GR533x Mesh Simple On Off 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

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Simple On Off 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。

目录

1 简介

为了保证各类蓝牙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。

在标准Model无法满足应用时，用户可根据实际需求自定义Models。例如，应用程序需增加新功能，而标准Model并不包含这些功能。

本文以Simple On Off（SOO）Models为例，介绍如何开发以及使用自定义Models。

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/

2 SOO Models概述

Goodix提供参考版SOO Models，用于展示自定义Models的实现及使用。

SOO Models包括：

- SOO 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/关状态。
- SOO Server Model：用于接收来自客户端模型的消息，并根据客户端的请求，回复、更改服务器模型的开/关状态。

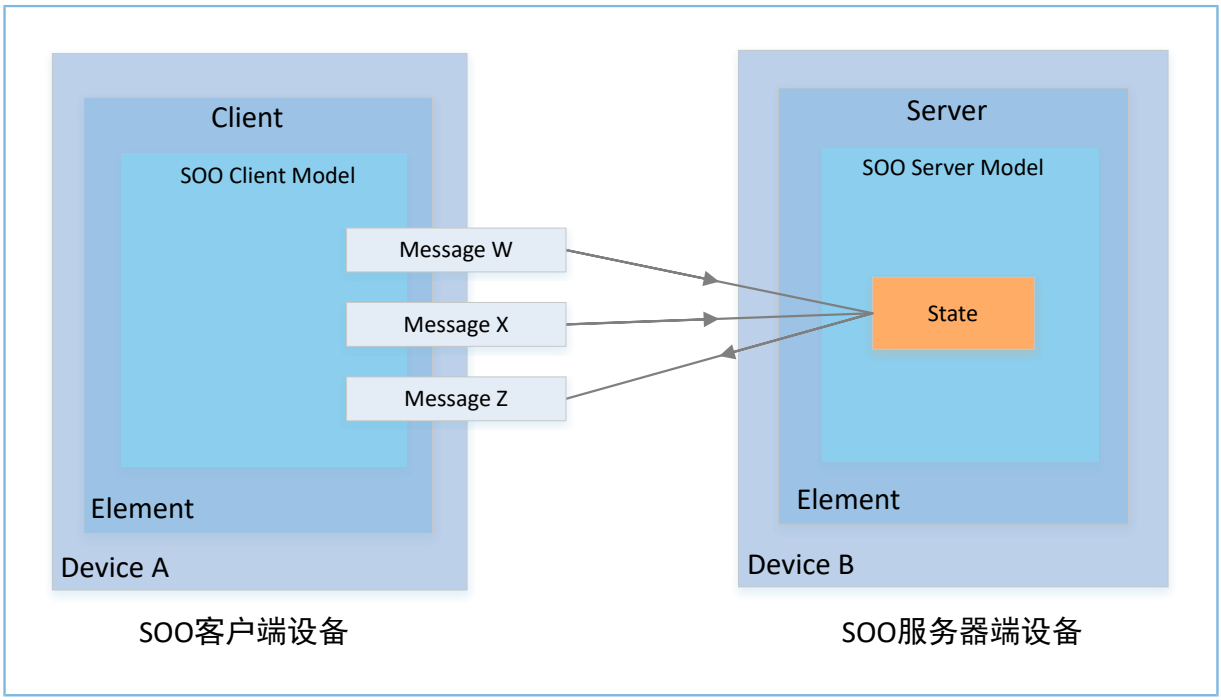

图 2-1 SOO客户端/服务端模型之间消息交互

2.1 发布消息类型

SOO Client Model支持的发布消息类型如下表所示。

表 2-1 SOO Client Model发布消息类型

消息类型	消息描述
SIMPLE_ONOFF_OPCODE_GET	获取Server设备当前的开/关状态
SIMPLE_ONOFF_OPCODE_SET	设置Server设备当前的开/关状态，并等待回复
SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED	设置Server设备当前的开/关状态，无需等待回复

SOO Server Model支持的发布消息类型如下表所示。

表 2-2 SOO Server Model发布消息类型

消息类型	消息描述
SIMPLE_ONOFF_OPCODE_STATUS	发布Server设备当前的开/关状态

3 运行示例

本章介绍如何运行SOO Client Model示例工程，并测试/验证运行结果。

SOO Model示例工程包括：

- SOO Client Models示例工程：位于`SDK_Folder\projects\mesh\Vendor\mesh_app_simple_on_off_client`
- SOO Server Models示例工程：位于`SDK_Folder\projects\mesh\Vendor\mesh_app_simple_on_off_server`

说明:

SDK_Folder为GR533x SDK的根目录。

3.1 准备工作

运行SOO Models示例之前，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SOO Client设备和SOO Server设备。

使用Type-C USB数据线连接SK板与PC。

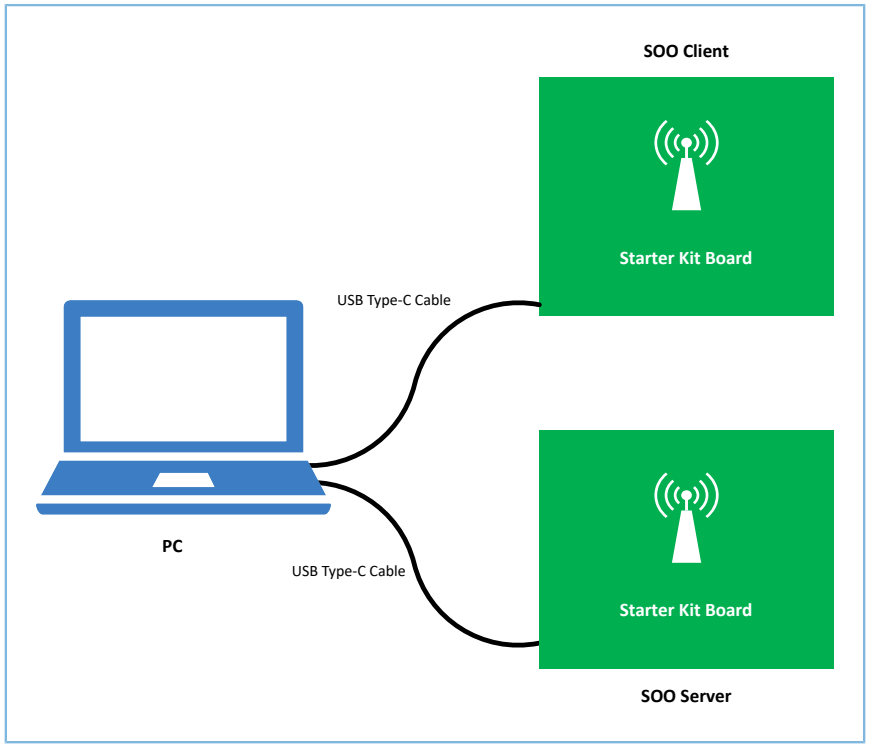

图 3-1 硬件连接示意图

3.3 下载固件

使用Keil工具 编译mesh_app_simple_on_off_client、mesh_app_simple_on_off_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。

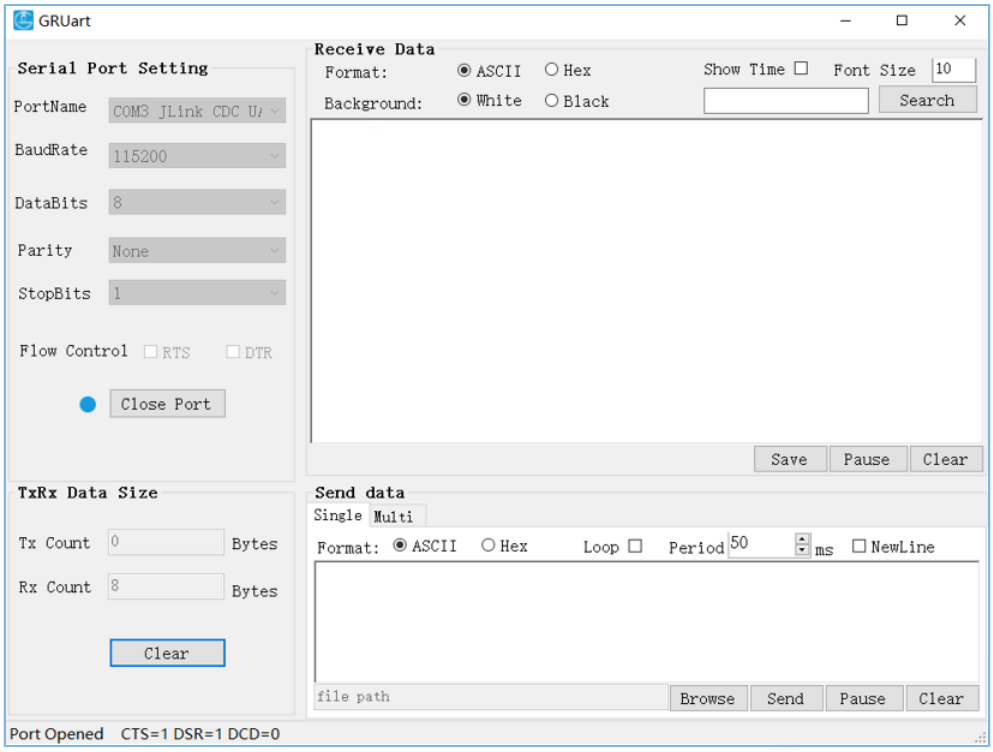

图 3-2 串口配置

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SOO Models之间的通信。

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

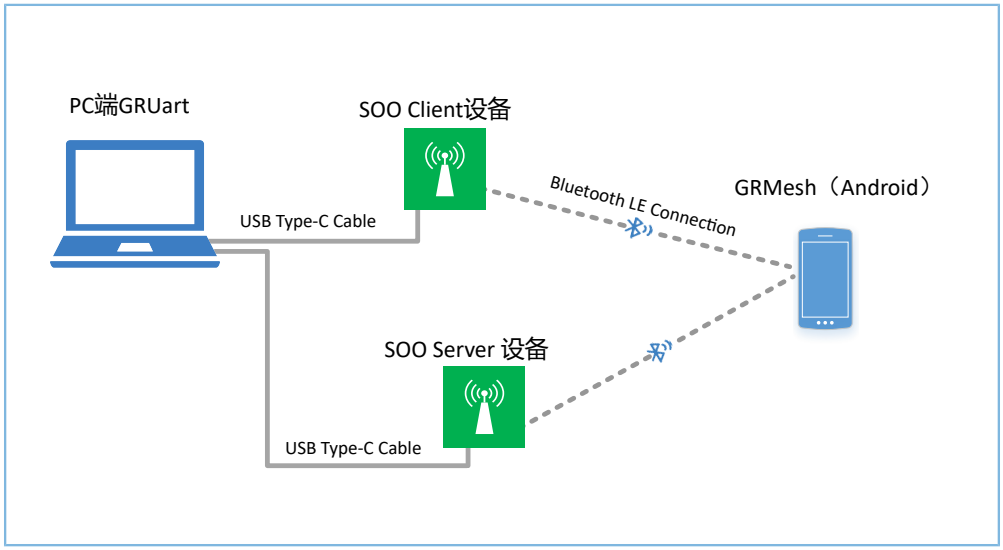

图 3-3 SOO Models测试场景图

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SOO Client设备（Goodix_Simple_Switch）和 SOO Server设备（Goodix_Simple_Light）,即两块SK板。

2. 使用GRMesh 开通配置 Goodix_Simple_Switch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_Simple_Switch”和“Goodix_Simple_Light”的设备。

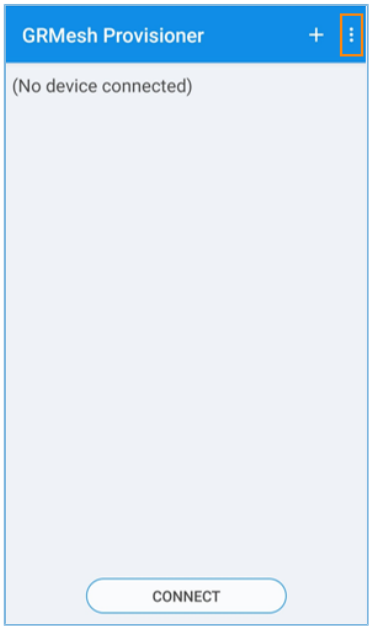

图 3-4 添加新设备

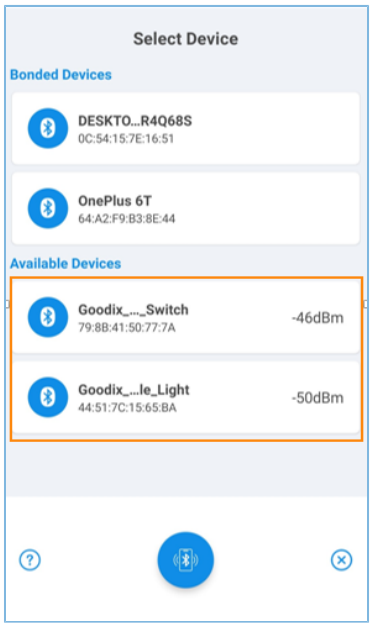

图 3-5 发现新设备

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

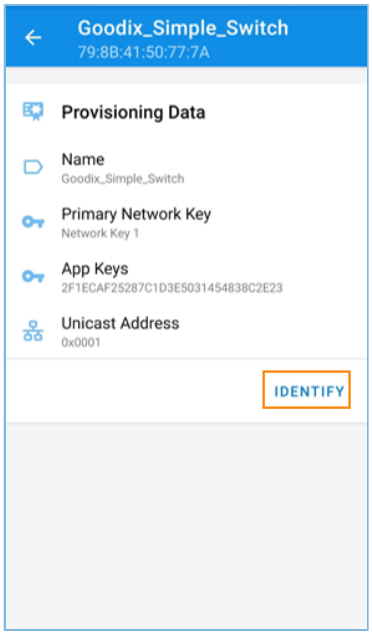

图 3-6 点击“IDENTIFY”

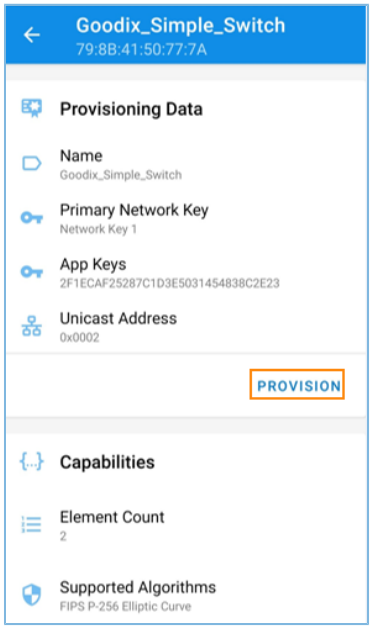

图 3-7 点击“PROVISION”

(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_Simple_Switch设备的开通配置，即将SOO Client设备加入到Mesh网络。

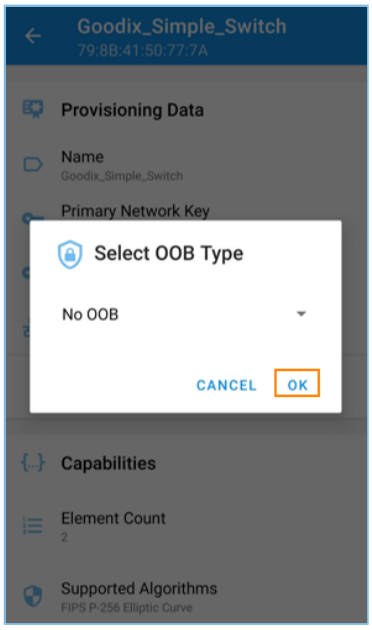

图 3-8 点击“OK”

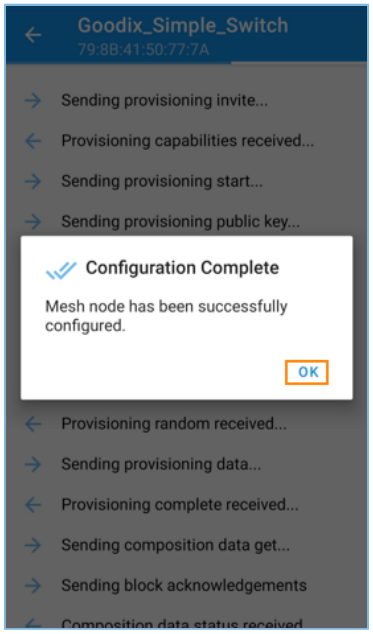

图 3-9 开通配置完成

3. 在“GRMesh Provisioner”界面，点击“Goodix_Simple_Switch > Element:0x0002 > Vendor Model”，配置Mesh网络中Goodix_Simple_Switch节点第一个元素中的Vendor Model。

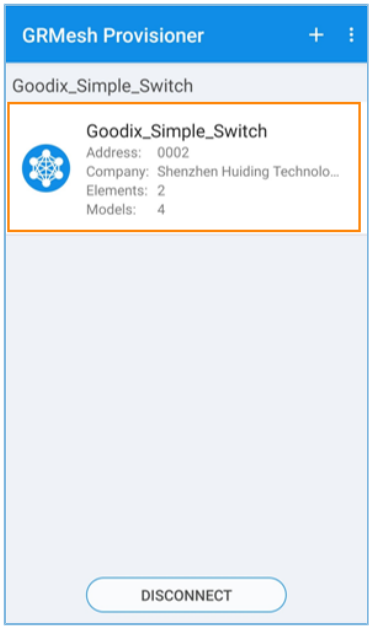

图 3-10 配置节点

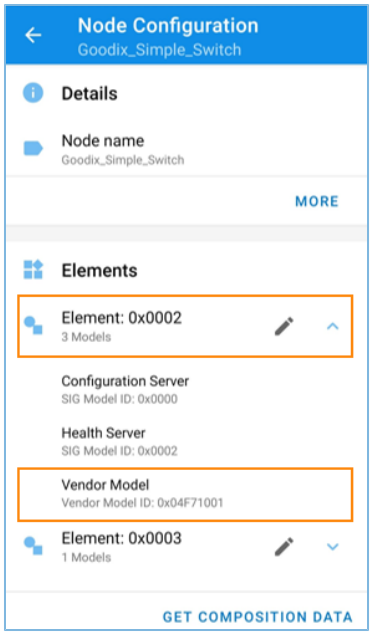

图 3-11 点击“Vendor Model”

(1) 在“Vendor Model”界面，配置“Bound App Keys”为“index 0”，“Publish Address”为单播地址“0x0004”（对应Goodix_Simple_Light中第一个元素的地址）。

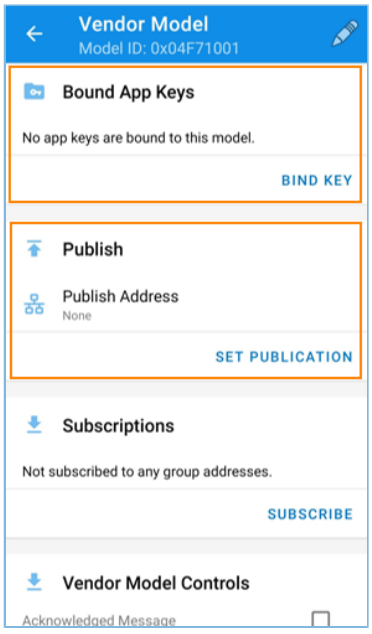

图 3-12 Vendor Model界面

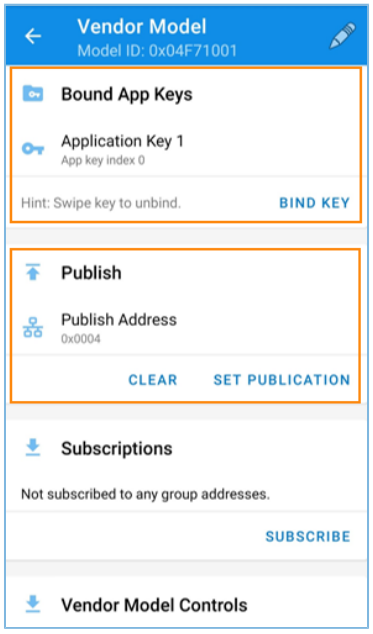

图 3-13 配置App Keys和地址

4. 重复步骤2～3，开通配置Goodix_Simple_Light设备，并设置第一个元素的Vendor Model的“Publish Address”设置为单播地址“0x0002”（对应Goodix_Simple_Switch第一个元素的地址）。

5. 完成Client设备与Server设备的开通配置后，可在“GRMeshProvisioner”界面查看到设备信息，如下图。

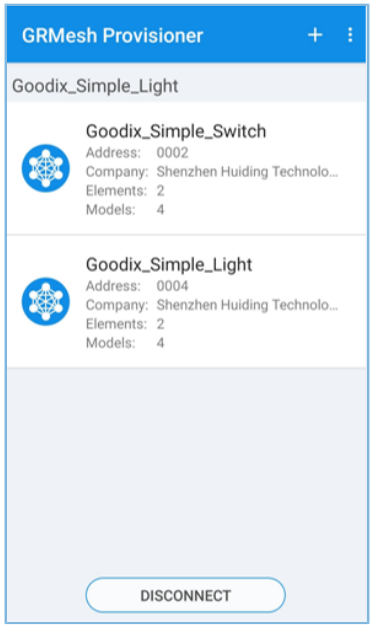

图 3-14 配置完成

3.6.2 GRUart验证SOO Models

使用GRUart 测试 SOO Client设备和SOO 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SOO Client设备串口，一个作为SOO Server设备串口。

2. 在Client设备串口输入“0001”。

   (1) Client设备发送“SIMPLE_ONOFF_OPCODE_SET”消息（携带开命令）。

   (2) Server设备接收到消息后，设置其当前状态为“开”并进行回复。

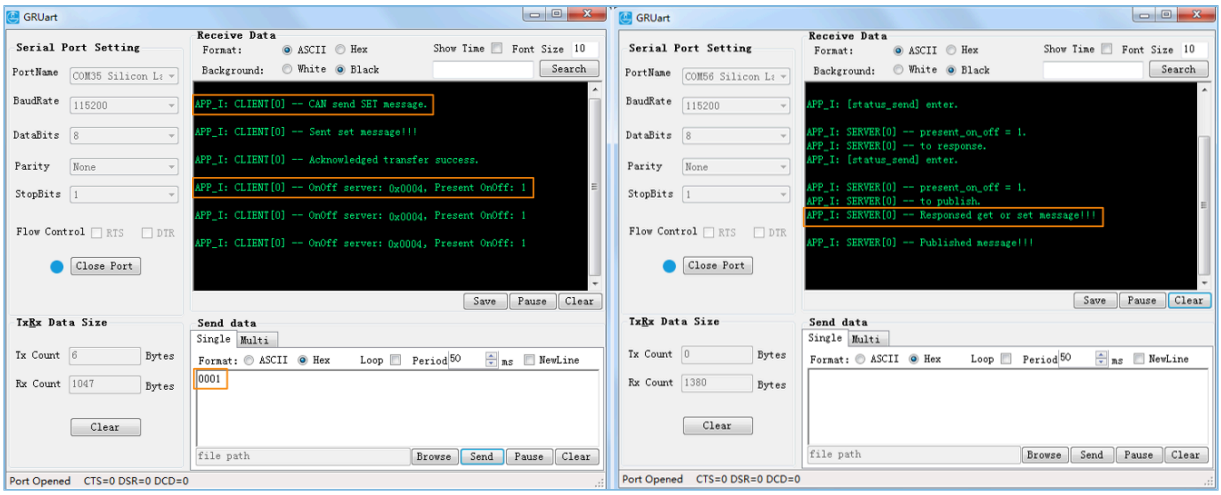

图 3-15 Client设备（左）与Server设备（右）消息交互

3. 在Client设备串口输入“0002”。

   (1) Client设备发送“SIMPLE_ONOFF_OPCODE_SET”消息（携带关命令）。

   (2) Server设备接收到消息后，设置其当前状态为“关”并进行回复。

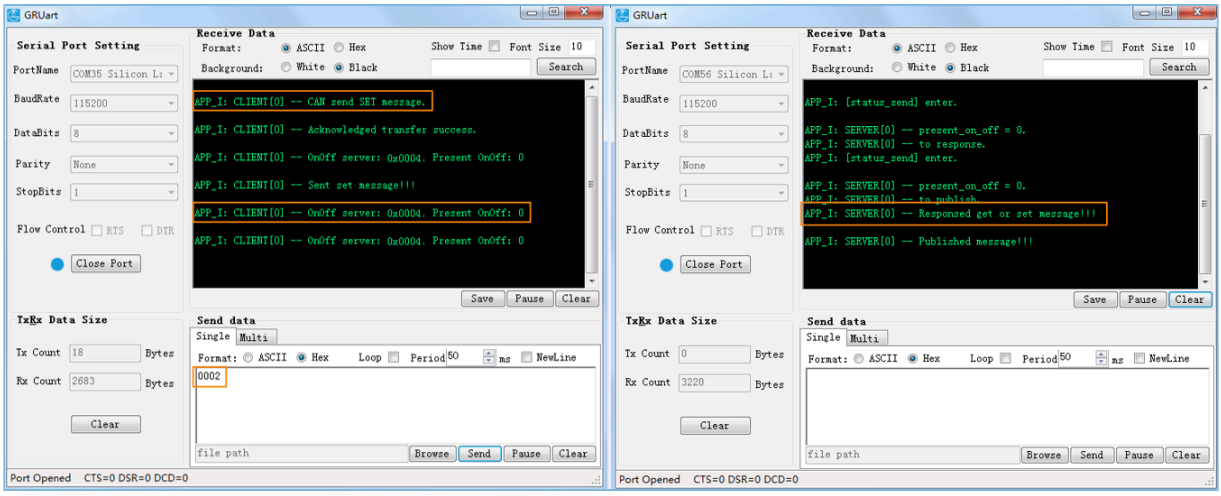

图 3-16 Client设备（左）与Server设备（右）消息交互

4 应用详解

本章主要介绍SOO Models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SOO 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SOO Models。

4.1.1 SOO Client Model工程

SOO Client Model示例的源代码和工程文件位于：

`SDK_Folder\projects\mesh\Vendor\mesh_app_simple_on_off_client`，其中工程文件位于keil_5文件夹。

在Keil中打开*mesh_app_simple_on_off_client.uvprojx*，可查看到SOO 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_simple_on_off_client文件说明

Group	文件	描述
gr_models	simple_onoff_client.c	SOO Models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	SOO Models注册及应用逻辑处理
	custom_config.h	Application工程通用配置
	mesh_stack_config.h	Mesh协议相关配置

4.1.2 SOO Server Model工程

SOO Server Model示例的源代码和工程文件位于：

`SDK_Folder\projects\mesh\Vendor\mesh_app_simple_on_off_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_simple_on_off_server.uvprojx*，可查看到SOO 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_simple_on_off_server文件说明

Group	文件	描述
gr_models	simple_onoff_server.c	Simple On Off Server Model实现
user_callback	user_gap_callback.c	获取连接和断连事件

Group	文件	描述
	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	Simple On Off Server Model注册及应用逻辑处理
	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 SOO Models消息处理

本节主要介绍SOO Models消息处理的实现代码，以展示如何实现自定义Models消息的处理 。

4.2.1.1 SOO Client Model

本节示例代码的路径为：

SDK_Folder\components\mesh\models\Vendor\simple_onoff\src\simple_onoff_client.c

4.2.1.1.1 可处理消息表

可处理消息表simple_on_off_client_opcode_list包含SOO Client Model可接收并处理的Mesh消息。

```
static const uint16_t simple_on_off_client_opcode_list[] =
{
    SIMPLE_ONOFF_OPCODE_STATUS,
};
```

4.2.1.1.2 回调函数处理表

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

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

```
static const mesh_model_cb_t simple_on_off_client_msg_cb =
{
    .cb_rx             = simple_on_off_client_rx_cb,
    .cb_sent           = simple_on_off_client_sent_cb,
    .cb_publish_period  = NULL,
};
```

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

```
static void simple_on_off_client_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 = m_opcode_handlers[company_opcode
                                                - SIMPLE_ONOFF_OPCODE_STATUS].handler;

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

static const mesh_opcode_handler_t m_opcode_handlers[] =
{
    {SIMPLE_ONOFF_OPCODE_STATUS, status_handle},
};
static void status_handle(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    simple_onoff_client_t * p_client = (simple_onoff_client_t *) p_args;

    if (p_rx_msg->msg_len == SIMPLE_ONOFF_STATUS_LEN)
    {
        p_client->settings.p_callbacks->onoff_status_cb(p_client, p_rx_msg,
                                        (simple_onoff_status_msg_pkt_t *)p_rx_msg->msg);
    }
}
```

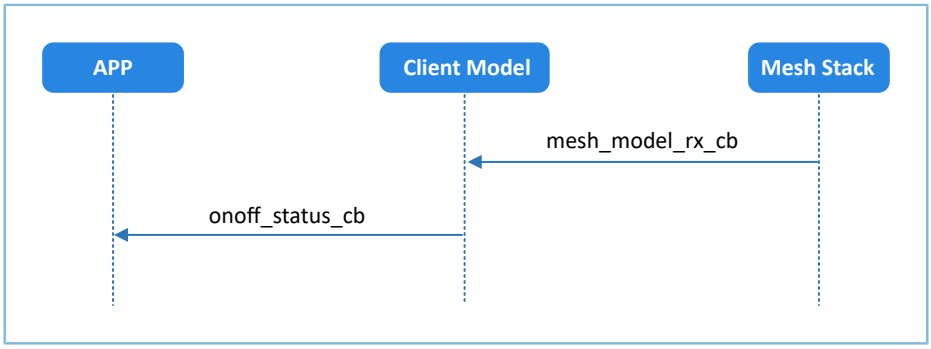

图 4-1 SIMPLE_ONOFF_OPCODE_STATUS消息处理

4.2.1.1.3 发送消息处理

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

- SIMPLE_ONOFF_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包括期待回复消息的opcode以及等待时间。

```
uint16_t simple_onoff_client_set(simple_onoff_client_t * p_client,
                                        const simple_onoff_set_msg_pkt_t * p_params)
```

```
{
    mesh_model_send_info_t model_msg_send;
    bool reliable_trans_state = false;
    uint8_t tx_hdl = SIMPLE_ONOFF_CLIENT_SET_SEND_TX_HDL +
                     p_client->model_instance_index * SIMPLE_ONOFF_CLIENT_TX_HDL_TOTAL;

    mesh_model_reliable_info_t reliable_info =
    {
        .reply_opcode = MESH_ACCESS_OPCODE_VENDOR(SIMPLE_ONOFF_OPCODE_STATUS,
                                                  SIMPLE_ONOFF_COMPANY_ID),
        .status_cb = p_client->settings.p_callbacks->ack_transaction_status_cb,
        .timeout_ms = p_client->settings.timeout_ms,
    };

    if (p_client == NULL || p_params == NULL)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    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,
                           SIMPLE_ONOFF_OPCODE_SET,
                           tx_hdl, (uint8_t *)p_params, SIMPLE_ONOFF_SET_LEN);

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

完成SET消息发送后，SOO Client Model会通过回调函数通知用户应用操作状态（如图 4-2中的方式1、方式2）。若用户应用想提前终止该操作，可调用simple_onoff_client_setget_cancel函数实现，SOO Client Model也会通过回调函数通知用户应用操作状态（如图 4-2中的方式3）。

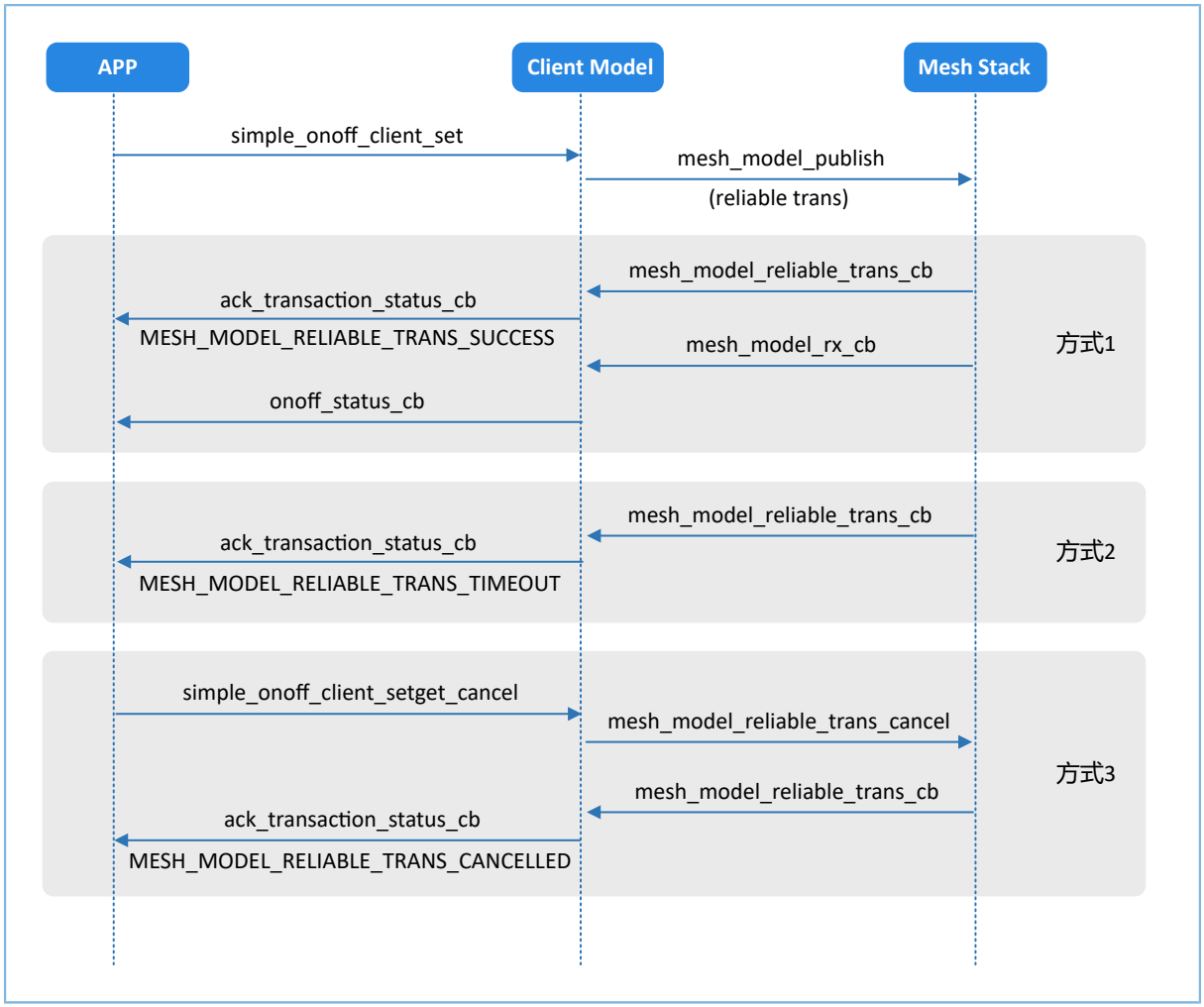

图 4-2 SIMPLE_ONOFF_OPCODE_SET消息发送

- SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED消息

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

```
uint16_t simple_onoff_client_set_unack(simple_onoff_client_t * p_client,
                                       const simple_onoff_set_msg_pkt_t * p_params)
{
    mesh_model_send_info_t model_msg_send;
    uint8_t tx_hdl = SIMPLE_ONOFF_CLIENT_SET_UNRELIABLE_SEND_TX_HDL
                     + p_client->model_instance_index * SIMPLE_ONOFF_CLIENT_TX_HDL_TOTAL;

    if (p_client == NULL || p_params == NULL)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    message_create(&model_msg_send, p_client->model_lid,
                   SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED,
                   tx_hdl, (uint8_t *)p_params, SIMPLE_ONOFF_SET_LEN);
```

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

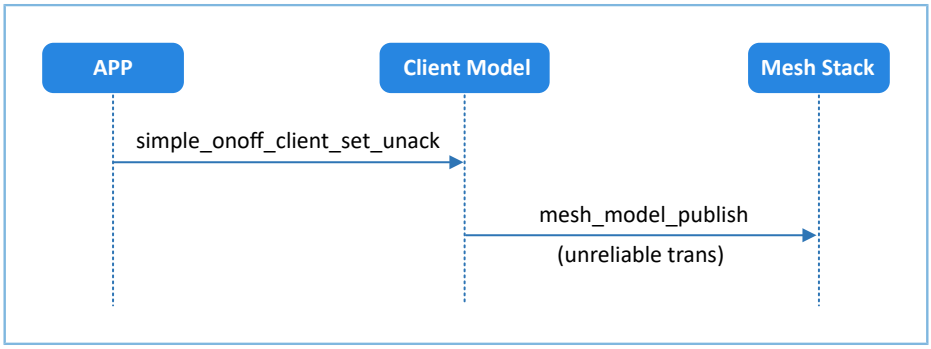

图 4-3 SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED消息发送

- SIMPLE_ONOFF_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包括期待回复消息的opcode以及等待时间。

```
uint16_t simple_onoff_client_get(simple_onoff_client_t * p_client)
{
    mesh_model_send_info_t model_msg_send;
    bool reliable_trans_state = false;
    uint8_t tx_hdl = SIMPLE_ONOFF_CLIENT_GET_SEND_TX_HDL
          + p_client->model_instance_index * SIMPLE_ONOFF_CLIENT_TX_HDL_TOTAL;

    mesh_model_reliable_info_t reliable_info =
    {
        .reply_opcode = MESH_ACCESS_OPCODE_VENDOR(SIMPLE_ONOFF_OPCODE_STATUS,
                                                  SIMPLE_ONOFF_COMPANY_ID),
        .status_cb = p_client->settings.p_callbacks->ack_transaction_status_cb,
        .timeout_ms = p_client->settings.timeout_ms,
    };

    if (p_client == NULL)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    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,
                           SIMPLE_ONOFF_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，SOO Client Model会通过回调函数通知用户应用操作状态（如图 4-4中的方式1、方式2）。若用户应用想提前终止该操作，可调用simple_onoff_client_setget_cancel函数实现，SOO Client Model也会通过回调函数通知用户应用操作状态（如图 4-4中的方式3）。

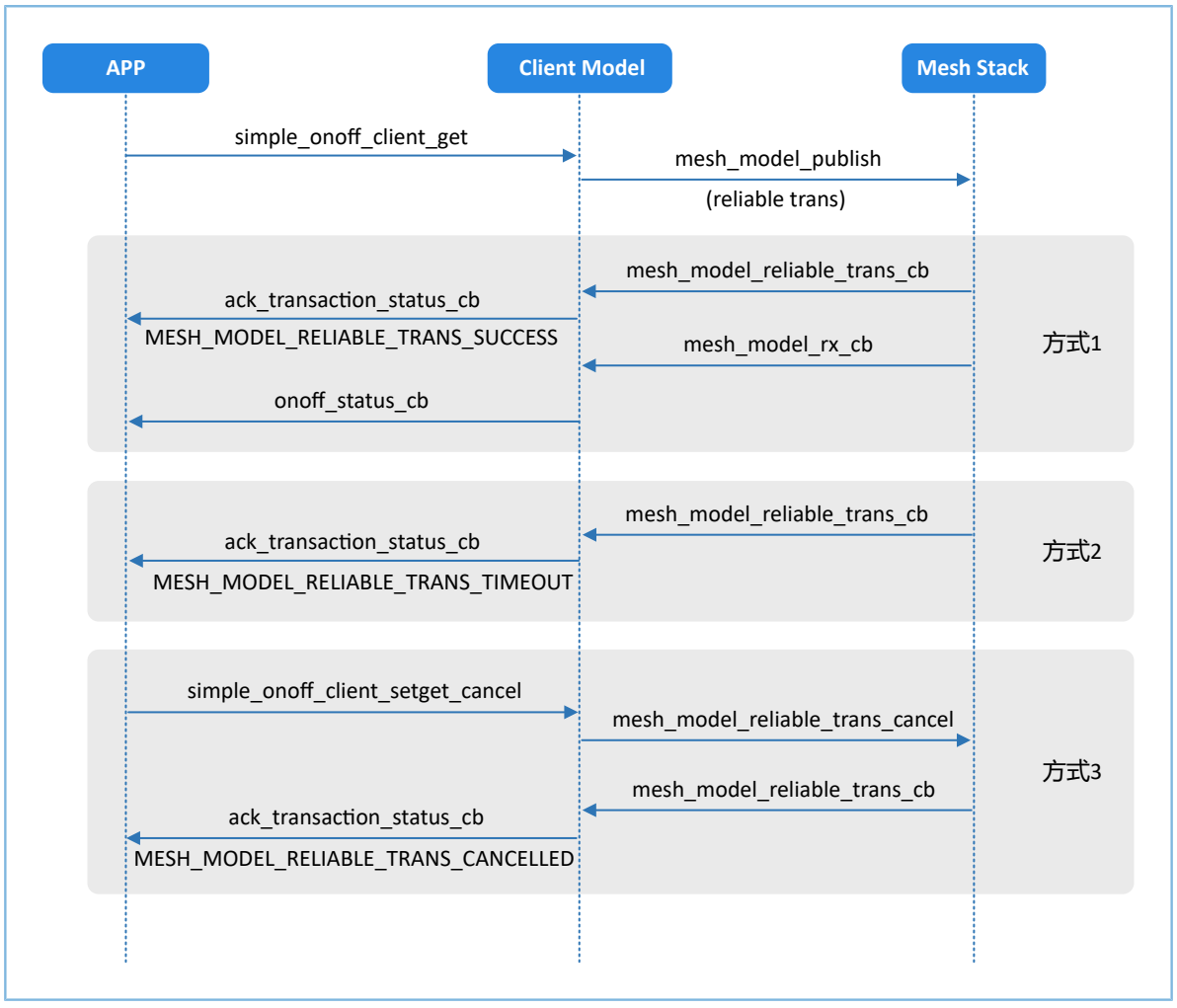

图 4-4 SIMPLE_ONOFF_OPCODE_GET消息发送

4.2.1.2 SOO Server Model

本节示例代码的路径为：

SDK_Folder\components\mesh\models\Vendor\simple_onoff\src\simple_onoff_server.c

4.2.1.2.1 可处理消息表

可处理消息表simple_on_off_server_opcode_list包含SOO Server Model可接收并处理的Mesh消息。

```
static const uint16_t simple_on_off_server_opcode_list[] =
{
    SIMPLE_ONOFF_OPCODE_GET,
    SIMPLE_ONOFF_OPCODE_SET,
    SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED,
};
```

4.2.1.2.2 回调函数处理表

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

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

```
static const mesh_model_cb_t simple_on_off_server_msg_cb =
{
    .cb_rx             = simple_on_off_server_rx_cb,
    .cb_sent           = simple_on_off_server_sent_cb,
    .cb_publish_period  = NULL,
};
```

simple_on_off_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 simple_on_off_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 = m_opcode_handlers[company_opcode -
                                            SIMPLE_ONOFF_OPCODE_GET].handler;

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

static const mesh_opcode_handler_t m_opcode_handlers[] =
{
    {SIMPLE_ONOFF_OPCODE_GET, handle_get_cb},
    {SIMPLE_ONOFF_OPCODE_SET, handle_set_cb},
    {SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED, handle_set_cb},
};
```

- handle_set_cb函数可处理以下消息：
  - SIMPLE_ONOFF_OPCODE_SET
  - SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED

```
static void handle_set_cb(const mesh_model_msg_ind_t *p_rx_msg,  void *p_args)
{
    uint32_t send_status = MESH_ERROR_NO_ERROR;
    simple_onoff_server_t  * p_server = (simple_onoff_server_t  *) p_args;
     APP_LOG_INFO("SERVER[%d] -- Receive message, want to set on-off state!!!", p_server->model_instance_index);

       simple_onoff_set_msg_pkt_t  *  p_msg_params_packed  =  (simple_onoff_set_msg_pkt_t  *) p_rx_msg->msg;
    simple_onoff_status_msg_pkt_t out_data =
    {
        .present_on_off = p_msg_params_packed->on_off,
    };

    if (set_params_validate(p_rx_msg, p_msg_params_packed))
    {
        if (model_tid_validate(&p_server->tid_filter, p_rx_msg, p_msg_params_packed->tid))
        {
            p_server->client_address = p_rx_msg->src;
           p_server->settings.p_callbacks->onoff_cbs.set_cb(p_server->model_instance_index, p_msg_params_packed->on_off);

            // response
            if (SIMPLE_ONOFF_OPCODE_SET == p_rx_msg->opcode.company_opcode)
            {
                send_status = status_send(p_server, p_rx_msg, &out_data);
                if (MESH_ERROR_NO_ERROR != send_status)
                {
                      APP_LOG_WARNING("SERVER[%d] -- Response status faild, error code: 0x%04x", p_server->model_instance_index, send_status);
                }
            }

            // publish
            send_status = status_send(p_server, NULL, &out_data);
            if (MESH_ERROR_NO_ERROR != send_status)
            {
                 APP_LOG_WARNING("SERVER[%d] -- Publish status faild, error code: 0x%04x", p_server->model_instance_index, send_status);
            }
        }
    }
}
static uint32_t status_send(simple_onoff_server_t * p_server,
```

```
                                const mesh_model_msg_ind_t *p_rx_msg,
                                const simple_onoff_status_msg_pkt_t * p_params)
{
    simple_onoff_status_msg_pkt_t msg_pkt;
   uint8_t tx_hdl = (NULL == p_rx_msg) ? SIMPLE_ONOFF_SERVER_PUBLISH_SEND_TX_HDL + p_server->model_instance_index * SIMPLE_ONOFF_SERVER_TX_HDL_TOTAL
                                       : SIMPLE_ONOFF_SERVER_RSP_SEND_TX_HDL + p_server->model_instance_index * SIMPLE_ONOFF_SERVER_TX_HDL_TOTAL;

    if (p_params->present_on_off > SIMPLE_ONOFF_MAX)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    msg_pkt.present_on_off = p_params->present_on_off;

    mesh_model_send_info_t msg_send =
    {
        .model_lid = p_server->model_lid,
                        .opcode   =   MESH_ACCESS_OPCODE_VENDOR(SIMPLE_ONOFF_OPCODE_STATUS, SIMPLE_ONOFF_COMPANY_ID),
        .tx_hdl = tx_hdl,
        .p_data_send = (uint8_t *) &msg_pkt,
        .data_send_len = SIMPLE_ONOFF_STATUS_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]  --  present_on_off  =  %d.",  p_server->model_instance_index, msg_pkt.present_on_off);

    if (NULL == p_rx_msg)
    {

        APP_LOG_INFO("SERVER[%d] -- to publish.", p_server->model_instance_index);
        return mesh_model_publish(&msg_send, NULL);
    }
    else
    {
        APP_LOG_INFO("SERVER[%d] -- to response.", p_server->model_instance_index);
        return mesh_model_rsp_send(&msg_send);
    }
}
```

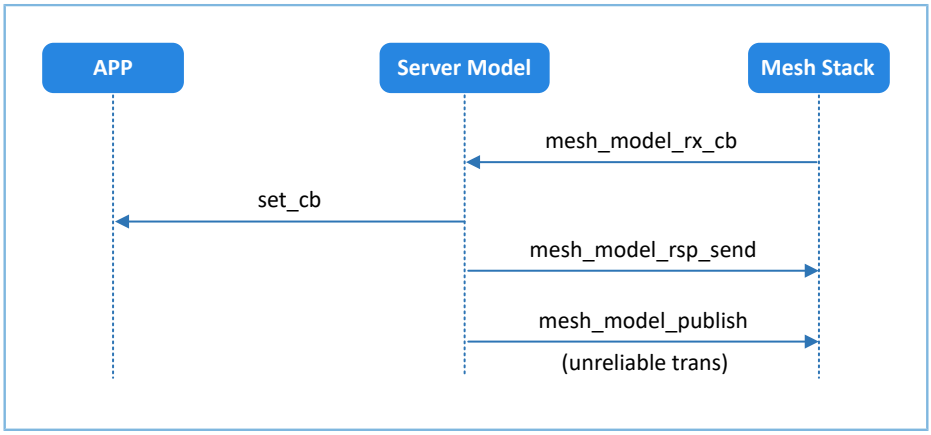

图 4-5 SIMPLE_ONOFF_OPCODE_SET消息处理

*_SET和*_SET_UNACKNOWLEDGE消息的处理有所不同，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。

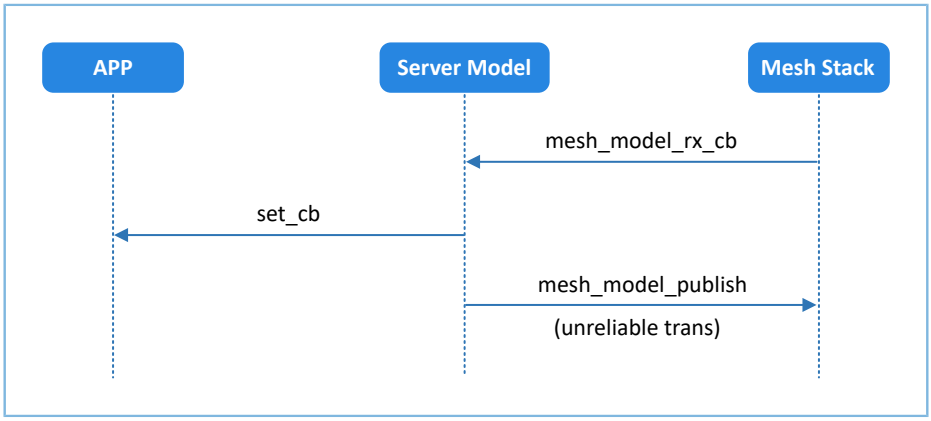

图 4-6 SIMPLE_ONOFF_OPCODE_SET_UNACKNOWLEDGED消息处理

- handle_get_cb函数

  该函数可处理SIMPLE_ONOFF_OPCODE_GET消息。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)
{
    simple_onoff_server_t * p_server = (simple_onoff_server_t *) p_args;

    APP_LOG_INFO("[%s] enter.", __func__);
     APP_LOG_INFO("SERVER[%d] -- Receive message, want to get on-off state!!!", p_server-
>model_instance_index);

    simple_onoff_status_msg_pkt_t out_data = {0};

    if (get_params_validate(p_rx_msg))
    {
        p_server->client_address = p_rx_msg->src;
```

```
            p_server->settings.p_callbacks->onoff_cbs.get_cb(p_server->model_instance_index,
 &out_data.present_on_off);
        uint32_t send_status = status_send(p_server, p_rx_msg, &out_data);
        if (MESH_ERROR_NO_ERROR != send_status)
        {
           APP_LOG_ERROR("SERVER[%d] -- Publish status faild, error code: 0x%04x", p_server-
>model_instance_index, send_status);
        }
    }
}
```

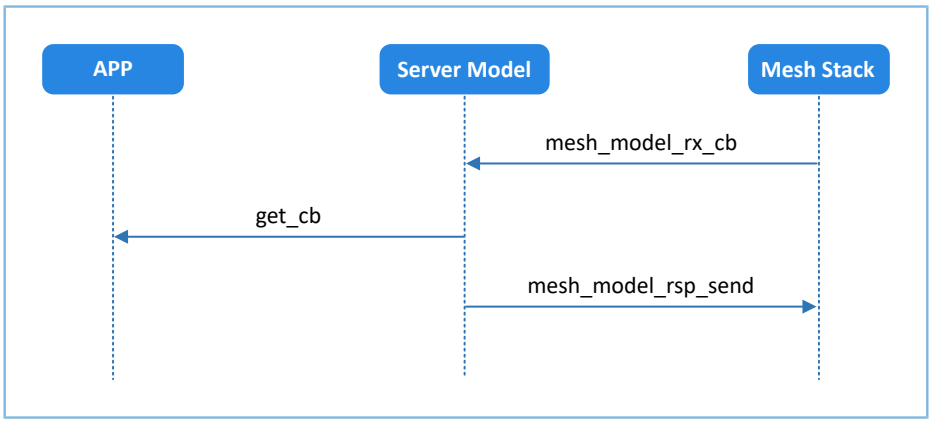

图 4-7 SIMPLE_ONOFF_OPCODE_GET消息处理

4.2.1.2.3 发送消息处理

SOO Server Model可主动发送SIMPLE_ONOFF_OPCODE_STATUS消息。status_send函数调用mesh_model_publish进行消息发布。

```
uint16_t simple_onoff_server_status_publish(simple_onoff_server_t * p_server,
                                    const simple_onoff_status_msg_pkt_t * p_params)
{
    if (NULL == p_server || NULL == p_params)
    {
        return MESH_ERROR_SDK_INVALID_PARAM;
    }

    return status_send(p_server, NULL, p_params);
}
```

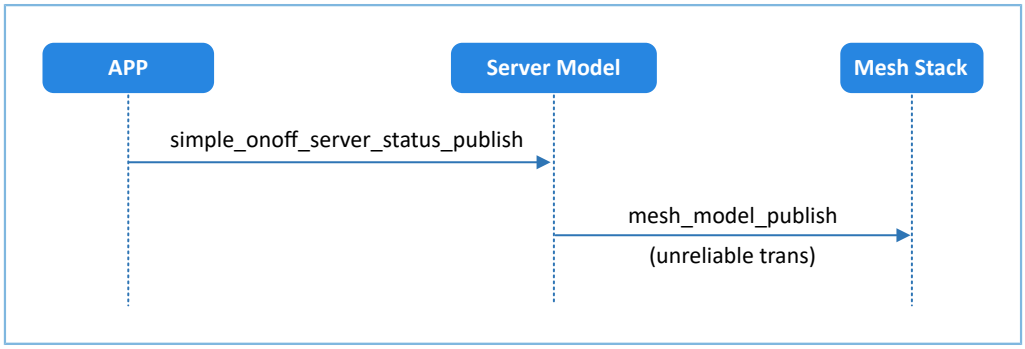

图 4-8 SIMPLE_ONOFF_OPCODE_STATUS消息发送