一文学会MQTT

最后更新时间：2024年04月11日

[简介]

> MQTT是什么?

MQTT（Message Queuing Telemetry Transport）是一种基于**发布-订阅（Publish-Subscribe）**模式的**轻量级**通讯协议，采用**客户端-代理（Client-Broker）**模型进行通信，基于TCP协议，属于应用层协议。

它最初由 IBM 在 1999 年开发，目前已成为**物联网（IoT）**领域中最流行的通讯协议之一。

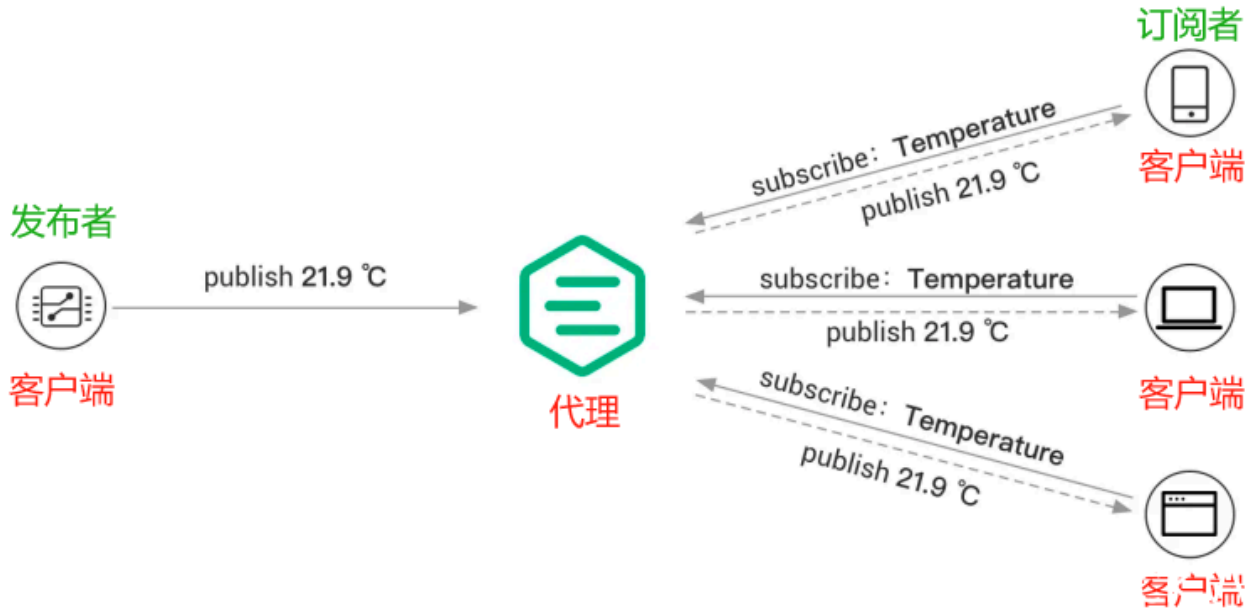
MQTT 的发布-订阅机制可以很轻易地满足我们一对一、一对多、多对一的通信需要。

> 发布-订阅模式是什么?

在 MQTT 中，发布-订阅模式是指消息的**发布者**/发送者（**Publisher**）和**订阅者**/接收者（**Subscriber**）之间的通讯方式。

发布-订阅模式与**客户端-服务器**模式的不同之处在于：发布者和订阅者之间无需建立直接连接，而是通过 MQTT **代理**（**Broker**） 来负责消息的路由和分发。

下图展示了 MQTT 发布-订阅过程。温度传感器作为客户端连接到 MQTT Broker，并通过发布操作将温度数据发布到一个特定主题（例如 `Temperature` ）。MQTT Broker 接收到该消息后会负责将其转发给订阅了相应主题（ `Temperature` ）的订阅者客户端。



> **客户端-代理**模型是什么?

客户端-代理模型指的是 MQTT 客户端（包括发布者和订阅者）与 MQTT 代理（Broker）之间的通讯模式。具体来说：

- **客户端**可以是**发布者**，也可以是**订阅者**，也可以同时具备这两个身份
- **代理**类似于服务器，负责接收**发布者**发布到**特定主题**的消息并转发给相应的**订阅者**

> 主题是什么?

主题（**Topic**）是消息的分类或者话题，用唯一标识字符串来标识消息的内容或者类型。MQTT 协议根据主题来转发消息。

主题可以是层次结构的，使用正斜杠（ `/` ）进行分隔，类似于 URL 路径。例如 `weather/temperature` 是一个层次结构的主题，表示天气的温度信息。

MQTT 主题支持以下两种通配符： `+` 和 `#` 。

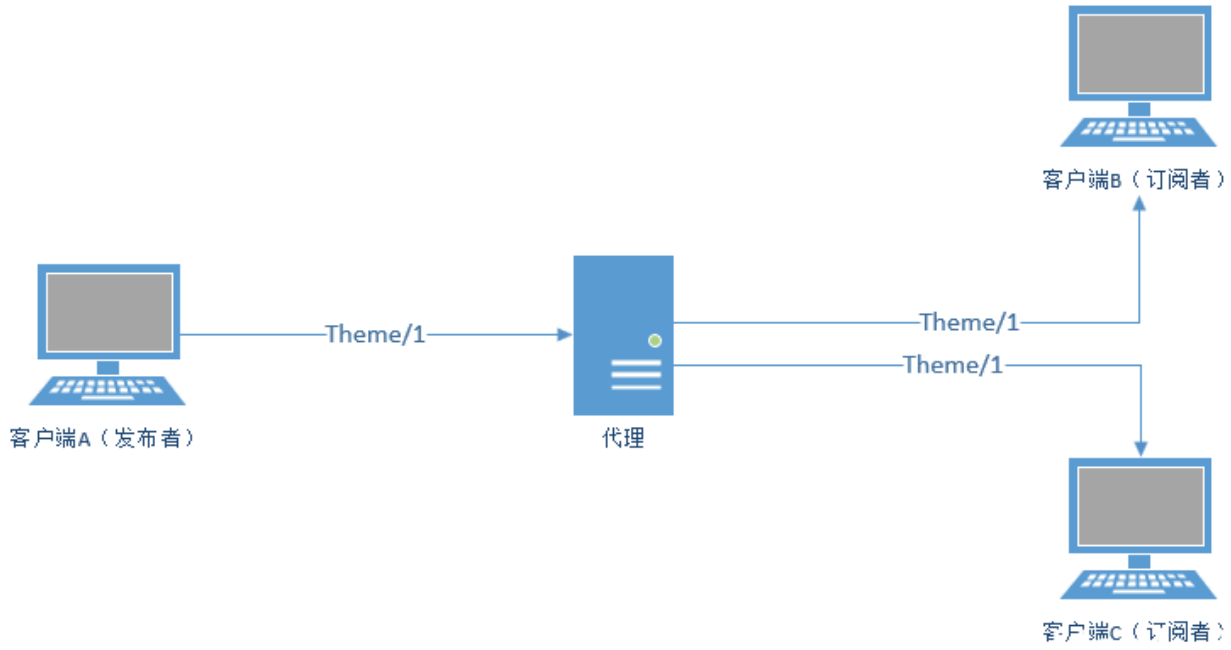
- `+` ：表示单层通配符，例如 `a/+` 匹配 `a/x` 或 `a/y` 。
- `#` ：表示多层通配符，例如 `a/#` 匹配 `a/x` 、 `a/b/c/d` 。

注意：通配符主题只能用于订阅，不能用于发布。

一个主题可以有多个订阅者，代理会将该主题下的消息转发给所有订阅者；一个主题也可以有多个发布者，代理将按照消息到达的顺序转发。

例如**客户端A**发布了一条消息 XXX 到特定的**主题**（**Topic**），而**客户端B**和**客户端C**订阅了这个主题。当客户端A发布消息到该主题时，客户端B和客户端C都会接收到这条消息。这样，客户端A作为发布者，而客户

端B和客户端C作为订阅者，他们通过订阅相同的主题来实现消息的传递和接收。



以 `$SYS/` 开头的主题为系统主题，系统主题主要用于获取 MQTT 服务器自身运行状态、消息统计、客户端上下线事件等数据。目前，MQTT 协议暂未明确规定 `$SYS/` 主题标准，但大多数 MQTT 服务器都遵循该标准建议。

例如，EMQX 服务器支持通过以下主题获取集群状态。

主题	说明
$SYS/brokers	EMQX 集群节点列表
$SYS/brokers/emqx@127.0.0.1/version	EMQX 版本
$SYS/brokers/emqx@127.0.0.1/uptime	EMQX 运行时间
$SYS/brokers/emqx@127.0.0.1/datetime	EMQX 系统时间
$SYS/brokers/emqx@127.0.0.1/sysdescr	EMQX 系统信息

更多内容请参考：通过案例理解 MQTT 主题与通配符

> QoS是什么?

MQTT定义了服务质量QoS（Quality of Service，）用于控制消息在不同网络环境下保证消息的可靠性。

QoS 级别有三种：

- **0**：最低级别的 QoS，表示"最多传递一次"。消息发布者发布消息后，不会收到任何确认，也不保证消息会被准确地传递给订阅者。消息可能会丢失或者重复传递。消息最多传送一次。如果当前客户端不可用，它将丢失这条消息。
- **1**：中等级别的 QoS，表示"至少传递一次"。消息发布者发布消息后，会收到一个确认（PUBACK），但不保证消息只传递一次。消息可能会重复传递，但不会丢失。
- **2**：最高级别的 QoS，表示"只传递一次"。消息发布者发布消息后，会进行消息传递的完全确认，确保消息只传递一次给订阅者。这种级别的 QoS 保证了消息的完整性和精确性，但需要更多的网络带宽和资源。

MQTT协议的优缺点

优点	缺点
轻量级：协议简单、开销小，适合在资源受限的设备上使用。	消息传递不可靠：QoS 0 级别的消息可能会丢失，不适用于要求严格的消息传递可靠性的场景。
易于实现：提供了多种编程语言的客户端库，方便开发者进行应用程序开发。	不适用于大规模消息传输：在大量消息传输的场景下，可能会导致代理性能问题。
异步通讯：支持发布者和订阅者之间的异步通讯，提高了系统的灵活性和响应速度。	安全性限制：默认情况下，MQTT 不提供消息加密和身份验证，需要额外的安全措施来保护通讯安全。

ⓘ 与HTTP相比

- HTTP 报文相较与 MQTT 需要占用更多的网络开销
- HTTP 是一种无状态协议，服务器在处理请求时不会记录客户端的状态，也无法实现从连接异常断开中恢复
- 请求-响应模式需要通过轮询才能获取数据更新，而 MQTT 通过订阅即可获取实时数据更新
- MQTT 5.0 增加了请求-响应特性，以实现订阅者收到消息后向某个主题发送应答，发布者收到应答后再进行后续操作

与消息队列相比

- 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

> 使用场景

- 物联网（IoT）应用程序中的设备通讯和数据传输。如手机控车、共享充电宝、共享单车、设备监控等等。
- 智能家居系统。例如控制灯光、温度、安全系统等。
- 传感器网络中的数据采集和监控，农业监测、环境检测、医疗监测等。例如远程监测患者的生命体征数据、医疗设备的运行状态等，并及时向医护人员发送警报和通知

手机控车

在手机控车场景中，用户通过手机应用程序向车辆发送控制指令（例如启动、熄火、锁车、解锁等）。以手机解锁汽车为例，用户发送解锁指令的数据流程通常包括以下步骤：

1. **用户操作触发控车指令**：用户登录手机App后点击界面的解锁按钮。
2. **手机应用程序发送指令**：手机App通过 HTTPS 向TSP服务器发起解锁车辆请求或。请求中可能包含指令的类型（例如启动汽车）、车辆标识（例如车辆VIN）等信息。
3. **后端服务器接收指令**：TSP服务器接收到用户发送的解锁请求后对用户进行鉴权。
4. **服务器向车辆发送指令**：鉴权通过后TSP服务器作为**发布者**，通过 MQTT 协议向特定主题（如 `vin/xxxx/unlock` ）发送消息（可能包括车辆的唯一标识符（例如车辆VIN）、指令类型、指令参数等信息）。
5. **车辆接收指令并执行**：车辆作为**订阅者**订阅该主题（ `vin/xxxx/unlock` ）以接收指令。当车辆接收到服务器发送的 MQTT 消息后，根据指令类型和参数执行相应的操作，解锁车辆等操作。

车辆信息监控

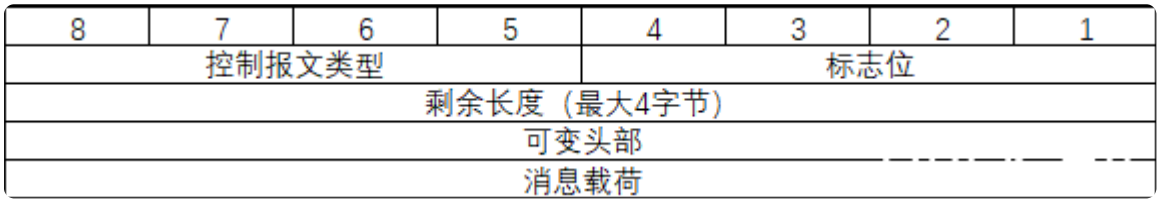
在车辆信息监控场景中，车辆作为**发布者**，定期发布车辆的状态信息（例如车速、位置、油耗、车辆健康状态等）。监控系统作为 MQTT 的**订阅者**，订阅车辆状态信息所在的主题，并实时地获取和展示车辆的状态信息。

[数据报文结构]

MQTT的数据报文结构非常简单，由**固定头部**、**可选头部**和**消息载荷**组成。

- 固定头部由**控制报文类型**、**标志位**和**剩余长度**组成，控制报文类型占第一个字节的高四位，标志位占第一个字节的低四位，剩余长度最多为4字节，固定头部最大长度为5字节
- 可变头部的长度取决于报文的类型和包含的字段。不同类型的报文包含的字段不同，长度不同。
- 消息有效载荷的长度取决于实际的消息内容。它可以是空的，也可以是几个字节到几千字节不等，取决于应用场景和需要传输的数据量。

以下是 MQTT 数据报文的基本结构：



[使用MQTT]

使用MQTT前需要先部署MQTT代理服务，然后使用客户端工具测试连通性，接着选择您喜欢的编程语言去实现您想要的功能。

> 代理服务器选择

以下是一些常见的MQTT代理：

MQTT 代理	下载地址	优点	
Mosquitto	https://github.com/eclipse/mosquitto	轻量级，易于部署 开源，免费 支持多种操作系统	功可
HiveMQ	https://www.hivemq.com	高性能，适用于大规模 IoT 应用 提供企业级支持	商部署
EMQX	https://github.com/emqx/emqx	支持 MQTT、MQTT-SN、CoAP 和 HTTP 协议 高可用性和可扩展性	社区部署
RabbitMQ	https://github.com/rabbitmq/rabbitmq-server	功能丰富，支持多种通讯协议	学习

MQTT 代理	下载地址	优点	
		可靠性和高可用性	对
ActiveMQ	https://github.com/apache/activemq	开源， 具有丰富的功能和插件支持 可靠性和可用性	学习 对
VerneMQ	https://github.com/vernemq/vernemq	开源， 具有高可用性和可伸缩性 专注于 IoT 和实时通讯	社区 部署
Apache ActiveMQ Artemis	https://github.com/apache/activemq-artemis	开源， 具有低延迟和高吞吐量 可扩展性和可靠性	部署 学

关于 MQTT Broker 的更多详情，请参阅EMQX提供的文章 2023 年最全面的 MQTT Broker 比较指南。

客户端工具

通常在部署完MQTT代理后需要使用客户端工具进行连接测试和消息测试，通常使用MQTTX。它包含桌面程序、命令行和在线Web客户端。

连接参数

ⓘ 连接地址

MQTT 的连接地址通常包含： 服务器 IP 或者域名、服务器端口、连接协议。

连接协议可以是基于TCP的(`mqtt` / `mqtts`)，也可以是基于WebSocket的(`ws` / `wss`)。

ⓘ 客户端ID

MQTT 服务器使用 Client ID 识别客户端，连接到服务器的每个客户端都必须要有唯一的 Client ID。Client ID 的长度通常为 1 至 23 个字节的 UTF-8 字符串。

注意：如果客户端使用一个重复的 Client ID 连接至服务器，将会把已使用该 Client ID 连接成功的客户端踢下线。

用户名和密码

MQTT 协议可以通过用户名和密码来进行相关的认证和授权，但是如果此信息未加密，则用户名和密码将以明文方式传输。如果设置了用户名与密码认证，那么最好要使用 `mqtts` 或 `wss` 协议。

连接超时时长

连接超时时长，收到服务器连接确认前的等待时间，等待时间内未收到连接确认则为连接失败。

保活周期

保活周期，是一个以秒为单位的时间间隔。客户端在无报文发送时，将按 Keep Alive 设定的值定时向服务端发送心跳报文，确保连接不被服务端断开。

在连接建立成功后，如果服务器没有在 Keep Alive 的 1.5 倍时间内收到来自客户端的任何包，则会认为和客户端之间的连接出现了问题，此时服务器便会断开和客户端的连接。

清除会话

为 false 时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。为 true 时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。

遗嘱消息

遗嘱消息是 MQTT 为那些可能出现**意外断线**的设备提供的将**遗嘱**优雅地发送给其他客户端的能力。设置了遗嘱消息消息的 MQTT 客户端异常下线时，MQTT 服务器会发布该客户端设置的遗嘱消息。

> EMQX使用教程

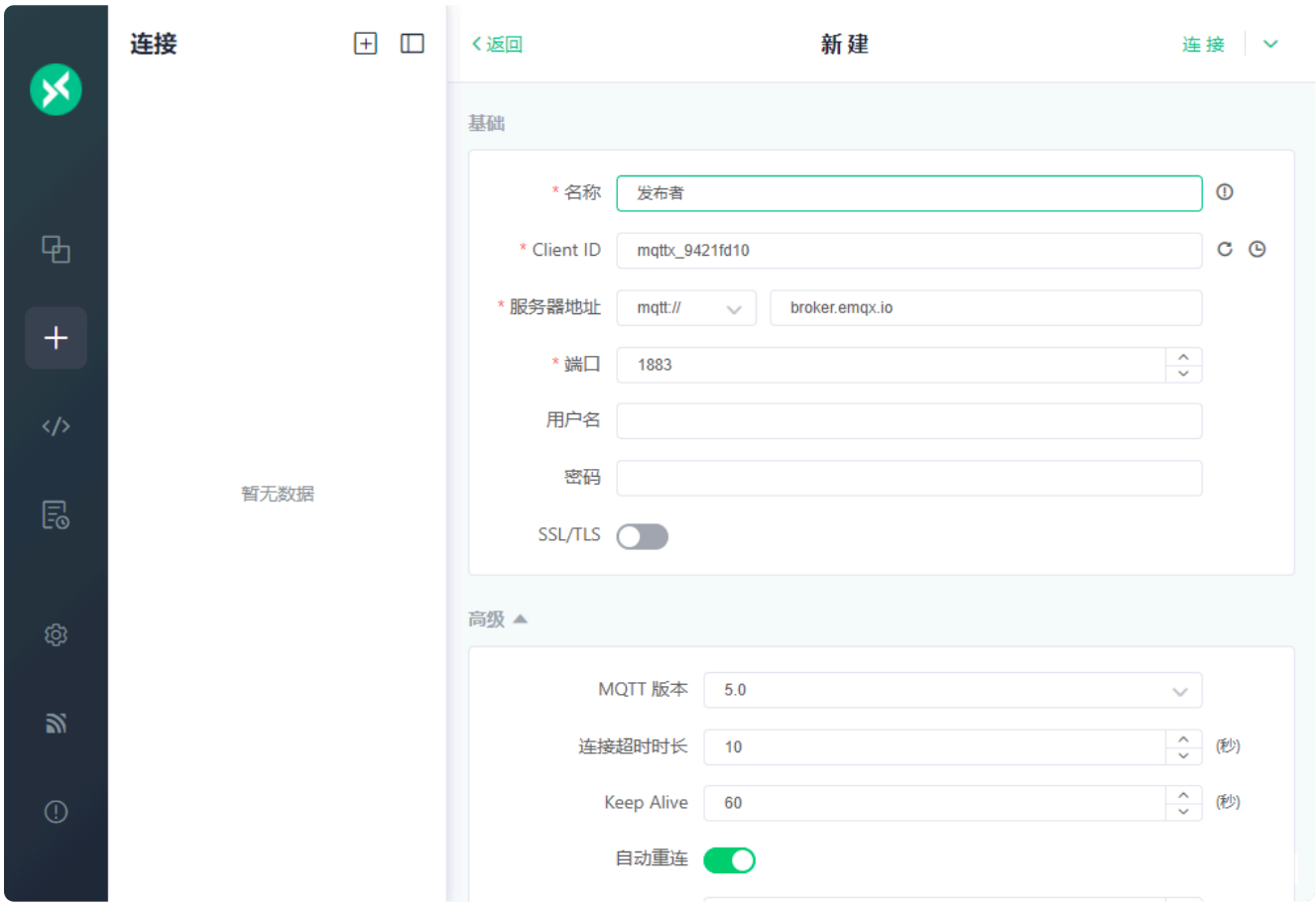
代理服务器部署

EMQX作为一个开源的 MQTT 消息代理，具有许多优势，使其在物联网和实时通讯应用中备受青睐。您可以采用rpm、deb等安装包进行安装，也可以采用docker进行安装，还可以使用EMQX Cloud，免去自己部署的麻烦。官方提供的docker安装命令如下：

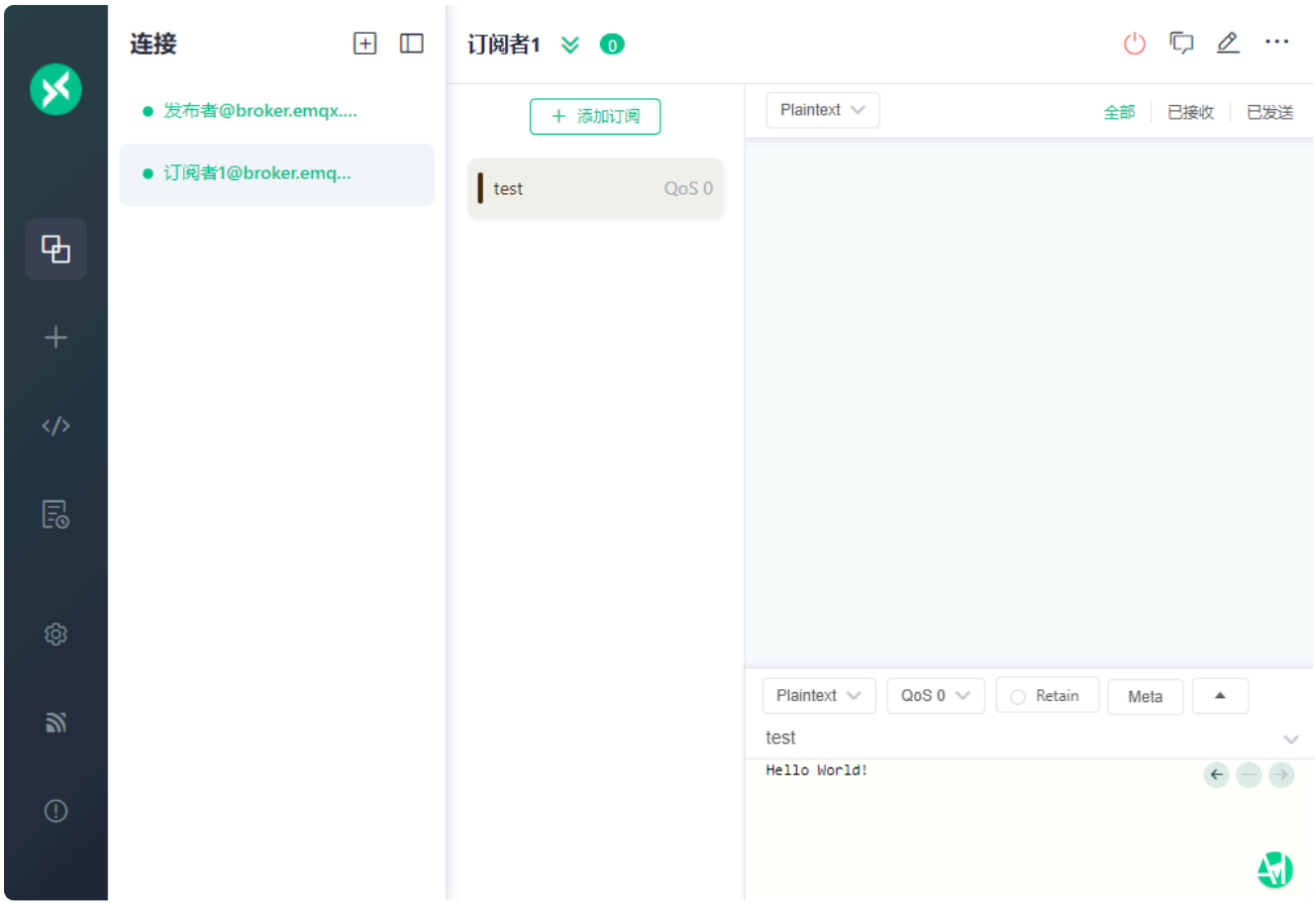
```
docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8084:8084 -p 8883:8883 -p 180
```

简单使用

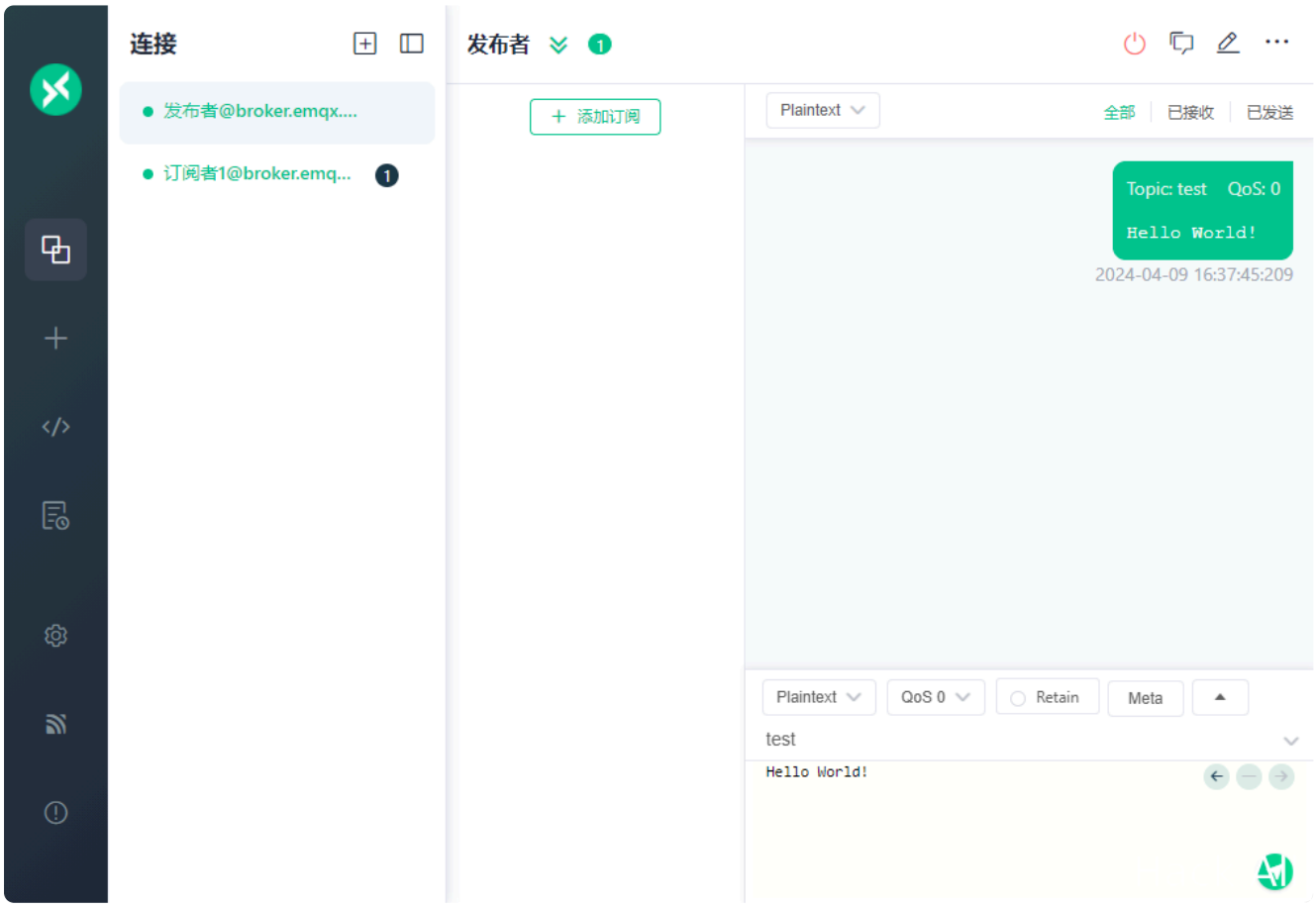
发布者和订阅者都需要创建一个 MQTT 连接来连接到代理服务器。采用默认配置创建发布者的连接：

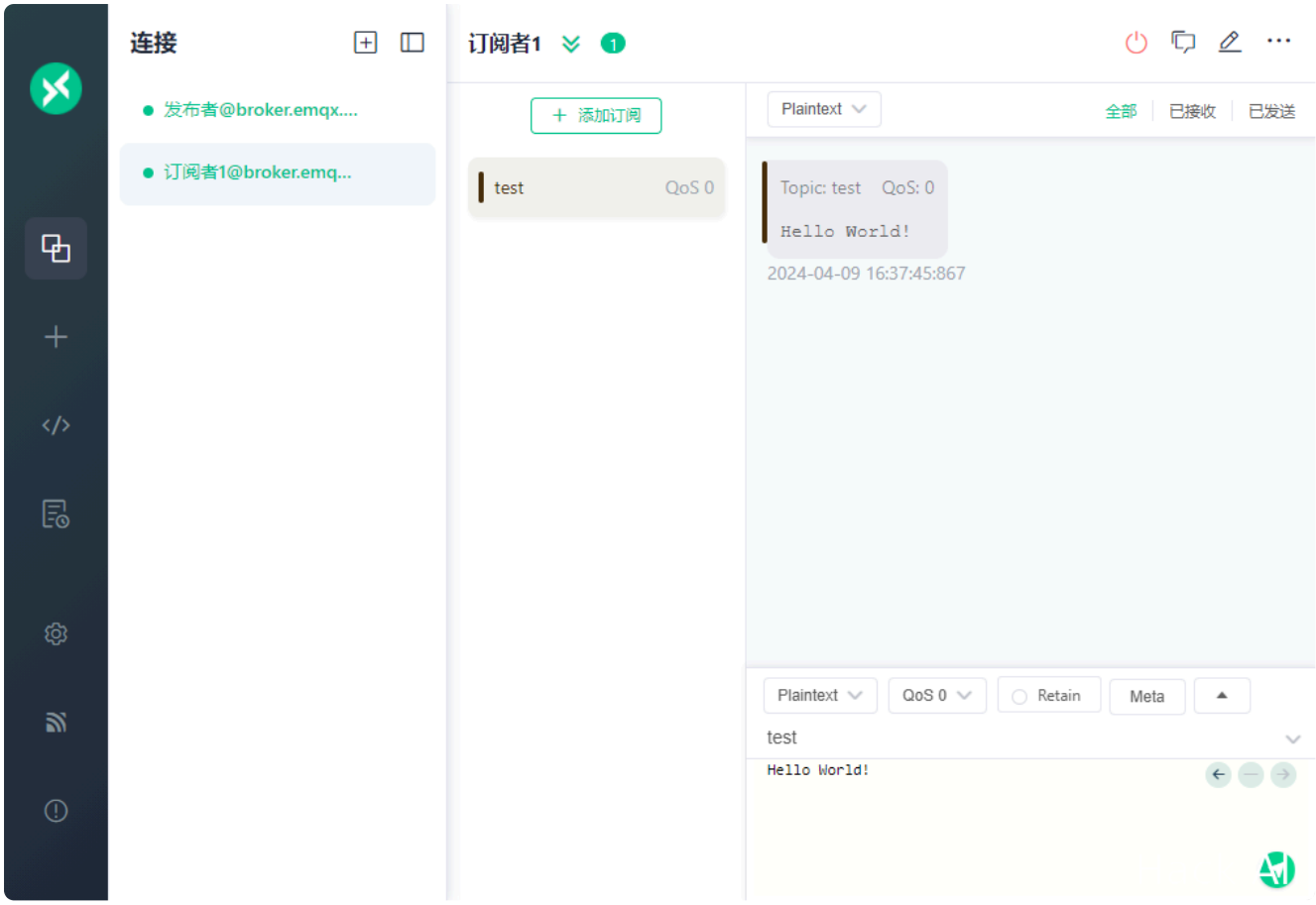


然后采用默认连接创建订阅者的连接。然后点击添加订阅即可订阅特定主题的消息：



发布者发布消息到主题test时，MQTT代理将消息转发到相应的订阅者：





编程使用

Python 3使用MQTT

使用 `pip3 install -i https://pypi.doubanio.com/simple paho-mqtt` 安装MQTT客户端模块:

```
C:\Users\Administrator>python -V
Python 3.11.3

C:\Users\Administrator>pip -V
pip 22.3.1 from D:\Python3\Lib\site-packages\pip (python 3.11)

C:\Users\Administrator>pip3 install -i https://pypi.doubanio.com/simple paho-mqtt
Looking in indexes: https://pypi.doubanio.com/simple
Collecting paho-mqtt
  Downloading https://mirrors.cloud.tencent.com/pypi/packages/5c/14/665ea683e6328af4418e55aec6ea9572adf4e0000ef70988db7fc99142c9/paho_mqtt-2.0.0-py3-none-any.whl (66 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.9/66.9 kB 602.7 kB/s eta 0:00:00
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-2.0.0

[notice] A new release of pip available: 22.3.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```


发布者代码:

```
import random
import time
```

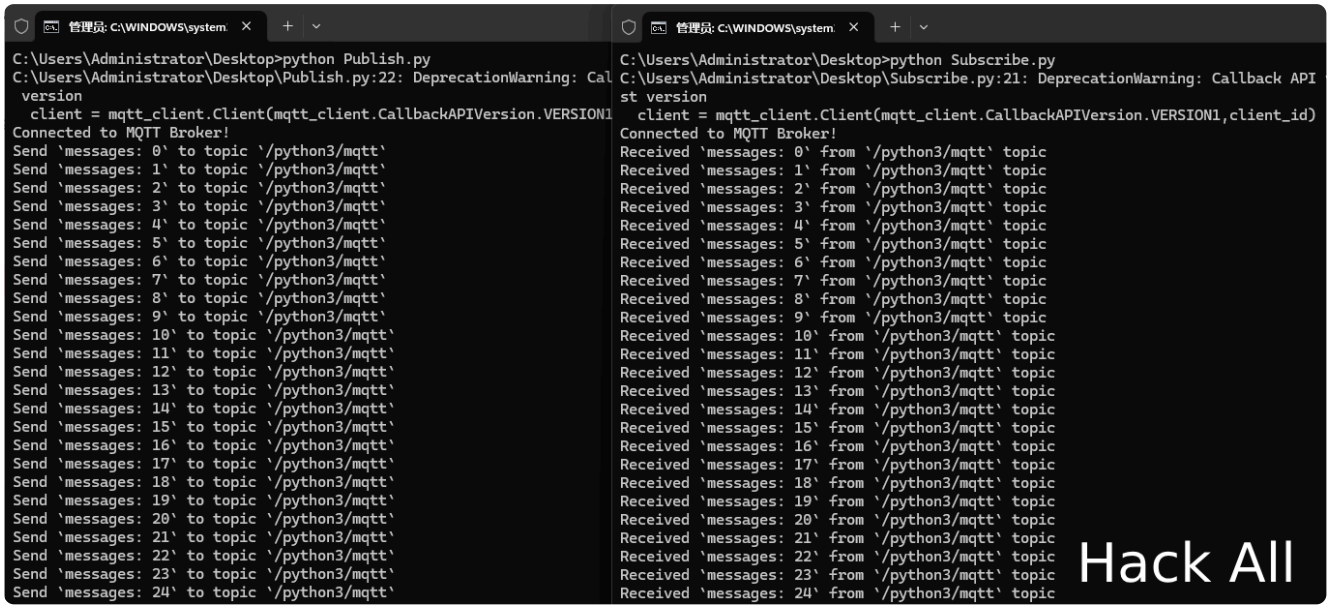
```python
from paho.mqtt import client as mqtt_client

# MQTT代理地址、端口、主题
broker = 'broker.emqx.io'
port = 1883
topic = "/python3/mqtt"
# 生成随机客户端ID
client_id = f'python-mqtt-{random.randint(0, 1000)}'

# 连接代理
def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    # 创建客户端，指定API版本
    client = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION1,client_id)
    # 绑定回调函数
    client.on_connect = on_connect
    # 连接代理
    client.connect(broker, port)
    return client


# 发布消息
def publish(client):
    msg_count = 0
    while True:
        # 每隔2秒发送消息
        time.sleep(2)
        msg = f"messages: {msg_count}"
        # 将QoS设置为2
        result = client.publish(topic, msg, qos=2)
        # result: [0, 1]
        status = result[0]
        if status == 0:
            print(f"Send `{msg}` to topic `{topic}`")
        else:
            print(f"Failed to send message to topic {topic}")
        msg_count += 1


def run():
    client = connect_mqtt()
    client.loop_start()
    publish(client)
```

```
if __name__ == '__main__':
    run()
```

订阅者代码:

```
import random

from paho.mqtt import client as mqtt_client

# MQTT代理地址、端口、主题
broker = 'broker.emqx.io'
port = 1883
topic = "/python3/mqtt"
# 生成随机客户端ID
client_id = f'python-mqtt-{random.randint(0, 100)}'

# 连接代理
def connect_mqtt() -> mqtt_client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)

    # 创建客户端，指定API版本
    client = mqtt_client.Client(mqtt_client.CallbackAPIVersion.VERSION1,client_id)
    # 绑定回调函数
    client.on_connect = on_connect
    # 连接代理
    client.connect(broker, port)
    return client


# 订阅主题
def subscribe(client: mqtt_client):
    def on_message(client, userdata, msg):
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")

    client.subscribe(topic)
    client.on_message = on_message


def run():
    client = connect_mqtt()
    subscribe(client)
    # 循环监听和接收服务器发送的消息
    client.loop_forever()
```

```
if __name__ == '__main__':
    run()
```



其他编程（Java、PHP、Golang、Nodejs、React、Django、Flask等）的使用方法请查阅：

https://www.emqx.com/zh/blog/category/mqtt-programming

EMQX的更多用法请查阅：官方文档

[如何安全的使用MQTT?]

MQTT 协议作为一种轻量级的消息传输协议，其安全性取决于实施的安全措施以及协议本身的设计。以下是一些可能存在的 MQTT 漏洞：

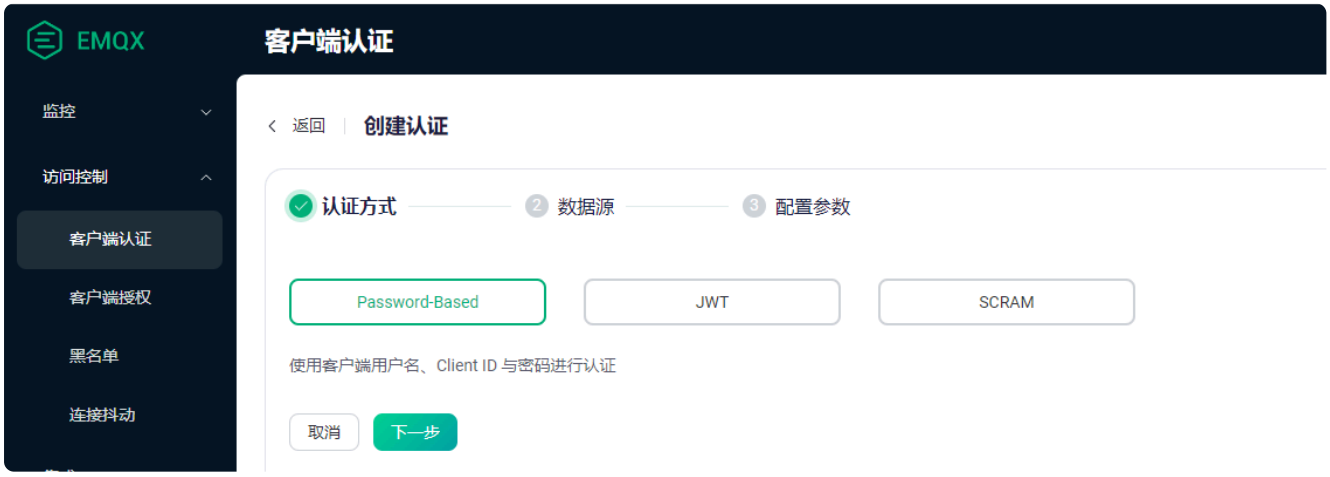
1. **未加密通信**：如果 MQTT 连接未使用 TLS/SSL 加密，那么通信内容可能会被窃听者截获，造成信息泄露的风险。
2. **未授权访问**：未对 MQTT 连接进行身份验证，或者使用弱密码进行身份验证，会导致未经授权的用户连接到 MQTT 代理，可能进行未授权的发布或订阅操作。
3. **拒绝服务攻击**：恶意客户端可能会发送大量无效的连接请求或消息，以耗尽服务器资源，导致拒绝服务攻击。
4. **主题猜测**：攻击者可能通过监视主题订阅情况来推断出敏感信息的结构和内容，从而进行信息收集或其他攻击。
5. **中间人攻击**：如果 MQTT 连接未加密，攻击者可以截获通信流量并进行中间人攻击，篡改或伪造通信内容。
6. **缓冲区溢出**：未经过正确的输入验证和处理的消息可能导致缓冲区溢出漏洞，从而被攻击者利用进行远程代码执行或拒绝服务攻击。

以下是确保 MQTT 使用安全性的一些关键步骤：

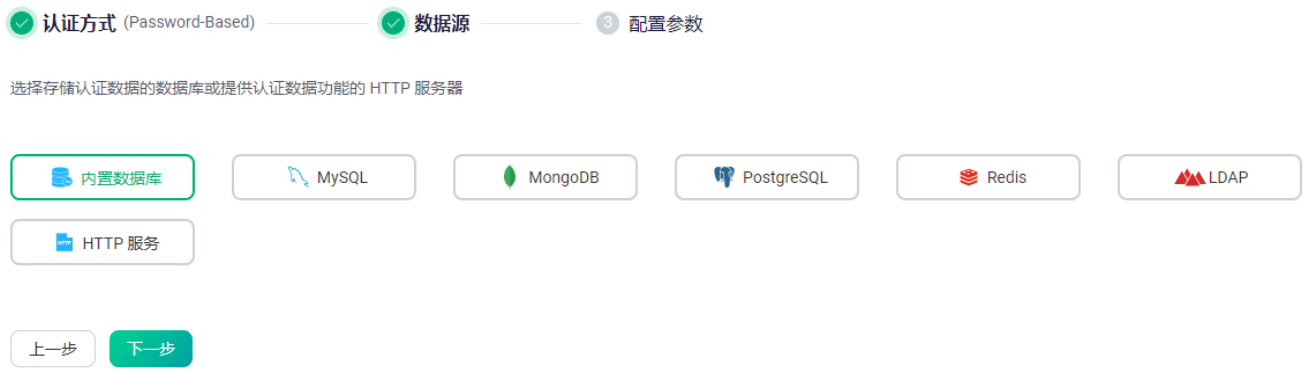
1. **使用 TLS/SSL 加密**：使用 TLS/SSL 加密 MQTT 连接，确保数据在传输过程中是加密的。这可以防止窃听者窃取或篡改数据。
2. **身份验证**：使用用户名和密码、客户端证书、OAuth 2.0 、Token等进行身份验证，只有经过授权的用户才能连接到 MQTT 代理。
3. **访问控制列表（ACL）**：使用 ACL 来限制客户端对主题的访问。通过 ACL，可以控制哪些客户端可以访问哪些主题，从而防止未经授权的访问。
4. **持久化会话**：启用持久化会话可以确保在客户端重新连接时，之前订阅的主题和未发布的消息不会丢失。这有助于确保数据的可靠传输。
5. **网络隔离**：将 MQTT 代理置于安全的网络环境中，并采取措施限制对其访问。这可以减少潜在的攻击面。
6. **定期更新和监控**：定期更新 MQTT 代理和相关软件，并监控其运行状态和网络活动。及时应用安全补丁和更新，以防止已知的安全漏洞被利用。
7. **安全审计和日志记录**：定期进行安全审计，检查系统配置和安全措施是否符合最佳实践，并记录重要的安全事件和活动，以便进行调查和分析。

> 配置身份验证

以EMQX为例，依次点击：访问控制→客户端认证→创建，可以选择基于用户名与密码、JWT和SCRAM认证方式：

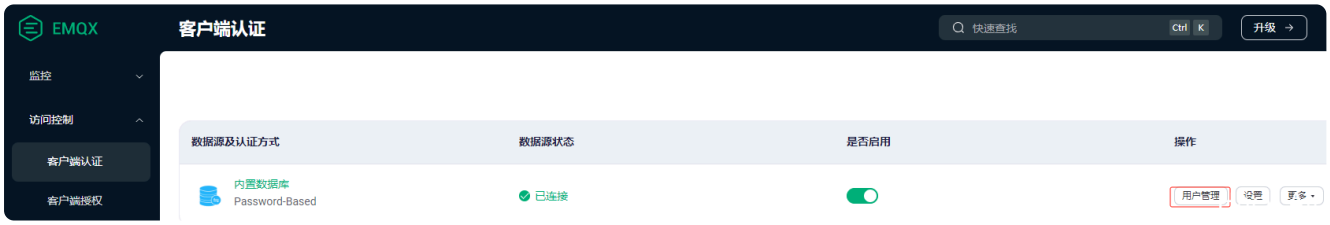


> ⓘ **基于用户名密码**

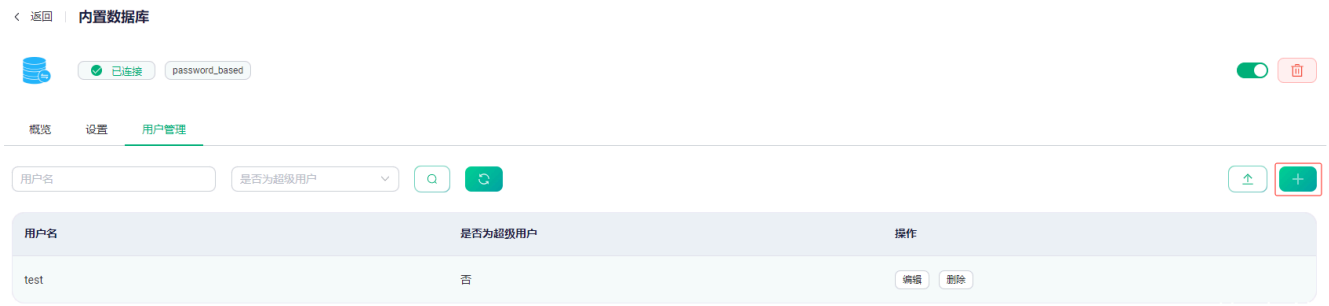


可选择内置数据库、外部数据库、LDAP和HTTP等多种数据源。以内置数据库为例，配置账号类型、密码密码加密方式和加盐方式：

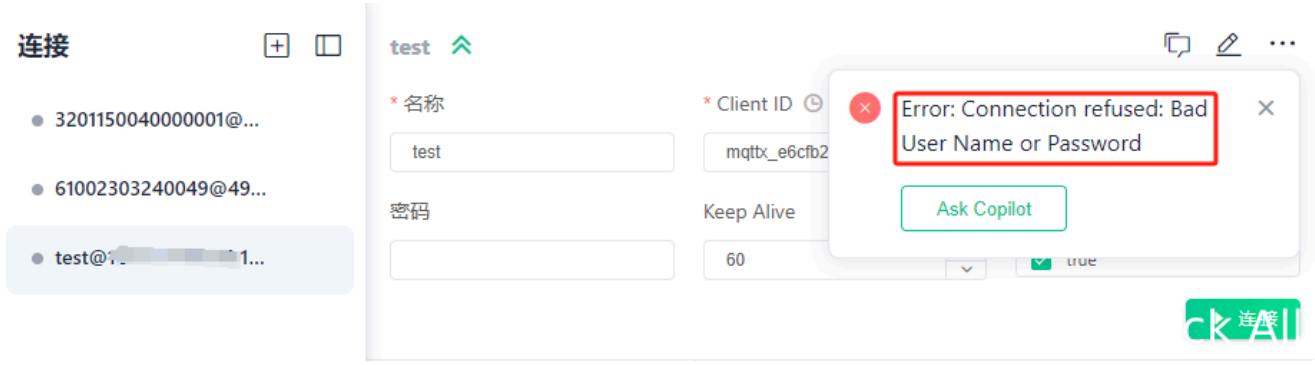
点击 创建 即可完成。



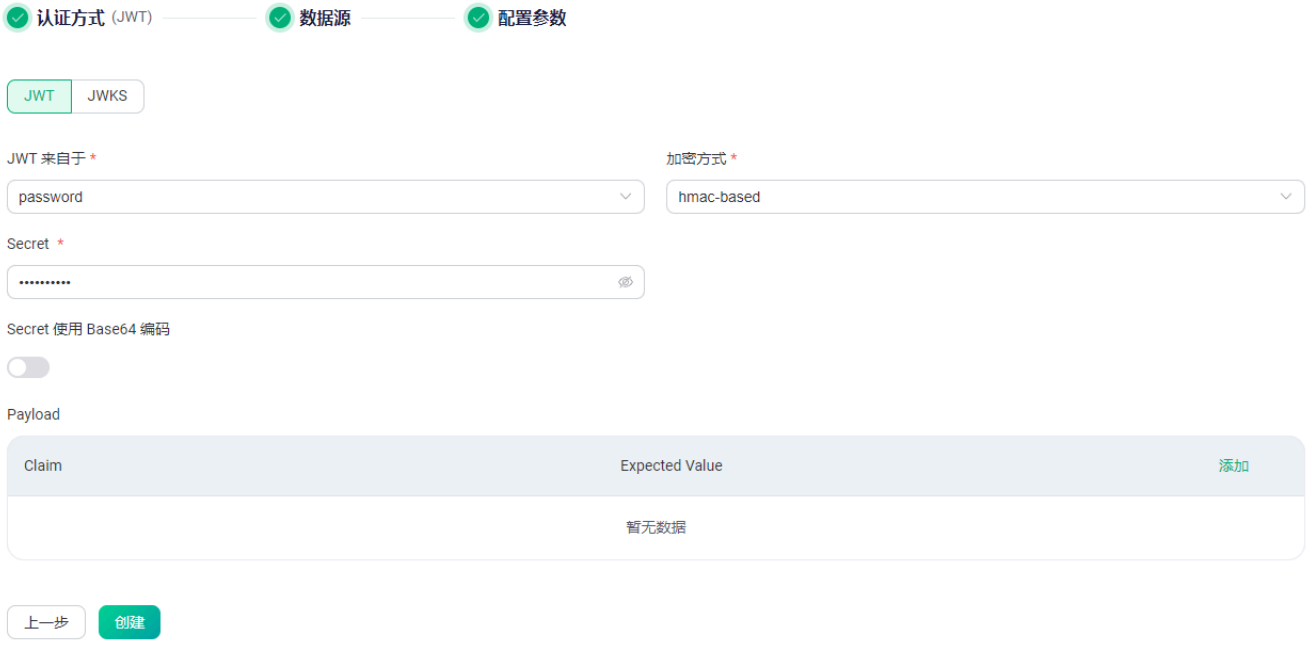
然后点击 用户管理→添加 配置用户名和密码：



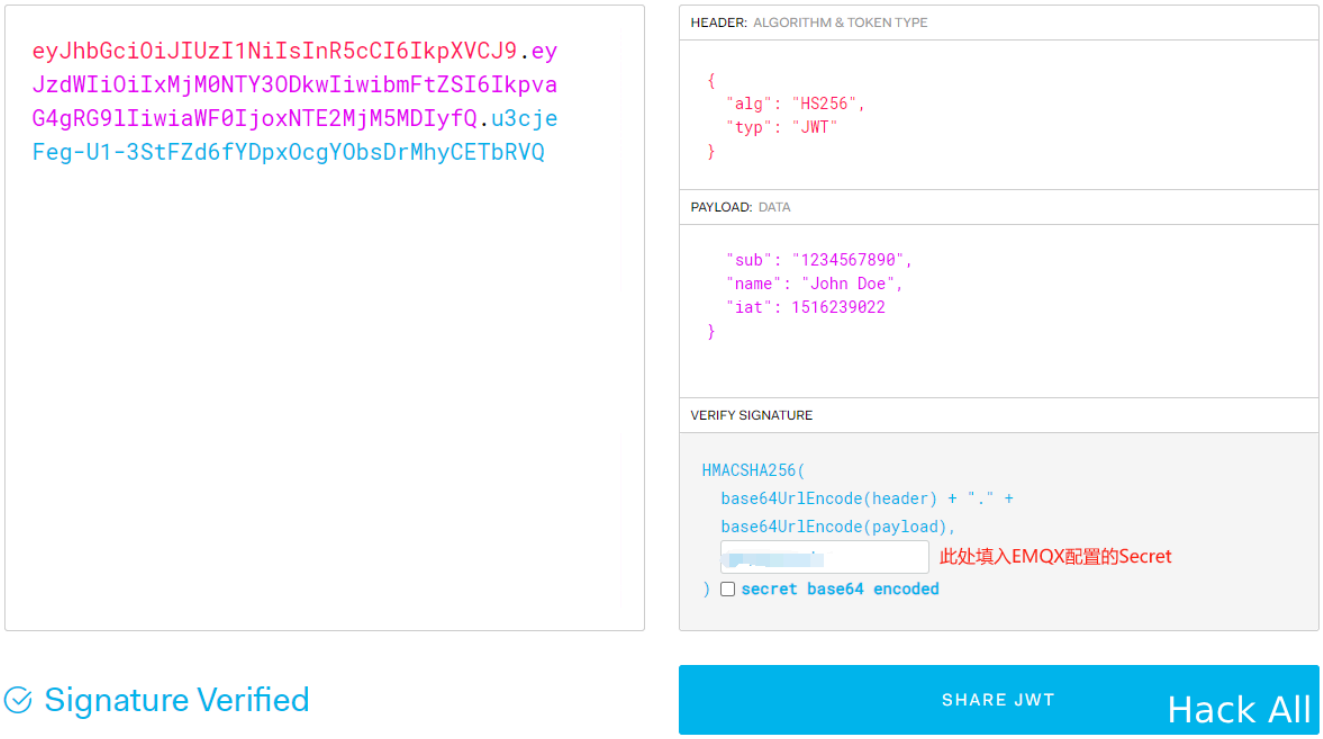
此时需要正确的用户名和密码才能连接，否则连接失败：



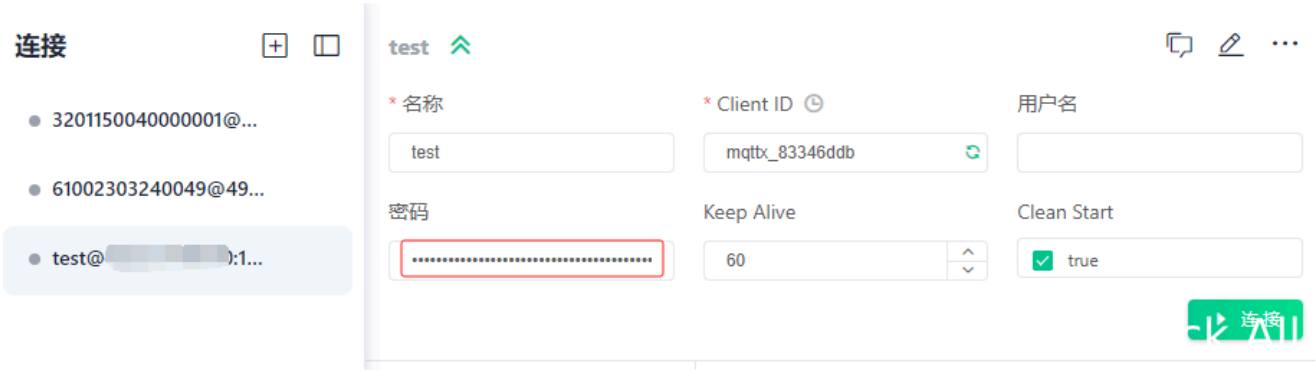
JWT无需数据源，直接配置参数即可，支持**JWT**和**JWKS**认证：



加密方式可以选择 `hmac-based` 或 `public-key` ， Payload可以自定义添加。创建成功后客户端需要使用相同的Secret生成JWT签名数据进行认证。测试时可以在jwt.io生成，如果EMQX启用了 `Secret使用Base64编码` 则需要勾选 `secret base64 encode` ：



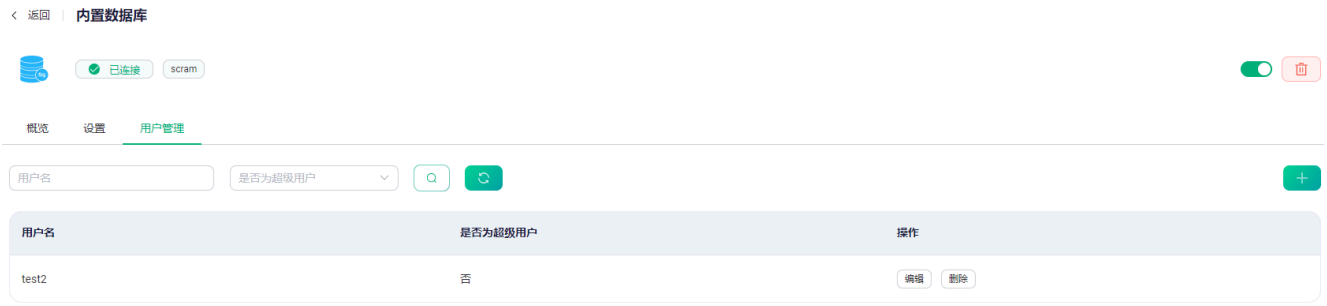
然后将签名数据填入 密码 字段即可连接成功：



EMQX更多JWT认证配置可查阅：https://www.emqx.io/docs/zh/latest/access-control/authn/jwt.html

SCRAM为MQTT 5.0特有的增强认证。加密方式支持 sha256 和 sha512：

创建成功后点击 用户管理→添加 配置用户名和密码：



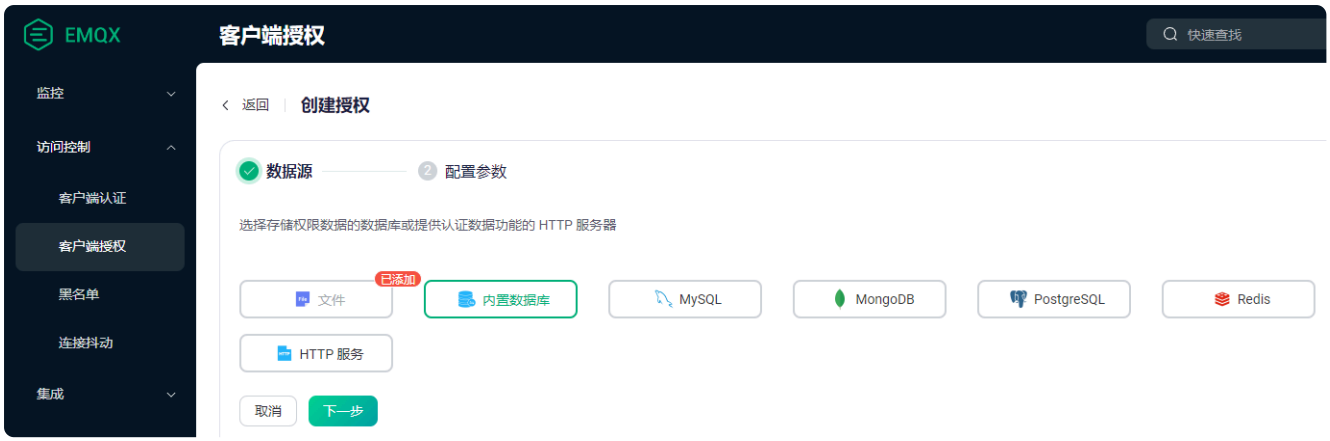
但是目前MQTTX不支持scram认证，EMQX官方也未提供详细的使用方法和示例代码，需要自行编写代码实现。

此外还支持 X.509 证书认证和 PSK 认证。关于EMQX认证的更多内容请查阅：https://www.emqx.io/docs/zh/latest/access-control/authn/authn.html

配置授权

以EMQX为例，授权是指对 MQTT 客户端的发布和订阅操作进行权限控制。EMQX 的授权机制基本原理是：当客户端尝试发布或订阅时，EMQX 会根据特定流程或用户定义的查询语句，从数据源中获取该客户端的权限数据，将权限与要执行的操作进行匹配，根据匹配结果来允许或拒绝本次操作。

依次点击： 访问控制→客户端授权→创建 ，可选择文件（ACL文件）、内置数据库、外部数据库和HTTP服务等：



ACL文件

基于文件的授权权限列表简单轻量，适合配置通用的规则。对于上百条或者更面向客户端的规则，推荐使用其他授权来源。

授权规则以 `Erlang 元组` 数据列表的形式存储在文件中。

基本语法和概念如下：

- 元组是用**花括号**包起来的一个列表，各个元素用**逗号**分隔
- 每条规则应以 `.` 结尾
- 注释行以 `%%` 开头，在解析过程中会被丢弃

ACL文件语法格式为：`{权限，客户端，操作，主题}`

第一个元素表示该条规则对应的**权限**；可选值：

- `allow` （允许）
- `deny` （拒绝）

第二个元素用来指定适用此条规则的**客户端**，比如：

- `{username, "dashboard"}`：用户名为 `dashboard` 的客户端；也可写作 `{user, "dashboard"}`
- `{username, {re, "^dash"}}`：用户名匹配正则表达式 `^dash` 的客户端
- `{clientid, "dashboard"}`：客户端 ID 为 `dashboard` 的客户端，也可写作 `{client, "dashboard"}`
- `{clientid, {re, "^dash"}}`：客户端 ID 匹配正则表达式 `^dash` 的客户端
- `{ipaddr, "127.0.0.1"}`：源地址为 `127.0.0.1` 的客户端；支持 CIDR 地址格式。注意：如果 EMQX 部署在负载均衡器后侧，建议为 EMQX 的监听器开启

`proxy_protocol` 配置，否则 EMQX 可能会使用负载均衡器的源地址。

- `{ipaddrs, ["127.0.0.1", ..., ]}`：来自多个源地址的客户端，不同 IP 地址之间以 `,` 区分
- `all`：匹配所有客户端
- `{'and', [Spec1, Spec2, ...]}`：满足列表中所有规范的客户端。
- `{'or', [Spec1, Spec2, ...]}`：满足列表中任何规范的客户端。

第三个元素用来指定该条规则对应的操作：

- `publish`：发布消息
- `subscribe`：订阅主题
- `all`：发布消息和订阅主题

- 从 v5.1.1 版本开始，EMQX 支持检查发布与订阅操作中的 QoS 与保留消息标志位，您可以在第三个元素中加上 `qos` 或 `retain` 来指定检查的 QoS 或保留消息标志位，例如：
  - `{publish, [{qos, 1}, {retain, false}]}`：拒绝发布 QoS 为 1 的保留消息
  - `{publish, {retain, true}}`：拒绝发布保留消息
  - `{subscribe, {qos, 2}}`：拒绝以 QoS2 订阅主题

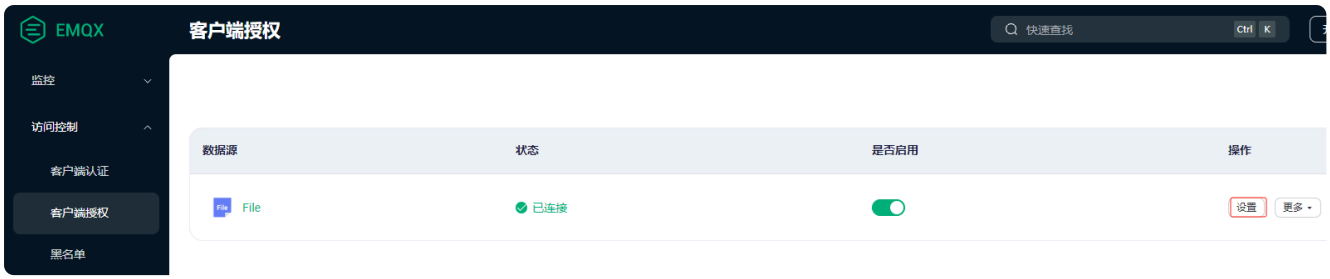
第四个元素用于指定当前规则适用的 MQTT 主题，支持通配符（主题过滤器），可以使用主题占位符：

- `"t/${clientid}"`：使用了主题占位符，当客户端 ID 为 `emqx_c` 的客户端触发检查时，将精确匹配 `t/emqx_c` 主题
- `"$SYS/#"`：通过通配符匹配 `$SYS/` 开头的所有主题，如 `$SYS/foo` 、 `$SYS/foo/bar`
- `{eq, "foo/#"}`：精确匹配 `foo/#` 主题，主题 `foo/bar` 将无法匹配，此处 `eq` 表示全等比较（equal）

另外还有 2 种特殊的规则，通常会用在 ACL 文件的末尾作为默认规则使用。

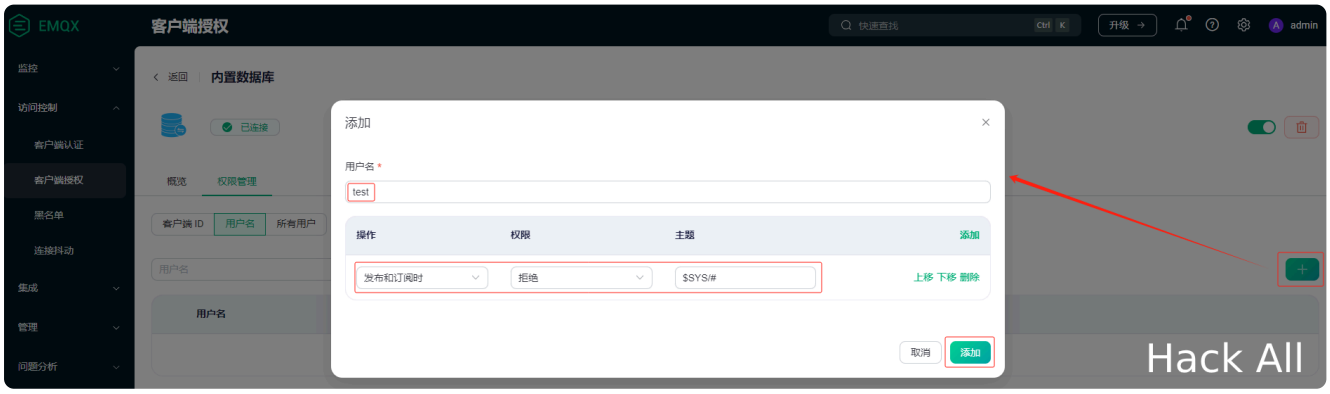
- `{allow, all}`：允许所有请求
- `{deny, all}`：拒绝所有请求

添加文件授权方式后 设置 即可编辑授权文件：



内置数据库

内置数据库授权无需配置参数，直接创建即可，点击 权限管理 即可配置授权。可以选择通过 客户端ID 、 用户名 或 所有用户 进行配置，如禁止用户test发布和订阅 $SYS/# 主题：



此时用户test无法订阅和发布 $SYS/# 主题：

如果仅允许用户admin订阅和发布特定主题时，配置允许admin用户订阅和发布特定主题，配置所有用户拒绝特定主题：

关于EMQX授权的更多内容请查阅：https://www.emqx.io/docs/zh/latest/access-control/authz/authz.html

连接抖动

EMQX 支持自动封禁那些短时间内频繁连接的客户端，并且在一段时间内拒绝这些客户端的连接，以避免此类客户端过多占用服务器资源。

连接抖动功能只会封禁**客户端 ID**，并不封禁用户名和 IP 地址，即该机器只要更换客户端 ID 就能够继续连接。但可以通过**黑名单**功能根据客户端ID、IP地址（段）、用户名和表达式等进行封禁。

依次点击： 访问控制→连接抖动→启用抖动检测→保存修改 即可：

此时多次连接即可触发连接抖动：

总的来说黑名单功能比较生硬，连接抖动功能又比较鸡肋，如果用户名和密码设置较为简单还是有机会被攻击者暴力破解的，所以一定要设置复杂度较高的密码。

NEXT POST
搭建私有Docker仓库