docker emqx]# docker pull emqx/emqx:v4.0.5
```

2：使用docker命令运行得到docker容器

```
[root@docker emqx]# docker run -tid --name emqx -p 1883:1883 -p 8083:8083 -p 8081:8081 -p 8883:8883 -p 8084:8084 -p 18083:18083  emqx/emqx:v4.0.5
```

访问Dashboard 查看启动效果!

4.6.2 基本命令

如果采用的是非docker部署的，那么EMQ X提供了一些常用的命令行工具，方便用户对EMQ X进行启动、关闭、进入控制台等操作。

- emqx start

  后台启动 EMQ X Broker
- emqx stop

  关闭 EMQ X Broker
- emqx restart

  重启 EMQ X Broker
- emqx console

  使用控制台启动 EMQ X Broker
- emqx foreground

  使用控制台启动 EMQ X Broker，与 `emqx console` 不同，`emqx foreground` 不支持输入 Erlang 命令

- emqx ping

  Ping EMQ X Broke

4.6.3 目录结构

不同安装方式得到的 EMQ X 其目录结构会有所不同，具体如下:

描述	使用 ZIP 压缩包安装(同docker)	使用二进制包安装
可执行文件目录	./bin	/usr/lib/emqx/bin
数据文件	./data	/var/lib/emqx/data
Erlang 虚拟机文件	./erts-*	/usr/lib/emqx/erts-*
配置文件目录	./etc	/etc/emqx
依赖项目录	./lib	/usr/lib/emqx/lib
日志文件	./log	/var/log/emqx
启动相关的脚本、schema 文件	./releases	/usr/lib/emqx/releases

以上目录中，用户经常接触与使用的是 `bin`、`etc`、`data`、`log` 目录。

bin 目录

emqx、*emqx.cmd*：EMQ X 的可执行文件

emqx_ctl、*emqx_ctl.cmd*：EMQ X 管理命令的可执行文件

etc 目录

EMQ X 通过 `etc` 目录下配置文件进行设置，主要配置文件包括:

配置文件	说明
emqx.conf	EMQ X 配置文件
acl.conf	EMQ X 默认 ACL 规则配置文件
plugins/*.conf	EMQ X 各类插件配置文件
certs/*	EMQ X SSL 证书文件
emqx.lic	License 文件仅限 EMQ X Enterprise

data 目录

EMQ X 将运行数据存储在 `data` 目录下，主要的文件包括:

configs/app..config*

EMQ X 读取 `etc/emqx.conf` 和 `etc/plugins/*.conf` 中的配置后，转换为 Erlang 原生配置文件格式，并在运行时读取其中的配置。

loaded_plugins

`loaded_plugins` 文件记录了 EMQ X 默认启动的插件列表，可以修改此文件以增删默认启动的插件。`loaded_plugins` 中启动项格式为 `{<Plugin Name>, <Enabled>}.`，`<Enabled>` 字段为布尔类型，EMQ X 会在启动时根据 `<Enabled>` 的值判断是否需要启动该插件。

```
$ cat loaded_plugins
{emqx_management,true}.
{emqx_recon,true}.
{emqx_retainer,true}.
{emqx_dashboard,true}.
{emqx_rule_engine,true}.
{emqx_bridge_mqtt,false}.
```

mnesia

Mnesia 数据库是 Erlang 内置的一个分布式 DBMS，可以直接存储 Erlang 的各种数据结构。

EMQ X 使用 Mnesia 数据库存储自身运行数据，例如告警记录、规则引擎已创建的资源和规则、Dashbaord 用户信息等数据，这些数据都将被存储在 `mnesia` 目录下，因此一旦删除该目录，将导致 EMQ X 丢失所有业务数据。

可以通过 `emqx_ctl mnesia` 命令查询 EMQ X 中 Mnesia 数据库的系统信息。

log 目录

*emqx.log.**：EMQ X 运行时产生的日志文件

crash.dump：EMQ X 的崩溃转储文件，可以通过 `etc/emqx.conf` 修改配置。

*erlang.log.**：以 `emqx start` 方式后台启动 EMQ X 时，控制台日志的副本文件。

4.6.4 配置说明

EMQ X 的配置文件通常以 `.conf` 作为后缀名，你可以在 `etc` 目录找到这些配置文件，主要配置文件包括：

配置文件	说明
etc/emqx.conf	EMQ X 配置文件
etc/acl.conf	EMQ X 默认 ACL 规则配置文件
etc/plugins/*.conf	EMQ X 扩展插件配置文件

需要注意的是，安装方式不同 `etc` 目录所处的路径可能不同，

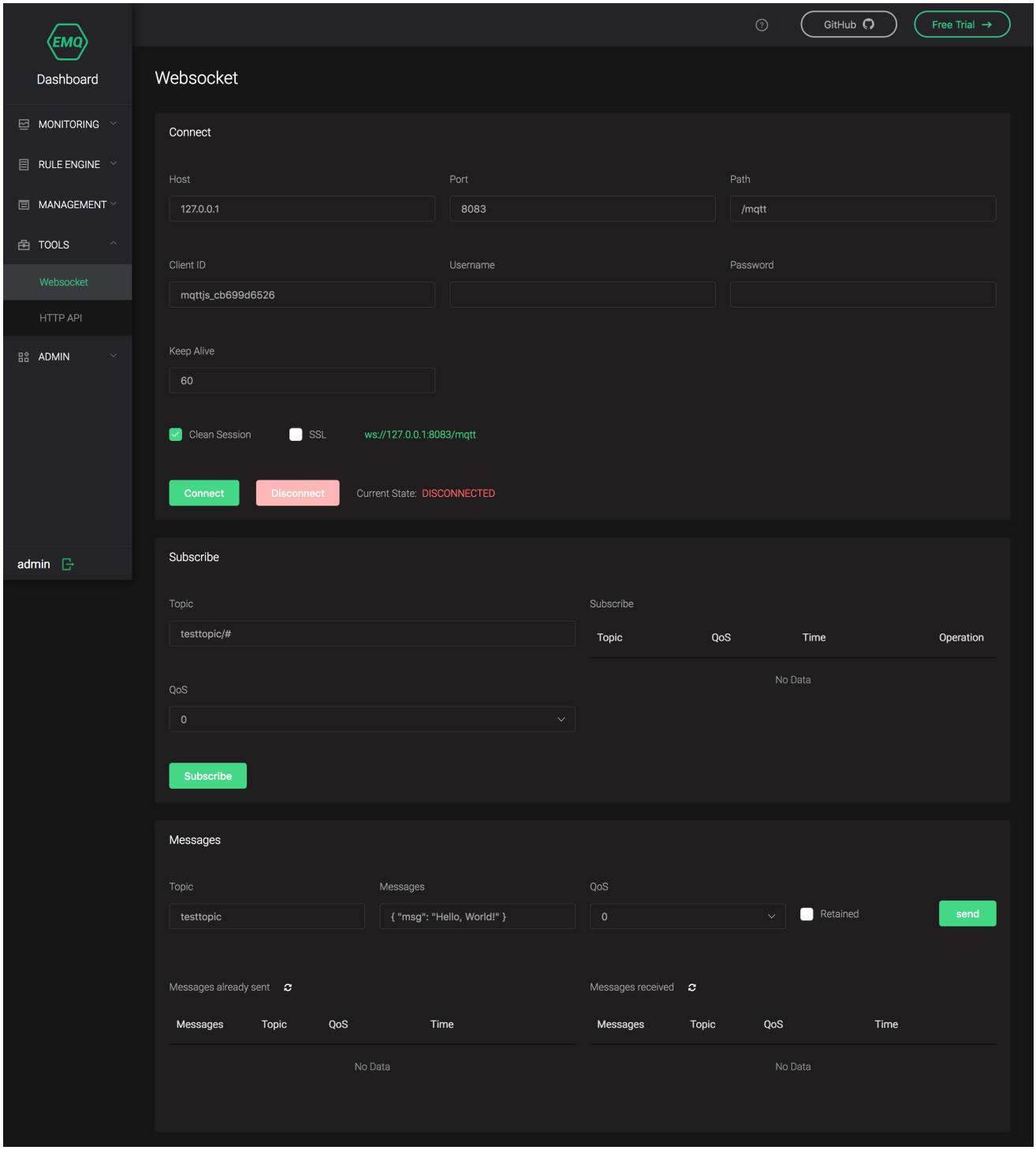
语法规则

- 采用类似 sysctl 的 k = v 通用格式
- 单个配置项的所有信息都在同一行内，换行意味着创建一个新的配置项
- 键可以通过 `.` 进行分层，支持按树形结构管理配置项

- 值的类型可以是 `integer`, `fload`, `percent`, `enum`, `ip`, `string`, `atom`, `flag`, `duration` and `bytesize`
- 任何以＃开头的行均被视为注释

4.7 客户端websocket消息收发

在EMQ X Broker提供的 Dashboard 中 `TOOLS` 导航下的 `Websocket` 页面提供了一个简易但有效的 WebSocket 客户端工具，它包含了连接、订阅和发布功能，同时还能查看自己发送和接收的报文数据，我们期望它可以帮助您快速地完成某些场景或功能的测试验证：



> - MQTT是为了物联网场景设计的基于TCP的Pub/Sub协议，有许多为物联网优化的特性，比如适应不同网络的QoS、层级主题、遗言等等。

> - WebSocket是为了HTML5应用方便与服务器双向通讯而设计的协议，HTTP握手然后转TCP协议，用于取代之前web服务器推送数据的Server Push、Comet、长轮询等老旧实现。
>
> 两者之所有有交集，是因为一个应用场景：如何通过HTML5应用来作为MQTT的客户端，以便接受设备消息或者向设备发送信息