GR533x Mesh Light Lightness Models示例手册

版本： **1.1**

发布日期： **2023-11-06**

深圳市汇顶科技股份有限公司

版权所有 © 2023 深圳市汇顶科技股份有限公司。保留一切权利。

非经本公司书面许可，任何单位和个人不得对本手册内的任何部分擅自摘抄、复制、修改、翻译、传播，或将其全部或部分用于商业用途。

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田保税区腾飞工业大厦B座12-13层

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档介绍了GR533x SDK中Light Lightness Models示例的运行、验证以及关键代码等，旨在帮助用户快速进行二次开发。

读者对象

本文适用于以下读者：

- 芯片用户
- 开发人员
- 测试人员
- 技术支持工程师

版本说明

本文档为第2次发布，对应的产品系列为GR533x。

修订记录

版本	日期	修订内容
1.0	2023-10-18	首次发布
1.1	2023-11-06	更新GRUart获取方式。

版权所有 © 2023 深圳市汇顶科技股份有限公司

目录

前言 I

1 简介 1

2 LLN Models概述 2

2.1 发布消息类型 3

3 运行示例 5

3.1 准备工作 5

3.2 连接硬件 5

3.3 下载固件 6

3.4 运行程序 6

3.5 配置串口 6

3.6 测试验证 7

3.6.1 GRMesh配置设备 8

3.6.2 GRUart验证LLN Models 11

4 应用详解 14

4.1 工程目录 14

4.1.1 LLN Client Model工程 14

4.1.2 LLN Server Model工程 14

4.2 代码介绍 15

4.2.1 LLN Models消息处理 15

4.2.1.1 LLN Client Model 16

4.2.1.1.1 可处理消息表 16

4.2.1.1.2 回调函数处理表 16

4.2.1.1.3 发送消息处理 19

4.2.1.2 LLN Server Model 23

4.2.1.2.1 可处理消息表 23

4.2.1.2.2 回调函数处理表 23

4.2.1.2.3 发送消息处理 29

4.2.1.3 LLNS Server Model 30

4.2.1.3.1 可处理消息表 30

4.2.1.3.2 回调函数处理表 30

4.2.1.3.3 发送消息处理 31

4.2.2 LLN Models状态功能处理 33

4.2.2.1 LLN model状态绑定 33

5 常见问题 38

5.1 工程下载失败 38

5.2 设备初始化失败 38

版权所有 © 2023 深圳市汇顶科技股份有限公司

5.3 GRMesh未发现设备.....39
5.4 Client和Server设备之间不能收发数据.....39
5.5 未触发Binding.....39

版权所有 © 2023 深圳市汇顶科技股份有限公司

1 简介

为保证各类蓝牙Mesh设备之间消息交互的兼容性，蓝牙技术联盟（Bluetooth Special Interest Group，Bluetooth SIG）定义了一系列蓝牙Mesh应用领域的通用标准Models，如Generics、Sensors、Time and Scenes、Lighting等。利用这些标准Models，蓝牙Mesh设备能轻松控制对端蓝牙Mesh设备或获取相关信息，实现Mesh在不同场景下的应用。

本文以Lighting Models所包含的Light Lightness（LLN）Models为例，介绍如何使用标准Models以及基于标准Models进行开发。

进行操作前，可参考以下文档。

表 1-1 文档参考

名称	描述
GR533x开发者指南	介绍GR533x SDK以及基于SDK的应用开发和调试
Bluetooth Core Spec	Bluetooth官方标准核心规范
Mesh协议/Mesh Model规范	Bluetooth官方Mesh协议及规范：www.bluetooth.com/specifications/mesh-specifications/
J-Link用户指南	J-Link的使用说明：www.segger.com/downloads/jlink/UM08001_JLink.pdf
Keil用户指南	Keil详细操作说明：www.keil.com/support/man/docs/uv4/

版权所有 © 2023 深圳市汇顶科技股份有限公司

2 LLN Models概述

Goodix提供参考版Light Lightness（LLN）Models，用于展示标准Models的实现及使用。

LLN Models包括：

- 客户端模型（LLN Client Model）：用于向服务器模型发送消息，以获取、更改服务器模型的开/关、亮度等级、默认亮度和亮度范围等信息。
- 服务器模型包括以下两种模型：
  - LLN Server Model：接收来自客户端模型的消息，并根据客户端的请求，回复、更改服务器模型中的开/关、亮度等级。
  - Light Lightness Setup（LLNS）Server Model：接收来自客户端模型的消息，并根据客户端的请求，回复、更改服务器模型中的默认亮度、亮度范围。

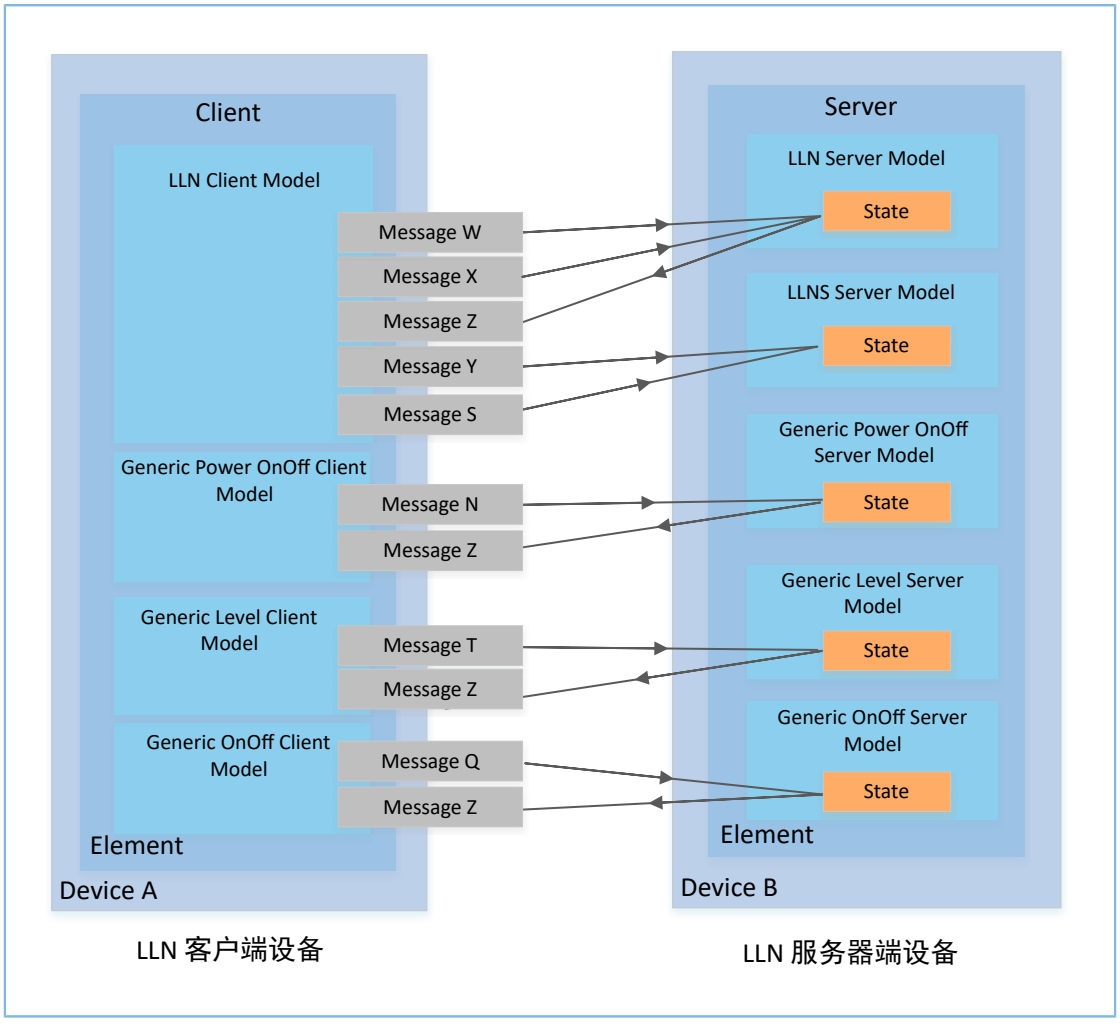

图 2-1 LLN客户端模型和服务端模型之间消息交互

如图 2-1所示，一个完整的LLN Server Element包括：

- 两个主Server Models：
  - LLN Server Model

- LLNS Server Model

- 三个扩展Server Models：
  - Generic Power OnOff Server Model
  - Generic Level Server Model
  - Generic OnOff Server Model

主Server Models继承了扩展Server Models的行为，且这两类Models的的状态具有绑定关系。

为了完成与服务器设备的交互，LLN客户端设备应包含与Server Models对应的Client Models，包括：

- LLN Client Model：对应LLN Model和LLNS Servers Model
- Generic Power OnOff Client Model：对应Generic Power OnOff Server Model
- Generic Level Client Model：对应Generic Level Server Model
- Generic OnOff Client Model：对应Generic OnOff Server Model

2.1 发布消息类型

LLN Client Model支持的发布消息类型如下表所示。

表 2-1 LLN Client Model发布消息类型

消息类型	消息描述
LIGHT_LIGHTNESS_OPCODE_GET	获取Server设备当前的Lightness Actual状态，并等待回复。
LIGHT_LIGHTNESS_OPCODE_SET	设置Server设备的当前的Lightness Actual状态，并等待回复。
LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED	设置Server设备当前的Lightness Actual状态，无需等待回复。
LIGHT_LIGHTNESS_LINEAR_OPCODE_GET	获取Server设备当前的Lightness Linear状态，并等待回复。
LIGHT_LIGHTNESS_LINEAR_OPCODE_SET	设置Server设备当前的Lightness Linear状态，并等待回复。
LIGHT_LIGHTNESS_LINEAR_OPCODE_SET_UNACKNOWLEDGED	设置Server设备当前的Lightness Linear状态，无需等待回复。
LIGHT_LIGHTNESS_LAST_OPCODE_GET	获取Server设备的Lightness Last状态，并等待回复。
LIGHT_LIGHTNESS_DEFAULT_OPCODE_GET	获取Server设备的Lightness Default状态，并等待回复。
LIGHT_LIGHTNESS_RANGE_OPCODE_GET	获取Server设备的Lightness Range状态，并等待回复。
LIGHT_LIGHTNESS_DEFAULT_OPCODE_SET	设置Server设备的当前Lightness Default状态，并等待回复。

版权所有 © 2023 深圳市汇顶科技股份有限公司

消息类型	消息描述
LIGHT_LIGHTNESS_DEFAULT_OPCODE_SET_UNACKNOWLEDGED	设置Server端的当前Lightness Default状态，无需等待回复。
LIGHT_LIGHTNESS_RANGE_OPCODE_SET	设置Server端的当前Lightness Range状态并等待回复。
LIGHT_LIGHTNESS_RANGE_OPCODE_SET_UNACKNOWLEDGED	设置Server端的当前Lightness Range状态，无需等待回复。

LLN Server Model支持的发布消息类型如下表所示。

表 2-2 LLN Server Model发布消息类型

消息类型	消息描述
LIGHT_LIGHTNESS_OPCODE_STATUS	发布Server设备当前的Lightness Actual状态。
LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS	发布Server设备当前的Lightness Linear状态。
LIGHT_LIGHTNESS_LAST_OPCODE_STATUS	发布Server设备的Lightness Last状态。
LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS	发布Server设备的Lightness Default状态。
LIGHT_LIGHTNESS_RANGE_OPCODE_STATUS	发布Server设备的Lightness Range状态。

说明:

LLNS Server Model不支持消息发布，仅接收、处理LLN Client Model发送的消息。如需发布消息，则由对应的LLN Server Model发布该消息。

版权所有 © 2023 深圳市汇顶科技股份有限公司

3 运行示例

本章介绍如何运行LLN Models示例工程，并测试/验证运行结果。

LLN Models示例工程包括：

- LLN Client Models示例工程：位于`SDK_Folder\projects\mesh\SIG\mesh_app_light_lightness_client`
- LLN Server Models示例工程：位于`SDK_Folder\projects\mesh\SIG\mesh_app_light_lightness_server`

说明:

SDK_Folder为GR533x SDK的根目录。

3.1 准备工作

运行LLN Models示例工程之前，请完成以下准备工作。

- 硬件准备

表 3-1 硬件准备

名称	描述
J-Link工具	SEGGER公司推出的JTAG仿真器，如需更多了解，请访问：www.segger.com/products/debug-probes/j-link/
开发板	GR5331 Starter Kit开发板（2块）
数据线	Type-C USB数据线

- 软件准备

表 3-2 软件准备

名称	描述
Windows	Windows 7/Windows 10操作系统
J-Link Driver	J-Link驱动程序，下载网址：www.segger.com/downloads/jlink/
Keil MDK5	IDE工具，下载网址：www.keil.com/download/product/
GRMesh（Android）	Mesh调试工具
GRUart	串口调试工具，下载网址：www.goodix.com/zh/download?objectId=64&objectType=software

3.2 连接硬件

使用两块GR5331 Starter Kit开发板（以下简称“SK板”），分别作为LLN Client设备和LLN Server设备。

使用Type-C USB数据线连接SK板与PC。

版权所有 © 2023 深圳市汇顶科技股份有限公司

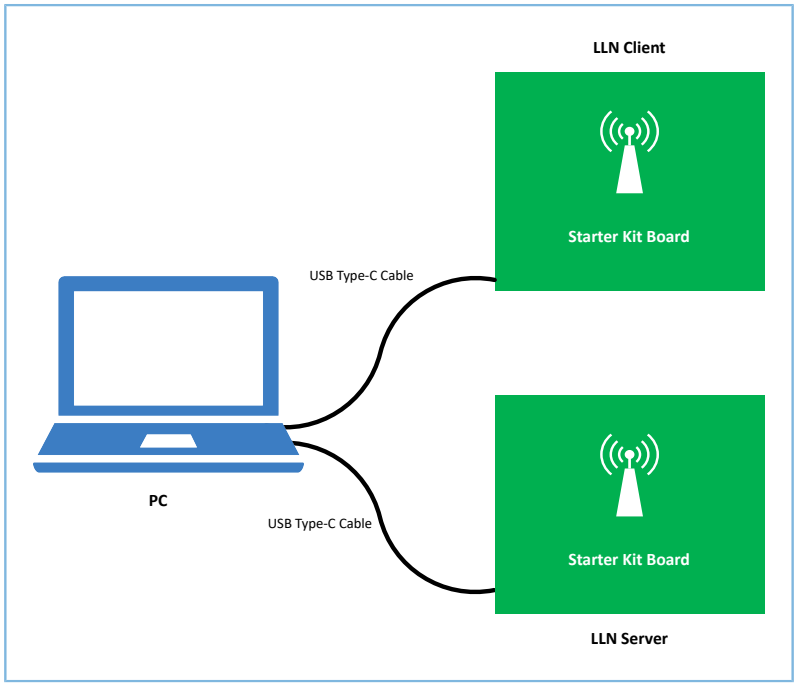

图 3-1 硬件连接示意图

3.3 下载固件

使用Keil工具编译mesh_app_light_lightness_client、mesh_app_light_lightness_server工程，并将生成的目标文件分别下载至两块SK板。

3.4 运行程序

固件下载至SK板后，按下开发板上的复位键，即可运行示例程序。

3.5 配置串口

启动GRUart，按照下表中的参数配置串口。

表 3-3 GRUart串口配置参数

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
需根据实际选择	115200	8	None	1	不勾选

GRUart配置完成后如下图所示。

版权所有 © 2023 深圳市汇顶科技股份有限公司

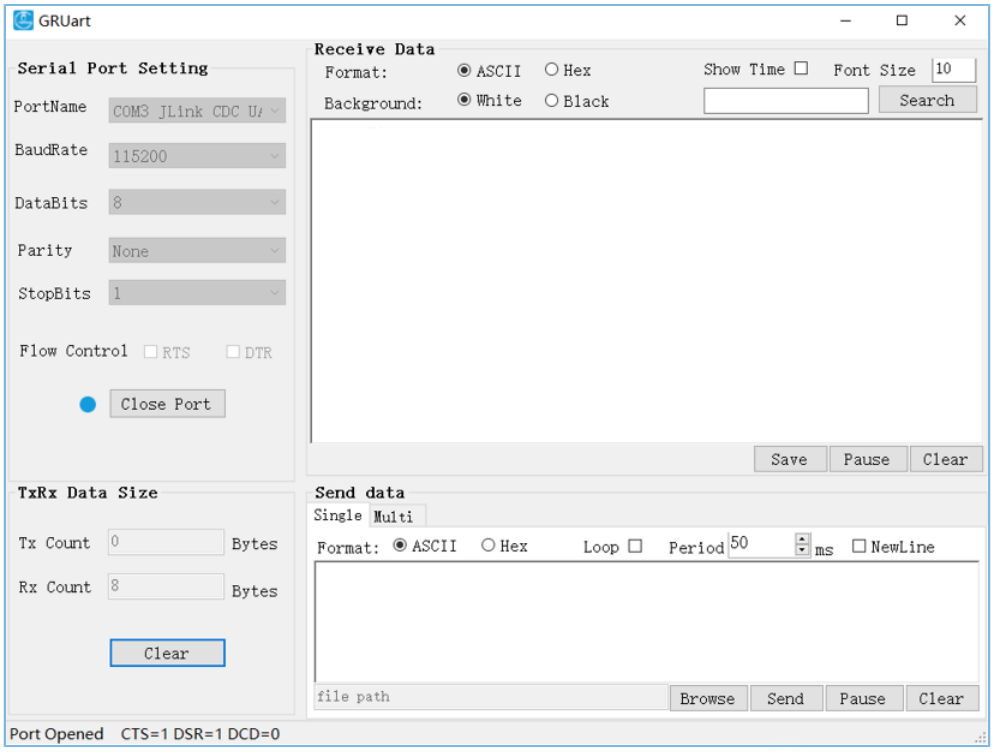

图 3-2 串口配置

3.6 测试验证

以上工作准备就绪后，可按以下步骤测试、验证LLN Models之间的通信。

1. 使用GRMesh配置LLN Client设备和LLN Server设备。
2. 通过GRUart打印输出的日志信息，验证LLN Client设备与LLN Server设备间的消息交互是否正常。

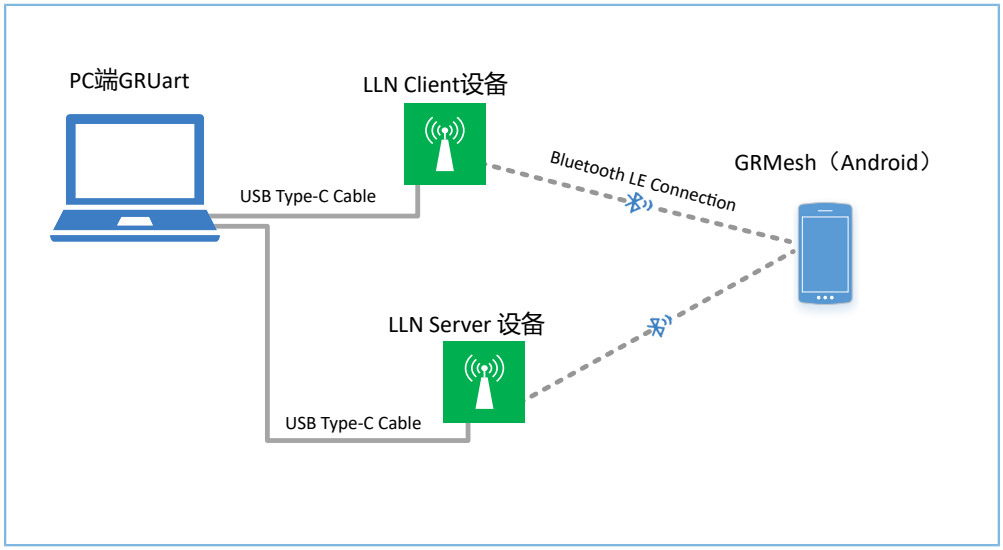

图 3-3 LLN Models测试场景图

版权所有 © 2023 深圳市汇顶科技股份有限公司

3.6.1 GRMesh配置设备

使用GRMesh配置Client设备、Server设备，操作步骤如下：

1. 上电LLN Client设备（Goodix_LLNC）和LLN Server设备（Goodix_LLNS），即两块SK板。

2. 使用GRMesh开通配置Goodix_LLNC设备。

   (1) 在GRmesh首页，点击右上角的“+”按钮，添加新设备。扫描发现广播名为Goodix_LLNC的设备。

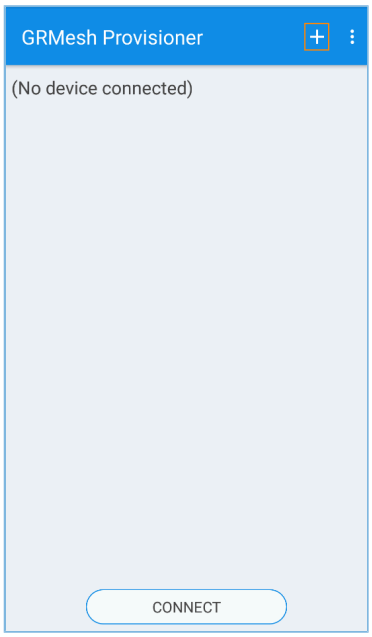

图 3-4 添加新设备

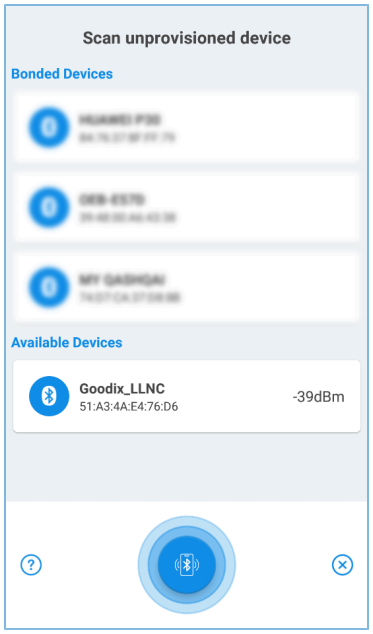

图 3-5 发现新设备

   (2) 点击“Goodix_LLNC > IDENTIFY > PROVISION”。

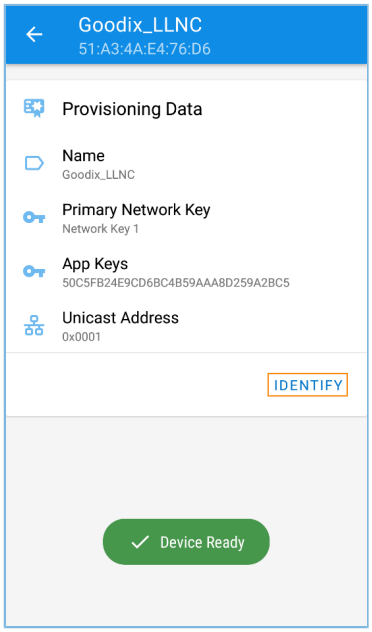

图 3-6 点击“IDENTIFY”

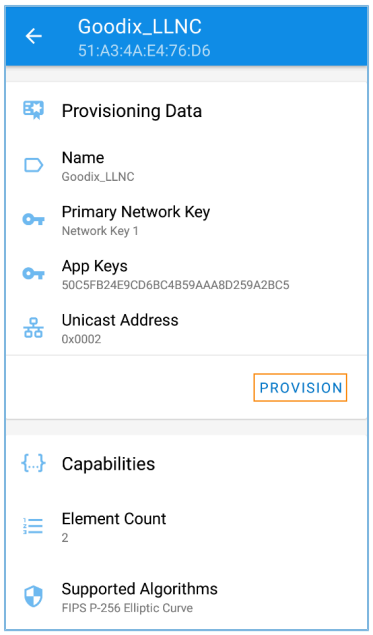

图 3-7 点击“PROVISION”

版权所有 © 2023 深圳市汇顶科技股份有限公司

(3) 在弹出的“Select OOB Type”窗口中，设置配对方式为“No OOB”并点击“OK”，完成Goodix_LLNC设备的开通配置，即将LLN Client设备加入到Mesh网络。

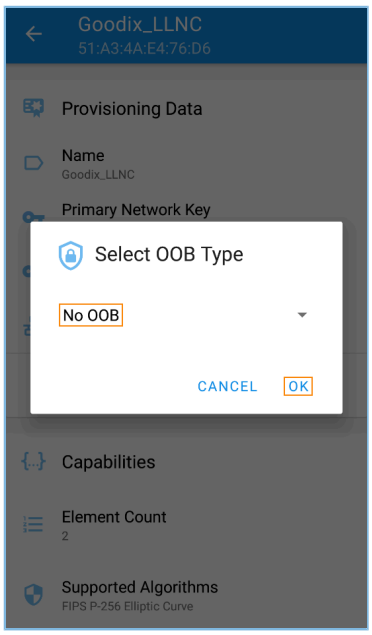

图 3-8 点击“OK”

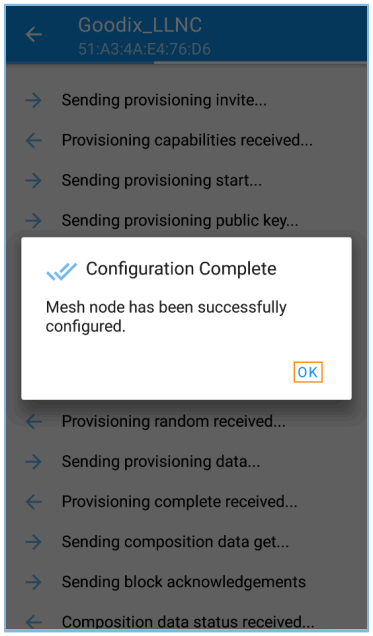

图 3-9 开通配置完成

3. 在“GRMesh Provisioner”界面，点击“Goodix_LLNC > Element:0x0002 > Light Lightness Client Model”，配置Goodix_LLNC节点中第一个元素的Light Lightness Client Model。

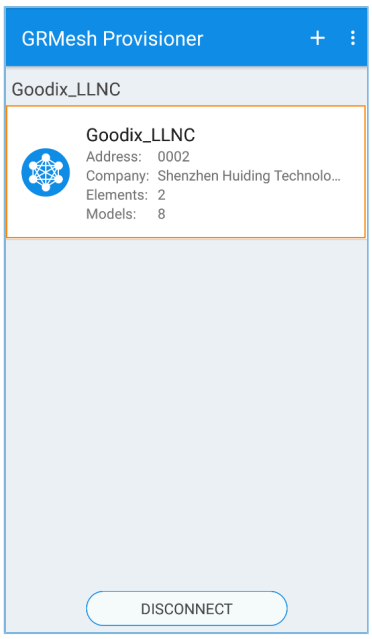

图 3-10 配置节点

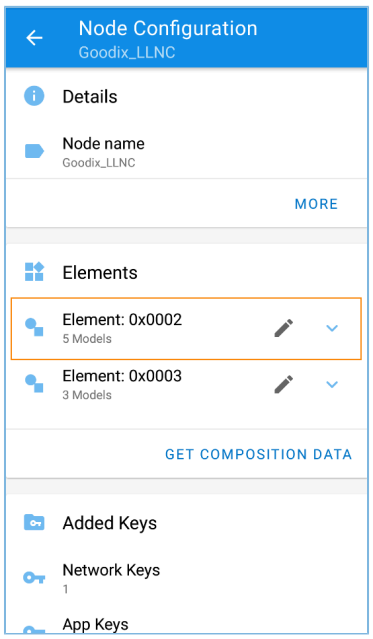

图 3-11 点击“Vendor Model”

在“Light Lightness Client”界面，设置“Bound App Keys”为“index 0”，“Publish Address”的单播地址为“0x0004”（对应Goodix_LLNS中第一个元素的地址）。

版权所有 © 2023 深圳市汇顶科技股份有限公司

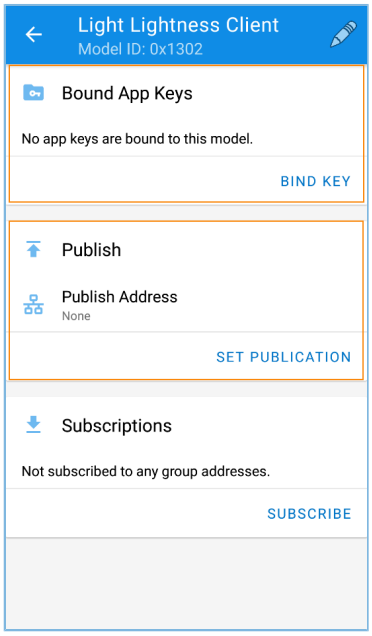

图 3-12 Vendor Model界面

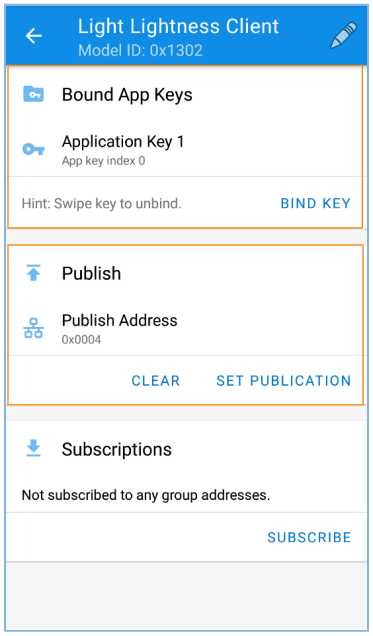

图 3-13 配置App Keys和地址

4. 参考步骤2 ～ 3， 配置Goodix_LLNC设备中第一个元素的Generic On Off Client model和Generic Level Client model。

5. 参考步骤2 ～ 3，开通配置Goodix_LLNS设备，并配置第一个元素的Light Lightness Server Model的“Publish Address”（单播地址）为“0x0002”（对应Goodix_LLNC中第一个元素的地址）。

6. 设置Goodix_LLNS设备的第一个元素的Generic On Off Server Model、Generic Level Server Model和LightLightness Setup Server Model的“Bound App Keys”为“index 0”。

7. 开通配置Client设备与Server设备后，可在“GRMesh Provisioner”界面查看到设备信息，如下图所示。

版权所有 © 2023 深圳市汇顶科技股份有限公司

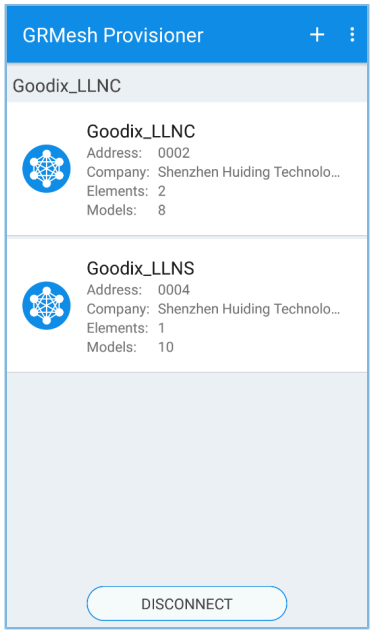

图 3-14 配置完成

3.6.2 GRUart验证LLN Models

使用GRUart测试LLN Client设备和LLN Server设备的通信交互，具体步骤如下：

1. 打开两个GRUart窗口：一个作为LLN Client设备串口，一个作为LLN Server设备串口。
2. 在Client设备串口输入“00010490015400”。
   (1) Client设备发送发送“LIGHT_LIGHTNESS_OPCODE_SET”消息（携带`Lightness Actual`命令），Lightness Actual (400, 0x0190)、transition time (2000 ms, 0x0054)和delay time（默认值0 ms）。
   (2) Server设备接收到消息后，设置transition time和delay time，并立即回复“LIGHT_LIGHTNESS_OPCODE_STATUS”至Client端设备。

版权所有 © 2023 深圳市汇顶科技股份有限公司

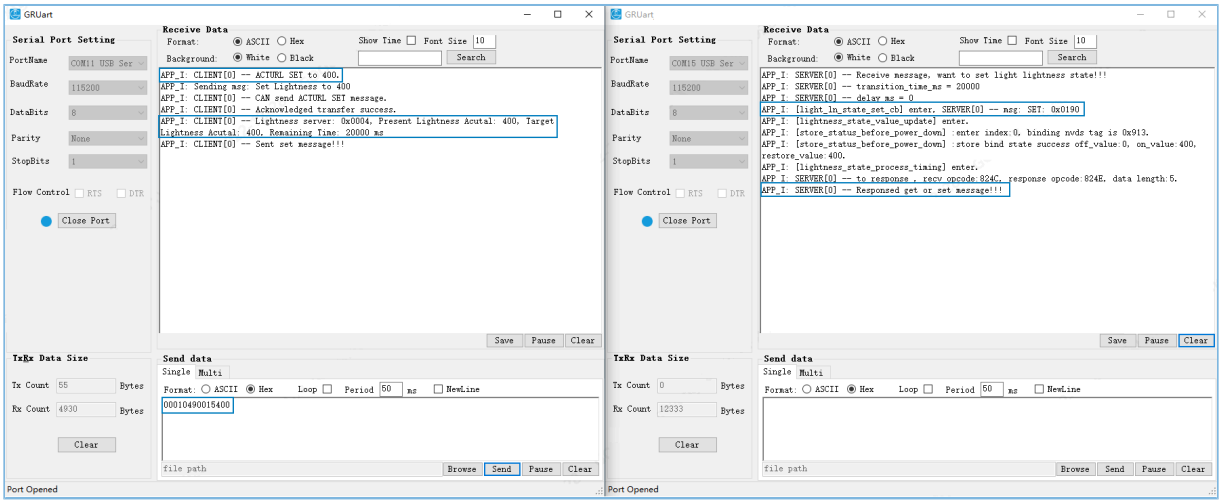

图 3-15 Client设备（左）与Server设备（右）的消息交互-1

当transition time超时（大于2000 ms），Server设备将“Lightness Actual State”设定为对应值，并发布该状态至Client端，如下图所示。

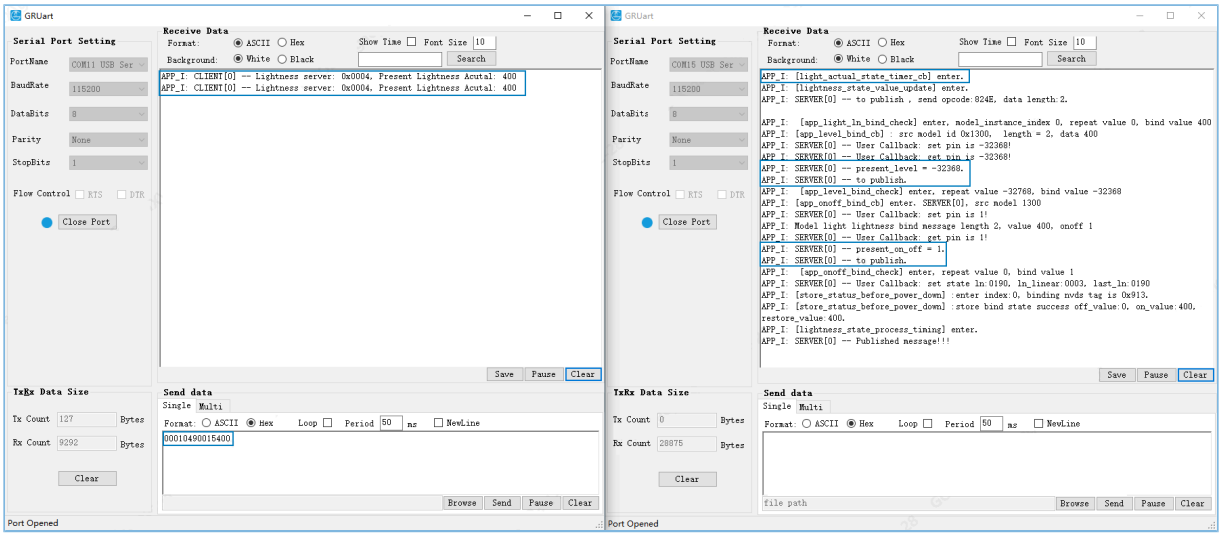

图 3-16 Client设备（左）与Server设备（右）的消息交互-2

3. 在Client设备串口输入“000B04640000F0”。

   (1) Client设备发送“LIGHT_LIGHTNESS_RANGE_OPCODE_SET”消息（携带Lightness range命令），Range Min (100, 0x0064)，Range Max (61440, 0xF000)。

   (2) Server设备接收到消息后，设置其lightness range状态为对应值，并通过Server Model进行回复。

版权所有 © 2023 深圳市汇顶科技股份有限公司

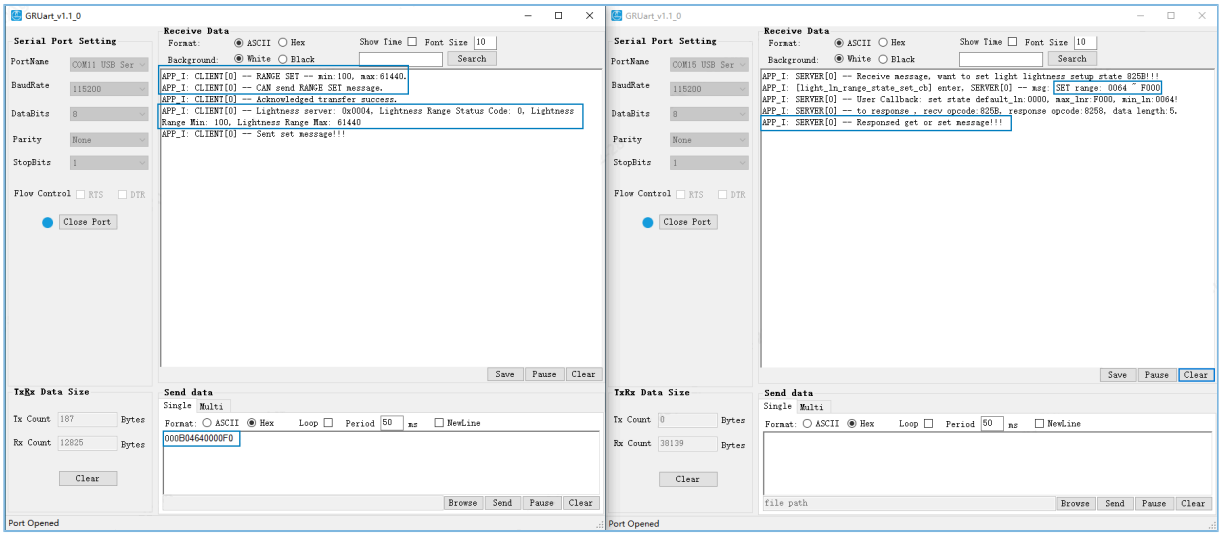

图 3-17 Client设备（左）与Server设备（右）的消息交互-3

版权所有 © 2023 深圳市汇顶科技股份有限公司

4 应用详解

本章主要介绍Light_lightness_Model示例的工程目录以及关键功能的实现代码。

4.1 工程目录

GR533x SDK提供了LLNClient Models和LLN Server Models示例的源代码以及工程文件，可方便用户基于该示例工程，实现自定义的LLN Models。

4.1.1 LLN Client Model工程

LLN Client Model示例的源代码和工程文件位于：`SDK_Folder\projects\mesh\SIG\mesh_app_light_lightness_client`，其中工程文件位于keil_5文件夹。

在Keil中打开*mesh_app_light_lightness_client.uvprojx*，可查看到LLN Client Model示例的工程目录结构，相关文件说明如下表所示。

表 4-1 mesh_app_light_lightness_client文件说明

Group	文件	描述
gr_models	light_lightness_client.c	LLN Client Model实现
	Generic_level_client.c	Generic Level Client Model实现
	Generic_onoff_client.c	Generic OnOff Client Model实现
	Generic_power_onoff_client.c	Generic Power OnOff Client Model实现
user_callback	user_gap_callback.c	获取连接和断连事件
	user_mesh_callback.c	获取开通配置事件
user_platform	user_periph_setup.c	App Log、串口和设备地址的配置
user_app	main.c	main()入口函数
	user_app.c	Client Models注册及用户逻辑处理
	App_light_lightness_client.c	LLN Client Model应用初始化以及注册
	App_level_client.c	Generic Level Client Model应用初始化以及注册
	App_onoff_client.c	Generic OnOff Client Model应用初始化以及注册

4.1.2 LLN Server Model工程

LLN Server Model示例的源代码和工程文件位于：`SDK_Folder\projects\mesh\SIG\mesh_app_light_lightness_server`，其中工程文件位于keil_5文件夹。

在Keil中打开工程文件*mesh_app_light_lightness_server.uvprojx*，可查看到LLN Server Model示例的工程目录结构，相关文件说明如下表所示。

表 4-2 mesh_app_light_lightness_server文件说明

Group	文件	描述
gr_models	light_lightness_server.c	LLN Server Model实现

版权所有 © 2023 深圳市汇顶科技股份有限公司

Group	文件	描述
	light_lightness_setup_server.c	LLNS Server Model实现
	generic_level_server.c	Generic Level Server Model实现
	generic_onoff_server.c	Generic OnOff Server Model实现
	generic_power_onoff_server.c	Generic Power OnOff Server Model实现
	generic_power_onoff_setup_server.c	Generic Power OnOff Setup Server Model实现
	generic_power_onoff_behavior.c	Generic Power OnOff方法实现
	generic_default_transition_time_server.c	Generic Default Transition Time Server Model实现
	scene_server.c	Scene Server Model实现
	scene_setup_server.c	Scene Setup Server Model实现
user_callback	user_gap_callback.c	获取连接和断连事件
	user_mesh_callback.c	获取开通配置事件
user_platform	user_periph_setup.c	App Log、串口和设备地址的配置
user_app	main.c	main()入口函数
	user_app.c	Server Models注册及用户逻辑处理
	app_light_lightness_server.c	LLN Server Model应用初始化、注册及逻辑处理
	app_light_lightness_setup_server.c	LLNS Server Model应用初始化、注册及逻辑处理
	app_level_server.c	Generic Level Server Model应用初始化、注册及逻辑处理
	app_onoff_server.c	Generic OnOff Server Model应用初始化、注册及逻辑处理
	app_power_onoff_server.c	Generic Power OnOff Server Model应用初始化、注册及逻辑处理
	app_power_onoff_setup_server.c	Generic Power OnOff Setup Server Model应用初始化、注册及逻辑处理
	app_scene_server.c	Scene Server Model应用初始化、注册及逻辑处理
	app_scene_setup_server.c	Scene Setup Server Model应用初始化、注册及逻辑处理
	custom_config.h	Application工程通用配置
	mesh_stack_config.h	Mesh协议相关配置

4.2 代码介绍

4.2.1 LLN Models消息处理

本节主要介绍LLN Models消息处理的实现代码，以展示如何实现标准Models消息处理。

版权所有 © 2023 深圳市汇顶科技股份有限公司

4.2.1.1 LLN Client Model

本节示例代码的路径为：SDK_Folder\components\mesh\models\SIG\Lighting\light_lightness\src\light_lightness_client.c

4.2.1.1.1 可处理消息表

可处理消息表light_ln_client_opcode_list包含LLN Client Model可接收并处理的Mesh消息。

```
static const uint16_t light_ln_client_opcode_list[] =
{
    LIGHT_LIGHTNESS_OPCODE_STATUS,
    LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS,
    LIGHT_LIGHTNESS_LAST_OPCODE_STATUS,
    LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS,
    LIGHT_LIGHTNESS_RANGE_OPCODE_STATUS,
};
```

4.2.1.1.2 回调函数处理表

回调函数处理表light_ln_client_msg_cb包括3个成员：

- cb_rx：LLN Client Model接收的消息回调。
- cb_sent：LLN Client Model发布/回应消息是否完成的回调。
- cb_publish_period：LLN Client Model的周期发布状态发生变化时的回调。

```
static const mesh_model_cb_t light_ln_client_msg_cb =
{
    .cb_rx              = light_ln_client_rx_cb,
    .cb_sent            = light_ln_client_sent_cb,
    .cb_publish_period  = NULL,
};
```

light_ln_client_rx_cb()函数对接收的Mesh消息进行处理，其实现代码如下所示：

```
static void light_ln_client_rx_cb(mesh_model_msg_ind_t *p_model_msg, void *p_args)
{
    uint16_t company_opcode = p_model_msg->opcode.company_opcode;
    uint8_t index = 0;
    switch(company_opcode)
    {
        case LIGHT_LIGHTNESS_OPCODE_STATUS:
            index = 0;
            break;
        case LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS:
            index = 1;
            break;
        case LIGHT_LIGHTNESS_LAST_OPCODE_STATUS:
            index = 2;
            break;
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
        case LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS:
            index = 3;
            break;
        case LIGHT_LIGHTNESS_RANGE_OPCODE_STATUS:
            index = 4;
            break;
    }

static const mesh_opcode_handler_t m_opcode_handlers[] =
{
    {LIGHT_LIGHTNESS_OPCODE_STATUS, actual_linear_status_handle},
    {LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS, actual_linear_status_handle},
    {LIGHT_LIGHTNESS_LAST_OPCODE_STATUS, last_status_handle},
    {LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS, dft_status_handle},
    {LIGHT_LIGHTNESS_RANGE_OPCODE_STATUS, range_status_handle},
};
static void actual_linear_status_handle(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    light_ln_client_t * p_client = (light_ln_client_t *) p_args;
    light_ln_status_params_t in_data = {0};

        if  (p_rx_msg->msg_len  ==  LIGHT_LIGHTNESS_STATUS_MINLEN  ||  p_rx_msg->msg_len  ==
 LIGHT_LIGHTNESS_STATUS_MAXLEN)
    {
          light_ln_status_msg_pkt_t  *  p_msg_params_packed  =  (light_ln_status_msg_pkt_t  *)
 p_rx_msg->msg;

        if (p_rx_msg->msg_len == LIGHT_LIGHTNESS_STATUS_MINLEN)
        {
            in_data.present_ln = p_msg_params_packed->present_ln;
            in_data.target_ln = p_msg_params_packed->present_ln;
            in_data.remaining_time_ms = 0;
        }
        else
        {
            in_data.present_ln = p_msg_params_packed->present_ln;
            in_data.target_ln = p_msg_params_packed->target_ln;
              in_data.remaining_time_ms = model_transition_time_decode(p_msg_params_packed-
>remaining_time);
        }
        if(p_rx_msg->opcode.company_opcode == LIGHT_LIGHTNESS_OPCODE_STATUS)
        {
          p_client->settings.p_callbacks->ln_actual_status_cb(p_client, p_rx_msg, &in_data);
        }
        else
        {
          p_client->settings.p_callbacks->ln_linear_status_cb(p_client, p_rx_msg, &in_data);
        }
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
    }
}
static void last_status_handle(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    light_ln_client_t * p_client = (light_ln_client_t *) p_args;
    light_ln_last_status_params_t in_data = {0};

    if (p_rx_msg->msg_len == LIGHT_LIGHTNESS_LAST_STATUS_LEN)
    {
      light_ln_last_status_msg_pkt_t * p_msg_params_packed = (light_ln_last_status_msg_pkt_t
 *) p_rx_msg->msg;
        in_data.ln = p_msg_params_packed->ln;
        p_client->settings.p_callbacks->ln_last_status_cb(p_client, p_rx_msg, &in_data);
    }
}

static void dft_status_handle(const mesh_model_msg_ind_t *p_rx_msg, void *p_args)
{
    ...
    if (p_rx_msg->msg_len == LIGHT_LIGHTNESS_DFT_STATUS_LEN)
    {
        light_ln_dft_status_msg_pkt_t * p_msg_params_packed =
                                        (light_ln_dft_status_msg_pkt_t *) p_rx_msg->msg;
        in_data.ln = p_msg_params_packed->ln;
        p_client->settings.p_callbacks->ln_dft_status_cb(p_client,
                                                         p_rx_msg,
                                                         &in_data);
    }
}
static void range_status_handle(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    light_ln_client_t * p_client = (light_ln_client_t *) p_args;
    light_ln_range_status_params_t in_data = {0};

    if (p_rx_msg->msg_len == LIGHT_LIGHTNESS_RANGE_STATUS_LEN)
    {
                            light_ln_range_status_msg_pkt_t    *    p_msg_params_packed    =
 (light_ln_range_status_msg_pkt_t *) p_rx_msg->msg;
        in_data.max_ln = p_msg_params_packed->max_ln;
        in_data.min_ln = p_msg_params_packed->min_ln;
        in_data.status_code = p_msg_params_packed->status_code;
        p_client->settings.p_callbacks->ln_range_status_cb(p_client, p_rx_msg, &in_data);
    }
}
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

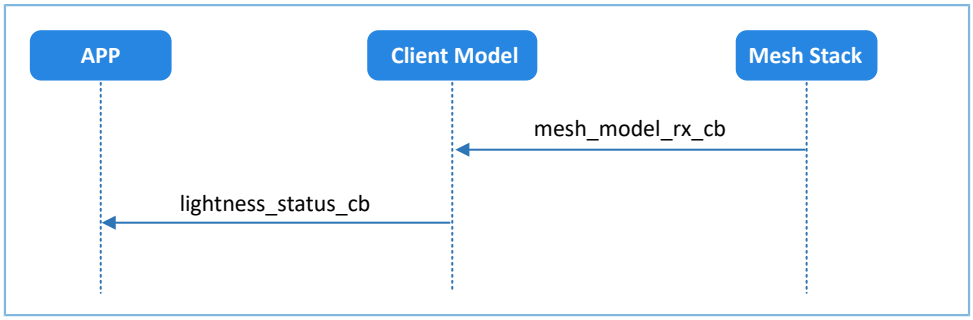

图 4-1 LIGHT_LIGHTNESS_OPCODE_STATUS消息处理

4.2.1.1.3 发送消息处理

LLN Client Model的发送消息可分为三类：GET、SET和SET_UNACK消息。

- LIGHT_LIGHTNESS_OPCODE_SET消息

  发送该消息时，mesh_model_publish()函数的第2个参数需携带reliable_info信息。reliable_info包括所期待回复消息的opcode以及等待时间。

```
uint16_t light_ln_client_set(light_ln_client_t * p_client,
                             const light_ln_set_params_t * p_params,
                             const model_transition_t * p_transition,
                             const cmd_type_t cmd_type)
{
    light_ln_set_msg_pkt_t set;
    bool reliable_trans_state = false;
    uint8_t tx_hdl = LIGHT_LIGHTNESS_CLIENT_SET_SEND_TX_HDL + p_client->model_instance_index
 * LIGHT_LIGHTNESS_CLIENT_TX_HDL_TOTAL;
    mesh_model_send_info_t model_msg_send;
    mesh_model_reliable_info_t reliable_info =
    {
        .reply_opcode.company_opcode = (cmd_type == ACTURL) ?
         LIGHT_LIGHTNESS_OPCODE_STATUS : LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS,
        .reply_opcode.company_id= MESH_ACCESS_COMPANY_ID_NONE,
        .status_cb = p_client->settings.p_callbacks->ack_transaction_status_cb,
        .timeout_ms = p_client->settings.timeout_ms,
    };
    ......
    if (MESH_ERROR_NO_ERROR == mesh_model_reliable_trans_is_on(
                                                          p_client->model_lid,
                                                          &reliable_trans_state))
    {
        if (reliable_trans_state)
        {
            return MESH_ERROR_SDK_RELIABLE_TRANS_ON;
        }
        else
        {
```

```
            uint8_t msg_length = message_set_packet_create(&set,
                                                           p_params,
                                                           p_transition);
            message_create(&model_msg_send, p_client->model_lid,
                           (cmd_type == ACTURL) ? LIGHT_LIGHTNESS_OPCODE_SET :
                            LIGHT_LIGHTNESS_LINEAR_OPCODE_SET,
                            tx_hdl, (uint8_t *)&set, msg_length);

            return mesh_model_publish(&model_msg_send, &reliable_info);
        }
    }
    else
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }
}
```

完成SET消息发送后，LLN Client Model会通过回调函数通知用户应用操作状态，如下图所示。

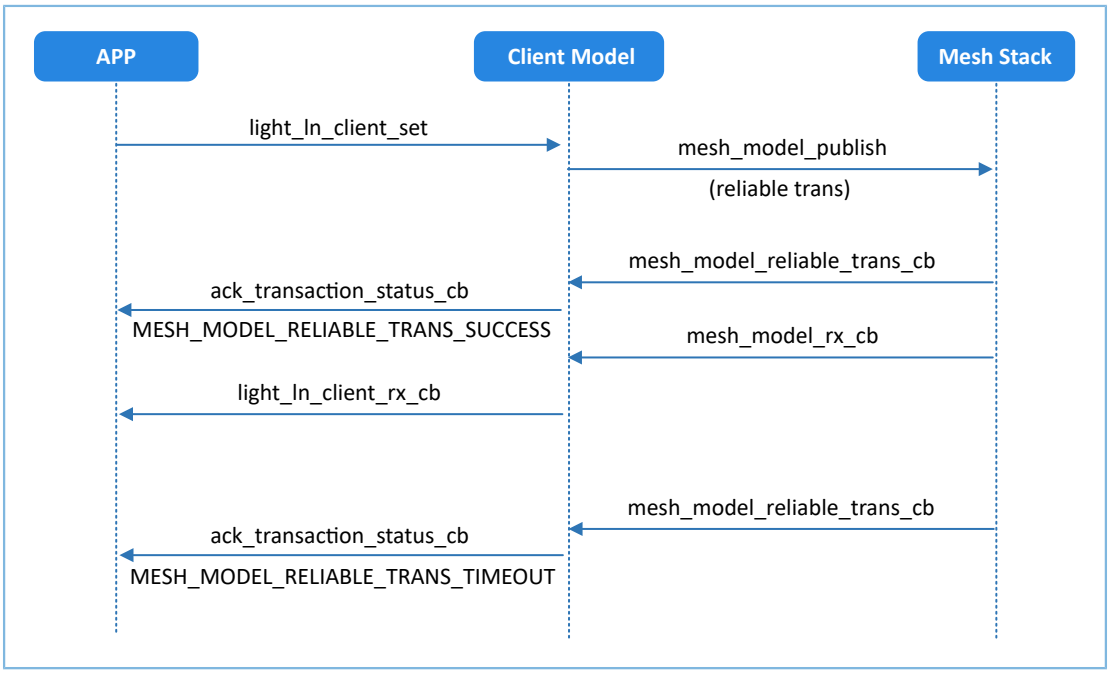

图 4-2 LIGHT_LIGHTNESS_OPCODE_SET消息发送

- LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED消息

  发送该消息时，mesh_model_publish函数的第2个参数需设置为NULL。当无需Server回复应答消息时，Client可发送该消息。

```
uint16_t light_ln_client_set_unack(light_ln_client_t * p_client,
                                   const light_ln_set_params_t * p_params,
                                   const model_transition_t * p_transition,
                                   const cmd_type_t cmd_type)
{
    light_ln_set_msg_pkt_t set_un;
    uint8_t tx_hdl = LIGHT_LIGHTNESS_CLIENT_SET_UNRELIABLE_SEND_TX_HDL +
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
            p_client->model_instance_index * LIGHT_LIGHTNESS_CLIENT_TX_HDL_TOTAL;
    mesh_model_send_info_t model_msg_send;
    ...

    uint8_t msg_length = message_set_packet_create(&set_un,
                                                   p_params, p_transition);

    message_create(&model_msg_send, p_client->model_lid,
             (cmd_type == ACTURL) ? LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED :
                                    LIGHT_LIGHTNESS_LINEAR_OPCODE_SET_UNACKNOWLEDGED,
                                        tx_hdl, (uint8_t *)&set_un, msg_length);

    return mesh_model_publish(&model_msg_send, NULL);
}
```

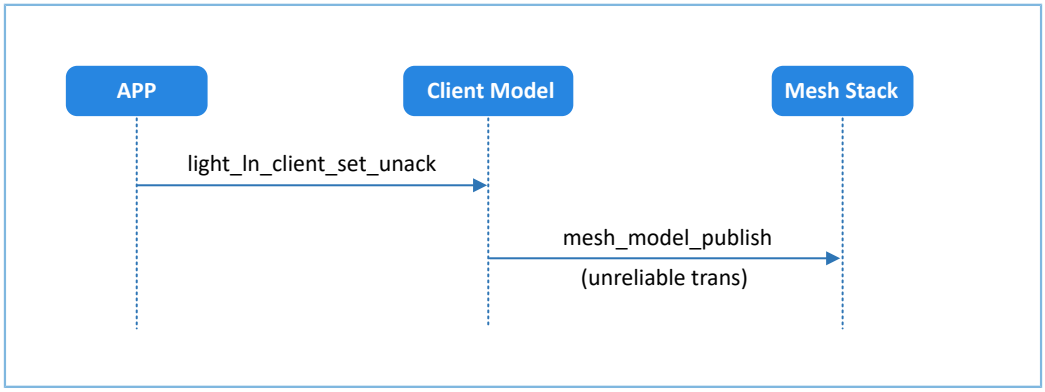

图 4-3 LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED消息发送

- LIGHT_LIGHTNESS_OPCODE_GET消息

  发送该消息时，mesh_model_publish()函数的第2个参数需携带reliable_info信息。reliable_info包括所期待回复消息的opcode以及等待时间。

```
uint16_t light_ln_client_get(light_ln_client_t * p_client,
                             const cmd_type_t cmd_type)
{
    mesh_model_send_info_t model_msg_send;
    bool reliable_trans_state = false;
    uint8_t tx_hdl = LIGHT_LIGHTNESS_CLIENT_GET_SEND_TX_HDL +
              p_client->model_instance_index * LIGHT_LIGHTNESS_CLIENT_TX_HDL_TOTAL;
    mesh_model_reliable_info_t reliable_info =
    {
        .reply_opcode.company_opcode =
                            (cmd_type == ACTURL) ? LIGHT_LIGHTNESS_OPCODE_STATUS :
                                                   LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS,
        .reply_opcode.company_id= MESH_ACCESS_COMPANY_ID_NONE,
        .status_cb = p_client->settings.p_callbacks->ack_transaction_status_cb,
        .timeout_ms = p_client->settings.timeout_ms,
    };

    ...
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
    if (MESH_ERROR_NO_ERROR ==
                            mesh_model_reliable_trans_is_on(p_client->model_lid,
                                                            &reliable_trans_state))
    {
        if (reliable_trans_state)
        {
            return MESH_ERROR_SDK_RELIABLE_TRANS_ON;
        }
        else
        {
            message_create(&model_msg_send, p_client->model_lid,
                           (cmd_type == ACTURL) ? LIGHT_LIGHTNESS_OPCODE_GET :
                                                  LIGHT_LIGHTNESS_LINEAR_OPCODE_GET,
                                                  tx_hdl, NULL, 0);

            return mesh_model_publish(&model_msg_send, &reliable_info);
        }
    }
    else
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }
}
```

完成GET消息发送后，LLN Client Model会通过回调函数通知用户应用操作状态，如下图所示。

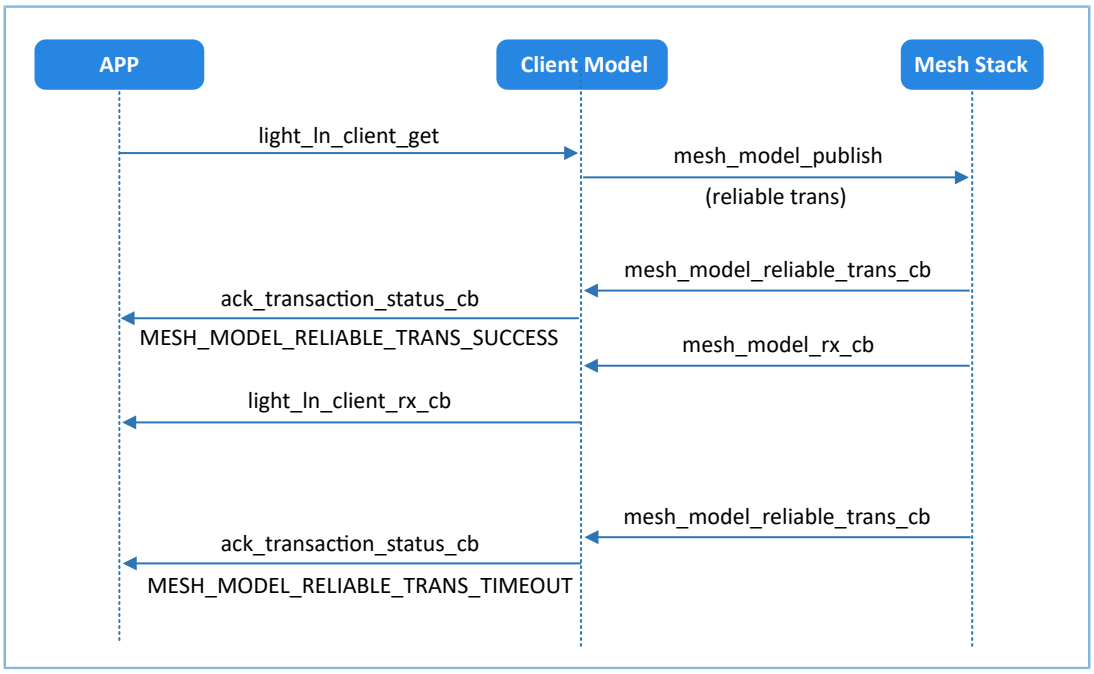

图 4-4 LIGHT_LIGHTNESS_OPCODE_GET消息发送

版权所有 © 2023 深圳市汇顶科技股份有限公司

4.2.1.2 LLN Server Model

本节示例代码的路径为：SDK_Folder\components\mesh\models\SIG\Lighting\light_lightness\src\light_lightness_server.c。

4.2.1.2.1 可处理消息表

可处理消息表light_lightness_server_opcode_list包含LLN Server Model可接收并处理的Mesh消息。

```
static const uint16_t light_lightness_server_opcode_list[] =
{
    LIGHT_LIGHTNESS_OPCODE_GET,
    LIGHT_LIGHTNESS_OPCODE_SET ,
    LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED,
    LIGHT_LIGHTNESS_LINEAR_OPCODE_GET,
    LIGHT_LIGHTNESS_LINEAR_OPCODE_SET,
    LIGHT_LIGHTNESS_LINEAR_OPCODE_SET_UNACKNOWLEDGED,
    LIGHT_LIGHTNESS_LAST_OPCODE_GET,
    LIGHT_LIGHTNESS_DEFAULT_OPCODE_GET,
    LIGHT_LIGHTNESS_RANGE_OPCODE_GET,
};
```

4.2.1.2.2 回调函数处理表

回调函数处理表light_lightness_server_msg_cb包括3个成员：

- cb_rx：LLN Server Model接收的消息回调。
- cb_sent：LLN Server Model发布/回应消息是否完成的回调。
- cb_publish_period：LLN Server Model的周期发布状态发生变化时的回调。

```
static const mesh_model_cb_t light_lightness_server_msg_cb =
{
    .cb_rx              = light_lightness_server_rx_cb,
    .cb_sent            = light_lightness_server_sent_cb ,
    .cb_publish_period  = NULL ,
};
```

light_lightness_server_rx_cb()函数对接收的Mesh消息进行处理，其实现代码如下所示。

```
static void light_lightness_server_rx_cb(mesh_model_msg_ind_t *p_model_msg,
                                         void *p_args)
{
    uint16_t company_opcode = p_model_msg->opcode.company_opcode;

    mesh_opcode_handler_cb_t handler = NULL;

    for(uint8_t i = 0; i<light_lightness_server_register_info.num_opcodes; i++)
    {
        if(company_opcode == m_opcode_handlers[i].opcode)
        {
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
            handler = m_opcode_handlers[i].handler;
            break;
        }
    }

    if (NULL != handler)
    {
        handler(p_model_msg, p_args);
    }
}

static const mesh_opcode_handler_t m_opcode_handlers[] =
{
    {LIGHT_LIGHTNESS_OPCODE_GET, handle_get_cb},
    {LIGHT_LIGHTNESS_OPCODE_SET, handle_set_cb},
    {LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED, handle_set_cb},
    {LIGHT_LIGHTNESS_LINEAR_OPCODE_GET, handle_get_cb},
    {LIGHT_LIGHTNESS_LINEAR_OPCODE_SET, handle_set_cb},
    {LIGHT_LIGHTNESS_LINEAR_OPCODE_SET_UNACKNOWLEDGED, handle_set_cb},
    {LIGHT_LIGHTNESS_LAST_OPCODE_GET, handle_get_cb},
    {LIGHT_LIGHTNESS_DEFAULT_OPCODE_GET, handle_get_cb},
    {LIGHT_LIGHTNESS_RANGE_OPCODE_GET, handle_get_cb},
};
```

- handle_set_cb()函数可处理以下消息：
  - LIGHT_LIGHTNESS_OPCODE_SET
  - LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED
  - LIGHT_LIGHTNESS_LINEAR_OPCODE_SET
  - LIGHT_LIGHTNESS_LINEAR_OPCODE_SET_UNACKNOWLEDGED

  下面以LIGHT_LIGHTNESS_OPCODE_SET消息处理为例，介绍handle_set_cb()函数的代码实现。

```
static void handle_set_cb(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    light_ln_server_t  * p_server = (light_ln_server_t  *) p_args;
     APP_LOG_INFO("SERVER[%d] -- Receive message, want to set light lightness state!!!",
 p_server->model_instance_index);

    if (set_params_validate(p_rx_msg))
    {
        light_ln_state_set_cb_t set_cb_local = NULL;
        light_ln_set_params_t in_data = {0};
        model_transition_t in_data_tr = {0};
        light_ln_status_params_u out_data = {0};
        light_ln_set_msg_pkt_t * p_msg_params_packed = (light_ln_set_msg_pkt_t
                                                        *) p_rx_msg->msg;
        bool ack_flag = false;
```

```
        in_data.ln = gx_read16p ((void const *)&p_msg_params_packed->ln);
        in_data.tid = p_msg_params_packed->tid;

        if (model_tid_validate(&p_server->tid_tracker, p_rx_msg,
            p_rx_msg->opcode.company_opcode, in_data.tid))
        {
            ......

            switch(p_rx_msg->opcode.company_opcode)
            {
                case LIGHT_LIGHTNESS_OPCODE_SET:
                    ack_flag = true;
                    ......

                    set_cb_local = p_server->settings.p_callbacks->light_ln_cbs.set_cb;
                    break;
                ......

                default:
                    break;
            }

            if (NULL != set_cb_local)
            {
                set_cb_local(p_server,
                             p_rx_msg,
                             &in_data,
                             ((p_rx_msg->msg_len == LIGHT_LIGHTNESS_SET_MINLEN)
                             &&(p_server->p_dtt_ms == NULL)) ? NULL : &in_data_tr,
                             (ack_flag) ? &out_data : NULL);
            }

            if (ack_flag)
            {
                (void) status_send(p_server, p_rx_msg, &out_data);
            }
        }
    }
}

static uint32_t status_send(light_ln_server_t * p_server,
                            const mesh_model_msg_ind_t *p_rx_msg,
                            const light_ln_status_params_u * p_params)
{
    light_ln_status_msg_pkt_t *msg_pkt = NULL;
    light_ln_last_status_msg_pkt_t *msg_pkt_l = NULL;
    light_ln_dft_status_msg_pkt_t *msg_pkt_d = NULL;
    light_ln_range_status_msg_pkt_t *msg_pkt_r = NULL;
    uint8_t *p_send_data = NULL;
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
    uint16_t send_op = LIGHT_LIGHTNESS_OPCODE_STATUS;
    uint16_t data_send_len = 0;
    mesh_error_t status = MESH_ERROR_SDK_INVALID_PARAM;
      uint8_t  tx_hdl  =  (NULL  ==  p_rx_msg)  ?  LIGHT_LIGHTNESS_SERVER_PUBLISH_SEND_TX_HDL  +
 p_server->model_instance_index * LIGHT_LIGHTNESS_SERVER_TX_HDL_TOTAL
                                        : LIGHT_LIGHTNESS_SERVER_RSP_SEND_TX_HDL + p_server-
>model_instance_index * LIGHT_LIGHTNESS_SERVER_TX_HDL_TOTAL;


    if (NULL != p_rx_msg)
    {
        switch(p_rx_msg->opcode.company_opcode)
        {
        ......

            case LIGHT_LIGHTNESS_OPCODE_SET:
                msg_pkt = (light_ln_status_msg_pkt_t *)sys_malloc(
                                                    sizeof(light_ln_status_msg_pkt_t));
                p_send_data = (uint8_t *)msg_pkt;
                ......

                gx_write16p( (void const *)&(msg_pkt->present_ln),
                             p_params->ln.present_ln);
                data_send_len += 2;
                ......
            break;
            ......

            default:
            break;
        }
}
    ......

    mesh_model_send_info_t msg_send =
    {
        .model_lid = p_server->model_lid,
        .opcode = MESH_ACCESS_OPCODE_SIG(send_op),
        .tx_hdl = tx_hdl,
        .p_data_send = (uint8_t *) p_send_data,
        .data_send_len = data_send_len,
        .dst = (NULL == p_rx_msg) ? MESH_INVALID_ADDR : p_rx_msg->src,
        .appkey_index =
                (NULL == p_rx_msg) ? MESH_INVALID_KEY_INDEX : p_rx_msg->appkey_index,
    };

    if (NULL == p_rx_msg)
    {
        APP_LOG_INFO("SERVER[%d] -- to publish : send opcode = %04X,
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
                    data_send_len = %d.",
                    p_server->model_instance_index, send_op, data_send_len);
        status = mesh_model_publish(&msg_send, NULL);
    }
    else
    {
        APP_LOG_INFO("SERVER[%d] -- recv opcode = %04X, response opcode = %04X,
                    data_send_len = %d.",
                    p_server->model_instance_index,
                    p_rx_msg->opcode.company_opcode, send_op, data_send_len);
        status = mesh_model_rsp_send(&msg_send);
    }

    if(p_send_data)
    {
        sys_free(p_send_data);
        p_send_data = NULL;
    }

    return status;
}
```

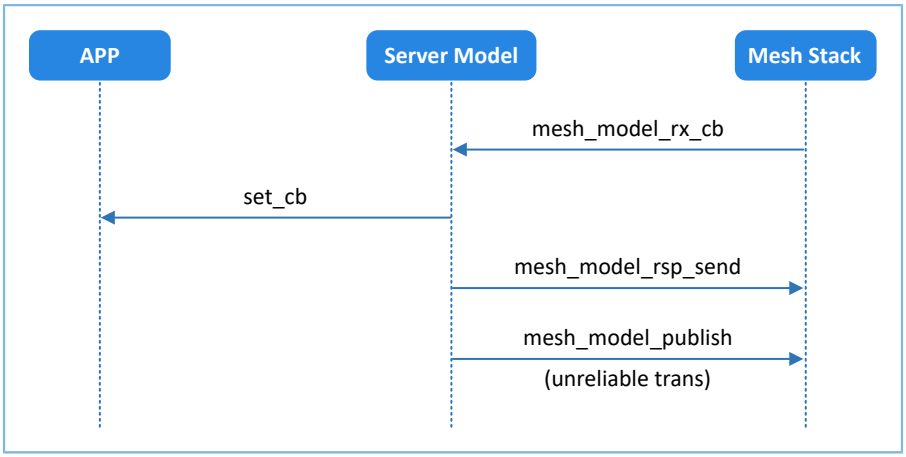

图 4-5 LIGHT_LIGHTNESS_OPCODE_SET消息处理

*_SET和*_SET_UNACKNOWLEDGE消息处理有所不同，前者比后者需多调用一次status_send()函数，进而调用mesh_model_rsp_send进行消息回复。两者均会在最后调用一次status_send()函数进而调用mesh_model_publish进行消息发布。

版权所有 © 2023 深圳市汇顶科技股份有限公司

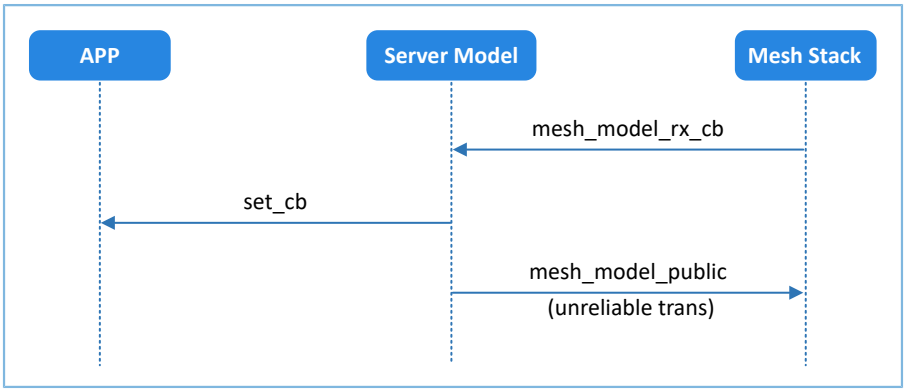

图 4-6 LIGHT_LIGHTNESS_OPCODE_SET_UNACKNOWLEDGED消息处理

- handle_get_cb()函数可处理以下消息：
  - LIGHT_LIGHTNESS_OPCODE_GET
  - LIGHT_LIGHTNESS_LINEAR_OPCODE_GET
  - LIGHT_LIGHTNESS_LAST_OPCODE_GET
  - LIGHT_LIGHTNESS_DEFAULT_OPCODE_GET
  - LIGHT_LIGHTNESS_RANGE_OPCODE_GET

  status_send()函数调用mesh_model_rsp_send进行消息回复。

```
static void handle_get_cb(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    light_ln_server_t * p_server = (light_ln_server_t *) p_args;
    light_ln_status_params_u out_data = {0};
    light_ln_state_get_cb_t get_cb_local = NULL;

    APP_LOG_INFO("[%s] enter.", __func__);
    APP_LOG_INFO("SERVER[%d] -- Receive message,
                  want to get state opcode %04X!!!",
                  p_server->model_instance_index,
                  p_rx_msg->opcode.company_opcode);

    if (get_params_validate(p_rx_msg))
    {
        switch (p_rx_msg->opcode.company_opcode)
        {
            case LIGHT_LIGHTNESS_OPCODE_GET:
                get_cb_local =
                            p_server->settings.p_callbacks->light_ln_cbs.get_cb;
                break;

            case LIGHT_LIGHTNESS_LINEAR_OPCODE_GET:
                get_cb_local =
                            p_server->settings.p_callbacks->light_ln_linear_cbs.get_cb;
                break;
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
            case LIGHT_LIGHTNESS_LAST_OPCODE_GET:
                get_cb_local =
                             p_server->settings.p_callbacks->light_ln_last_cbs.get_cb;
                break;

            case LIGHT_LIGHTNESS_DEFAULT_OPCODE_GET:
                get_cb_local =
                            p_server->settings.p_callbacks->light_ln_dft_cbs.get_cb;
                break;

            case LIGHT_LIGHTNESS_RANGE_OPCODE_GET:
                get_cb_local =
                            p_server->settings.p_callbacks->light_ln_range_cbs.get_cb;
                break;

            default:
                break;
        }

        if(get_cb_local)
        {
            get_cb_local(p_server, p_rx_msg, &out_data);
           (void) status_send(p_server, p_rx_msg, &out_data);
        }
    }
}
```

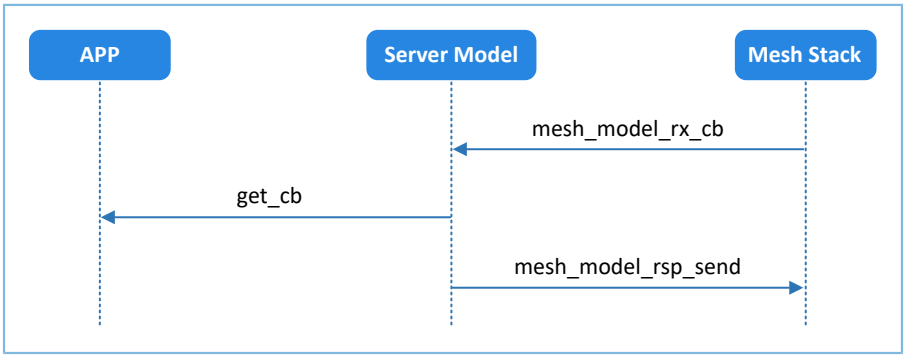

图 4-7 LIGHT_LIGHTNESS_OPCODE_GET消息处理

4.2.1.2.3 发送消息处理

LLN Server Model可主动发送以下消息：

- LIGHT_LIGHTNESS_LINEAR_OPCODE_STATUS
- LIGHT_LIGHTNESS_OPCODE_STATUS
- LIGHT_LIGHTNESS_LAST_OPCODE_STATUS
- LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS
- LIGHT_LIGHTNESS_RANGE_OPCODE_STATUS

版权所有 © 2023 深圳市汇顶科技股份有限公司

status_send()函数调用mesh_model_publish进行消息发布。

```
uint16_t light_ln_server_status_publish(light_ln_server_t * p_server,
                                        const light_ln_status_params_t * p_params)
{
    if (NULL == p_server || NULL == p_params)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    return status_send(p_server, NULL,
                       (const light_ln_status_params_u *) p_params);
}
```

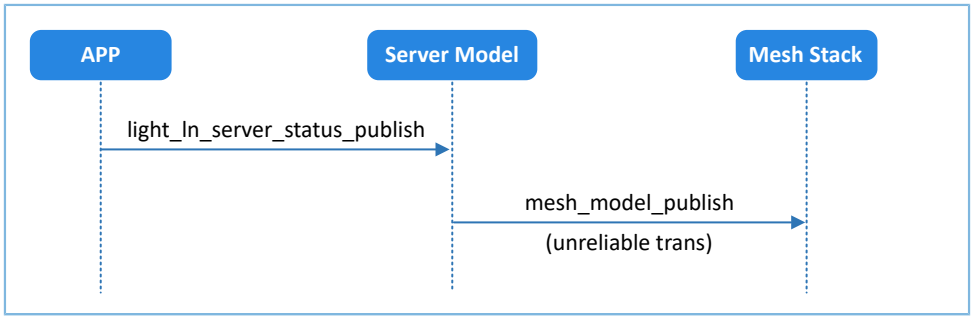

图 4-8 LIGHT_LIGHTNESS_OPCODE_STATUS消息发送

4.2.1.3 LLNS Server Model

本节示例代码的路径为：SDK_Folder\components\mesh\models\SIG\Lighting\light_lightness\src\light_lightness_setup_server.c。

4.2.1.3.1 可处理消息表

可处理消息表light_lightness_setup_server_opcode_list包含LLNS Server Model可接收并处理的Mesh消息。

```
static const uint16_t light_lightness_setup_server_opcode_list[] =
{
    LIGHT_LIGHTNESS_DEFAULT_OPCODE_SET,
    LIGHT_LIGHTNESS_DEFAULT_OPCODE_SET_UNACKNOWLEDGED,
    LIGHT_LIGHTNESS_RANGE_OPCODE_SET,
    LIGHT_LIGHTNESS_RANGE_OPCODE_SET_UNACKNOWLEDGED,
};
```

4.2.1.3.2 回调函数处理表

LLNS Server Model不支持发布消息。当状态变更时，需通过LLN Server Model发布消息。

回调函数处理表light_lightness_setup_server_msg_cb包括3个成员：

- cb_rx：LLNS Server Model接收的来自LLN Client的消息回调。
- cb_sent：LLNS Server Model发布/回应消息是否完成的回调。若未发布消息，则cb_sent赋值为NULL。

版权所有 © 2023 深圳市汇顶科技股份有限公司

- cb_publish_period：LLNS Server Model的周期发布状态发生变化时的回调。若未发布消息，则cb_publish_period赋值为NULL。

```
static const mesh_model_cb_t light_lightness_setup_server_msg_cb =
{
    .cb_rx = light_lightness_setup_server_rx_cb,
    .cb_sent = NULL,
    .cb_publish_period = NULL
};
```

4.2.1.3.3 发送消息处理

当LLNS Server Model接收Range state set或default state set消息并处理后，不会直接应答该消息，而是采用LLN Server Model应答消息机制。

将发送的“model_lid”设定为“LLN Server Model”，实现LLN Server Model应答机制。

```
mesh_model_send_info_t msg_send =
{
  .model_lid = p_server->ln_server->model_lid,
  .opcode = MESH_ACCESS_OPCODE_SIG(send_op),
  .tx_hdl = tx_hdl,
  .p_data_send = (uint8_t *) p_send_data,
  .data_send_len = data_send_len,
  .dst = (NULL == p_rx_msg) ? MESH_INVALID_ADDR : p_rx_msg->src,
  .appkey_index =
          (NULL == p_rx_msg) ? MESH_INVALID_KEY_INDEX : p_rx_msg->appkey_index,
};
```

LLNS Server Model的status_send函数的实现代码如下：

```
static uint32_t status_send(light_ln_setup_server_t * p_server,
                                  const mesh_model_msg_ind_t *p_rx_msg,
                                  const light_ln_status_params_u * p_params)
{
    light_ln_dft_status_msg_pkt_t *msg_pkt_d = NULL;
    light_ln_range_status_msg_pkt_t *msg_pkt_r = NULL;
    uint8_t *p_send_data = NULL;
    uint16_t send_op = LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS;
    uint16_t data_send_len = 0;
    mesh_error_t status = MESH_ERROR_SDK_INVALID_PARAM;
   uint8_t tx_hdl = LIGHT_LIGHTNESS_SERVER_RSP_SEND_TX_HDL + p_server->model_instance_index
 * LIGHT_LIGHTNESS_SERVER_TX_HDL_TOTAL;

    if (NULL == p_rx_msg)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }
    else
    {
        switch(p_rx_msg->opcode.company_opcode)
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
        {
            case LIGHT_LIGHTNESS_DEFAULT_OPCODE_SET:
                send_op = LIGHT_LIGHTNESS_DEFAULT_OPCODE_STATUS;
                                                msg_pkt_d   =   (light_ln_dft_status_msg_pkt_t
*)sys_malloc(sizeof(light_ln_dft_status_msg_pkt_t));
                p_send_data = (uint8_t *)msg_pkt_d;
                if(NULL == p_send_data)
                {
                    return MESH_ERROR_INSUFFICIENT_RESOURCES;
                }

                gx_write16p( (void const *)&(msg_pkt_d->ln), p_params->ln_dft.ln);
                data_send_len += 2;
            break;

            case LIGHT_LIGHTNESS_RANGE_OPCODE_SET:

                if (p_params->ln_range.status_code != STATUS_CODES_SUCCESS)
                {
                    return MESH_ERROR_NO_ERROR;
                }
                send_op = LIGHT_LIGHTNESS_RANGE_OPCODE_STATUS;
                                                msg_pkt_r   =   (light_ln_range_status_msg_pkt_t
*)sys_malloc(sizeof(light_ln_range_status_msg_pkt_t));
                p_send_data = (uint8_t *)msg_pkt_r;
                if(NULL == p_send_data)
                {
                    return MESH_ERROR_INSUFFICIENT_RESOURCES;
                }

                msg_pkt_r->status_code = p_params->ln_range.status_code;
                data_send_len += 1;
               gx_write16p( (void const *)&(msg_pkt_r->min_ln), p_params->ln_range.min_ln);
                data_send_len += 2;
               gx_write16p( (void const *)&(msg_pkt_r->max_ln), p_params->ln_range.max_ln);
                data_send_len += 2;
            break;

            default:
            break;
        }

        if ((0 == data_send_len) || (NULL == p_send_data))
        {
            return MESH_ERROR_INSUFFICIENT_RESOURCES;
        }

        if (p_server->ln_server)//setup server send status
        {
```

```
            mesh_model_send_info_t msg_send =
            {
                .model_lid = p_server->ln_server->model_lid,
                .opcode = MESH_ACCESS_OPCODE_SIG(send_op),
                .tx_hdl = tx_hdl,
                .p_data_send = (uint8_t *) p_send_data,
                .data_send_len = data_send_len,
                .dst = (NULL == p_rx_msg) ? MESH_INVALID_ADDR : p_rx_msg->src,
                   .appkey_index = (NULL == p_rx_msg) ? MESH_INVALID_KEY_INDEX : p_rx_msg-
>appkey_index,
            };

              APP_LOG_INFO("SERVER[%d] -- to response , recv opcode:%04X, response opcode:
%04X, data length:%d.",
                           p_server->model_instance_index, p_rx_msg->opcode.company_opcode,
 send_op, data_send_len);
            status = mesh_model_rsp_send(&msg_send);
        }
        else
        {
            APP_LOG_ERROR("[ERROR] -- No server to send status");
            status = MESH_ERROR_COMMAND_DISALLOWED;
        }

        if(p_send_data)
        {
            sys_free(p_send_data);
            p_send_data = NULL;
        }

        return status;
    }
}
```

4.2.2 LLN Models状态功能处理

本节主要介绍LLN Models状态处理的实现代码，以展示如何实现标准Models状态的处理逻辑。

4.2.2.1 LLN model状态绑定

Lightness state分别与generic level state、generic onoff state存在绑定（binding）关系。若用户自定义Model时，如需绑定状态，可参考以下示例代码，并根据实际需求进行修改。

1. Binding列表初始化

   在SDK_Folder\projects\mesh\SIG\mesh_app_light_lightness_server\Src\user\user_app.c中，注册model之前，需先调用mesh_model_bind_list_init()函数，初始化binding列表，以便后续state注册绑定关系时使用。

```
#define USER_MODEL_REG_NUM_MAX 0x20
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
static void mesh_init_cmp_callback(void)
{
    //set run time
    mesh_run_time_set(1000, 0);
    //init bind model list
    mesh_model_bind_list_init(USER_MODEL_REG_NUM_MAX);
    //register model
    if (MESH_ERROR_NO_ERROR == app_model_server_init())
    {
        APP_LOG_INFO("Register model successful!");
        mesh_enable();
    }
}
```

2. Binding model以及回调函数注册

   在SDK_Folder\projects\mesh\SIG\common\src\app_light_lightness_server.c中，Server Model初始化时，需调用mesh_model_bind_list_add()函数，将model binding信息注册到binding列表中。当其他model state需与lightness state绑定时，将由注册的app_light_ln_bind_cb回调，并传输数据。LLN应用根据传输的信息，进行相应处理。

```
#define MODEL_ID_LIGHTS_LN          (0x1300)
……
uint16_t app_light_ln_server_init(app_light_lightness_server_t *p_server,
                                  uint8_t element_offset,
                                  app_light_ln_set_cb_t set_cb,
                                  app_light_ln_get_cb_t get_cb)
{
    …
mesh_model_bind_list_add(MODEL_ID_LIGHTS_LN,
                         element_offset, (void *)p_server,
                         app_light_ln_bind_cb);
…
}
```

3. Binding信息处理

   在SDK_Folder\projects\mesh\SIG\common\src\app_light_lightness_server.c中，注册的回调函数app_light_ln_bind_cb()，根据传入的“model id”、“state”数据等信息，处理自身的绑定状态变更。以下为generic onoff server调用app_onoff_bind_check()后，lightness回调处理generic onoff state的binding实现代码。

```
#define MODEL_ID_GENS_OO            (0x1000)

static bool app_light_ln_bind_cb(void *p_server, uint32_t src_model_id,
                                 void *p_data, uint16_t data_len)
{
    app_light_lightness_server_t * pl_server =
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
                                    (app_light_lightness_server_t *)p_server;
    bool ret = false;

    APP_LOG_INFO("[%s] enter, SERVER[%d], src model %04X data length %d",
                 __func__, pl_server->server.model_instance_index,
                 src_model_id, data_len);

    if (pl_server != NULL)
    {
        switch(src_model_id)
        {
            …
            case MODEL_ID_GENS_OO:
            {
                if (data_len == 1)
                {
                    uint8_t onoff = *(uint8_t *)p_data;

                    if ((onoff == 0x00) && (pl_server->state.state.present_ln !=
                                                            0))//onoff :off
                    {
                        pl_server->state.state.present_ln = 0x00;
                        pl_server->state.state.present_ln_linear = 0x00;
                        lightness_state_value_update(pl_server, true);

                        ret = true;
                    }
                    else if ((onoff == 0x01) &&
                             (pl_server->state.state.present_ln == 0)) //onoff:on
                    {
                        if(pl_server->state.state.default_ln == 0x0000)
                        {
                            pl_server->state.state.present_ln =
                                                pl_server->state.state.last_ln;
                        }
                        else
                        {
                            pl_server->state.state.present_ln =
                                                pl_server->state.state.default_ln;
                        }
                        pl_server->state.state.present_ln_linear =
                               CV_LIGHTS_LN_LINEAR(pl_server->state.state.present_ln);
                        lightness_state_value_update(pl_server, true);

                        ret = true;
                    }
                    APP_LOG_INFO("model gen onoff bind message length %d,
                                   value %d, lightness actual %d", data_len, onoff,
                                   pl_server->state.state.present_ln);
```

```
                }
                break;
            }
            …
            default:
                break;
        }

        if(ret)
        {
            app_light_ln_status_publish(pl_server);
            app_light_ln_bind_check(pl_server, src_model_id);
        }
    }

    return ret;
}
```

4. Binding状态检查

LLN Server Model在运行过程中，需处理其他Server Model（如Generic OnOff Server Model）的binding消息。并且，当lightness状态变更后，LLN Server Model需将新的lightness状态发送给其他Server Model，以便该Server Model处理自身的binding关系。例如，LLN Server Model通知Generic OnOff Server Model处理Generic OnOff State的binding，可调用mesh_model_bind_check()函数实现该操作。

```
static void lightness_state_value_update(
                        app_light_lightness_server_t * p_server, bool immed)
{
    …
      if (!p_server->value_updated)
      {
          p_server->light_ln_set_cb(p_server->server.model_instance_index,
                                    &(p_server->state.state));
          p_server->value_updated = true;
          app_light_ln_bind_check(p_server,MODEL_ID_LIGHTS_LN);
      }
  …
}

static void app_light_ln_bind_check(app_light_lightness_server_t *p_server,
                                    uint32_t trigger_model)
{
    …
    if (trigger_model != MODEL_ID_GENS_OO)
    {
        /* check bind generic onoff state */
        mesh_model_bind_check(p_server->server.model_instance_index,
                              MODEL_ID_GENS_OO, MODEL_ID_LIGHTS_LN,
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

```
                                &p_server->state.state.present_ln,
                                sizeof(p_server->state.state.present_ln));
    }
...
    return ;
}
```

版权所有 © 2023 深圳市汇顶科技股份有限公司

5 常见问题

本章主要描述运行LLN Models示例时，可能出现的问题、原因及处理方法。

5.1 工程下载失败

- 问题描述

  使用Keil下载工程失败。

- 问题分析

  Keil工程下载配置不正确。

- 处理方法

  确保下载工具配置正确，如下图所示。

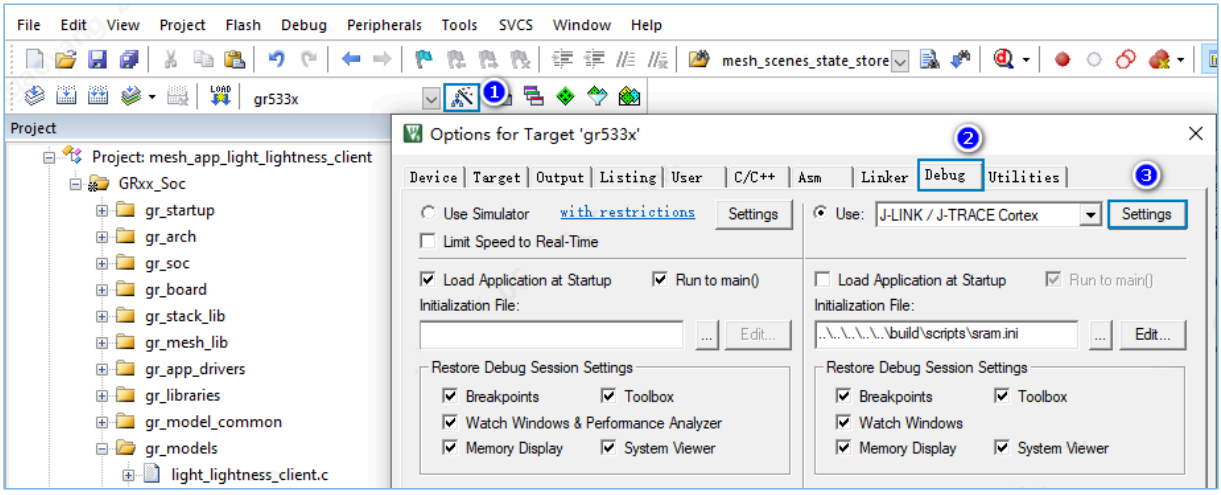

图 5-1 Keil下载配置

5.2 设备初始化失败

- 问题描述

  下载目标文件并运行程序后，串口未打印“Enable successful”，设备初始化失败。

- 问题分析

  配置的工程中，注册model流程失败。

- 处理方法

  - 配置的工程中注册lightness model数量不能大于宏定义LIGHT_LIGHTNESS_INSTANCE_COUNT。
  - 配置的宏定义LIGHT_LIGHTNESS_INSTANCE_COUNT不能大于宏定义LIGHT_LIGHTNESS_SERVER_INSTANCE_COUNT_MAX。
  - 配置的所有注册model数量，不能大于MESH_MODEL_NUM_MAX。

版权所有 © 2023 深圳市汇顶科技股份有限公司

5.3 GRMesh未发现设备

- 问题描述

  准备配网时，使用GRMesh（Android）找不到设备。

- 问题分析

  固件没有正常运行或是未清除之前配网信息。

- 处理方法

  - 确认工程中，设备名是否有改动，设备名设定长度应短于16 Byte。
  - 将GR533x芯片全擦除，再下载。
  - 查看串口输出，是否有“enable successful”。

5.4 Client和Server设备之间不能收发数据

- 问题描述

  配网成功后，不能收发数据。

- 问题分析

  固件未正确下载到开发板运行，或者串口波特率设置不正确导致终端无法正常打印输出日志信息。

- 处理方法

  - 确保配网Client和Server之间发布地址和订阅地址相互对应。
  - 确保Client和Server的App Key一致。

5.5 未触发Binding

- 问题描述

  Model状态改变后，未触发与其Binding状态的相应改变。

- 问题分析

  Binding注册或Binding检查时，调用函数参数不正确。

  - 处理方法
    - 确保binding注册时，model_instance_index回调函数的调用正确。
    - 确保binding检查时，正确填入src model id、dest model id和model index。
    - 确保binding检查时，传入的数据和数据长度的正确性。
    - 确保在回调处理函数中，有处理对应的model id传入的消息。

版权所有 © 2023 深圳市汇顶科技股份有限公司