# 消息队列 MQTT 版

# 开发指南

腾讯云

# 文档目录

# 开发指南
# 数据面 HTTP 接口说明
# 查询客户端在线状态

最近更新时间：2025-04-28 15:17:22

## 概述

消息队列 MQTT 版支持通过主动查询和客户端事件两种方式来获取 MQTT 客户端在线状态。

## 原理介绍

消息队列 MQTT 版提供以下两种方式查询客户端在线状态：

| 方式 | 说明 | 使用场景 | 差异对比 |
| --- | --- | --- | --- |
| 主动查询 | 实例开放 RESTful API，实时查询客户端在线状态。 | 1. 业务流程中需要根据客户端是否在线决定后续运行逻辑。<br>2. 运维过程需要判断特定客户端当前是否在线。 | 主动查询是查询当前客户端的实时状态，相比上下线事件通知的方式更精确。 |
| 客户端事件 | 使用消息通知，在客户端上线和下线事件触发时，MQTT 服务端产生对应事件并发布到对应系统主题中，业务应用可以根据需要订阅相关事件消息。<br>该方式属于异步感知客户端的状态，且感知到的是上下线事件，而非在线状态，云端应用需要根据事件发生的时间序列分析出客户端的状态。 | 1. 业务需要在客户端上线或者下线时触发一些预定义的动作。<br>2. 业务需要对客户端的上下线数据进行统计分析，并根据客户端的在线状态推送消息。 | 客户端采用消息解耦，状态判断更加复杂，且误判可能性更大（最终一致性）。虽然存在一定复杂度和误判概率，但更加适合大规模的客户端的状态统计。 |

## 使用方法

### 主动查询

| 主动查询 |
| --- |
| |

### 请求示例

通过以下 RESTful API 进行查询：

```
curl --header "Authorization: Basic dXNlcjA6c2VjcmV0MA==" https://${instance-access-point}/client-id/${client-id}/status
```

请求参数说明：

| 变量 | 语义 | 示例值 |
| --- | --- | --- |
| ${instance-access-point} | 实例接入点地址，从控制台的 **集群管理** > **基本信息**页面获取。 | mqtt-xxxx-gz-public.mqtt.tencenttdmq.com |
| ${client-id} | 需要查询的目标 client-id，具体规范定义可参见 Client Identifier (ClientID)，结果需经过 URL 编码。 | VIN0000001 |

### 返回的 Status Code 说明：

1. 支持 HTTP Method： GET/POST。
2. 支持 HTTP/HTTPS。
3. 通过 HTTP Basic Authentication实现认证， 参见 Basic Authentication 。
4. 认证、授权：用户名和密码必须合法，且该账户必须有 CONNECT 权限。
5. HTTP 响应：

| HTTP Status Code（状态码） | 描述 |
| --- | --- |
| 200 | 请求正常。 |
| 401 | 认证失败，用户名或者密码不正确。 |
| 403 | 鉴权失败，缺少 CONNECT 权限。 |
| 429 | 流控，Too Many Requests。 |

### 输出示例

如果请求正常处理，返回以下内容：

```
{ "online" : true }
```

**输出参数说明：**

| 参数 | 说明 |
| --- | --- |
| online | 客户端在线状态。<br>• true：在线。<br>• false：不在线。 |

**客户端事件**

详细参见 客户端事件 。

## 计费说明

查询一次客户端在线状态， 等同于一次 QoS=0 的消息发布，具体可参见 价格说明 。查询频率受限于实例 QPS Quota 约束。

# 通过 HTTP POST 发布 MQTT 消息

最近更新时间：2025-03-12 16:04:52

## 概述

客户端可以通过 HTTP(s) POST 方式发送 REST 请求发布 MQTT 消息。该方式支持 HTTP 1.0和1.1协议。

## URL

实例支持 HTTP URL 如下：

```
http(s)://${instance-access-pooint}/topics/${url-encoded-topic-name}?qos=${qos}&retained=${retained}&client_id=${ClientId}&username=${Username}
```

| 变量 | 语义 | 示例值 | 可选 |
| --- | --- | --- | --- |
| instance-access-point | 实例接入点地址，从 消息队列 MQTT 版 控制台 单击**资源管理** > **集群管理** > 实例 ID 进入基本信息页面获取。 | mqtt-example-sh-public.mqtt.tencenttdmq.com | 否 |
| url-encoded-topic-name | 要发送的 Topic 名称，URL encoded。 | home | 否 |
| qos | 发送 MQTT 消息时选择 QoS，默认值为1，即 at-least-once。 | 1 | 是 |
| retained | 发送的消息是否是 retained 消息。 | false | 否 |
| client_id | 使用的 client-id 发送 MQTT 消息，参与授权验证。<br>• 如果在一机一证场景，使用证书 Common Name 字段。<br>• 其它场景默认值是 http-${connection-id}。 | curl001 | 是 |
| username | 连接、发送时使用的用户名，参与授权验证。<br>如果使用 Basic Authentication Header，取 | SampleUser | 是 |

| | | | |
|---|---|---|---|
| | Basic Authentication Header 中用户名部分。 | | |

## 认证

目前支持 Basic HTTP Authentication 和基于 X509 证书的“一机一证”两种认证方式。暂未支持基于 JWT 的认证方式。

> **说明：**
> **Basic HTTP Authentication**
> In basic HTTP authentication, a request contains a header field in the form of Authorization: Basic <credentials>, where <credentials> is the Base64 encoding of ID and password joined by a single colon ':'.

### 示例 1

缺省 username 字段。

```
curl --verbose --request POST -H "Authorization: Basic
dXNlcjA6c2VjcmV0MA==" --data "{ \"message\": \"Hello, world\" }"
"http://mqtt.cloud.tencent.com/topics/home?qos=1&client_id=curl-
x&retained=true"
```

### 示例 2

提供 HTTP(s) 发送时使用的 username。

> **注意：**
> Username 会参与 ACL 授权。

```
curl --verbose -k --request POST -H "Authorization: Basic
dXNlcjA6c2VjcmV0MA==" --data "{ \"message\": \"Hello, world\" }"
"https://mqtt.cloud.tencent.com/topics/home?qos=1&client_id=curl-
x&retained=true&username=sample-user"
```

## BYOC

支持基于 X509一机一证使用场景。

### 示例 1

**说明：**

- 客户端信任的证书为：CA.crt。
- 客户端证书链为：client.chain.crt。
- 客户端私钥为：client.pkcs8.key。

```
curl --tlsv1.2 \
    --cacert CA.crt \
    --cert client.chain.crt \
    --key client.pkcs8.key \
    --verbose \
    --request POST \
    --data "{ \"message\": \"Hello, world\" }" \
    "https://mqtt.cloud.tencent.com/topics/home?qos=1"
```

## 授权

HTTP POST 方式类似 WebSocket，正常参与所有的授权流程。如果使用 HTTP 1.0协议，每次发送都会执行完整的 CONNECT，PUBLISH 流程，也会执行对应的的鉴权、授权操作。

# 配置自定义域名

最近更新时间：2025-08-01 16:34:12

用户可以使用自定义域名连接 TDMQ MQTT 集群。 这个方式有一下几个优势：

- 当 MQTT 接入点提供给用户的客户时，保持用户品牌统一；
- 迁移后端集群时，通过重新解析 DNS 即可完成，保持接入点不变；
- 可以复用企业已有的 PKI 设施。

配置实例使用自定义域名包含以下步骤：

1. 上传服务端证书到 SSL证书服务；
2. 配置服务端证书对、信任证书到 MQTT 实例；
3. 创建 DNS 记录；

## 上传证书对、信任证书到 SSL 证书服务

为了确保证书正常轮转更新、过期提醒，不影响业务连续性，MQTT实例仅支持配置SSL证书服务托管的证书。

### 证书要求

1. 服务端证书(End Entity Certificate)必须包含 X.509 v3 Extended Key Usage 扩展，扩展必须包含 serverAuth(TLS Web Server Authentication), 参考 RFC 5280；即证书应包含必要的 Subject Alternative Name(SAN) 扩展和 Common Name(CN)。如果您的证书系 CA 机构签发，一般均包含该扩展。

   > ⚠ 注意：
   > - 如果证书同时包含 CN 和 SAN, TLS Server Name Indication(SNI) 选择证书时，优先在 SAN 扩展包含域名中选择；如果 SAN 不存在，兼容使用 CN 进行选择。
   > - 主流浏览器已忽略 CN 匹配，如 Chrome 58后忽略 CN，使用 Web Secure Socket 作为 MQTT 传输层时，务必确保 SAN 扩展和 Server Auth 值。

2. 证书链最多包含5个证书；
3. 证书链文件不超过16KiB。

### 证书上传、托管、购买

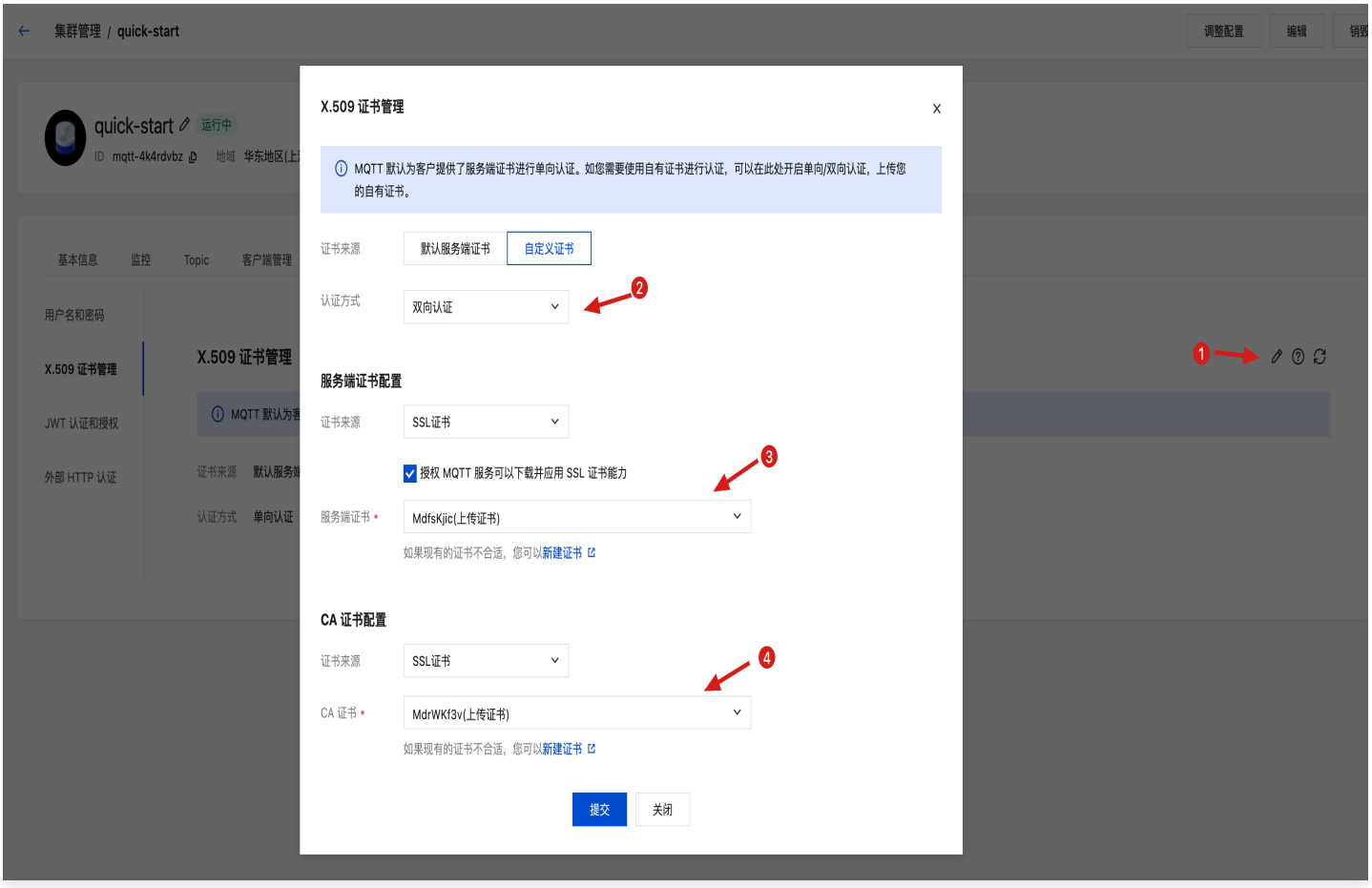
详情请参见 上传 SSL 证书指引 。

## 配置证书、信任证书到MQTT实例

MQTT 控制台 > 集群管理/实例页面，切到认证管理/X.509证书管理页面：


根据是否验证客户端证书，选择合适的认证模式：

- 单向认证：不校验客户端证书
- 双向认证：校验客户端证书




假设选择的服务端证书 Subject Alternative Name 包含以下域名：`mqtt-abc.compay.com`。

# 创建 DNS 记录

参考 云解析-CNAME 记录，创建一个 CNAME 记录，将 `mqtt-abc.compay.com` 解析到实例的接入点提供的域名：`mqtt-xxx-sh-public.mqtt.tencenttdmq.com`。

后续通过 `mqtt-abc.company.com` 访问。

# 注册设备身份

最近更新时间：2025-07-30 18:14:42

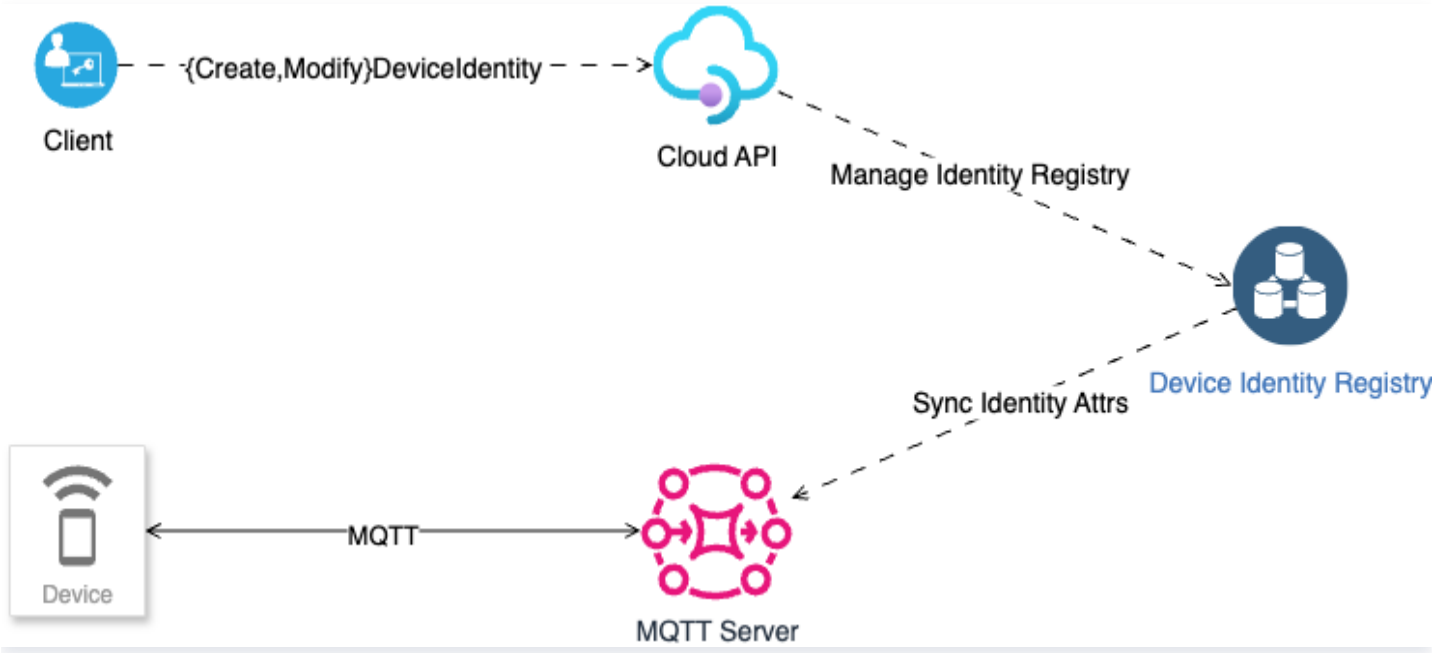
> **说明：**
> 当前功能灰度中，如控制台提示当前集群暂未开启该功能，请提交工单联系我们处理。

## 背景

MQTT 产品为每个实例提供设备身份注册表，方便用户为具体设备/客户端提供进附加配置，从而实现“一机一密”，“传播属性”等特性。

## 实现原理

设备身份注册表通过 CAPI 开放一组管理接口，实现设备身份的创建、更新、删除等管理。




## Device Identity

设备身份目前包含以下字段：

| 字段 | 描述 | 备注 |
| --- | --- | --- |
| DeviceID | 设备 ID，等同于 Client ID | 关联设备配置和 MQTT Client Session |
| Status | Device Identity 状态 | 当 Status 为 Disabled时，忽略该配置 |

| | | |
|---|---|---|
| PropagatingProperties | 传播属性 | 详见 传播属性 |
| PrimaryKey | 主秘钥 | 详见 一机一密 |
| SecondaryKey | 次秘钥 | 详见 一机一密 |
| CreateTime | 设备身份记录创建时间 | - |

# 配置传播属性

最近更新时间：2025-07-30 18:14:42

> **说明：**
> 当前功能灰度中，如控制台提示当前集群暂未开启该功能，请提交工单联系我们处理。

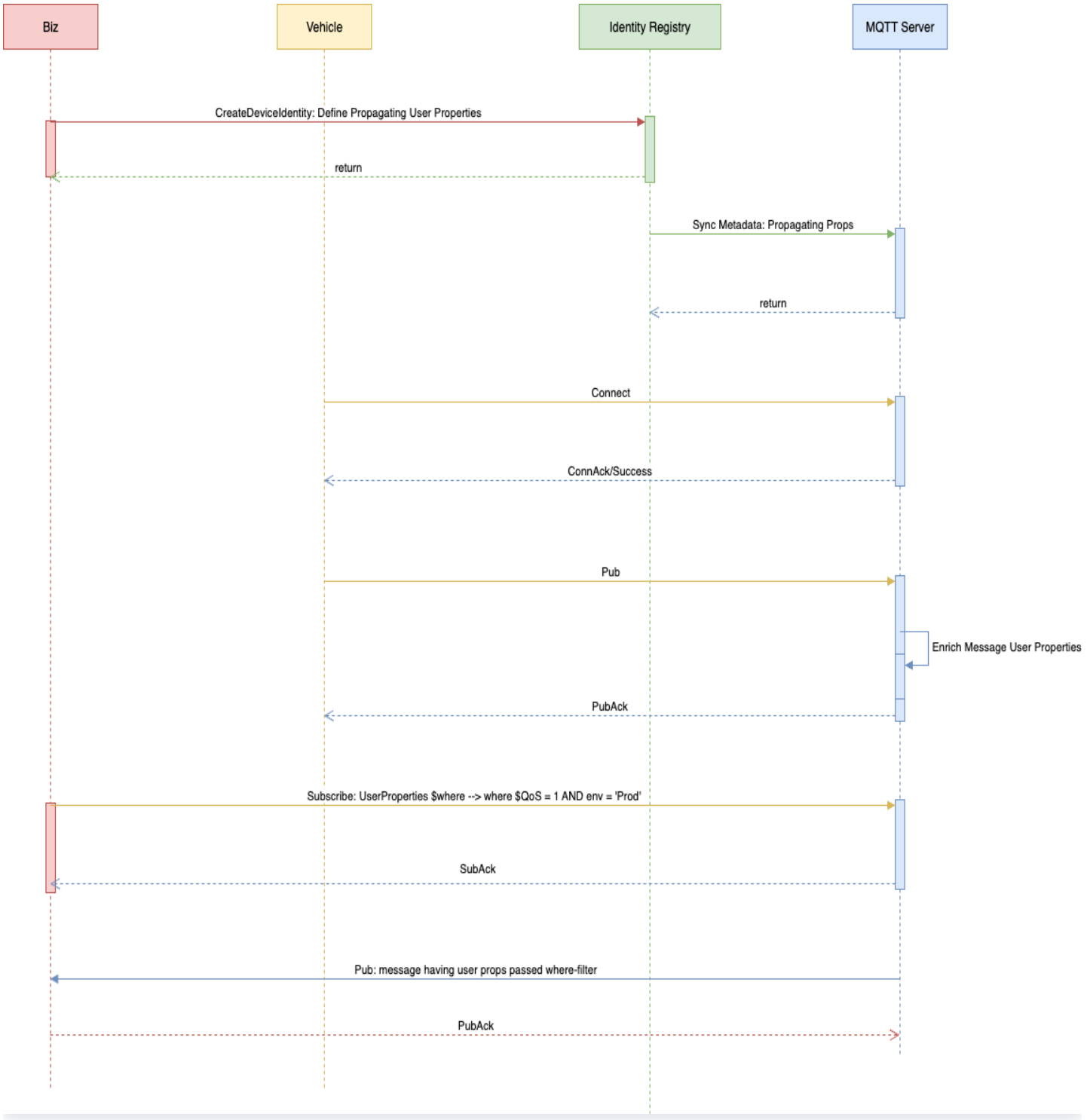
## 功能描述

用户可以通过设备身份注册表为设备/客户端配置 传播属性。当匹配的客户端发布消息时，会额外添加传播属性包含的 User Property 列表，达到不修改设备代码的情况下，增加元数据的目标。

## 使用场景

1. 设备无法支持 OTA 升级；
2. 灰度发布，A/B测试等场景， 需要对设备打环境标。

## 实现原理




如上述序列图所示:

1. 用户通过云 API 创建设备身份，定义传播属性；
2. 设备身份配置信息同步到 MQTT Server 节点；
3. 匹配的客户端发布消息时， MQTT Server 根据设备身份元数据， 为消息添加（Append）传播属性定义的 User Property 列表；
4. MQTT 5.0 订阅者订阅消息时， 投递包含传播属性的 User Properties。

# 配置 SQL 过滤

最近更新时间：2025-08-06 19:50:21

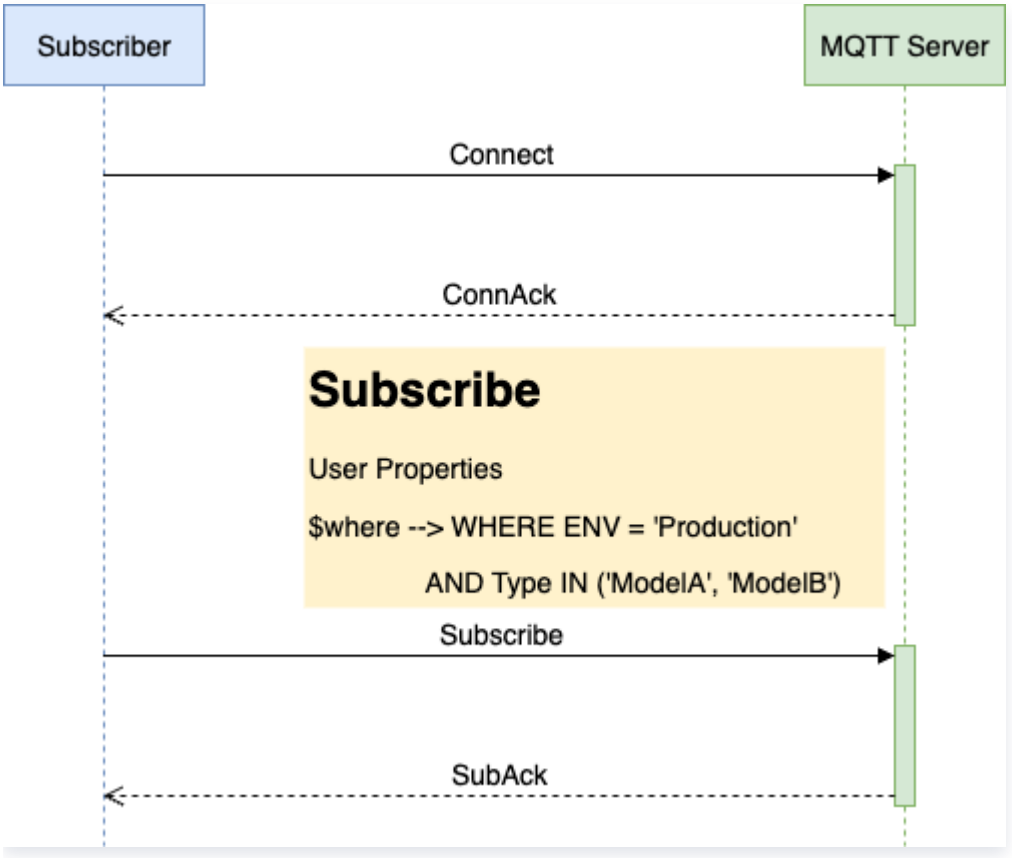
> **说明：**
> 当前功能灰度中，如控制台提示当前集群暂未开启该功能，请提交工单联系我们处理。

## 背景

MQTT 标准规范定义了 Topic Filter 的概念，让订阅者可以根据 MQTT Topic Name 层级结构选择需要订阅的消息。层级结构 和 Wildcard 表达式，提供了比较灵活的消息过滤表达能力。但在一些场景中，例如灰度发布，A/B测试，系统升级等场景，Topic Filter 无法满足更灵活的需求。

## 实现原理

MQTT 5.0 引入了 Subscribe User Property，MQTT 产品扩展支持了 Subscribe User Property 的过滤语义。




当用户 Subscribe User Property 包含 key 为 "$where"，Value 为 WHERE子句时，MQTT Server 在推送消息时，会根据 WHERE 子句对消息进行过滤，仅投递符合过滤条件的消息到订阅者。

## 使用限制

1. Subscribe User Properties 中只能有一个过滤语句；当有多个 $where --> WHERE-CLAUSE 属性时，仅第一个生效；
2. 消息用户属性中，按照协议 允许多个同名的 Key-Value对 ，同名 Key 的多个键值对情况下，仅最后一个参与过滤表达式计算；
3. 如果过滤表达式包含某个字段在消息属性中未出现, 则该字段的值为 NULL；
4. WHERE 子句支持的操作符包括: AND、OR、NOT、=、!=、>、>=、<、<=、IN、IS NULL、LIKE、CASE WHEN...THEN...ELSE...END；
5. WHERE 子句支持的函数包括: UPPER, LOWER, LENGTH, ABS, COALESCE;
6. 字符串字面量仅支持单引号, 例如 **where type = 'string-literal'**。

## 示例

```
package com.tencent.tdmq.mqtt.quickstart.paho.v5.async;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;
import org.eclipse.paho.mqttv5.client.IMqttToken;
import org.eclipse.paho.mqttv5.client.MqttAsyncClient;
import org.eclipse.paho.mqttv5.client.MqttCallback;
import org.eclipse.paho.mqttv5.client.MqttConnectionOptions;
import org.eclipse.paho.mqttv5.client.MqttDisconnectResponse;
import org.eclipse.paho.mqttv5.client.persist.MemoryPersistence;
import org.eclipse.paho.mqttv5.common.MqttException;
import org.eclipse.paho.mqttv5.common.MqttMessage;
import org.eclipse.paho.mqttv5.common.MqttSubscription;
import org.eclipse.paho.mqttv5.common.packet.MqttProperties;
import org.eclipse.paho.mqttv5.common.packet.UserProperty;

public class BasicQuickStart {
    public static void main(String[] args) throws MqttException,
InterruptedException {
        String serverUri = "tcp://mqtt-xxx.mqtt.tdmqcloud.com:1883";
        String clientId = "deviceBasic";

        String topic = "home/room/1";
        String[] topicFilters = new String[] {"home/#"};
        int[] qos = new int[] {1};
```

```
        MqttAsyncClient client = new MqttAsyncClient(serverUri,
clientId, new MemoryPersistence());
        MqttConnectionOptions options = new MqttConnectionOptions();
        options.setUserName("YOUR-USERNAME");
        options.setPassword("YOUR-
PASSWORD".getBytes(StandardCharsets.UTF_8));
        options.setCleanStart(true);
        options.setSessionExpiryInterval(TimeUnit.DAYS.toSeconds(1));

        client.setCallback(new MqttCallback() {
            @Override
            public void disconnected(MqttDisconnectResponse response) {
                System.out.println("Disconnected: " +
response.getReasonString());
            }

            @Override
            public void mqttErrorOccurred(MqttException e) {
                e.printStackTrace();
            }

            @Override
            public void messageArrived(String s, MqttMessage message)
throws Exception {
                byte[] payload = message.getPayload();
                String content;
                if (4 == payload.length) {
                    ByteBuffer buf = ByteBuffer.wrap(payload);
                    content = String.valueOf(buf.getInt());
                } else {
                    content = new String(payload,
StandardCharsets.UTF_8);
                }
                System.out.printf("Message arrived, topic=%s, QoS=%d
content=[%s], properties=%s%n",
                    topic, message.getQos(), content,
message.getProperties());
            }

            @Override
            public void deliveryComplete(IMqttToken token) {

            }
```

```
            @Override
            public void connectComplete(boolean reconnect, String
serverURI) {
                System.out.println(reconnect ? "Reconnected" :
"Connected" + " to " + serverURI);
            }

            @Override
            public void authPacketArrived(int i, MqttProperties
properties) {
            }
        });
        client.connect(options).waitForCompletion();
        try {
            // Subscribe
            MqttSubscription[] subscriptions = new
MqttSubscription[topicFilters.length];
            for (int i = 0; i < topicFilters.length; i++) {
                subscriptions[i] = new MqttSubscription(topicFilters[i],
qos[i]);
            }
            MqttProperties subscribeProperties = new MqttProperties();
            List<UserProperty> userProperties = new ArrayList<>();
            UserProperty userProperty = new UserProperty("$where",
"where $QoS = 1 AND k1 = 'v1'");
            userProperties.add(userProperty);
            subscribeProperties.setUserProperties(userProperties);
            client.subscribe(subscriptions, null, null,
subscribeProperties).waitForCompletion();
        } catch (MqttException e) {
            e.printStackTrace();
        }


        int total = 128;
        for (int i = 0; i < total; i++) {
            byte[] payload = new byte[4];
            ByteBuffer buffer = ByteBuffer.wrap(payload);
            buffer.putInt(i);
            MqttMessage message = new MqttMessage(payload);
            message.setQos(1);
            MqttProperties properties = new MqttProperties();
```

```
            properties.setContentType("application/json");
            properties.setResponseTopic("response/topic");
            message.setProperties(properties);
            System.out.printf("Prepare to publish message %d%n", i);
            // P2P topic format: {first-topic}/p2p/{target-client-id}
            client.publish(topic, message);
            System.out.printf("Published message %d%n", i);
            TimeUnit.MILLISECONDS.sleep(100);
        }
        TimeUnit.MINUTES.sleep(3);
        client.disconnect();
    }
}
```

# 配置点对点订阅

最近更新时间：2025-04-28 15:17:22

## 场景说明

除了 MQTT 标准协议定义的发布/订阅（Pub/Sub）消息模型外，消息队列 MQTT 版还支持点对点（Point to Point，简称 P2P）模式。

> **说明：**
> 当前仅**专业版**提供 P2P 能力。

## 什么是 P2P 消息模式?

当需要发送消息给指定的一个消费者时，可以使用点对点消息模式。对比主要提供单个 Publisher 多个 Subscriber 的 1:N，或者 M:N 场景方案的 Pub/Sub 消息模式， P2P 消息模式提供了高效的点对点通信方案。
使用 P2P 模式时，Publisher 已经明确该消息的目标接收者信息，并且该消息只需要被指定的单一客户端消费。
发送者发送消息时通过设置符合**命名规则的 Topic** 指定接收者，接收者无需提前订阅即可消费到该消息。
使用 P2P 模式不仅可以节省接收者注册订阅关系的成本，还可以降低推送延迟。

## 发布 P2P 消息

```
String firstTopic = ...;
String targetClientId = ...;
String topic = firstTopic + "/p2p/" + targetClientId;
MqttMessage message = ...;
mqttClient.publish(topic, message);
```

## 订阅 P2P 消息

接收消息的客户端无需任何订阅处理，正确初始化和连接到集群的目标客户端即可收到 P2P 消息。

## Paho Golang SDK 使用注意事项：

Paho Golang SDK 在收到消息后, 会根据收到消息的 Topic 进行 matchAndDispatch 。

```
// matchAndDispatch takes a channel of Message pointers as input and
starts a go routine that
// takes messages off the channel, matches them against the internal
route list and calls the
// associated callback (or the defaultHandler, if one exists and no
other route matched). If
```

```
// anything is sent down the stop channel the function will end.
func (r *router) matchAndDispatch(messages <-chan
*packets.PublishPacket, order bool, client *client) <-chan
*PacketAndToken {
    var wg sync.WaitGroup
    ackOutChan := make(chan *PacketAndToken) // Channel returned to
caller; closed when messages channel closed
    var ackInChan chan *PacketAndToken       // ACKs generated by
ackFunc get put onto this channel

    stopAckCopy := make(chan struct{})    // Closure requests stop of go
routine copying ackInChan to ackOutChan
    ackCopyStopped := make(chan struct{}) // Closure indicates that it
is safe to close ackOutChan
    goRoutinesDone := make(chan struct{}) // closed on wg.Done()
    if order {
       ackInChan = ackOutChan // When order = true no go routines are
used so safe to use one channel and close when done
    } else {
       // When order = false ACK messages are sent in go routines so
ackInChan cannot be closed until all goroutines done
       ackInChan = make(chan *PacketAndToken)
       go func() { // go routine to copy from ackInChan to ackOutChan
until stopped
          for {
             select {
             case a := <-ackInChan:
                ackOutChan <- a
             case <-stopAckCopy:
                close(ackCopyStopped) // Signal main go routine that it
is safe to close ackOutChan
                for {
                   select {
                   case <-ackInChan: // drain ackInChan to ensure all
goRoutines can complete cleanly (ACK dropped)
                      DEBUG.Println(ROU, "matchAndDispatch received
acknowledgment after processing stopped (ACK dropped).")
                   case <-goRoutinesDone:
                      close(ackInChan) // Nothing further should be sent
(a panic is probably better than silent failure)
                      DEBUG.Println(ROU, "matchAndDispatch order=false
copy goroutine exiting.")
                      return
```

```
                    }
                }
            }
        }
    }()
}

go func() { // Main go routine handling inbound messages
    for message := range messages {
        // DEBUG.Println(ROU, "matchAndDispatch received message")
        sent := false
        r.RLock()
        m := messageFromPublish(message, ackFunc(ackInChan,
client.persist, message))
        var handlers []MessageHandler
        for e := r.routes.Front(); e != nil; e = e.Next() {
            if e.Value.(*route).match(message.TopicName) {
                if order {
                    handlers = append(handlers, e.Value.
(*route).callback)
                } else {
                    hd := e.Value.(*route).callback
                    wg.Add(1)
                    go func() {
                        hd(client, m)
                        if !client.options.AutoAckDisabled {
                            m.Ack()
                        }
                        wg.Done()
                    }()
                }
                sent = true
            }
        }
        if !sent {
            if r.defaultHandler != nil {
                if order {
                    handlers = append(handlers, r.defaultHandler)
                } else {
                    wg.Add(1)
                    go func() {
                        r.defaultHandler(client, m)
                        if !client.options.AutoAckDisabled {
```

```
                        m.Ack()
                    }
                    wg.Done()
                }()
            }
        } else {
            DEBUG.Println(ROU, "matchAndDispatch received message
and no handler was available. Message will NOT be acknowledged.")
        }
    }
    r.RUnlock()
    for _, handler := range handlers {
        handler(client, m)
        if !client.options.AutoAckDisabled {
            m.Ack()
        }
    }
    // DEBUG.Println(ROU, "matchAndDispatch handled message")
}
if order {
    close(ackOutChan)
} else { // Ensure that nothing further will be written to
ackOutChan before closing it
    close(stopAckCopy)
    <-ackCopyStopped
    close(ackOutChan)
    go func() {
        wg.Wait() // Note: If this remains running then the user
has handlers that are not returning
        close(goRoutinesDone)
    }()
}
DEBUG.Println(ROU, "matchAndDispatch exiting")
}()
return ackOutChan
}
```

**收到的消息可能不匹配任何一个 topic-filter，为了能正确处理这类消息，需要配置 defaultHandler：**

`ClientOptions:SetDefaultPublishHandler(messagePubHandler)`，**使 fallback 逻辑符合预期**。

```
package main

import (
```

```
    "fmt"
    mqtt "github.com/eclipse/paho.mqtt.golang"
    "time"
)

var messagePubHandler mqtt.MessageHandler = func(client mqtt.Client, msg
mqtt.Message) {
    fmt.Printf("Received message: %s from topic: %s\n", msg.Payload(),
msg.Topic())
}

var connectHandler mqtt.OnConnectHandler = func(client mqtt.Client) {
    fmt.Println("Connected")
}

var connectLostHandler mqtt.ConnectionLostHandler = func(client
mqtt.Client, err error) {
    fmt.Printf("Connect lost: %v", err)
}

func sub(client mqtt.Client) {
    topic := "home/room"
    // Message handler per topic-filter
    token := client.Subscribe(topic, 1, messagePubHandler)
    result := token.Wait()
    fmt.Printf("Subscribed to topic %s, %s", topic, result)
}

func publish(client mqtt.Client) {
    num := 10
    for i := 0; i < num; i++ {
        text := fmt.Sprintf("Message %d", i)
        token := client.Publish("home/test", 0, false, text)
        token.Wait()
        time.Sleep(time.Second)
    }
}

func main() {
    // Acquire your instance access point from MQTT Console
    var broker = "mqtt-xxx.mqtt.tencenttdmq.com"
    var port = 1883
    opts := mqtt.NewClientOptions()
```

```
    opts.AddBroker(fmt.Sprintf("tcp://%s:%d", broker, port))
    opts.SetClientID("mqttGolangClient")
    opts.SetUsername("YOUR-USERNAME")
    opts.SetPassword("YOUR-PASSWORD")
    // Need to configure defaultHandler to make P2P message fallback
properly
    opts.SetDefaultPublishHandler(messagePubHandler)
    opts.OnConnect = connectHandler
    opts.OnConnectionLost = connectLostHandler
    client := mqtt.NewClient(opts)
    if token := client.Connect(); token.Wait() && token.Error() != nil {
        panic(token.Error())
    }
    sub(client)
    publish(client)
    time.Sleep(time.Minute * 3)
}
```