第一讲
MQTT（I）
Lecture 1
MQTT (I)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

MQTT

- MQTT是英文单词消息队列遥测传输的首字母缩写，是物联网最常用的消息传递协议。该协议定义了一组规则，描述IoT 设备如何通过 Internet 发布和订阅数据。MQTT用于物联网和工业物联网设备之间的消息传递和数据交换，例如嵌入式设备、传感器、工业 PLC 等。该协议是事件驱动的，并使用发布/订阅模式连接设备。发送方（发布者）和接收方（订阅者）通过主题进行通信，相互解耦。它们之间的连接由MQTT代理处理。MQTT代理过滤所有传入消息并将它们正确分发给订阅者。

  MQTT stands for MQ Telemetry Transport, which is the most commonly used messaging protocol for the Internet of Things (IoT). The protocol defines a set of rules that describe how IoT devices can publish and subscribe to data over the Internet. MQTT is used for messaging and data exchange between IoT and industrial IoT (IIoT) devices, such as embedded devices, sensors, industrial PLCs, etc. The protocol is event driven and connects devices using the publish /subscribe (Pub/Sub) pattern. The sender (Publisher) and the receiver (Subscriber) communicate via Topics and are decoupled from each other. The connection between them is handled by the MQTT broker. The MQTT broker filters all incoming messages and distributes them correctly to the Subscribers.

MQTT特点
Features of MQTT

- MQTT 协议由 IBM的Andy Stanford-Clark 和 Arlen Nipper于1999年发明。他们需要一个协议来最小化电池损耗和最小带宽，以便通过卫星与石油管道连接。因此，在MQTT的设计与实现过程中，根植了如下特点：
  - 轻量与高效
  - 双向通信
  - 扩展到数以百万计的设备
  - 可靠的消息传递
  - 支持不可靠的网络
  - 支持安全性

- The MQTT protocol was invented in 1999 by Andy Stanford-Clark and Arlen Nipper working in IBM. They needed a protocol for minimal battery loss and minimal bandwidth to connect with oil pipelines via satellite. The following features are rooted in the design and implementation of MQTT:
  - Lightweight and Efficient
  - Bi-directional Communications
  - Scale to Millions of Things
  - Reliable Message Delivery
  - Support for Unreliable Networks
  - Security Enabled

MQTT演进
Evolvement of MQTT

- MQTT较长时间在IBM内部使用，在2010年，IBM开放了自身维护的MQTT v3.1方案。在该版本发布大约3年后，IBM将其交由标准化开放组织OASIS翼下进行标准化。OASIS于2014年10月29日宣布官方批准的MQTT标准，将版本号从3.1更改为3.1.1以表明对先前版本的更改。经过约5年的努力，OASIS于2019年3月批准了新的MQTT 5规范。该新的MQTT版本为MQTT引入了部署在云平台上的物联网应用程序所需的新功能，以及那些需要更高可靠性和错误处理来实现关键任务消息传递的功能。

  MQTT was used within IBM company internally after the initial draft. In 2010, IBM released the maintained MQTT v3.1 specification. Approximately 3 years, it was announced that MQTT would be standardized under the wings of OASIS, an open organization with the purpose of advancing standards. On October 29, 2014 MQTT became an officially approved OASIS Standard, with version change from 3.1 to 3.1.1 to show the updates. After 5 years effort, in March 2019, OASIS ratified the new MQTT 5 specification. This new MQTT version introduced new features to MQTT that are required for IoT applications deployed on cloud platforms, and those that require more reliability and error handling to implement mission-critical messaging.

发布/订阅模型
The Publish/Subscribe Pattern

- 客户端-服务器模型是一种重要的分布式计算模型。在此模型中，一般会采用端到端通信的模式。这种高耦合的方式并不适用于所有场景。发布/订阅模式或模型提供了传统客户端-服务器架构的替代方案，其将发布/订阅模型将发送消息的客户端（发布者）与接收消息的客户端（订阅者）进行分离，从而使发布者和订阅者不发生直接联系（事实上，他们甚至不知道对方的存在）。发布者与订阅者之间的连接由第三方组件（代理）来处理。代理的工作是过滤所有传入的消息并将它们正确地分发给订阅者。

  The client-sever model represents an important pattern in distributive computation. In this model, a client communicates directly with an endpoint. However, this tightly-coupling pattern cannot suit every scenarios. The publish/subscribe pattern (also known as pub/sub) provides an alternative to traditional client-server architecture, where it decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers). The publishers and subscribers never contact each other directly (in fact, they are not even aware that the other exists). The connection between them is handled by a third component (the broker), who’s job is to filter all incoming messages and distribute them correctly to subscribers.

发布/订阅模型
The Publish/Subscribe Pattern

- 如前所述，发布/订阅模型最重要的方面是消息的发布者与接收者（订阅者）的解耦。这种解耦有几个维度：
  - 空间解耦：发布者和订阅者不需要相互了解（例如不需要交换IP地址和端口）。
  - 时间解耦：发布者和订阅者不需要同时运行。
  - 同步解耦：在发布或接收过程中不需要中断两个组件的操作。
- 总之，发布/订阅模型消除了消息发布者和接收者/订阅者之间的直接通信。

- As aforementioned, the most important aspect of pub/sub is the decoupling of the publisher of the message from the recipient (subscriber). This decoupling has several dimensions:
  - Space decoupling: Publisher and subscriber do not need to know each other (for example, no exchange of IP address and port).
  - Time decoupling: Publisher and subscriber do not need to run at the same time.
  - Synchronization decoupling: Operations on both components do not need to be interrupted during publishing or receiving.
- In summary, the pub/sub model removes direct communication between the publisher of the message and the recipient/subscriber.

发布/订阅模型
The Publish/Subscribe Pattern

- 消息过滤是代理在发布/订阅过程中发挥的关键作用之一。代理通过如下过滤选项，过滤所有消息，以便每个订阅者只接收感兴趣的消息：
  - 选项 1：基于主题的过滤，此过滤基于作为每条消息一部分的主题。接收客户端向代理订阅感兴趣的主题。从那时起，代理确保接收客户端获取发布到订阅主题的所有消息。通常，主题是具有层次结构的字符串，允许基于有限数量的表达式进行过滤。
  - 选项 2：基于内容的过滤在基于内容的过滤中，代理根据特定的内容过滤语言过滤消息。接收客户端订阅过滤他们感兴趣的消息的查询。这种方法的一个显着缺点是必须事先知道消息的内容，并且不能加密或轻易更改。
  - 选项 3：基于类型的过滤当使用面向对象的语言时，基于消息（事件）的类型/类别进行过滤是一种常见的做法。例如，订阅者可以收听所有类型为 Exception 或任何子类型的消息。

发布/订阅模型
The Publish/Subscribe Pattern

- Message filtering is one of the roles that the broker plays in the pub/sub process. The broker has several filtering options and acts accordingly to filter all the messages so that each subscriber receives only messages of interest:
  - OPTION 1: SUBJECT-BASED FILTERING

    This filtering is based on the subject or topic that is part of each message. The receiving client subscribes to the broker for topics of interest. From that point on, the broker ensures that the receiving client gets all message published to the subscribed topics. In general, topics are strings with a hierarchical structure that allow filtering based on a limited number of expressions.
  - OPTION 2: CONTENT-BASED FILTERING

    In content-based filtering, the broker filters the message based on a specific content filter-language. The receiving clients subscribe to filter queries of messages for which they are interested. A significant downside to this method is that the content of the message must be known beforehand and cannot be encrypted or easily changed.
  - OPTION 3: TYPE-BASED FILTERING

    When object-oriented languages are used, filtering based on the type/class of a message (event) is a common practice. For example,, a subscriber can listen to all messages, which are of type Exception or any sub-type.

MQTT实践
MQTT Embodiment

- MQTT 体现了我们提到的发布/订阅模型的所有方面：
  - MQTT在空间上解耦了发布者和订阅者。要发布或接收消息，发布者和订阅者只需要知道代理的主机名/IP 和端口即可。
  - MQTT实现了时间解耦。尽管大多数 MQTT 用例以近乎实时的方式传递消息，但如果需要，代理可以为不在线的客户端存储消息。（存储消息必须满足两个条件：客户端已连接到持久会话并订阅了服务质量大于 0 的主题）。
  - MQTT基于异步工作模式。因为大多数客户端库都是异步工作的，并且基于回调或类似模型，所以在等待消息或发布消息时不会阻塞任务。在某些用例中，同步是可取的并且是可能的。虽然为了等待某个消息，一些库具有同步 API，但是流程通常是异步的。

MQTT实践
MQTT Embodiment

- MQTT embodies all the aspects of pub/sub that we've mentioned:
  - MQTT decouples the publisher and subscriber spatially. To publish or receive messages, publishers and subscribers only need to know the hostname/IP and port of the broker.
  - MQTT decouples by time. Although most MQTT use cases deliver messages in near-real time, if desired, the broker can store messages for clients that are not online. (Two conditions must be met to store messages: the client had connected with a persistent session and subscribed to a topic with a Quality of Service greater than 0).
  - MQTT works asynchronously. Because most client libraries work asynchronously and are based on callbacks or a similar model, tasks are not blocked while waiting for a message or publishing a message. In certain use cases, synchronization is desirable and possible. Although to wait for a certain message, some libraries have synchronous APIs, but the flow is usually asynchronous.

MQTT实践
MQTT Embodiment

- MQTT使用基于主题的消息过滤。每条消息都包含一个主题，代理可以使用该主题来确定订阅客户端是否应该收到消息。当然，具体MQTT代理实现可允许通过自定义扩展系统设置基于内容的过滤。
- 为了应对发布/订阅系统的不同情况，MQTT 具有三个服务质量 (QoS) 级别。可以通过指定QoS要求消息成功地从客户端传递到代理或从代理传递到客户端。
- 在特定情况下，可能没有人订阅特定主题，这种情况下，代理必须知道如何处理这种情况，具体实现可能通过插件系统解决此类情况。
- 为了保持分层主题树的灵活性，非常仔细地设计主题树并为未来的用例留出空间是很重要的。
- 生产环境中的设置可能要求一些代理的实现允许更多的操作，例如将每条消息记录到数据库中以进行历史分析。

MQTT实践
MQTT Embodiment

- MQTT uses subject-based filtering of messages. Every message contains a topic (subject) that the broker can use to determine whether a subscribing client gets the message or not. Some implementations of MQTT broker might allow the user to extend the system to set up content-based filtering.
- To handle the challenges of a pub/sub system, MQTT has three Quality of Service (QoS) levels, which allows by specifying the value to make sure that a message gets successfully delivered from the client to the broker or from the broker to a client.
- There is the chance that nobody subscribes to the particular topic. If this is a problem, the broker must know how to handle the situation, for example, to have a plugin system that can resolve such cases.
- To keep the hierarchical topic tree flexible, it is important to design the topic tree very carefully and leave room for future use cases.
- Production setups might require the broker to be configured taking action or simply logging every message into a database for historical analyses.

MQTT与消息队列
MQTT and Message Queue

- MQTT的名称以及该协议是否作为消息队列实现存在很多混淆。实际上，MQTT中的MQ指的是IBM的MQ series产品，与“消息队列”不直接相关。无论名称来自何处，了解 MQTT 与传统消息队列之间的区别都是有用的：
  - 消息队列存储消息直到它们被消费。当使用消息队列时，每条传入消息都存储在队列中，直到它被客户端（消费者）拾取。如果没有客户端接收到该消息，则该消息将停留在队列中并等待被消费。在MQTT中，消息可以对应没有人订阅主题，在消息队列中，消息不可能不被任何客户端处理。
  - 一条消息只能由一个客户端消费。在传统的消息队列中，一条消息只能由一个消费者处理，负载在队列的所有消费者之间进行均衡。在 MQTT 中，订阅主题的每个订阅者都会收到消息。
  - 队列必须被显式命名和创建，创建队列比主题更严格。在可以使用队列之前，必须使用单独的命令显式创建队列。只有在队列被命名和创建后，才能发布或消费消息。相比之下，MQTT 主题非常灵活，可以动态创建。

MQTT与消息队列
MQTT and Message Queue

- There is a lot of confusion about the name MQTT and whether the protocol is implemented as a message queue or not. Actually, MQ in MQTT refers to the MQ series product from IBM and has nothing to do with “message queue“. Regardless of where the name comes from, it’s useful to understand the differences between MQTT and a traditional message queue:
  - A message queue stores message until they are consumed. When you use a message queue, each incoming message is stored in the queue until it is picked up by a client (often called a consumer). If no client picks up the message, the message remains stuck in the queue and waits to be consumed. In a message queue, it is not possible for a message not to be processed by any client, as it is in MQTT if nobody subscribes to a topic.
  - A message is only consumed by one client. In a traditional message queue, a message can be processed by one consumer only. The load is distributed between all consumers for a queue. In MQTT the behavior is quite different: every subscriber that subscribes to the topic gets the message.
  - Queues are named and must be created explicitly. A queue is far more rigid than a topic. Before a queue can be used, the queue must be created explicitly with a separate command. Only after the queue is named and created is it possible to publish or consume messages. In contrast, MQTT topics are extremely flexible and can be created on the fly.

MQTT客户端
MQTT Client

- 在基于MQTT的应用中，MQTT客户端或客户端通常是指发布者和订阅者。特别声明是发布者或订阅者，是为了区分客户端是发布消息还是订阅接收消息，尽管发布和订阅功能也可以在同一个 MQTT客户端中实现。MQTT客户端是运行MQTT库并通过网络连接到MQTT代理的任何设备，从非常小的、资源受限的微控制器，到运行图形界面、具有复杂功能的服务器。基本上，任何通过TCP/IP 堆栈使用MQTT的设备都可以称为MQTT客户端。

  MQTT client or simply client usually refers to publishers and subscribers in the applications based on MQTT. The publisher and subscriber labels refer to whether the client is currently publishing messages or subscribed to receive messages, although publish and subscribe functionality can also be implemented in the same MQTT client. An MQTT client can be any device that runs an MQTT library and connects to an MQTT broker over a network, from a very small, resource-constrained micro controller, up to a full-fledged server with a graphical UI and high-throughput computational capabilities. Basically, any device that speaks MQTT over a TCP/IP stack can be called an MQTT client.

MQTT代理
MQTT Broker

- MQTT代理是基于MQTT实现发布/订阅协议的核心。代理负责接收所有的消息，过滤消息，确定每条消息的订阅者，并将消息发送给这些订阅的客户端。代理还保存所有具有持久会话的客户端的会话数据，包括订阅和错过的消息。代理的另一个职责是客户端的身份验证和授权。
- 通常，代理是可扩展的，这有助于自定义身份验证、授权和集成到后端系统。针对客户端来说，订阅所有消息并不是一个好的选择，因此需要代理处理大量客户端，需要将消息进行分析后传递给下游的处理系统。简而言之，代理通常是直接暴露在互联网上的组件，是每条消息都必须通过的中心枢纽。因此，代理必须具有高度可扩展性、可集成到后端系统、易于监控并且（当然）具有抗故障能力。一些代理实现可能通过使用最先进的事件驱动网络处理、开放扩展系统和标准监控提供程序来满足这些要求。根据实现的不同，代理甚至可以处理多达数百万个并发连接的 MQTT 客户端。

MQTT代理
MQTT Broker

- MQTT broker is at the heart of any publish/subscribe protocol. The broker is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients. The broker also holds the session data of all clients that have persistent sessions, including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients.
- Usually, the broker is extensible, which facilitates custom authentication, authorization, and integration into backend systems. Subscribing to all message is not really an option for an MQTT client, it requires the broker to handle a lot of clients, by analyzing the messages to pass them to downstream systems for processing. In brief, the broker is the component that is directly exposed on the internet and the central hub through which every message must pass, therefore, it is important that your broker is highly scalable, integratable into backend systems, easy to monitor, and (of course) failure-resistant. Some implementations meet these requirements by using state-of-the-art event-driven network processing, an open extension system, and standard monitoring providers. Depending on the implementation, a broker can handle up to millions of concurrently connected MQTT clients.

MQTT主题
MQTT Topic

- 在MQTT中，主题一词是指代理用于为每个连接的客户端过滤消息的UTF-8字符串。主题由一个或多个主题级别组成。每个主题级别由正斜杠（主题级别分隔符）分隔。与消息队列相比，MQTT主题非常轻量级。客户端在发布或订阅之前不需要创建所需的主题。代理接受每个有效主题而无需任何事先初始化。注意每个主题必须至少包含1个字符，并且主题字符串允许有空格。同时主题区分大小写，例如，myhome/temperature和MyHome/Temperature是两个不同的主题。此外，单独的正斜杠是一个有效的主题。

  In MQTT, the word topic refers to an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator). In comparison to a message queue, MQTT topics are very lightweight. The client does not need to create the desired topic before they publish or subscribe to it. The broker accepts each valid topic without any prior initialization. Note that each topic must contain at least 1 character and that the topic string permits empty spaces. Topics are case-sensitive. For example, myhome/temperature and MyHome/Temperature are two different topics. Additionally, the forward slash alone is a valid topic.

MQTT主题
MQTT Topic

- 当客户端订阅一个主题时，它可以订阅已发布消息的确切主题，也可以使用通配符同时订阅多个主题。通配符只能用于订阅主题，不能用于发布消息。有两种不同类型的通配符：单级和多级。单级通配符用+表示，多级通配符用#表示。一个例外是以 $ 开头的主题。以 $ 符号开头的主题保留用于MQTT代理的内部统计信息。虽然目前此类主题没有官方标准化，但客户端无法向这些主题发布消息。

  When a client subscribes to a topic, it can subscribe to the exact topic of a published message or it can use wildcards to subscribe to multiple topics simultaneously. A wildcard can only be used to subscribe to topics, not to publish a message. There are two different kinds of wildcards: single-level and multi-level. Single level wildcard is denoted by + and multi-level wildcard is denoted by #. One exception is topics beginning with $. Topics that start with a $ symbol have a different purpose. The $-symbol topics are reserved for internal statistics of the MQTT broker. Although at the moment, there is no official standardization for such topics, clients cannot publish messages to these topics.

MQTT主题
MQTT Topic

- 加号表示主题中的单级通配符，顾名思义，单级通配符替换一个主题级别。如果主题包含任意字符串而不是通配符，则任何主题都与具有单级通配符的主题匹配。例如，订阅 myhome/groundfloor/+/temperature 可以产生以下结果：

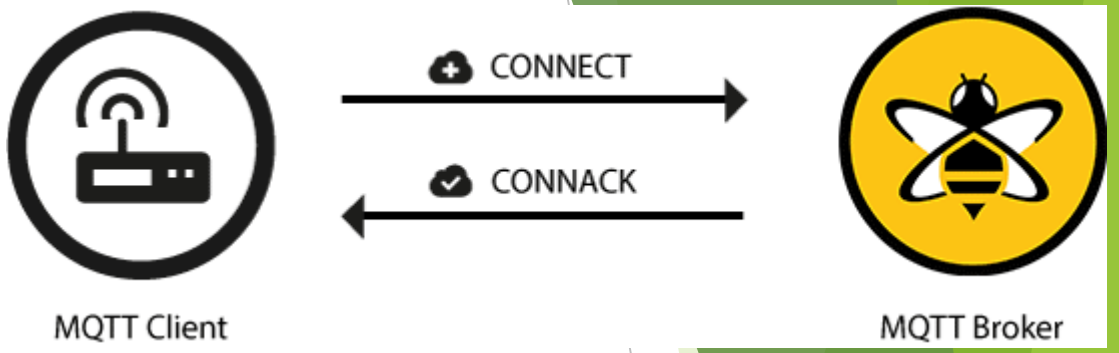
  The plus symbol represents a single-level wildcard in a topic. As the name suggests, a single-level wildcard replaces one topic level. Any topic matches a topic with single-level wildcard if it contains an arbitrary string instead of the wildcard. For example a subscription to myhome/groundfloor/+/temperature can produce the following results:

MQTT主题
MQTT Topic

- 井号表示主题中的多级通配符，多级通配符涵盖了多个主题级别。多级通配符一般作为主题中的最后一个字符，并且前置正斜杠。注意，当客户端使用多级通配符订阅主题时，它会接收以通配符之前的模式开头的主题的所有消息，特别的，如果仅将多级通配符指定为主题 (#)，将会收到发送到 MQTT 代理的所有消息，这可能会对高吞吐量的应用带来负面影响。

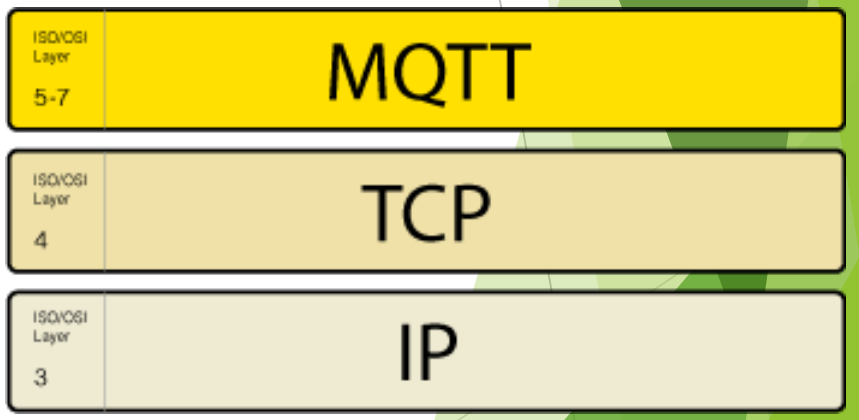
  The hash symbol represents the multi-level wild card in the topic. The multi-level wildcard covers many topic levels, it is generally placed as the last character in the topic and preceded by a forward slash. When a client subscribes to a topic with a multi-level wildcard, it receives all messages of a topic that begins with the pattern before the wildcard character, specifically, if you specify only the multi-level wildcard as a topic (#), you receive all messages that are sent to the MQTT broker. These might impact the application with high throughput.

MQTT连接
MQTT Connection

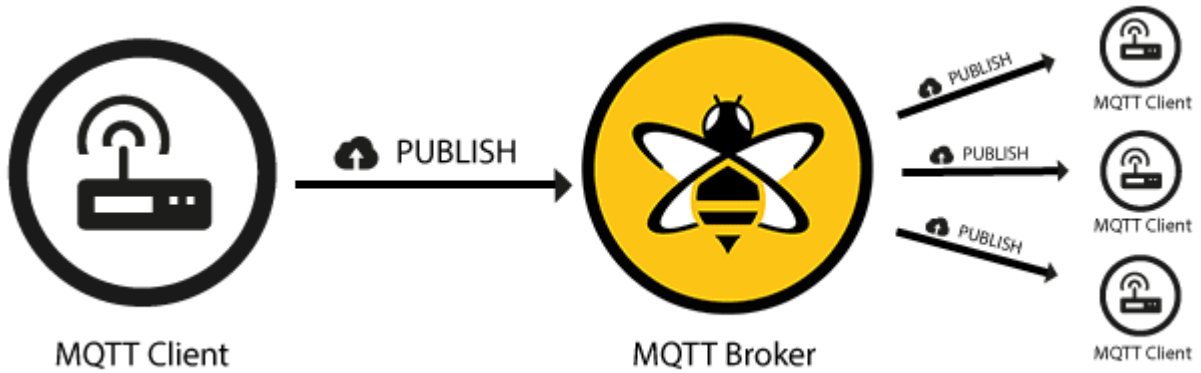


- MQTT协议是基于TCP/IP的应用层协议，因此客户端和代理都需要安装TCP/IP协议栈。MQTT连接只发生在客户端和代理之间，客户端不直接相互连接。为了发起连接，客户端向代理发送 CONNECT 消息。代理发送CONNACK消息进行响应并同时携带状态代码。建立连接后，代理将保持打开状态，直到客户端发送断开连接命令或连接中断。

  The MQTT protocol is based on TCP/IP, so both the client and the broker need to have a TCP/IP stack. The MQTT connection is always between one client and the broker. Clients never connect to each other directly. To initiate a connection, the client sends a CONNECT message to the broker. The broker responds with a CONNACK message and a status code. Once the connection is established, the broker keeps it open until the client sends a disconnect command or the connection breaks.



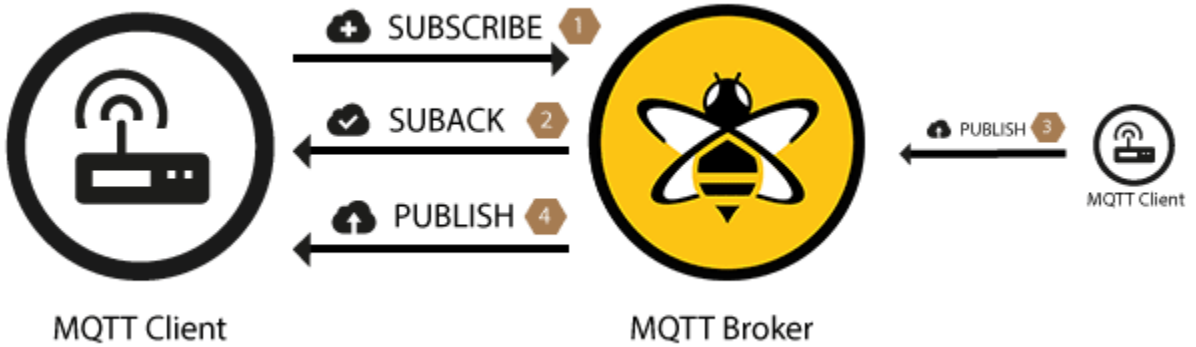
MQTT发布
MQTT Publish



- MQTT客户端可以在连接到代理后立即发布消息。每条消息都必须包含一个主题，MQTT代理利用基于主题的消息过滤，将消息转发给感兴趣的客户端。通常，每条消息都有一个有效载荷，其中包含要以字节格式传输的数据。MQTT与数据中立，客户端的用例决定了有效负载的结构，即发送客户端（发布者）决定是否要发送二进制数据、文本数据，甚至是XML或JSON。发布者只关心将PUBLISH消息传递给代理。代理负责收到PUBLISH消息后，将消息传递给所有订阅者。但发布客户端并不会收到任何关于是否有人对发布的消息感兴趣或有多少客户端从代理收到消息的反馈。

  An MQTT client can publish messages as soon as it connects to a broker. Each message must contain a topic, and MQTT broker utilizes topic-based filtering to forward the message to interested clients. Typically, each message has a payload which contains the data to transmit in byte format. MQTT is data-agnostic, and the use case of the client determines how the payload is structured. The sending client (publisher) decides whether it wants to send binary data, text data, or even full-fledged XML or JSON. The publisher is only concerned about delivering the PUBLISH message to the broker. The broker is responsible for delivering the message to all subscribers once receives the PUBLISH message. The publishing client does not get any feedback about whether anyone is interested in the published message or how many clients received the message from the broker.

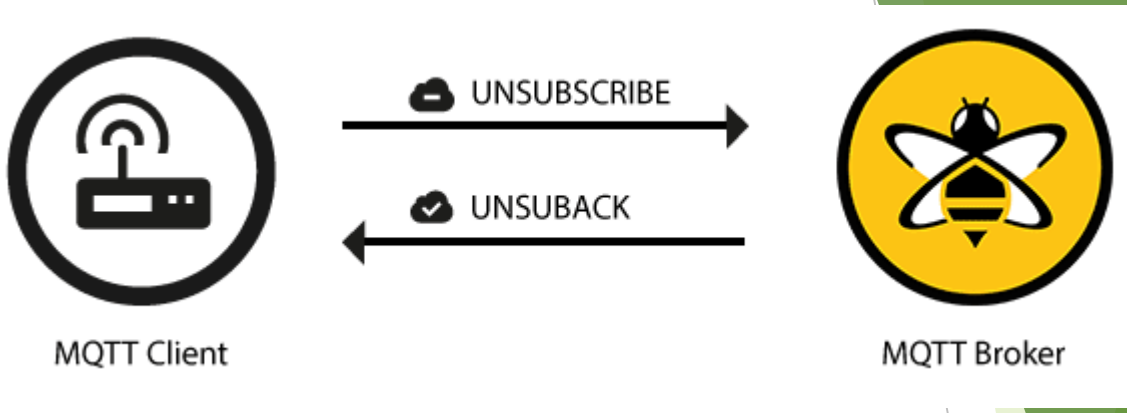
MQTT订阅
MQTT Subscribe



- 如果没有客户端订阅消息的主题，那么发布消息就没有意义，因为没有人收这样的消息。为了接收有关感兴趣主题的消息，客户端向MQTT代理发送一条SUBSCRIBE消息。这个订阅消息非常简单，它包含一个唯一的数据包标识符和一个订阅列表。为了确认每个订阅，代理向客户端发送一个SUBACK确认消息。此消息包含原始订阅消息的数据包标识符（以清楚地识别消息）和返回代码列表。

  If there are no clients to subscribe to the topics of the messages, publishing a message doesn’t make sense since no one ever receives it. In other words,. To receive messages on topics of interest, the client sends a SUBSCRIBE message to the MQTT broker. This subscribe message is very simple, it contains a unique packet identifier and a list of subscriptions. To confirm each subscription, the broker sends a SUBACK acknowledgement message to the client. This message contains the packet identifier of the original Subscribe message (to clearly identify the message) and a list of return codes.

MQTT退订
MQTT Unsubscribe



- 与SUBSCRIBE消息对应的是UNSUBSCRIBE消息。此消息删除代理上客户端的现有订阅。UNSUBSCRIBE消息与SUBSCRIBE消息类似，具有包标识符和主题列表。为了确认取消订阅，代理向客户端发送一个UNSUBACK确认消息。此消息仅包含原始UNSUBSCRIBE消息的数据包标识符（以清楚地识别消息）。客户端收到代理的UNSUBACK后，可以假设UNSUBSCRIBE消息中的订阅被删除。

  The counterpart of the SUBSCRIBE message is the UNSUBSCRIBE message. This message deletes existing subscriptions of a client on the broker. The UNSUBSCRIBE message is similar to the SUBSCRIBE message and has a packet identifier and a list of topics. To confirm the unsubscribe, the broker sends an UNSUBACK acknowledgement message to the client. This message contains only the packet identifier of the original UNSUBSCRIBE message (to clearly identify the message). After receiving the UNSUBACK from the broker, the client can assume that the subscriptions in the UNSUBSCRIBE message are deleted.