
PAHO-MQTT 用户手册

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019



WWW.RT-THREAD.ORG

Wednesday 30th January, 2019

目录

目录	i
1 MQTT 软件包介绍	1
1.1 文件目录结构	1
1.2 RT-Thread 软件包功能特点	1
1.3 MQTT 简述	2
1.4 MQTT 功能介绍	2
1.4.1 MQTT 客户端	2
1.4.2 MQTT 服务器	3
1.4.3 MQTT 协议中的方法	3
1.4.4 MQTT 协议中的订阅、主题、会话	3
2 MQTT 示例程序	4
2.1 示例代码讲解	4
2.2 运行示例	8
3 MQTT 工作原理	9
3.1 MQTT 协议工作原理	9
4 MQTT 使用说明	10
4.1 准备工作	10
4.2 使用流程	11
4.2.1 设置代理信息	11
4.2.2 配置 MQTT 客户端结构体	12
4.2.3 启动 MQTT 客户端	13
4.2.4 向指定 Topic 推送消息	13
4.3 运行效果	13
4.4 参考资料	13

5 MQTT API 介绍	14
5.1 订阅列表	14
5.2 callback	14
5.3 MQTT_URI	15
5.4 paho_mqtt_start 接口	15
5.5 MQTT Publish 接口	15

第 1 章

MQTT 软件包介绍

Paho MQTT 是 Eclipse 实现的 MQTT 协议的客户端，本软件包是在 Eclipse [paho-mqtt](#) 源码包的基础上设计的一套 MQTT 客户端程序。

1.1 文件目录结构

```
pahomqtt
├── docs
│   ├── figures                // 文档使用图片
│   │   ├── api.md            // API 使用说明
│   │   ├── introduction.md   // 介绍文档
│   │   ├── principle.md     // 实现原理
│   │   ├── README.md        // 文档结构说明
│   │   ├── samples.md       // 软件包示例
│   │   └── user-guide.md     // 使用说明
│   └── version.md           // 版本
├── MQTTClient-RT             // 移植文件
├── MQTTPacket                // 源文件
├── samples                   // 示例代码
│   └── mqtt_sample.c         // 软件包应用示例代码
├── tests                     // mqtt 功能测试程序
│   ├── LICENSE               // 软件包许可证
│   └── README.md             // 软件包使用说明
└── SConscript                // RT-Thread 默认的构建脚本
```

1.2 RT-Thread 软件包功能特点

RT-Thread MQTT 客户端功能特点如下：

- 断线自动重连

RT-Thread MQTT 软件包实现了断线重连机制，在断网或网络不稳定导致连接断开时，会维护登陆状态，重新连接，并自动重新订阅 Topic。提高连接的可靠性，增加了软件包的易用性。

- pipe 模型，非阻塞 API
降低编程难度，提高代码运行效率，适用于高并发数据量小的情况。
- 事件回调机制
在建立连接、收到消息或者断开连接等事件时，可以执行自定义的回调函数。
- TLS 加密传输
MQTT 可以采用 TLS 加密方式传输，保证数据的安全性和完整性。

1.3 MQTT 简述

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。

1.4 MQTT 功能介绍

MQTT 协议运行在 TCP/IP 或其他网络协议之上, 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, 其特点包括:

1. 使用发布/订阅消息模式, 它提供了一对多消息分发, 以实现与应用程序的解耦。
2. 对负载内容屏蔽的消息传输机制。
3. 对传输消息有三种服务质量 (QoS):
 1. 最多一次, 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
 2. 至多一次, 这一级别会确保消息到达, 但消息可能会重复。即: ≥ 1
 3. 只有一次, 确保消息只有一次到达。即: $= 1$ 。在一些要求比较严格的计费系统中, 可以使用此级别。
4. 数据传输和协议交换的最小化 (协议头部只有 2 字节), 以减少网络流量。
5. 通知机制, 异常中断时通知传输双方。

1.4.1 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:

- 与服务器建立连接
- 发布其他客户端可能会订阅的信息
- 接收其它客户端发布的消息
- 退订已订阅的消息

1.4.2 MQTT 服务器

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。它是位于信息发布者和订阅者之间，它可以：

- 接受来自客户的网络连接
- 接收客户发布的应用信息
- 处理来自客户端的订阅和退订请求
- 向订阅的客户转发应用程序消息

1.4.3 MQTT 协议中的方法

MQTT 协议中定义了一些方法（也被称为动作），用来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。

- Connect: 等待与服务器建立连接。
- Disconnect: 等待 MQTT 客户端完成所做的工作，并与服务器断开 TCP/IP 会话。
- Subscribe: 等待完成订阅。
- UnSubscribe: 等待服务器取消客户端的一个或多个 Topics 订阅。
- Publish: MQTT 客户端发送消息请求，发送完成后返回应用程序线程。

1.4.4 MQTT 协议中的订阅、主题、会话

- 订阅（Subscription）

订阅包含主题筛选器（Topic Filter）和最大服务质量（QoS）。订阅会与一个会话（Session）关联。一个会话可以包含多个订阅。每一个会话中的每个订阅都有一个不同的主题筛选器。

- 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。

- 主题名（Topic Name）

连接到一个应用程序消息的标签，该标签与服务器的订阅相匹配。服务器会将消息发送给订阅所匹配标签的每个客户端。

- 主题筛选器（Topic Filter）

一个对主题名通配符筛选器，在订阅表达式中使用，表示订阅所匹配到的多个主题。

- 负载（Payload）

消息订阅者所具体接收的内容。

- 应用消息 Application Message

MQTT 协议通过网络传输应用数据。应用消息通过 MQTT 传输时，它们有关联的服务质量（QoS）和主题（Topic）。

- 控制报文 MQTT Control Packet

通过网络连接发送的信息数据包。MQTT 规范定义了十四种不同类型的控制报文，其中一个（PUBLISH 报文）用于传输应用消息。

第 2 章

MQTT 示例程序

2.1 示例代码讲解

下面讲解 RT-Thread 提供的 MQTT 示例代码，测试服务器使用 Eclipse 的测试服务器，地址 `iot.eclipse.org`，端口 `1883`，MQTT 功能示例代码如下：

```
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#include <rtthread.h>

#define DBG_ENABLE
#define DBG_SECTION_NAME "[MQTT]"
#define DBG_LEVEL DBG_LOG
#define DBG_COLOR
#include <rtdbg.h>

#include "paho_mqtt.h"
#define MQTT_URI "tcp://iot.eclipse.org:1883" // 配置测试服务器地址
#define MQTT_USERNAME "admin"
#define MQTT_PASSWORD "admin"
#define MQTT_SUBTOPIC "/mqtt/test" // 设置订阅主题
#define MQTT_PUBTOPIC "/mqtt/test" // 设置推送主题
#define MQTT_WILLMSG "Goodbye!" // 设置遗言消息

/* 定义 MQTT 客户端环境结构体 */
static MQTTClient client;

/* MQTT 订阅事件自定义回调函数 */
static void mqtt_sub_callback(MQTTClient *c, MessageData *msg_data)
{
    *((char *)msg_data->message->payload + msg_data->message->payloadlen) = '\0';
    LOG_D("mqtt sub callback: %.*s %.*s",
          msg_data->topicName->lenstring.len,
```

```

        msg_data->topicName->lenstring.data,
        msg_data->message->payloadlen,
        (char *)msg_data->message->payload);

    return;
}

/* MQTT 订阅事件默认回调函数 */
static void mqtt_sub_default_callback(MQTTClient *c, MessageData *msg_data)
{
    *((char *)msg_data->message->payload + msg_data->message->payloadlen) = '\\0';
    LOG_D("mqtt sub default callback: %.*s %.*s",
          msg_data->topicName->lenstring.len,
          msg_data->topicName->lenstring.data,
          msg_data->message->payloadlen,
          (char *)msg_data->message->payload);
    return;
}

/* MQTT 连接事件回调函数 */
static void mqtt_connect_callback(MQTTClient *c)
{
    LOG_D("inter mqtt_connect_callback!");
}

/* MQTT 上线事件回调函数 */
static void mqtt_online_callback(MQTTClient *c)
{
    LOG_D("inter mqtt_online_callback!");
}

/* MQTT 下线事件回调函数 */
static void mqtt_offline_callback(MQTTClient *c)
{
    LOG_D("inter mqtt_offline_callback!");
}

/**
 * 这个函数创建并配置 MQTT 客户端。
 *
 * @param void
 *
 * @return none
 */
static void mq_start(void)
{
    /* 使用 MQTTPacket_connectData_initializer 初始化 condata 参数 */
    MQTTPacket_connectData condata = MQTTPacket_connectData_initializer;
    static char cid[20] = { 0 };

```

```
static int is_started = 0;
if (is_started)
{
    return;
}
/* 配置 MQTT 结构体内容参数 */
{
    client.uri = MQTT_URI;

    /* 产生随机的客户端 ID */
    rt_snprintf(cid, sizeof(cid), "rtthread%d", rt_tick_get());

    /* 配置连接参数 */
    memcpy(&client.condata, &condata, sizeof(condata));
    client.condata.clientID.cstring = cid;
    client.condata.keepAliveInterval = 60;
    client.condata.cleansession = 1;
    client.condata.username.cstring = MQTT_USERNAME;
    client.condata.password.cstring = MQTT_PASSWORD;

    /* 配置 MQTT 遗言参数 */
    client.condata.willFlag = 1;
    client.condata.will.qos = 1;
    client.condata.will.retained = 0;
    client.condata.will.topicName.cstring = MQTT_PUBTOPIC;
    client.condata.will.message.cstring = MQTT_WILLMSG;

    /* 分配缓冲区 */
    client.buf_size = client.readbuf_size = 1024;
    client.buf = malloc(client.buf_size);
    client.readbuf = malloc(client.readbuf_size);
    if (!(client.buf && client.readbuf))
    {
        LOG_E("no memory for MQTT client buffer!");
        goto _exit;
    }

    /* 设置事件回调函数 */
    client.connect_callback = mqtt_connect_callback;
    client.online_callback = mqtt_online_callback;
    client.offline_callback = mqtt_offline_callback;

    /* 设置订阅表和事件回调函数*/
    client.messageHandlers[0].topicFilter = MQTT_SUBTOPIC;
    client.messageHandlers[0].callback = mqtt_sub_callback;
    client.messageHandlers[0].qos = QOS1;

    /* 设置默认的订阅主题*/
```

```

        client.defaultMessageHandler = mqtt_sub_default_callback;
    }

    /* 运行 MQTT 客户端 */
    paho_mqtt_start(&client);
    is_started = 1;

_exit:
    return;
}

/**
 * 这个函数推送消息给特定的 MQTT 主题。
 *
 * @param send_str publish message
 *
 * @return none
 */
static void mq_publish(const char *send_str)
{
    MQTTMessage message;
    const char *msg_str = send_str;
    const char *topic = MQTT_PUBTOPIC;
    message.qos = QOS1; //消息等级
    message.retained = 0;
    message.payload = (void *)msg_str;
    message.payloadlen = strlen(message.payload);

    MQTTPublish(&client, topic, &message);

    return;
}

#ifdef RT_USING_FINSH
#include <finsh.h>
FINSH_FUNCTION_EXPORT(mq_start, startup mqtt client);
FINSH_FUNCTION_EXPORT(mq_publish, publish mqtt msg);
#ifdef FINSH_USING_MSH
MSH_CMD_EXPORT(mq_start, startup mqtt client);

int mq_pub(int argc, char **argv)
{
    if (argc != 2)
    {
        rt_kprintf("More than two input parameters err!!\n");
        return 0;
    }
    mq_publish(argv[1]);
}

```

```
    return 0;
}
MSH_CMD_EXPORT(mq_pub, publish mqtt msg);
#endif /* FINSH_USING_MSH */
#endif /* RT_USING_FINSH */
```

2.2 运行示例

在 msh 中运行上述功能示例代码，可以实现向服务器订阅主题和向特定主题推送消息的功能，功能示例代码运行效果如下：

- 启动 MQTT 客户端，连接代理服务器并订阅主题：

```
msh />mq_start /* 启动 MQTT 客户端连接 Eclipse 服务器 */
inter mqtt_connect_callback! /* 服务器连接成功，调用连接回调函数打印服务器信息 */
ipv4 address port: 1883
[MQTT] HOST = 'iot.eclipse.org'
msh />[MQTT] Subscribe #0 /mqtt/test OK! /* 订阅主题 /mqtt/test 成功 */
inter mqtt_online_callback! /* MQTT 上线成功，调用上线回调函数 */
msh />
```

- 作为发布者向指定主题发布消息：

```
msh />mq_pub hello-rtthread /* 向指定主题发送 hello-rtthread 消息 */
msh />mqtt sub callback: /mqtt/test hello-rtthread /* 收到消息，执行回调函数 */
msh />
```

第 3 章

MQTT 工作原理

3.1 MQTT 协议工作原理

在 MQTT 协议中有三种身份：发布者（Publish）、代理（Broker）（服务器）和订阅者（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，这三者的关系如下图所示：



图 3.1: MQTT 工作原理示意图

在 MQTT 协议的实际使用过程中，一般遵循以下流程：

- 发布者通过代理服务器向指定的 Topic 发布消息。
- 订阅者通过代理服务器订阅所需要的 Topic。
- 订阅成功后如果有发布者向订阅者订阅的 Topic 发布消息，那么订阅者就会收到代理服务器的推送消息，通过这种方式可以进行高效的数据交换。

第 4 章

MQTT 使用说明

4.1 准备工作

首先需要下载 MQTT 软件包，并将软件包加入到项目中。在 BSP 目录下使用 menuconfig 命令打开 Env 配置界面，在 RT-Thread online packages → IoT - internet of things 中选择 Paho MQTT 软件包，操作界面如下图所示：

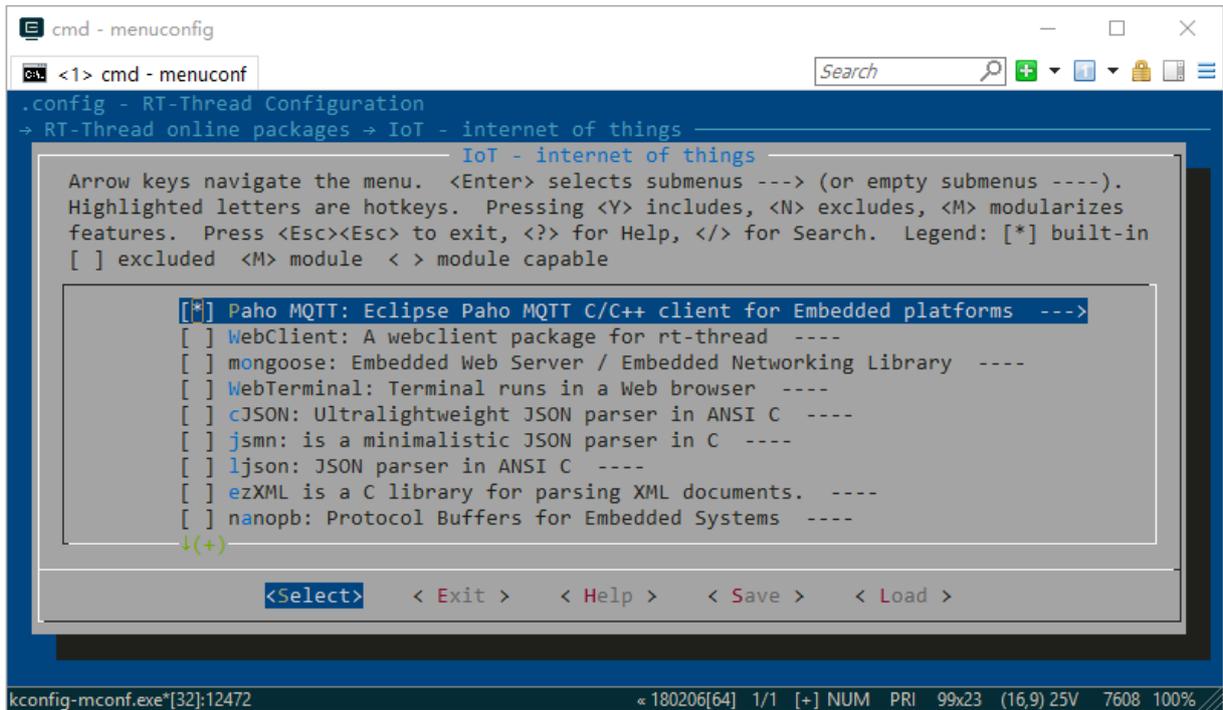


图 4.1: 选中 Paho MQTT 软件包

开启功能示例，便于测试 MQTT 功能：

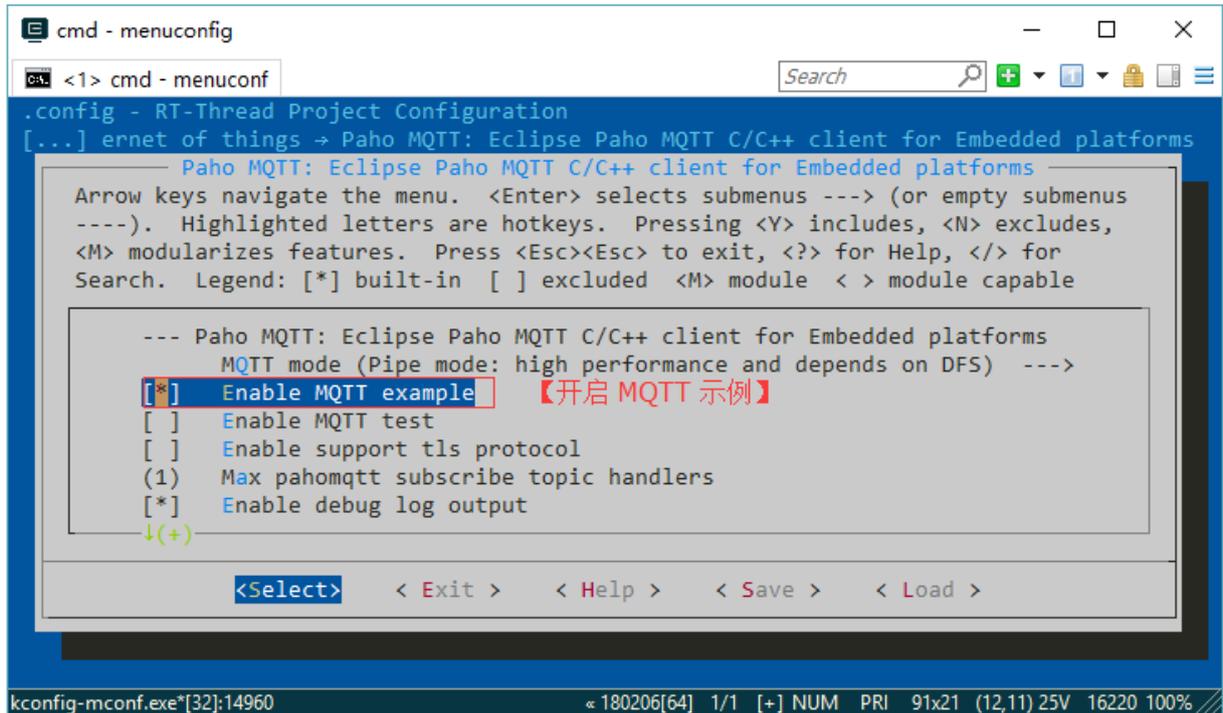


图 4.2: 开启 MQTT 软件包测试例程

配置项介绍如下：

```

--- Paho MQTT: Eclipse Paho MQTT C/C++ client for Embedded platforms
MQTT mode (Pipe mode: high performance and depends on DFS) --->#高级功能
[*] Enable MQTT example           #开启 MQTT 功能示例
[ ] Enable MQTT test              #开启 MQTT 测试例程
[ ] Enable support tls protocol   #开启 TLS 安全传输选项
(1) Max pahomqtt subscribe topic handlers #设置 Topic 最大订阅数量
[*] Enable debug log output       #开启调试Log输出
version (latest) --->            #选择软件包版本，默认为最新版

```

选择合适的配置项后，使用 `pkgs --update` 命令下载软件包并添加到工程中即可。

4.2 使用流程

这一节介绍 MQTT 软件包的配置参数和使用方法。

4.2.1 设置代理信息

首先要设置好代理服务器的地址，用户名、密码等必要信息。以 MQTT sample 为例有如下的设置：

```

#define MQTT_URI           "tcp://iot.eclipse.org:1883" //设置服务器地址
#define MQTT_USERNAME     "admin"                    //代理服务器用户名
#define MQTT_PASSWORD     "admin"                    //代理服务器密码
#define MQTT_SUBTOPIC     "/mqtt/test"               //订阅的 Topic
#define MQTT_PUBTOPIC     "/mqtt/test"               //推送的 Topic

```

```
#define MQTT_WILLMSG          "Goodbye!"          // 设置断开通知消息
```

4.2.2 配置 MQTT 客户端结构体

接下来需要初始化 MQTT 软件包客户端实例，将上一步设定的数据写入客户端实例的配置项，对客户端进行必要的配置，在这一步需要进行如下操作：

- 设置服务器地址，以及服务器账号密码等信息，示例代码如下：

```
/* 配置连接参数 */
memcpy(&client.condata, &condata, sizeof(condata));
client.condata.clientID.cstring = cid;
client.condata.keepAliveInterval = 60;
client.condata.cleansession = 1;
client.condata.username.cstring = MQTT_USERNAME;          // 设置账号
client.condata.password.cstring = MQTT_PASSWORD;          // 设置密码
```

- 设置消息等级、推送 Topic、以及断开通知消息等配置，示例如下：

```
/* 配置断开通知消息 */
client.condata.willFlag = 1;
client.condata.will.qos = 1;
client.condata.will.retained = 0;
client.condata.will.topicName.cstring = MQTT_PUBTOPIC;    // 设置推送主题
client.condata.will.message.cstring = MQTT_WILLMSG;       // 设置断开通知消息
```

- 设置事件回调函数，这里需要为事件设置回调函数，如连接成功事件、上线成功事件、下线事件等，示例代码如下：

```
/* 设置事件回调函数，回调函数需要自己编写，在例程中为回调函数留了空函数 */
client.connect_callback = mqtt_connect_callback;          // 设置连接回调函数
client.online_callback = mqtt_online_callback;           // 设置上线回调函数
client.offline_callback = mqtt_offline_callback;         // 设置下线回调函数
```

- 设置客户端订阅表，MQTT 客户端可以同时订阅多个 Topic，所以需要维护一个订阅表，在这一步需要为每一个 Topic 的订阅设置参数，主要包括 Topic 名称、该订阅的回调函数以及消息等级，代码示例如下：

```
/* 配置订阅表 */
client.messageHandlers[0].topicFilter = MQTT_SUBTOPIC; // 设置第一个订阅的 Topic
client.messageHandlers[0].callback = mqtt_sub_callback; // 设置该订阅的回调函数
client.messageHandlers[0].qos = QOS1;                  // 设置该订阅的消息等级
/* set default subscribe event callback */
client.defaultMessageHandler = mqtt_sub_default_callback; // 设置一个默认的回调函数，
    如果有订阅的 Topic 没有设置回调函数，则使用该默认回调函数
```

4.2.3 启动 MQTT 客户端

配置完成 MQTT 客户端实例后，需要启动客户端，代码示例如下：

```
/* 运行 MQTT 客户端 */
paho_mqtt_start(&client);
```

启动 MQTT 客户端之后，客户端会自动连接代理服务器，自动订阅已经设置的 Topic，根据事件执行回调函数进行数据的处理。

4.2.4 向指定 Topic 推送消息

连接服务器成功之后，便可以通过代理服务器向指定的 Topic 推送消息。推送消息时需要设置消息内容、Topic、消息等级等配置，示例代码如下：

```
MQTTMessage message;
const char *msg_str = send_str;
const char *topic = MQTT_PUBTOPIC;           // 设置指定 Topic
message.qos = QOS1;                          // 设置消息等级
message.retained = 0;
message.payload = (void *)msg_str;           // 设置消息内容
message.payloadlen = strlen(message.payload);
MQTTPublish(&client, topic, &message);      // 开始向指定 Topic 推送消息
```

4.3 运行效果

演示示例可以展示连接服务器、订阅 Topic、向指定 Topic 推送消息的功能，如下所示：

```
msh />mq_start                /* 启动 MQTT 客户端连接代理服务器 */
inter mqtt_connect_callback!   /* 连接成功，运行上线回调函数 */
ipv4 address port: 1883
[MQTT] HOST = 'iot.eclipse.org'
msh />[MQTT] Subscribe
inter mqtt_online_callback!    /* 上线成功，运行在线回调函数 */
msh />mq_pub hello-rtthread    /* 向指定 Topic 推送消息 */
msh />mqtt sub callback: /mqtt/test hello-rtthread /* 收到消息，执行回调函数 */
```

!!! note “注意事项” 需要注意正确填写 MQTT_USERNAME 和 MQTT_PASSWORD，如果 MQTT_USERNAME 和 MQTT_PASSWORD 填写错误，MQTT 客户端无法正确连接到 MQTT 服务器。

4.4 参考资料

- [MQTT 官网](#)
- [Paho 官网](#)
- [IBM MQTT 介绍](#)
- [Eclipse paho.mqtt 源码](#)

第 5 章

MQTT API 介绍

5.1 订阅列表

Paho MQTT 中采用订阅列表的形式进行多个 Topic 的订阅，订阅列表存储在 `MQTTClient` 结构体实例中，在 MQTT 启动前配置，如下所示：

```
... // 省略代码

MQTTClient client;

... // 省略代码

/* set subscribe table and event callback */
client.messageHandlers[0].topicFilter = MQTT_SUBTOPIC;
client.messageHandlers[0].callback = mqtt_sub_callback;
client.messageHandlers[0].qos = QOS1;
```

详细的代码讲解请参考 Samples 章节，订阅列表的最大数量可以由 `menuconfig` 中的 `Max pahoqtt subscribe topic handlers` 选项进行配置。

5.2 callback

paho-mqtt 使用 callback 的方式向用户提供 MQTT 的工作状态以及相关事件的处理，需要在 `MQTTClient` 结构体实例中注册使用。

callback 名称	描述
<code>connect_callback</code>	MQTT 连接成功的回调
<code>online_callback</code>	MQTT 客户端成功上线的回调
<code>offline_callback</code>	MQTT 客户端掉线的回调
<code>defaultMessageHandler</code>	默认的订阅消息接收回调

callback 名称	描述
messageHandlers[x].callback	订阅列表中对应的订阅消息接收回调

用户可以使用 `defaultMessageHandler` 回调默认处理接收到的订阅消息，也可以使用 `messageHandlers` 订阅列表，为 `messageHandlers` 数组中对应的每一个 Topic 提供一个独立的订阅消息接收回调。

5.3 MQTT_URI

paho-mqtt 中提供了 uri 解析功能，可以解析域名地址、ipv4 和 ipv6 地址，可解析 `tcp://` 和 `ssl://` 类型的 URI，用户只需要按照要求填写可用的 uri 即可。

- uri 示例：

```
domain 类型
tcp://iot.eclipse.org:1883

ipv4 类型
tcp://192.168.10.1:1883
ssl://192.168.10.1:1884

ipv6 类型
tcp://[fe80::20c:29ff:fe9a:a07e]:1883
ssl://[fe80::20c:29ff:fe9a:a07e]:1884
```

5.4 paho_mqtt_start 接口

```
int paho_mqtt_start(MQTTClient *client)
```

参数	描述
client	MQTT 客户端实例对象
return	0 : 成功; 其他: 失败

该函数启动 MQTT 客户端，根据配置项订阅相应的主题。

5.5 MQTT Publish 接口

- 功能：向指定的 Topic 主题发布 MQTT 消息。

```
int MQTTPublish(MQTTClient *c, const char *topicName, MQTTMessage *message)
```

参数	描述
c	MQTT 客户端实例对象
topicName	MQTT 消息发布主题
message	MQTT 消息内容
return	0 : 成功; 其他: 失败