umqtt- MQTT 客户端

版本：1.0

日期：2026-01-12

状态：受控/临时文件

文件保密级别：（请勾选 ■ ）		
绝密 □	保密 □	公开 ■

文件管控表

文件变更记录			
修订日期	最新版本	修订内容	编制
2026-01-12	1	首次发行	Wells Li

上海移远通信技术股份有限公司（以下简称“移远通信”）始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市闵行区田林路 1016 号科技绿洲 3 期（B 区）5 号楼　　邮编：200233
电话：+86 21 5108 6236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：http://www.quectel.com/cn/support/sales.htm。

如需技术支持或反馈我司技术文档中的问题，请随时登录网址：
http://www.quectel.com/cn/support/technical.htm 或发送邮件至：support@quectel.com。

前言

移远通信提供该文档内容以支持客户的产品设计。客户须按照文档中提供的规范、参数来设计产品。同时，您理解并同意，移远通信提供的参考设计仅作为示例。您同意在设计您目标产品时使用您独立的分析、评估和判断。在使用本文档所指导的任何硬软件或服务之前，请仔细阅读本声明。您在此承认并同意，尽管移远通信采取了商业范围内的合理努力来提供尽可能好的体验，但本文档和其所涉及服务是在“可用”基础上提供给您的。移远通信可在未事先通知的情况下，自行决定随时增加、修改或重述本文档。

使用和披露限制

许可协议

除非移远通信特别授权，否则我司所提供硬软件、材料和文档的接收方须对接收的内容保密，不得将其用于除本项目的实施与开展以外的任何其他目的。

版权声明

移远通信产品和本协议项下的第三方产品可能包含受移远通信或第三方材料、硬软件和文档版权保护的相关资料。除非事先得到书面同意，否则您不得获取、使用、向第三方披露我司所提供的文档和信息，或对此类受版权保护的资料进行复制、转载、抄袭、出版、展示、翻译、分发、合并、修改，或创造其衍生作品。移远通信或第三方对受版权保护的资料拥有专有权，不授予或转让任何专利、版权、商标或服务商标权的许可。为避免歧义，除了正常的非独家、免版税的产品使用许可，任何形式的购买都不可被视为授予许可。对于任何违反保密义务、未经授权使用或以其他非法形式恶意使用所述文档和信息的违法侵权行为，移远通信有权追究法律责任。

商标

除另行规定，本文档中的任何内容均不授予在广告、宣传或其他方面使用移远通信或第三方的任何商标、商号及名称，或其缩略语，或其仿冒品的权利。

第三方权利

您理解本文档可能涉及一个或多个属于第三方的硬软件和文档（“第三方材料”）。您对此类第三方材料的使用应受本文档的所有限制和义务约束。

移远通信针对第三方材料不做任何明示或暗示的保证或陈述，包括但不限于任何暗示或法定的适销性或特定用途的适用性、平静受益权、系统集成、信息准确性以及与许可技术或被许可人使用许可技术相关的不侵犯任何第三方知识产权的保证。本协议中的任何内容都不构成移远通信对任何移远通信产品或任何其他硬软件、设备、工具、信息或产品的开发、增强、修改、分销、营销、销售、提供销售或以其他方式维持生产的陈述或保证。此外，移远通信免除因交易过程、使用或贸易而产生的任何和所有保证。

隐私声明

为实现移远通信产品功能，特定设备数据将会上传至移远通信或第三方服务器（包括运营商、芯片供应商或您指定的服务器）。移远通信严格遵守相关法律法规，仅为实现产品功能之目的或在适用法律允许的情况下保留、使用、披露或以其他方式处理相关数据。当您与第三方进行数据交互前，请自行了解其隐私保护和数据安全政策。

免责声明

1) 移远通信不承担任何因未能遵守有关操作或设计规范而造成损害的责任。
2) 移远通信不承担因本文档中的任何因不准确、遗漏、或使用本文档中的信息而产生的任何责任。
3) 移远通信尽力确保开发中功能的完整性、准确性、及时性，但不排除上述功能错误或遗漏的可能。除非另有协议规定，否则移远通信对开发中功能的使用不做任何暗示或法定的保证。在适用法律允许的最大范围内，移远通信不对任何因使用开发中功能而遭受的损害承担责任，无论此类损害是否可以预见。
4) 移远通信对第三方网站及第三方资源的信息、内容、广告、商业报价、产品、服务和材料的可访问性、安全性、准确性、可用性、合法性和完整性不承担任何法律责任。

目录

1 模块概述

umqtt 是基于 Python 实现的轻量级 MQTT 客户端模块，专门为嵌入式系统和物联网设备设计。该模块提供了完整的 MQTT 协议支持，包括连接管理、消息发布/订阅、QoS 服务质量保证等核心功能。umqtt 包含两个子模块：

- `umqtt.simple`: 基础 MQTT 客户端实现
- `umqtt.robust`: 增强型 MQTT 客户端，提供自动重连和错误恢复机制

2 MQTT 协议简介

MQTT（Message Queuing Telemetry Transport）是一种轻量级的发布/订阅消息传输协议，专门为低带宽、高延迟或不稳定的网络环境设计。主要特点包括：

- **轻量级**: 协议头部最小仅 2 字节
- **发布/订阅模式**: 支持一对多消息分发
- **QoS 支持**: 提供三种消息服务质量级别
- **低功耗**: 适合电池供电的物联网设备

3 umqtt.simple 模块

3.1. 构造函数

3.1.1. 接口概述

创建 MQTT 客户端对象。

3.1.2. 函数原型

```
class
umqtt.simple.MQTTClient(client_id, server, port=0, user=None, password=None, keepalive=0, ssl=None)
```

3.1.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
client_id	是	string	-	客户端 ID，必须唯一
server	是	string	-	MQTT 服务器地址（IP 或域名）
port	否	int	0	服务器端口：0 表示自动选择（1883 非 SSL，8883 SSL）
user	否	string	None	用户名
password	否	string	None	密码
keepalive	否	int	0	心跳保活时间（秒），0 表示禁用
ssl	否	tls.SSLContext	None	TLS 安全连接配置： • None：禁用 TLS • tls.SSLContext 对象：启用 TLS 并使用自定义安全配置

3.1.4. 返回值描述

返回 MQTTClient 对象。

3.1.5. 示例

```
from umqtt import simple

# 创建 MQTT 客户端
client = simple.MQTTClient(
    client_id="device_001",
    server="mqtt.eclipseprojects.io",
    port=1883
)
```

3.2. connect

3.2.1. 接口概述

连接到 MQTT 服务器。

3.2.2. 函数原型

MQTTClient.connect(clean_session=True, timeout=None)

3.2.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
clean_session	否	bool	True	是否清理会话：True 清理，False 保留
timeout	否	int	None	连接超时时间（秒）

3.2.4. 返回值描述

成功返回 0，失败则抛出异常。

3.2.5. 示例

```
# 连接到 MQTT 服务器
try:
    session_present = client.connect()
    print(f"连接成功，会话状态: {session_present}")
except Exception as e:
    print(f"连接失败: {e}")
```

3.3. disconnect

3.3.1. 接口概述

断开与 MQTT 服务器的连接。

3.3.2. 函数原型

MQTTClient.disconnect()

3.3.3. 参数描述

无。

3.3.4. 返回值描述

无返回值。

3.3.5. 示例

```
# 断开连接
client.disconnect()
```

3.4. publish

3.4.1. 接口概述

发布消息到指定主题。

3.4.2. 函数原型

MQTTClient.publish(topic, msg, retain=False, qos=0)

3.4.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
topic	是	bytes/str	-	消息主题
msg	是	bytes/str	-	消息内容
retain	否	bool	False	是否保留消息
qos	否	int	0	服务质量：0-最多一次，1-至少一次

3.4.4. 返回值描述

无返回值。

3.4.5. 示例

```
# 发布消息
client.publish(b"sensors/temperature", b"25.6", qos=1)
client.publish("sensors/humidity", "60%", retain=True)
```

3.5. subscribe

3.5.1. 接口概述

订阅指定主题。

3.5.2. 函数原型

MQTTClient.subscribe(topic, qos=0)

3.5.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
topic	是	bytes/str	-	消息主题
qos	否	int	0	服务质量：0-最多一次，1-至少一次

3.5.4. 返回值描述

无返回值。

3.5.5. 示例

```
# 消息回调函数
def on_message(topic, msg):
    print(f"主题: {topic.decode()}")
    print(f"消息: {msg.decode()}")
# 设置消息回调
client.set_callback(on_message)
# 订阅主题
client.subscribe(b"sensors/#", qos=1)
client.subscribe("control/device001")
```

3.6. wait_msg

3.6.1. 接口概述

阻塞等待接收消息。

3.6.2. 函数原型

MQTTClient.wait_msg()

3.6.3. 参数描述

无。

3.6.4. 返回值描述

返回操作码（内部使用）。

3.6.5. 示例

```
# 设置消息回调函数
def on_message(topic, msg):
    print(f"收到消息: {topic} -> {msg}")
```

```
client.set_callback(on_message)

# 阻塞等待消息
while True:
    client.wait_msg()
```

3.7. check_msg

3.7.1. 接口概述

非阻塞检查是否有消息。

3.7.2. 函数原型

MQTTClient.check_msg()

3.7.3. 参数描述

无。

3.7.4. 返回值描述

返回操作码或 None（无消息时）。

3.7.5. 示例

```
# 非阻塞检查消息
while True:
    op = client.check_msg()
    if op is None:
        time.sleep(0.1)  # 无消息时短暂等待
```

4 umqtt.robust 模块

4.1. 构造函数

4.1.1. 接口概述

创建增强型 MQTT 客户端（支持自动重连）。

4.1.2. 函数原型

class umqtt.simple.MQTTClient(client_id, server, port=0, user=None, password=None, keepalive=0, ssl=None)

4.1.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
client_id	是	string	-	客户端 ID，必须唯一
server	是	string	-	MQTT 服务器地址（IP 或域名）
port	否	int	0	服务器端口：0 表示自动选择（1883 非 SSL，8883 SSL）
user	否	string	None	用户名
password	否	string	None	密码
keepalive	否	int	0	心跳保活时间（秒），0 表示禁用
ssl	否	tls.SSLContext	None	TLS 安全连接配置： • None：禁用 TLS • tls.SSLContext 对象：启用 TLS 并使用自定义安全配置

4.1.4. 返回值描述

返回 MQTTClient 对象。

4.1.5. 示例

```
from umqtt import robust

client = robust.MQTTClient(
    client_id="robust_client",
    server="mqtt.eclipseprojects.io"
)
```

4.2. 其他函数

其他函数的使用方式与 umqtt.simple 一致。

5 TLS 安全连接

5.1. 概述

umqtt 模块支持通过 TLS 协议建立安全的 MQTT 连接，使用 tls.SSLContext 对象进行安全配置。TLS 加密确保数据在传输过程中的机密性和完整性，有效防止中间人攻击和数据窃听，适用于对安全性要求较高的物联网应用场景。

5.2. 连接配置

5.2.1. 接口概述

配置 TLS 安全连接，必须使用 tls.SSLContext 对象。

5.2.2. 函数原型

```
MQTTClient(
    ...,
    ssl=tls_context
)
```

5.2.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
ssl	否	tls.SSLContext	None	TLS 安全连接配置： • None：禁用 TLS • tls.SSLContext 对象：启用 TLS 并使用自定义安全配置

5.3. TLS 上下文配置

5.3.1. 创建 TLS 上下文

5.3.1.1. 接口概述

创建 TLS 客户端上下文对象。

5.3.1.2. 函数原型

```
import tls
tls_context = tls.SSLContext(protocol)
```

5.3.1.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
protocol	是	int	-	TLS 协议版本： • tls.PROTOCOL_TLS_CLIENT：TLS 客户端协议

5.3.1.4. 示例

```
import tls

# 创建 TLS 客户端上下文
context = tls.SSLContext(tls.PROTOCOL_TLS_CLIENT)
```

5.3.2. 证书验证模式

5.3.2.1. 接口概述

设置服务器证书验证模式。

5.3.2.2. 函数原型

```
tls_context.verify_mode = mode
```

5.3.2.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
mode	是	int	-	验证模式： • tls.CERT_NONE：不验证服务器证书（不安全） • tls.CERT_REQUIRED：必须验证服务器证书（推荐）

5.3.2.4. 示例

```
import tls

context = tls.SSLContext(tls.PROTOCOL_TLS_CLIENT)
context.verify_mode = tls.CERT_REQUIRED  # 必须验证服务器证书
```

5.3.3. 证书验证模式

5.3.3.1. 接口概述

设置服务器证书验证模式。

5.3.3.2. 函数原型

```
tls_context.verify_mode = mode
```

5.3.3.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
mode	是	int	-	验证模式： • tls.CERT_NONE：不验证服务器证书（不安全） • tls.CERT_REQUIRED：必须验证服务器证书（推荐）

5.3.3.4. 示例

```
import tls

context = tls.SSLContext(tls.PROTOCOL_TLS_CLIENT)
context.verify_mode = tls.CERT_REQUIRED  # 必须验证服务器证书
```

5.4. 证书管理

5.4.1. 加载 CA 证书

5.4.1.1. 接口概述

加载可信的根证书（CA 证书），用于验证服务器证书。

5.4.1.2. 函数原型

```
tls_context.load_verify_locations(cadata)
```

5.4.1.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
cadata	是	str	-	CA 证书内容（PEM 格式字符串）

5.4.1.4. 示例

```
import tls

context = tls.SSLContext(tls.PROTOCOL_TLS_CLIENT)
context.verify_mode = tls.CERT_REQUIRED

# 直接传入 CA 证书内容（PEM 格式）
ca_content = """-----BEGIN CERTIFICATE-----
MIIFkjCCA3qgAwIBAgIQBQ3aPQqNQmT4C5N4Lp1OJzANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VcnQgR2xvYmFsIFJvb3QgRzIwHhcNMjAw
OTIzMDAwMDAwWhcNMzAwOTIyMjM1OTU5WjBeMQswCQYDVQQGEwJVUzEVMBMGA1U
```

```
EChMMRGlnaUNl
cnQgSW5jMSUwIwYDVQQDExxEaWdpQ2VydCBUcnVzdGVkIEc0IFJvb3QgQ0EgRzIwggIiMA
0GCSqG
SIb3DQEBAQUAA4ICDwAwggIKAoICAQCt2hF5pz5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5
z
5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z
5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z5z
...
-----END CERTIFICATE-----"""

context.load_verify_locations(ca_content)
```

5.4.2. 加载客户端证书和私钥

5.4.2.1. 接口概述

加载客户端证书和私钥，用于双向认证（客户端认证）。

5.4.2.2. 函数原型

```
tls_context.load_cert_chain(cert, key)
```

5.4.2.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
cert	是	str	-	客户端证书内容（PEM 格式字符串）
key	是	str	-	客户端私钥内容（PEM 格式字符串）

5.4.2.4. 示例

```
import tls

context = tls.SSLContext(tls.PROTOCOL_TLS_CLIENT)
context.verify_mode = tls.CERT_REQUIRED

# 加载 CA 证书
ca_content = """-----BEGIN CERTIFICATE-----
...  # CA 证书内容
-----END CERTIFICATE-----"""
context.load_verify_locations(ca_content)

# 客户端证书内容
client_cert = """-----BEGIN CERTIFICATE-----
...  # 客户端证书内容
-----END CERTIFICATE-----"""

# 客户端私钥内容
client_key = """-----BEGIN PRIVATE KEY-----
```

```
... # 客户端私钥内容
-----END PRIVATE KEY-----"""

# 加载客户端证书和私钥
context.load_cert_chain(client_cert, client_key)
```