MQTT基础

写在前面
MQTT基于订阅发布机制，包括消息服务器broker和client

> 2024.1.21日更新
> 本文介绍了MQTT协议并基于mosquitto代理和paho-mqtt库开发了mnist的app代码。包括mqtt-mnist-input、mqtt-mnist-infer、mqtt-mnist-output。
> 具体代码https://github.com/kongfuguagua/MQTT-MNIST.git

目录

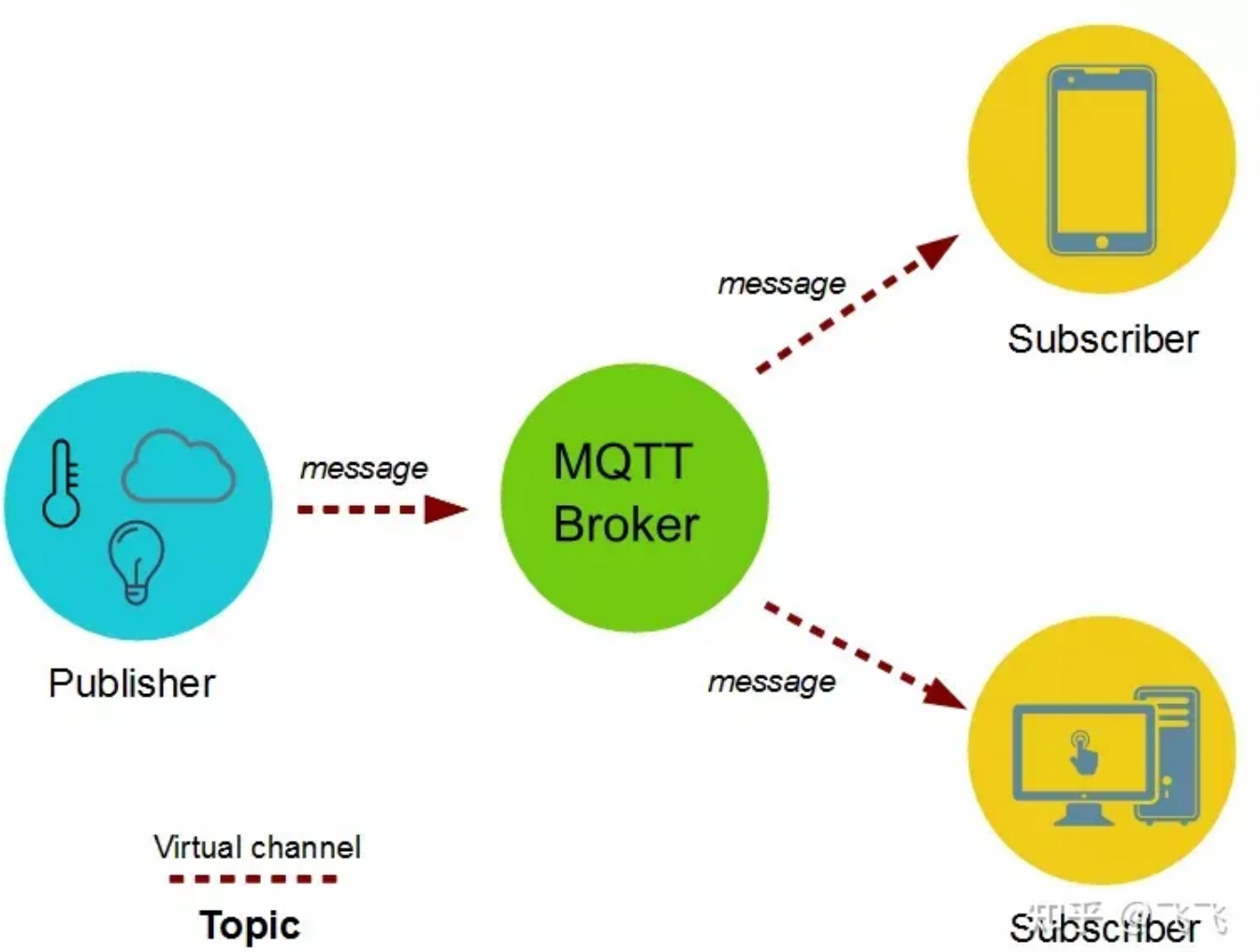
1MQTT介绍

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。MQTT最大优点在于，用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。

1.1订阅发布

发布订阅的思想在我们的生活中随处可见。以bilibili作为例子，up主制作视频发布在b站上，我们打开B站app观看视频，关注up主。B站即是一个broker（代理），up作为发布者publisher，用户作为subscriber。
MQTT使用的发布/订阅消息模式，它提供了一对多的消息分发机制，从而实现与应用程序的解耦。这是一种消息传递模式，**消息不是直接从发送器发送到接收器**（即点对点），而是由`MQTT`

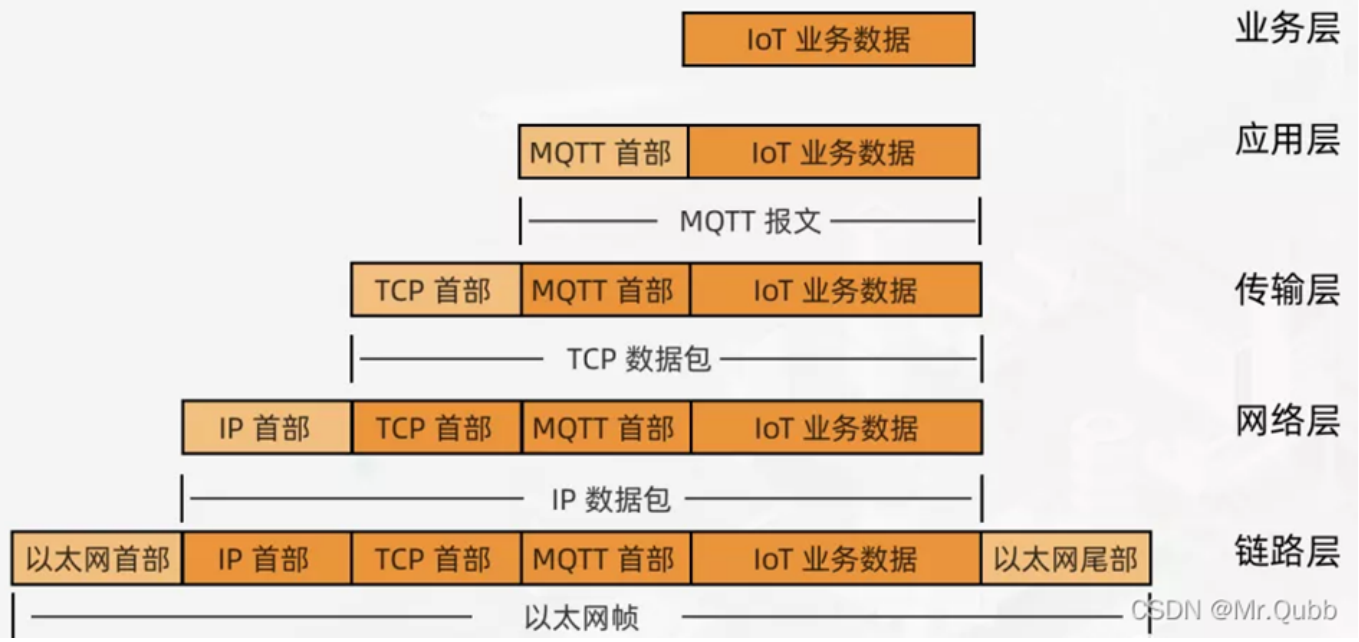
`server`（或称为 MQTT Broker）分发的。



1.2 MQTT vs TCP

MQTT是基于TCP/IP协议之上的应用层协议，与HTTP相似
1MQTT是应用层协议。优点在于实时性、轻量化和可选服务质量
2TCP是传输层协议。优点在于可靠
3MQTT存在broker，即消息服务器，作为信息交流的中心。

4TCP是流式协议， 每个IP存在65536个端口



1.3MQTT数据包

(1)QOS

服务质量是 MQTT 的一个重要特性。当我们使用 TCP/IP 时，连接已经在一定程度上受到保护。但是在无线网络中，中断和干扰很频繁，MQTT 在这里帮助避免信息丢失及其服务质量水平。这些级别在发布时使用。如果客户端发布到 MQTT 服务器，则客户端将是发送者，MQTT 服务器将是接收者。当MQTT服务器向客户端发布消息时，服务器是发送者，客户端是接收者。
QoS 0这一级别会发生消息丢失或重复，消息发布依赖于底层TCP/IP网络。即：<=1
QoS 1 承诺消息将至少传送一次给订阅者
QoS 2使用 QoS 2，我们保证消息仅传送到目的地一次。为此，带有唯一消息 ID 的消息会存储两次，首先来自发送者，然后是接收者。QoS 级别 2 在网络中具有最高的开销，因为在发送方和接收方之间需要两个流。

(2)MQTT 数据包结构

- `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；

整体MQTT的消息格式如下图所示；

1.4常见broker

目前MQTT代理的主流平台有下面几个：

- Mosquitto：https://mosquitto.org/
- VerneMQ：https://vernemq.com/
- EMQTT：http://emqtt.io/

2mosquitto配置

这里主要介绍WINDOWS和LINUX的mosquitto环境配置，容器化配置请看MQTT微服务链开发
Download | Eclipse Mosquitto下载页，自行选择版本和平台

2.1 mosquitto.conf

详细文档mosquitto.conf man page | Eclipse Mosquitto
mosquitto.conf是mosquitto的broker配置文件，必须在启动之前修改配置，否则无效。
常用配置为：

```
allow_anonymous true#测试场景允许匿名，如果false需要配置用户名和密码
listener 1883#监听端口号
#持久化配置
persistence true#持久化，数据会保存在broker
persistence_location /mosquitto/data#数据保存路径
log_dest file /mosquitto/log/mosquitto.log#日志保存路径
#bright配置
```

```
connection broker002#目标机名
address 10.102.182.96:1883#目标IP和port
topic # both 2#所有topic的双向通信
```

PC机配置只需要

```
allow_anonymous true#测试场景允许匿名，如果false需要配置用户名和密码
listener 1883#监听端口号
```

2.2mosquitto使用

mosquitto_pub

mosquitto_pub是推送消息，参数说明
-d 打印debug信息
-f 将指定文件的内容作为发送消息的内容
-h 指定要连接的域名 默认为localhost
-i 指定客户端clientid，默认为附加进程ID的mosquitto_pub
-I 指定clientId前缀
-m 消息内容
-n 发送一个空（null）消息
-p 连接端口号
-q 指定QoS的值（0,1,2）
-t 指定topic
-u 用户名
-P 用户密码
-V 指定MQTT协议版本
--will-payload 指定一个消息，该消息当客户端与broker意外断开连接时发出。该参数需要与--will-topic一起使用
--will-qos Will的QoS值。该参数需要与--will-topic一起使用
--will-retain 指定Will消息被当做一个retain消息（即消息被广播后，该消息被保留起来）。该参数需要与--will-topic一起使 用

mosquitto_sub

mosquitto_sub订阅主题，参数说明
- -c 指定客户端clean_session是否保存。
- -d 打印debug信息
- -h 指定要连接的域名 默认为localhost

- -i 指定客户端clientid
- -I 指定clientId前缀
- -k keepalive 每隔一段时间，发PING消息通知broker，仍处于连接状态。 默认为60秒.
- -q 指定希望接收到QoS为什么的消息 默认QoS为0
- -R 不显示陈旧的消息
- -t 订阅topic
- -v 打印消息
- --will-payload 指定一个消息，该消息当客户端与broker意外断开连接时发出。该参数需要与--will-topic一起使用
- --will-qos Will的QoS值。该参数需要与--will-topic一起使用
- --will-retain 指定Will消息被当做一个retain消息（即消息被广播后，该消息被保留起来）。该参数需要与--will-topic一起使>用
- --will-topic 用户发送Will消息的topic

2.3demo

```
mosquitto_sub -h localhost -t "topic/#" -i "client1"
mosquitto_pub -h localhost -t "topic/test" -i "client3" -m "How are you?"
```

3paho-mqtt简单使用

paho-mqtt是python的mqtt库
paho-mqtt · PyPI官方文档
sub

```
import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
```

```
client.connect("mqtt.eclipseprojects.io", 1883, 60)
client.subscribe('test')#订阅test
```

pub

```
import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("mqtt.eclipseprojects.io", 1883, 60)
client.publish('test','hello-world')#订阅test
```

4paho-mqtt二次开发

本章封装了mqtt，成为一个新类

```
class Mqtt:
    def __init__(self, clientIP, clientPort):
        self.__masterIP=masterIP
        self.clientPort = clientPort
        self.clientIP = clientIP
        self.client = mqtt.Client()
        self.client.on_connect = self.on_connect
        self.client.on_message = self.on_message
        self.image_index=0

    def on_connect(self, client, userdata, flags, rc):  #链接回调函数
        print("Connected with result code " + str(rc))

    def on_message(self, client, userdata, msg):  #发消息回调函数
        print(msg.topic + " " + ":" + str(msg.payload))
```

```
    def mqtt_connect(self):  # 链接MQTTbroker
        while 1:
            try:
                self.client.connect(self.clientIP, self.clientPort, 60)
                # self.SendOneSentence("client1")  # 先发个名字
                break
            except ConnectionRefusedError:
                print('由于目标计算机积极拒绝，无法连接')
                time.sleep(1)
            except Exception as e:
                print('client sock {} error: {}'.format((self.clientIP,
self.clientPort), e))
                # break

    def pub_topic(self):  #发布函数，发图片demo
        for i in range(5):
            image_data = self.image_preproess(r'{}.png'.format(i))
            self.client.publish('image', image_data)

    def image_preproess(self, filename, byte='high'):  #发图片预处理
        if os.path.isfile(filename):
            f = open(filename, "rb")
            fileContent = f.read()
            if byte == 'high':
                byteArr = bytes(fileContent)
            else:
                byteArr = base64.b64encode(fileContent)
        else:
            print('文件不存在')
            byteArr=None
        return byteArr

    def sub_topic(self):  # 订阅集合
        self.client.subscribe('test')
        self.client.message_callback_add('test', self.test_handle)# 订阅回调
        self.client.subscribe('image')
        self.client.message_callback_add('image', self.image_handle)

    def test_handle(self, client, userdata, msg):  # test主题回调
```

```
        a = threading.Thread(target=self.test_callback, args=(msg,))
        a.start()

    def image_handle(self, client, userdata, msg):  # image主题回调
        a = threading.Thread(target=self.image_callback, args=(msg,))
        a.start()

    def test_callback(self, msg):  # json接收demo
        print('线程号：', threading.get_ident())
        payload = json.loads(msg.payload)
        print(msg.topic)
        print(payload['name'])

    def image_callback(self, msg):  # 图片接收demo
        print('线程号：', threading.get_ident())
        f = open(r'./{}.png'.format(self.image_index), 'wb')
        self.image_index+=1
        payload = msg.payload
        f.write(payload)
        print(msg.topic)
```

5mnist实例

clone https://github.com/kongfuguagua/MQTT-MNIST.git
input

```
class Mnist_pub(Mqtt):
    def __init__(self, clientIP, clientPort, filelistname):
        super(Mnist_pub, self).__init__(clientIP, clientPort)
        self.getimagesaddr(filelistname)
        self.count = 0

    def getimagesaddr(self, filelistname):  #解析数据集
        f = open(filelistname, 'r')
        data_list = f.readlines()
        f.close()

        self.n_data = len(data_list)

        self.img_paths = []
```

```
        self.img_labels = []

        for data in data_list:
            self.img_paths.append(data[:-3])
            self.img_labels.append(data[-2])

    def pub_topic(self):  #发布图片
        while 1:
            image_data = self.image_preproess(
                os.path.abspath('./mnist_images/' + self.img_paths[self.count %
self.n_data]))
            self.count += 1
            self.client.publish('mnist', image_data)
            time.sleep(1)

    def main(self):  #主函数
        self.mqtt_connect()
        self.pub_topic()
```

infer

```
class server_infer(Mqtt, batch_net):
    def __init__(self,subIP, subPort, deal_images="deal_images"):
        self.__dealType = deal_images
        batch_net.__init__(self)
        Mqtt.__init__(self,subIP, subPort)
        self.NNLoad()

    def sub_topic(self):  # 订阅集合
        self.client.subscribe('mnist')
        self.client.message_callback_add('mnist', self.image_handle)

    def image_deal(self, filename):  #启动infer
        self.infer(filename)

    def NNoutput(self):  #结果输出
        print(self.predicted)
        self.pub_topic('result', self.predicted)

    def pub_topic(self, topic, ans):  #发布逻辑
```

```
        self.client.publish(topic, str(ans))

    def main(self):
        self.mqtt_connect()
        self.sub_topic()
        self.sub_loop_forever()
```

output

```
class MnistOutput(Mqtt):
    def __init__(self, masterIP, masterPort):
        super(MnistOutput, self).__init__(masterIP, masterPort)

    def sub_topic(self):  # 订阅集合
        self.client.subscribe('result')
        self.client.message_callback_add('result', self.result_handle)  # 订阅回调

    def result_handle(self, client, userdata, msg):  # result主题回调
        a = threading.Thread(target=self.result_callback, args=(msg,))
        a.start()

    def result_callback(self, msg):
        print('线程号：', threading.get_ident())
        print('{}:{}'.format(msg.topic,msg.payload))

    def main(self):
        self.mqtt_connect()
        self.sub_topic()
        self.sub_loop_forever()
```