GR551x AMS Profile示例手册

版本： **1.6**

发布日期： **2020-06-30**

深圳市汇顶科技股份有限公司

前言

编写目的

本文档介绍了如何使用和验证GR551x SDK中的AMS Client示例，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：

- GR551x用户
- GR551x开发人员
- GR551x测试人员
- iOS开发工程师
- 开发爱好者
- 文档工程师

版本说明

本文档为第4次发布，对应的产品系列为GR551x。

修订记录

版本	日期	修订内容
1.0	2019-12-08	首次发布
1.3	2020-03-16	更新了“3.6 测试验证”章节中串口输出时对象ID的名称
1.5	2020-05-30	更正了“4 应用详解”章节中的代码缩进格式
1.6	2020-06-30	基于SDK刷新版本

目录

1 简介

苹果媒体服务（Apple Media Service，AMS）应用于蓝牙设备，如手环、手表等智能设备。蓝牙设备可通过BLE连接获取iOS设备的媒体通知，也可以向iOS设备发送AMS相关控制命令。

本文档主要介绍如何基于GR551x平台实现AMS Client。

在进行操作前，建议参考以下文档。

表 1-1 文档参考

名称	描述
AMS Reference	AMS协议：Apple Media Service(AMS) Specification
GR551x开发者指南	GR551x软硬件介绍、快速使用及资源总览
Bluetooth Core Spec v5.1	Bluetooth官方标准核心规范5.1： https://www.bluetooth.com/specifications/bluetooth-core-specification/
Bluetooth GATT Spec	Bluetooth Profile和Service的详细信息： www.bluetooth.com/specifications/gatt
J-Link用户指南	J-Link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 Profile概述

AMS Profile定义了以下两种设备角色：

- Server端：iOS设备为中央设备（Central），提供Service以及数据源。
- Client端：蓝牙设备为外围设备（Peripheral），连接iOS设备后，发现iOS设备上的服务以及读写其数据。

Server端与Client端的交互流程如图 2-1所示：

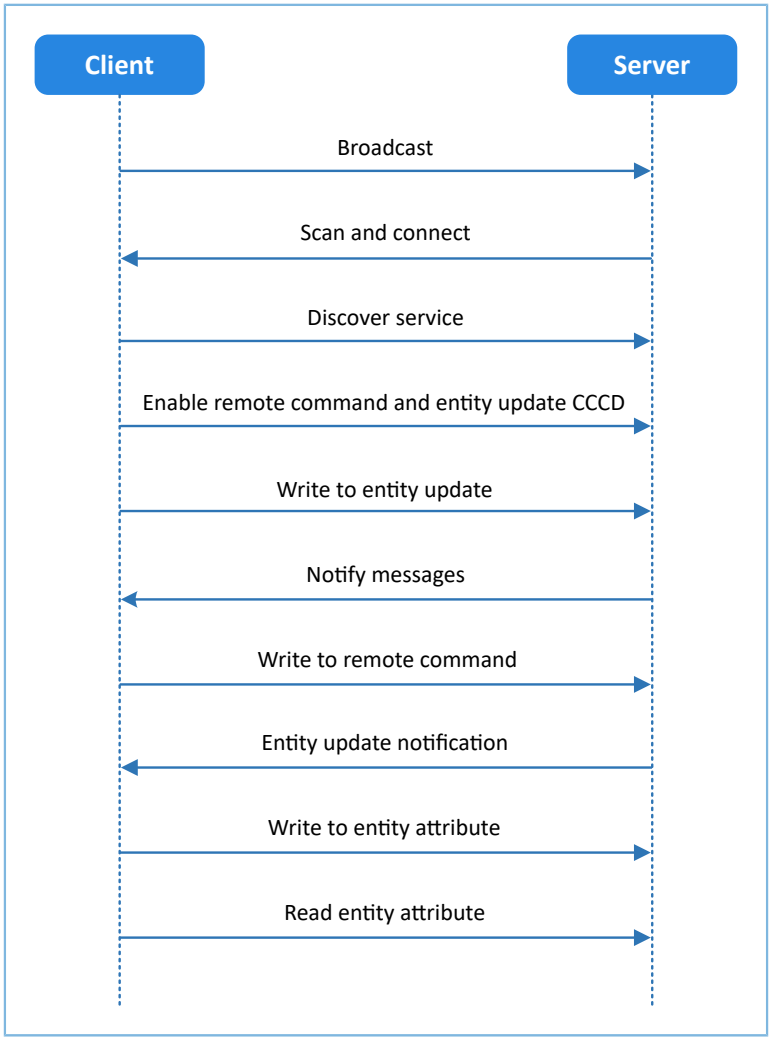

图 2-1 Client和Server 交互流程图

AMS 的特征（Characteristic）有：Remote Command、Entity Update、Entity Attribute，如下表 2-1 所示。

表 2-1 AMS Characteristic说明

Characteristic	UUID	Type	Support	Security	Properties
Remote Command	9B3C81D8-57B1-4A8A-B8DF-0E56F7CA51C2	128 bits	Mandatory	None	Write, Notify
Entity Update	2F7CABCE-808D-411F-9A0C-BB92BA96C102	128 bits	Mandatory	None	Write, Notify

Characteristic	UUID	Type	Support	Security	Properties
Entity Attribute	C6B2F38C-23AB-46D8-A6AB-A3A870BBD5D7	128 bits	Mandatory	None	Write, Read

各特征的作用如下：

- Remote Command：发送远程命令并接收可用远程命令更新。Write用于从蓝牙设备端发送远程命令到iOS设备端来实现远程控制。Notify用于当iOS设备端的可用远程命令发生变化时，iOS设备端向蓝牙设备端通知更新后的可用远程命令。
- Entity Update：设置关注对象并接收关注对象更新。Write用于在蓝牙设备端设置关注对象。Notify用于当iOS设备端的关注对象发生变化时，iOS设备端向蓝牙设备端通知关注对象更新后的值（受限于MTU长度，该值可能是不完整的）。
- Entity Attribute：设置和读取展示对象。Write用于在蓝牙设备端设置展示对象，Read用于在设置展示对象后，从iOS设备端读取展示对象的完整值。

3 初次运行

本章将介绍使用GR5515 Starter Kit开发板（以下简称GR5515 SK开发板）作为AMS Client、iOS设备作为AMS Server，初次运行AMS Client示例时的准备工作，相关连接及测试方法。

说明:

SDK_Folder为GR551x SDK的根目录。

3.1 准备工作

运行AMS Client示例之前，需要完成以下准备工作。

- 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	GR5515 Starter Kit开发板
数据线	Micro USB 2.0数据线
iOS设备	支持BLE 4.0及以上的iOS设备，如iPhone 4s/iPad 3及其以上版本

- 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/
GProgrammer（Windows）	GR551x Programming工具，位于`SDK_Folder\tools\GProgrammer`
GRUart（Windows）	GR551x串口调试工具，位于`SDK_Folder\tools\GRUart`

3.2 硬件连接

使用Micro USB 2.0数据线连接GR5515 SK开发板与计算机。

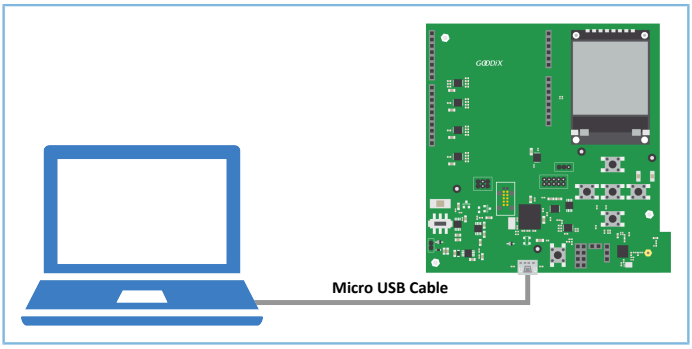

图 3-1 硬件连接示意图

3.3 下载固件

通过GProgrammer下载*ble_app_ams_c_fw.bin*固件至GR5515 SK开发板，具体操作方法请参考《GProgrammer用户手册》。

说明:

*ble_app_ams_c_fw.bin*位于：

`SDK_Folder\projects\ble\ble_peripheral\ble_app_ams_c\build\`

3.4 串口设置

启动GRUart，按照表 3-3 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

配置完成后，点击“Open Port”打开，如图 3-2所示。

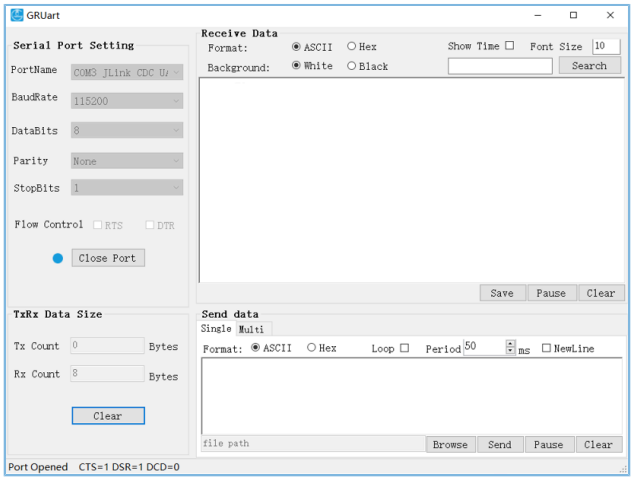

图 3-2 串口设置

此时还未有AMS服务通知产生，因此界面上无任何显示。

3.5 蓝牙连接

GR5515 SK开发板上电。打开iOS设备蓝牙功能，搜索周边蓝牙设备，发现广播名为“Goodix_AMS_C”的GR5515 SK开发板，如图 3-3所示。

图 3-3 发现广播名Goodix_AMS_C的设备

说明:

本文档测试用iOS设备为iPhone 6S，系统版本为12.4.1，可能与实际用户界面有差异。

点击“Goodix_AMS_C”连接GR5515 SK开发板，自动弹出如图 3-4所示的配对请求窗口。在该窗口中，输入代码“123456”（配对所需代码的设置方法，请参考4.2.1 连接、配对及绑定中app_sec_rcv_enc_req_cb()函数说明）后点击“配对”。

图 3-4 输入配对信息

配对成功后，提示“已连接”信息，如图 3-5所示。

图 3-5 设备配对成功

iOS设备与GR5515 SK开发板连接成功后，连接信息将显示在串口调试工具GRUart的界面上，如图 3-6所示。

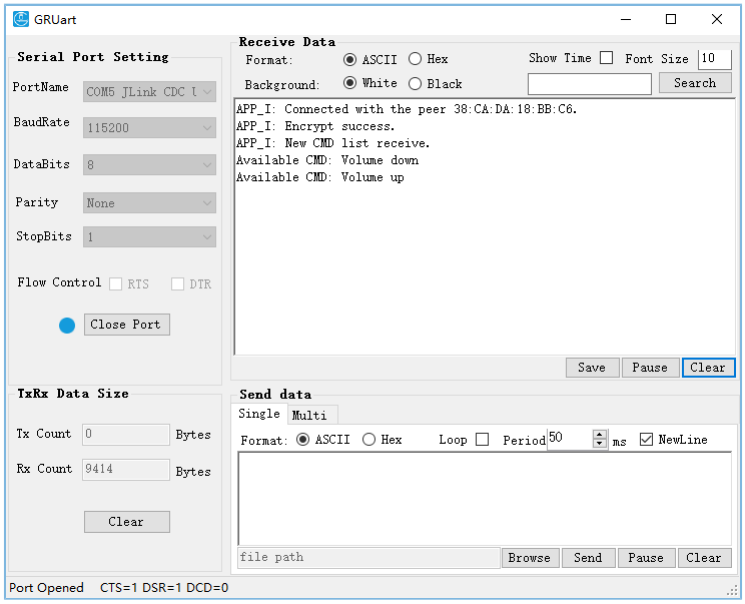

图 3-6 串口打印信息

3.6 测试验证

以上工作准备就绪后，iOS设备和GR5515 SK开发板即可开始AMS通讯测试。在本示例中针对AMS服务的测试，包括：

- GR5515 SK开发板获取iOS设备的媒体通知。
- GR5515 SK开发板向iOS设备发送AMS控制命令。

用户可根据GRUart界面显示的打印信息，验证AMS服务是否运行正常。（如需进一步了解AMS，请参考Apple Media Service(AMS) Specification）。

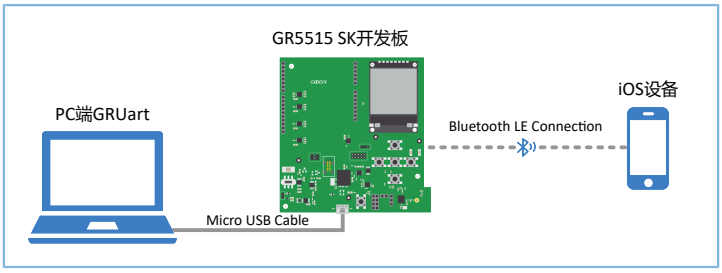

图 3-7 AMS测试场景图

3.6.1 获取iOS设备媒体通知

GR5515 SK开发板获取iOS设备媒体通知的测试步骤如下：

1. 在iOS设备上打开任意音乐播放器App。
2. 使用音乐播放器App播放任意歌曲，如图 3-8所示，示例中歌曲名为The Sound Of Silence，歌手名为Pat Metheny。

图 3-8 iOS设备上播放歌曲

3. 查看PC端GRUart的界面上的打印信息。

在本示例中，GRUart的界面上显示的打印信息（如图 3-9所示），包括：

- 该音乐播放器App支持的远程命令列表。
- 关注对象的更新信息。如关注了iOS设备媒体的Track-Title和Track-Artist对象，该对象发生改变时会发送相应的通知。（关于对象的详细信息，请参考Apple Media Service(AMS) Reference）。

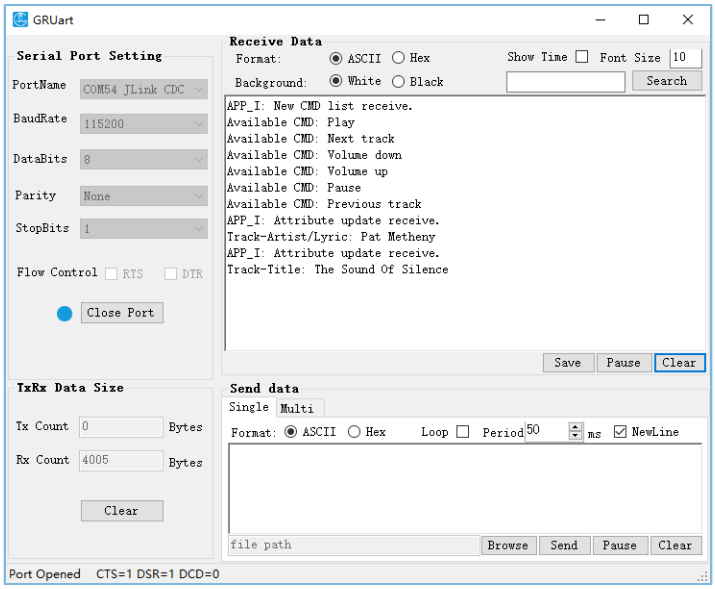

图 3-9 获取iOS设备媒体通知的打印信息

串口输出信息说明如下：

表 3-4 通知信息说明

名称	描述
APP_I: New CMD list receive	收到新的可用远程命令列表。
Available CMD: Next track	表明当前Next track命令可用，可发送“下一首”远程命令。
APP_I: Attribute update receive	关注的某个对象发生变化，收到该对象新的值。
Track-Artist/Lyric: Pat Metheny	Track-Artist对象发生变化，其新的值为“Pat Metheny”。

说明:

根据Apple Media Service(AMS) Reference的规定，Track-Artist的对象ID对应当前播放曲目的演唱者信息，但目前多数的音乐播放器App还通过该ID传递歌词信息。因此，在串口输出信息中，该ID对应的名称显示为：Track-Artist/Lyric。

通过图 3-9的打印信息，可确定GR5515 SK开发板和iOS设备之间的AMS服务运行正常。

3.6.2 向iOS设备发送控制命令

GR5515 SK开发板向iOS设备发送控制命令的步骤如下：

1. 在iOS设备上打开任意音乐播放器App。
2. 使用音乐播放器App播放任意一首歌曲。
3. 在音乐播放过程中，按下GR5515 SK开发板的“RIGHT”按键（即发送播放“下一首”的命令）。
4. 查看音乐播放器是否切换到播放列表中的下一首歌曲进行播放，以及GRUart界面显示的信息。

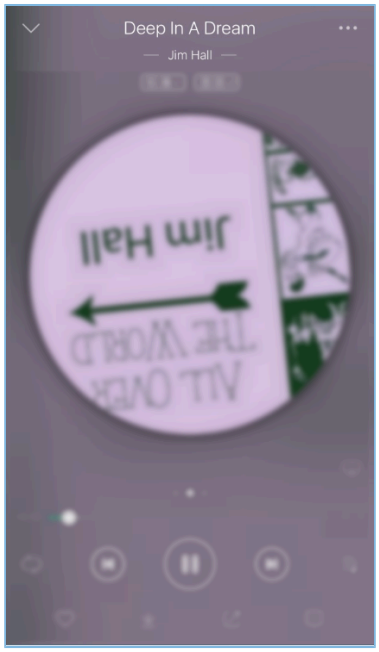

图 3-10 iOS设备上播放歌曲

在本示例中，GRUart的界面上显示的打印信息（如图 3-11所示），包括：

- 远程命令发送成功的信息。
- 当前已更新的歌曲名和演唱者信息（表征远程命令已经被执行）。

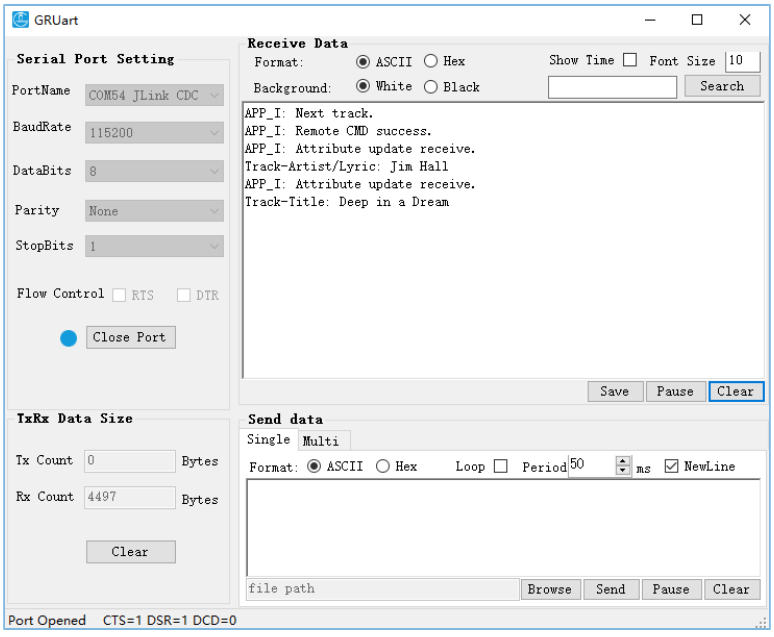

图 3-11 向iOS设备发控制命令的打印信息

通过图 3-11的打印信息，可确定GR5515 SK开发板和iOS设备之间的AMS服务运行正常。

说明:

GR5515 SK开发板按键的详细信息，请参考《GR5515 Starter Kit用户指南》“7 按键和指示灯”。

表 3-5 按键与命令对应关系说明

按键操作	对应命令
OK键（短按一次）	Play
OK键（短按两次）	Pause
OK键（长按一次）	Toggle play/pause
RIGHT键	Next track
LEFT键	Previous track
UP键	Volume up
DOWN键	Volume down

4 应用详解

本章主要将介绍AMS Client示例的交互流程及相关代码。

4.1 工程目录

AMS Client示例的源代码和工程文件位于：`SDK_Folder\projects\ble\ble_peripheral\ble_app_ams_c`，其中工程文件在Keil_5文件夹下。

双击打开*ble_app_ams_c.uvprojx*工程文件，在Keil中查看AMS Client示例ble_app_ams_c工程目录结构，相关文件介绍如表 4-1 所示。

表 4-1 ble_app_ams_c文件说明

Group	文件	描述
gr_profiles	ams_c.c	AMS相关GATT操作
user_callback	user_gap_callback.c	GAP Callback实现，如连接、断连、GAP参数更新等
	user_sm_callback.c	SM Callback实现，如配对绑定
user_platform	user_periph_setup.c	App Log、设备地址和电源管理模式的配置
user_app	main.c	main()入口函数
	user_app.c	实现AMS Client广播参数设置与事件处理
	user_arms_decode.c	实现对端输入数据的解码与输出

4.2 交互流程及代码介绍

本例程以AMS Client交互为例，分步讲解了AMS Client配对与绑定、发现服务、CCCD（Client Characteristic Configuration Descriptor）使能、通知消息处理和读写交互方式，便于用户深入了解各模块的主要代码。

AMS Client示例交互流程如图 4-1所示。

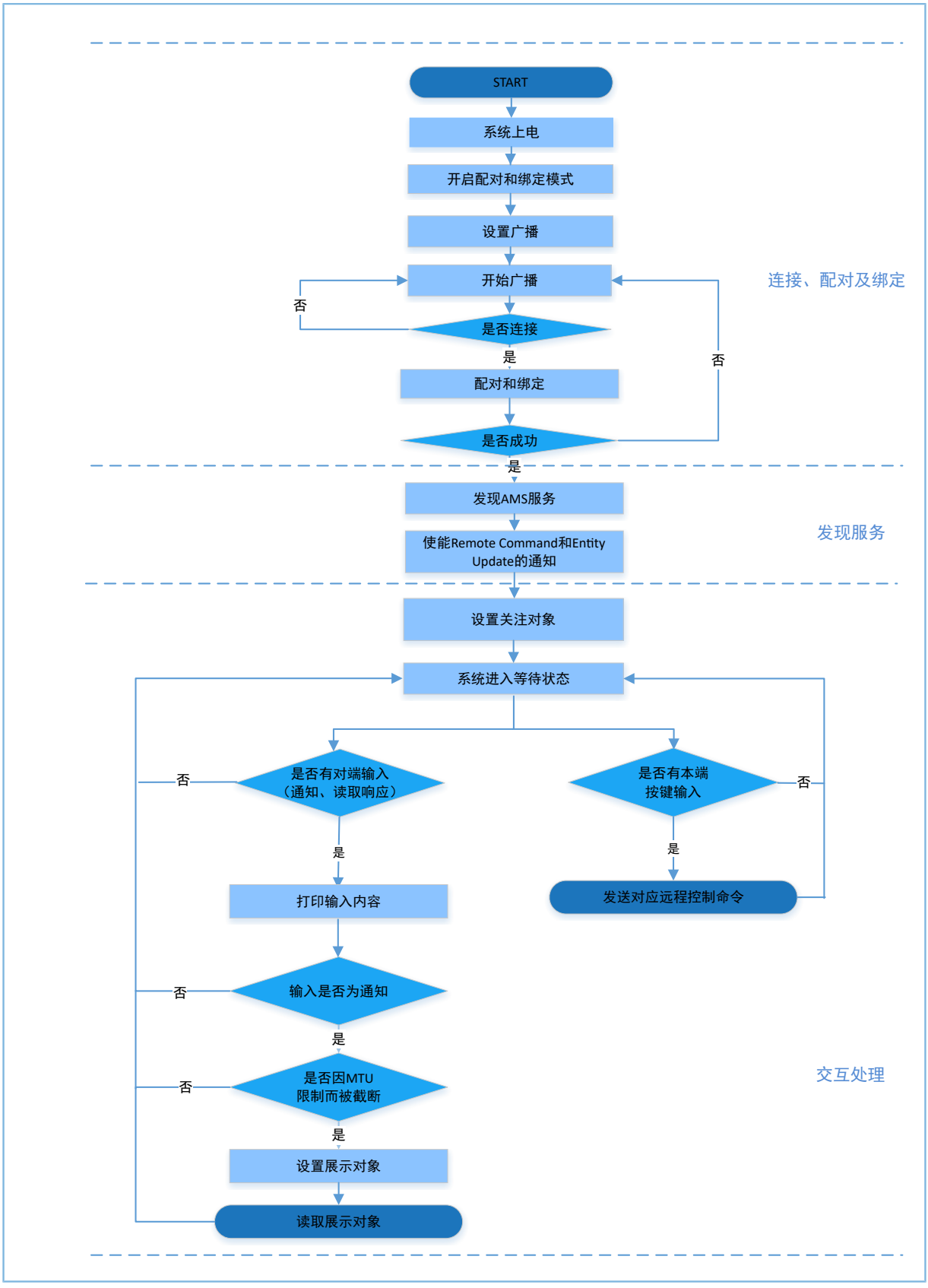

图 4-1 AMS交互流程图

4.2.1 连接、配对及绑定

搭载了GR5515芯片的设备（以下称为GR5515蓝牙设备）上电后，作为Peripheral角色进行广播。iOS设备端作为Central角色，可搜索周边蓝牙设备。

iOS设备打开蓝牙功能后，扫描到GR5515蓝牙设备并发起连接进行配对绑定。主要代码介绍如下。

路径：工程目录下的`user_app/user_app.c`

名称：gap_params_init();

gap_params_init()函数用于使能配对和绑定模式，设置当前安全级别（默认为Security Mode Level 3，用户可自定义）和广播参数。

说明:

关于LE Security Modes Level设置的详细信息，请参考Bluetooth Core Spec v5.1中的“10.2 LE security modes（Vol 3，Part C）”。

```
static void gap_params_init(void)
{
    sdk_err_t   error_code;

    ble_gap_pair_enable(true);

    error_code = ble_gap_privacy_params_set(150, true);
    APP_ERROR_CHECK(error_code);

    sec_param_t sec_param =
    {
        .level     = SEC_MODE1_LEVEL3,
        .io_cap    = IO_DISPLAY_ONLY,
        .oob       = false,
        .auth      = AUTH_BOND | AUTH_MITM | AUTH_SEC_CON,
        .key_size  = 16,
        .ikey_dist = KDIST_ENCKEY | KDIST_IDKEY,
        .rkey_dist = KDIST_ENCKEY | KDIST_IDKEY,
   };

    error_code = ble_sec_params_set(&sec_param);
    APP_ERROR_CHECK(error_code);

    s_gap_adv_param.adv_intv_max = APP_ADV_MAX_INTERVAL;
    s_gap_adv_param.adv_intv_min = APP_ADV_MIN_INTERVAL;
    s_gap_adv_param.adv_mode     = GAP_ADV_TYPE_ADV_IND;
    s_gap_adv_param.chnl_map     = GAP_ADV_CHANNEL_37_38_39;
    s_gap_adv_param.disc_mode    = GAP_DISC_MODE_NON_DISCOVERABLE;
    s_gap_adv_param.filter_pol   = GAP_ADV_ALLOW_SCAN_ANY_CON_ANY;

    error_code = ble_gap_device_name_set(BLE_GAP_WRITE_PERM_DISABLE,
                                         DEVICE_NAME, strlen(DEVICE_NAME));
```

```
    APP_ERROR_CHECK(error_code);

    error_code = ble_gap_adv_param_set(0, BLE_GAP_OWN_ADDR_STATIC, &s_gap_adv_param);
    APP_ERROR_CHECK(error_code);

    error_code = ble_gap_adv_data_set(0, BLE_GAP_ADV_DATA_TYPE_DATA,
                                      s_adv_data_set, sizeof(s_adv_data_set));
    APP_ERROR_CHECK(error_code);

    error_code = ble_gap_adv_data_set(0, BLE_GAP_ADV_DATA_TYPE_SCAN_RSP,
                                      s_adv_rsp_data_set, sizeof(s_adv_rsp_data_set));
    APP_ERROR_CHECK(error_code);

    s_gap_adv_time_param.duration    = 0;
    s_gap_adv_time_param.max_adv_evt = 0;
}
```

路径：工程目录下的user_callback/user_sm_callback.c

名称：app_sec_rcv_enc_req_cb();

在该回调函数中设置自定义配对密码“123456”并赋值到cfm_enc.data.tk.key[4]数组，代码如下面所示。

```
static void app_sec_rcv_enc_req_cb(uint8_t conn_idx, sec_enc_req_t *p_enc_req)
{
    ...
    switch (p_enc_req->req_type)
    {
      ...
        // user need to input the password
        case TK_REQ:
            cfm_enc.req_type = TK_REQ;
            cfm_enc.accept = true;
            tk = 123456;
            memset(cfm_enc.data.tk.key, 0, 16);
            cfm_enc.data.tk.key[0] = (uint8_t)((tk & 0x000000FF) >> 0);
            cfm_enc.data.tk.key[1] = (uint8_t)((tk & 0x0000FF00) >> 8);
            cfm_enc.data.tk.key[2] = (uint8_t)((tk & 0x00FF0000) >> 16);
            cfm_enc.data.tk.key[3] = (uint8_t)((tk & 0xFF000000) >> 24);
            break;
        default:
            break;
    }
    ble_sec_enc_cfm(conn_idx, &cfm_enc);
}
```

路径：工程目录下的user_callback/user_sm_callback.c

名称：app_sec_rcv_enc_ind_cb();

该函数在GR5515蓝牙设备和iOS设备配对成功后执行，用于通知系统配对完毕。如果配对成功，系统将直接调用ams_c_disc_srvc_start()函数发现服务。

```
static void app_sec_rcv_enc_ind_cb(uint8_t conn_idx, sec_enc_ind_t enc_ind, uint8_t auth)
{
    sdk_err_t error_code;
    if (ENC_SUCCESS != enc_ind)
    {
        return;
    }
    APP_LOG_INFO("Encrypt success.");

    error_code = ams_c_disc_srvc_start(conn_idx);
    APP_ERROR_CHECK(error_code);
}
```

4.2.2 发现服务

GR5515蓝牙设备在建立连接并配对成功后，发起搜索iOS设备AMS服务请求。

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_disc_srvc_start();

该函数按照指定的UUID查找/发现目标设备上对应的Service。查询结果将通过回调的方式返回（回调函数为*ams_c.c*中的ams_c_srvc_browse_cb()函数）。

```
sdk_err_t ams_c_disc_srvc_start(uint8_t conn_idx)
{
    ble_uuid_t ble_ams_uuid =
    {
        .uuid_len = BLE_ATT_UUID_128_LEN,
        .uuid     = (uint8_t *)ams_service_uuid,
    };
    return ble_gattc_prf_services_browse(s_ams_c_env.prf_id, conn_idx, &ble_ams_uuid);
}
```

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_srvc_browse_cb();

该函数用于获取iOS设备AMS的服务，通过UUID枚举服务中每一个属性对应的Handle。

```
static void ams_c_srvc_browse_cb(uint8_t conn_idx, uint8_t status, const
                                 ble_gattc_browse_srvc_t *p_browse_srvc)
{
    ams_c_evt_t  ams_c_evt;
    uint16_t     handle_disc;

    ams_c_evt.conn_idx = conn_idx;
    ams_c_evt.evt_type = AMS_C_EVT_DISCOVERY_FAIL;

    if(BLE_GATT_ERR_BROWSE_NO_ANY_MORE == status)
    {
        return;
```

```
    }
    if (BLE_SUCCESS == status)
    {
        if (0 == memcmp(p_browse_srvc->uuid, ams_service_uuid, BLE_ATT_UUID_128_LEN))
        {
            s_ams_c_env.handles.ams_srvc_start_handle = p_browse_srvc->start_hdl;
            s_ams_c_env.handles.ams_srvc_end_handle   = p_browse_srvc->end_hdl;
            for (uint32_t i = 0; i < p_browse_srvc->end_hdl -
                 p_browse_srvc->start_hdl; i++)
            {
                if (BLE_GATTC_BROWSE_ATTR_VAL == p_browse_srvc->info[i].attr_type)
                {
                    handle_disc = p_browse_srvc->start_hdl + i + 1;
                   if (0 == memcmp(p_browse_srvc->info[i].attr.uuid, ams_cmd_uuid,
                        BLE_ATT_UUID_128_LEN))
                    {
                        s_ams_c_env.handles.ams_cmd_handle = handle_disc;
                        s_ams_c_env.handles.ams_cmd_cccd_handle = handle_disc + 2;
                    }
                   else if (0 == memcmp(p_browse_srvc->info[i].attr.uuid,
                               ams_attr_update_uuid, BLE_ATT_UUID_128_LEN))

                    {
                        s_ams_c_env.handles.ams_attr_update_handle = handle_disc;
                        s_ams_c_env.handles.ams_attr_update_cccd_handle = handle_disc + 2;
                    }
                    else if (0 == memcmp(p_browse_srvc->info[i].attr.uuid,
                             ams_attr_display_uuid, BLE_ATT_UUID_128_LEN))
                    {
                        s_ams_c_env.handles.ams_attr_display_handle = handle_disc;
                    }
                }
                else if (p_browse_srvc->info[i].attr_type == BLE_GATTC_BROWSE_NONE)
                {
                    break;
                }
            }
            ams_c_evt.evt_type = AMS_C_EVT_DISCOVERY_CPLT;
        }
    }
    ams_c_evt_handler_excute(&ams_c_evt);
}
```

在完成服务发现后，应依次使能各特征的通知功能，具体方法请参考4.2.3 通知事件及解析。

4.2.3 通知事件及解析

特征Remote Command和Entity Update支持通知功能。若需要发送通知，应设置其特征描述符的值（即CCCD）来使能通知，特征描述符的Handle已经在服务发现中得到。主要代码介绍如下。

路径1：工程目录下的gr_profiles/ams_c.c

名称1：ams_c_cmd_notify_set();

路径2：工程目录下的gr_profiles/ams_c.c

名称2：ams_c_attr_update_notify_set();

ams_c_cmd_notify_set()函数代码如下，用于使能特征Remote Command的通知。ams_c_attr_update_notify_set()函数可参考ams_c_cmd_notify_set()，用于使能Entity Update的通知。

```
sdk_err_t ams_c_cmd_notify_set(uint8_t conn_idx, bool is_enable)
{
    gattc_write_attr_value_t write_attr_value;
    uint16_t ntf_value = is_enable ? PRF_CLI_START_NTF : PRF_CLI_STOP_NTFIND;

    if (BLE_ATT_INVALID_HDL == s_ams_c_env.handles.ams_cmd_cccd_handle)
    {
        return BLE_ATT_ERR_INVALID_HANDLE;
    }
    write_attr_value.handle  = s_ams_c_env.handles.ams_cmd_cccd_handle;
    write_attr_value.offset  = 0;
    write_attr_value.length  = 2;
    write_attr_value.p_value = (uint8_t *)&ntf_value;

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_att_ntf_ind_cb();

该函数以对应的数据结构接收Remote Command和Entity Update的通知信息。

```
static void ams_c_att_ntf_ind_cb(uint8_t conn_idx, const ble_gattc_ntf_ind_t *p_ntf_ind)
{
    ams_c_evt_t ams_c_evt;
    ams_c_evt.conn_idx = conn_idx;
    ams_c_evt.evt_type = AMS_C_EVT_INVALID;
    if (p_ntf_ind->handle == s_ams_c_env.handles.ams_cmd_handle)
    {
        ams_c_evt.evt_type = AMS_C_EVT_CMD_UPDATE_RECEIVE;
        ams_c_evt.param.cmd_list.p_cmd = p_ntf_ind->p_value;
        ams_c_evt.param.cmd_list.length = p_ntf_ind->length;
    }
    else if (p_ntf_ind->handle == s_ams_c_env.handles.ams_attr_update_handle)
    {
        ams_c_evt.evt_type = AMS_C_EVT_ATTR_UPDATE_RECEIVE;
        ams_c_attr_info_decode(p_ntf_ind->p_value, p_ntf_ind->length,
                               &ams_c_evt.param.attr_info);
    }
    ams_c_evt_handler_excute(&ams_c_evt);
}
```

4.2.4 读写交互

通过GR5515蓝牙设备向AMS服务各特征写入或读取信息，可以实现远程交互。以下将详细介绍AMS支持的读写交互。

4.2.4.1 远程控制

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_cmd_send();

该函数用于向特征Remote Command写入命令ID来实现远程控制。

```
sdk_err_t ams_c_cmd_send(uint8_t conn_idx, uint8_t cmd_id)
{
    ...
    gattc_write_attr_value_t write_attr_value;
    write_attr_value.handle = s_ams_c_env.handles.ams_cmd_handle;
    write_attr_value.offset = 0;
    write_attr_value.length = 1;
    write_attr_value.p_value = (uint8_t *)&cmd_id;

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

路径：工程目录下的gr_profiles/ams_c.h

名称：ams_c_cmd_id_t;

该结构体定义了各项远程命令ID的枚举值。

```
typedef enum
{
    AMS_CMD_ID_PLAY,                        /**< Command index of play. */
    AMS_CMD_ID_PAUSE,                       /**< Command index of pause. */
    AMS_CMD_ID_TOGGLE_PLAY_PAUSE,           /**< Command index of toggle. */
    AMS_CMD_ID_NEXT_TRACK,                  /**< Command index of next track. */
    AMS_CMD_ID_PREVIOUS_TRACK,              /**< Command index of previous track. */
    ...
} ams_c_cm
```

4.2.4.2 设置关注对象

路径：工程目录下的user_app/user_app.c

名称：attr_focus_set();

该函数向特征Entity Update写入对象ID来设置关注对象（对象ID包括Entity ID和Attribute ID两部分，如需了解对象ID详情请参考Apple Media Service(AMS) Reference）。关注对象设置完成后，当该对象发生变化时，Entity Update就会发送通知。

该函数定义了结构体作为关注对象ID的容器，再调用ams_c_attr_focus_set()来完成对特征Entity Attribute的写入。用户可根据需要来设置关注对象。

```
static void attr_focus_set(uint8_t conn_idx)
{
    sdk_err_t   error_code;
    ams_c_ett_attr_id_t track_attr_id =
    {
        .ett_id     = AMS_TRACK_ID,
        .attr_id    = {AMS_TRACK_ARTIST_ID, AMS_TRACK_TITTLE_ID},
        .attr_count = 2
    };
    error_code = ams_c_attr_focus_set(conn_idx, &track_attr_id);
    APP_ERROR_CHECK(error_code);
}
```

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_attr_focus_set();

该函数用于实现对特征Entity Attribute的写入。

```
sdk_err_t ams_c_attr_focus_set(uint8_t conn_idx, const ams_c_ett_attr_id_t *p_ett_attr_id)
{
    ...
    gattc_write_attr_value_t write_attr_value;
    write_attr_value.handle = s_ams_c_env.handles.ams_attr_update_handle;
    write_attr_value.offset = 0;
    write_attr_value.length = p_ett_attr_id->attr_count + 1;
    write_attr_value.p_value = (uint8_t *)&(p_ett_attr_id->ett_id);

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

4.2.4.3 设置展示对象

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_attr_display_set();

由于MTU的限制，特征Entity Update通知的对象数据可能是被截断的。要获得完整的对象数据，需要使用特征Entity Attribute。

该函数用于向特征Entity Attribute写入对象ID来设置展示对象。成功写入后，可通过读取特征Entity Attribute的值来获得该对象的完整数据。

```
sdk_err_t ams_c_attr_display_set(uint8_t conn_idx, const ams_c_attr_info_t *p_attr_info)
{
    ...
    gattc_write_attr_value_t write_attr_value;
    write_attr_value.handle = s_ams_c_env.handles.ams_attr_display_handle;
    write_attr_value.offset = 0;
    write_attr_value.length = 2;
    write_attr_value.p_value = (uint8_t *)&(p_attr_info->ett_id);
```

```
    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

4.2.4.4 读取展示对象

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_cplt_attr_read();

该函数用于在设置展示对象完成后，读取展示对象，得到该对象的完整值。

```
sdk_err_t ams_c_cplt_attr_read(uint8_t conn_idx)
{
    if (BLE_ATT_INVALID_HDL == s_ams_c_env.handles.ams_attr_display_handle)
    {
        return BLE_ATT_ERR_INVALID_HANDLE;
    }
    return ble_gattc_prf_read(s_ams_c_env.prf_id, conn_idx,
                              s_ams_c_env.handles.ams_attr_display_handle, 0);
}
```

路径：工程目录下的gr_profiles/ams_c.c

名称：ams_c_att_read_cb();

该函数以对应的数据结构接收读取结果。

```
static void ams_c_att_read_cb(uint8_t conn_idx, uint8_t status,
                              const ble_gattc_read_rsp_t *p_read_rsp)
{
    ...
    if (p_read_rsp->vals[0].handle == s_ams_c_env.handles.ams_attr_display_handle)
    {
        ams_c_evt.conn_idx = conn_idx;
        ams_c_evt.evt_type = AMS_C_EVT_CPLT_ATTR_READ_RSP;
        ams_c_evt.param.cplt_attr_data.p_data = p_read_rsp->vals[0].p_value;
        ams_c_evt.param.cplt_attr_data.length = p_read_rsp->vals[0].length;
    }

    ams_c_evt_handler_excute(&ams_c_evt);
}
```

5 常见问题

本章描述了在验证及应用AMS Client示例时，可能出现的问题、原因及处理方法。在以下描述中，手机指的是iOS设备。

5.1 串口终端没有输出信息

- 问题描述

  串口终端GRUart没有打印信息，或者打印乱码。

- 问题分析

  固件没有正确烧录到开发板运行，或者串口波特率设置不正确导致信息不能正常打印在终端。

- 处理方法

  1. 请确认GRUart的串口波特率已设置为115200，数据位为8，停止位为1，无校验位，无流控。同时检查串口线接入是否正确。
  2. 如果串口正常，请重新烧录*ble_app_ams_c_fw.bin*固件，并确认工程代码没有被修改，使用GProgrammer下载固件。

5.2 手机搜索不到蓝牙广播

- 问题描述

  打开手机App后，无法搜索到以Goodix_AMS_C为名称的广播。

- 问题分析

  蓝牙天线连接或固件异常。

- 处理方法

  1. 请确定使用的手机是否为iOS设备，手机蓝牙功能是否打开。如果都符合，则可能存在硬件问题，请检查GR5515蓝牙设备天线是否正常工作。
  2. 如果以上均为正常，用户可通过下载模板固件*ble_app_template_fw.bin*来检查硬件是否异常。模板固件位于：`SDK_Folder\projects\ble\ble_peripheral\ble_app_template\build\`。运行该固件后，若硬件正常，则能够搜索到以Goodix_Tem为名称的广播，否则可能存在硬件异常。

5.3 手机无法连接AMS Client设备蓝牙广播

- 问题描述

  名称为Goodix_AMS_C的广播，出现在蓝牙的“我的设备”列表中，但无法连接。

- 问题分析

  手机曾经连接并绑定过GR5515蓝牙设备，但该设备上的数据可能经过擦除和重新烧写，已丢失了绑定信息，使得绑定失效。

- 处理方法

  1. 在蓝牙的“我的设备”列表中，选择“Goodix_AMS_C”设备，再选择忽略此设备。
  2. 重新进行扫描、配对和绑定流程。