应用及自定义GR551x Sample Service

版本： **1.7**

发布日期： **2020-12-15**

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2层、13层

电话：+86-755-33338828　　传真：+86-755-33338099

网址：www.goodix.com

前言

编写目的

本文档介绍了如何使用和修改GR551x SDK中的Sample Service，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
- 开发爱好者
- 文档工程师

版本说明

本文档为第5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	优化了“3.3 添加Sample Service至新工程”章节描述
1.5	2020-05-30	更新了“3 应用Goodix Sample Service”章节中工程目录相关图片
1.6	2020-06-30	基于SDK刷新版本
1.7	2020-12-15	更新GRToolbox软件界面截图

目录

1 简介

为了保持各类蓝牙设备之间的兼容性，Bluetooth SIG（Bluetooth Special Interest Group，蓝牙技术联盟）在蓝牙涉及的领域中定义了一系列通用的标准Service。

利用这些标准Service，各类蓝牙设备能轻松控制对端蓝牙设备或获取相关信息。

然而，在某些情况下，有必要实现您自己的Service。例如，您的应用程序可能需要一些新的功能，而标准Service并不符合这些功能。

本文档将主要介绍如何应用以及修改Goodix Sample Service。

在进行操作前，建议参考如表 1-1 所示文档。

表 1-1 文档参考

名称	描述
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/
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 Sample Service概述

GR551x官方提供参考版Sample Service，用于Master与Slave端的基本Write和Notify通信。信息交互处理流程图如下所示。

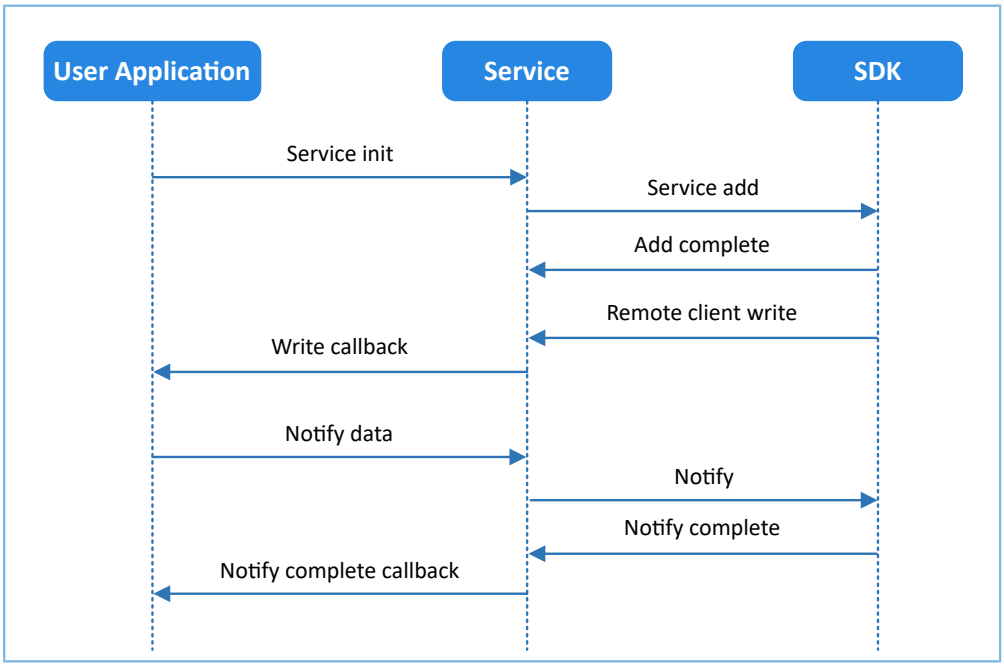

图 2-1 Service信息交互处理流程图

Sample Service提供RX和TX两个特征，前者的Property为Write，后者的Property为Notify。具体特征对比如表 2-1 所示。

表 2-1 Sample Service特征表

Description	UUID	Properties
Service	A6ED0101-D344-460A-8075-B9E8EC90D71B	-
RX Characteristic	A6ED0102-D344-460A-8075-B9E8EC90D71B	Write without Response
TX Characteristic	A6ED0103-D344-460A-8075-B9E8EC90D71B	Notify

3 应用Goodix Sample Service

本章将介绍如何在Keil中创建Goodix Sample Service新工程。

说明:

SDK_Folder为GR551x SDK的根目录。

3.1 准备工作

应用Sample Service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

- 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/
LightBlue（iOS）	iOS BLE调试工具，可通过App Store下载
GRToolbox（Android）	GR551x BLE调试工具，位于SDK_Folder\tools\GRToolbox

3.2 创建一个基于Template的新工程

进入SDK_Folder\projects\ble\ble_peripheral目录，拷贝ble_app_template目录到当前目录，将目录名和Keil目录中的工程名重命名为ble_app_template_mine，然后在Keil中打开该工程。如图 3-1所示。

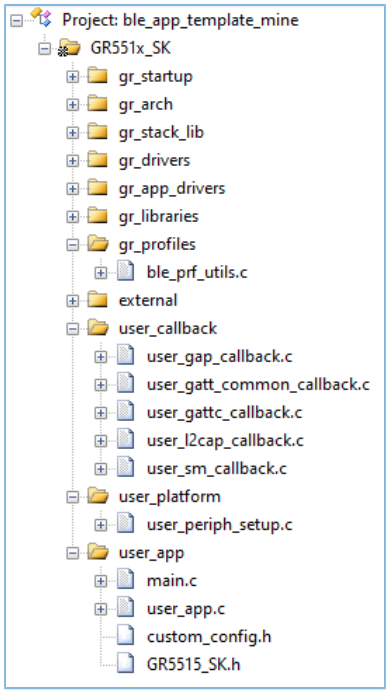

图 3-1 ble_app_template_mine工程结构目录

相关文件说明如表 3-3 所示。

表 3-3 ble_app_template_mine文件说明

Group	描述
gr_profiles	Profile源文件添加
user_callback	用户定义的BLE回调函数
user_platform	用户外设初始化
user_app	用户应用逻辑实现

3.3 添加Sample Service至新工程

拷贝SDK_Folder\components\profiles\sample目录到SDK_Folder\projects\ble\ble_peripheral\ble_app_template_mine\Src目录下，并按照如下步骤，将拷贝的sample目录中的*sample_service.c*文件添加到新工程中，并编译通过。

1. 添加*sample_service.c*到工程中，具体操作如下。

   (1) 点击Keil工具栏中的“Options for Target” 图标。

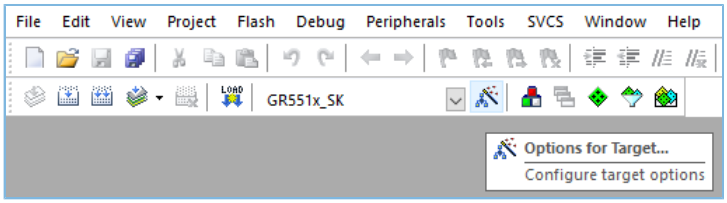

图 3-2 点击“Options for Target”图标

(2) 弹出“Options for Target GR551x_SK”窗口，选择“C/C++”标签页，在“Include Paths”栏中点击“Include Path”对应的 … 路径浏览按钮。

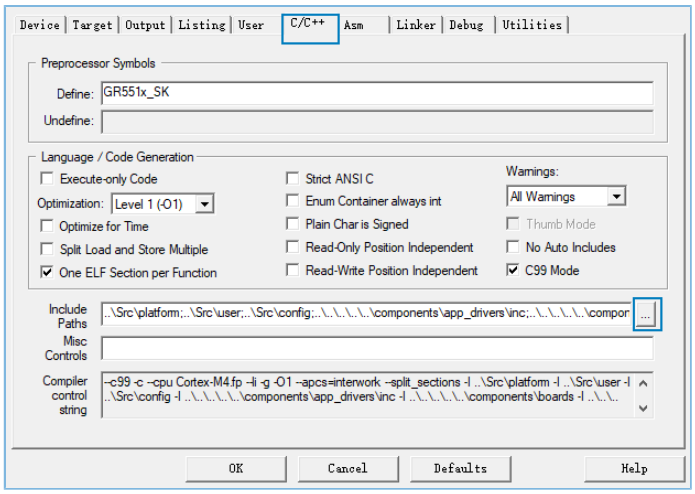

图 3-3 点击路径浏览按钮

(3) 在“Folder Setup”窗口拖动滚动条到Include路径列表底部，输入Sample Service的路径（..\Src\sample）；或双击底部任一空白路径，点击对应的 … 按钮，浏览Sample Service路径（..\Src\sample）并选择确定。

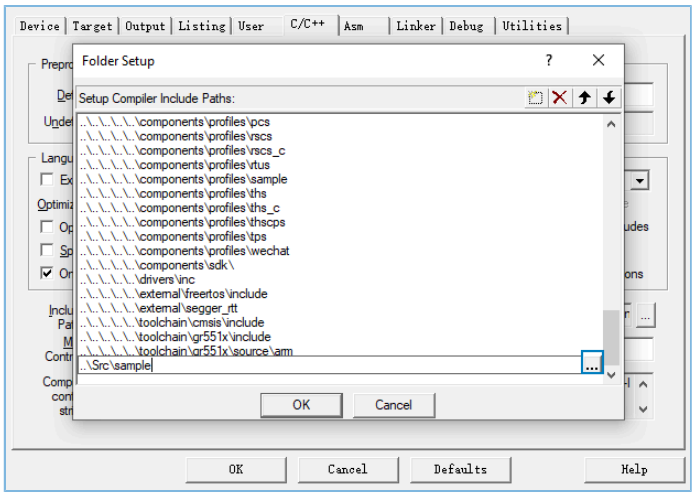

图 3-4 通过输入Sample Service路径选择

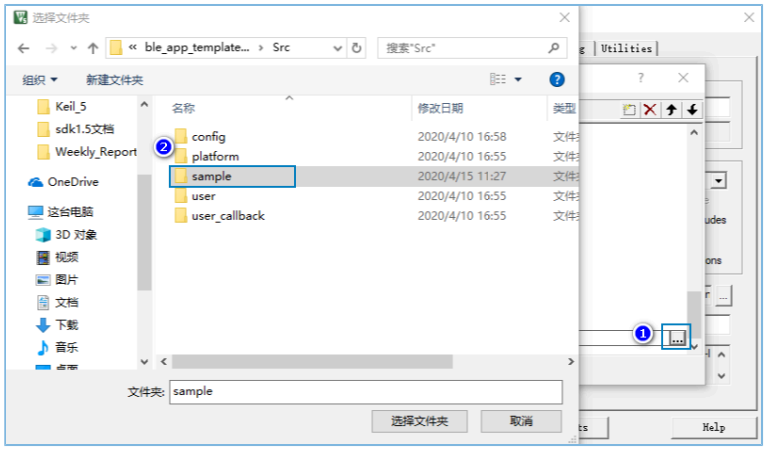

图 3-5 通过浏览Sample Service路径选择

2. 添加Sample Service源文件*sample_service.c*至Keil工程。在工程目录中选中gr_profiles，单击鼠标右键，选择“Add Existing Files to Groups 'gr_profiles' ”，选中`Src\Sample`目录下*sample_service.c*文件，点击“Add”，将其添加到gr_profiles文件夹中。

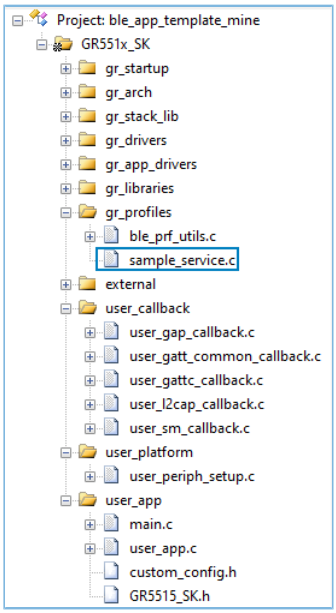

图 3-6 添加Sample Service源文件至Keil工程

3.4 应用Sample Service

本节将介绍如何初始化、测试及验证新工程（ble_app_template_mine）中的Sample Service。

3.4.1 初始化Sample Service

在*user_app.c*中的具体操作步骤如下。

- 添加Sample Service头文件。

```
#include "user_app.h"
#include "gr55xx_sys.h"
```

```
#include "app_log.h"
#include "app_error.h"
#include "sample_service.h"
```

- 实现Sample Service回调处理函数。

```
static void sample_envt_process(samples_evt_t *p_evt)
{
    switch(p_evt->evt_type)
    {
        case SAMPLES_EVT_TX_NOTIFICATION_ENABLED:
            break;
        case SAMPLES_EVT_TX_NOTIFICATION_DISABLED:
            break;
        case SAMPLES_EVT_RX_RECEIVE_DATA:
            break;
        case SAMPLES_EVT_TX_NOTIFY_COMPLETE:
            break;
        default:break;
    }
}
```

- 在services_init()函数中初始化环境变量，并调用接口添加Service。

```
static void services_init(void)
{
    samples_init_t sample_init[1];
    sample_init[0].evt_handler = sample_envt_process;
    samples_service_init(sample_init, 1);
}
```

3.4.2 测试及验证

完成以上操作后，编译工程并将目标文件下载至开发板，按照如下步骤测试和验证Sample Service。

1. 使用GRToolbox扫描设备，将发现广播名为“Goodix_Tem”的设备，如图 3-7所示。

说明:

iOS设备可采用LightBlue工具进行测试和验证。

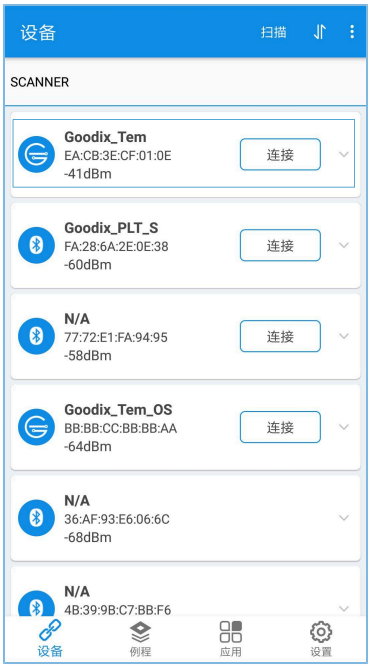

图 3-7 发现Goodix_Tem设备

2. 点击“Goodix_Tem”选项“连接”，查找Sample Service以及其两个特征是否在服务中。该特征的对比数据请参考表 2-1，若比对数值正确，则表明Sample Service应用成功。

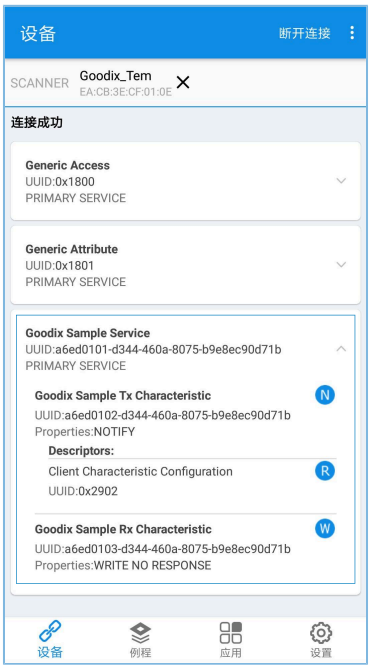

图 3-8 Sample Service应用成功

4 创建自定义Service

在某些应用情况下，标准Service无法满足需求。本章将基于Sample Service增加一个新特征（该特征具有Notify和Write without Response属性）为例，介绍如何应用自定义Service。

说明:

Sample Service修改后的Service，称为自定义Service。

在本章节中加粗的代码表示添加的新特征代码，未加粗代码为原有代码。

4.1 添加新特征

请按照如下步骤添加新特征。

1. 添加新特征UUID（在*sample_service.c*中添加）。

```
/**@brief The UUIDs of GUS characteristics. */
#define SAMPLE_SERVER_TX_UUID      {0x1B, 0xD7, 0x90, 0xEC, 0xE8, 0xB9, 0x75,
   0x80, 0x0A, 0x46, 0x44, 0xD3, 0x02, 0x01, 0xED, 0xA6}
#define SAMPLE_SERVER_RX_UUID      {0x1B, 0xD7, 0x90, 0xEC, 0xE8, 0xB9, 0x75,
   0x80, 0x0A, 0x46, 0x44, 0xD3, 0x03, 0x01, 0xED, 0xA6}
#define SAMPLE_SERVER_ADD_UUID     {0x1B, 0xD7, 0x90, 0xEC, 0xE8, 0xB9, 0x75,
   0x80, 0x0A, 0x46, 0x44, 0xD3, 0x04, 0x01, 0xED, 0xA6}
```

2. 在环境变量结构体中添加新特征Notify config变量（在*sample_service.c*中添加）。

```
/**@brief The UUIDs of GUS characteristics. */
/**@brief Samples Service environment variable. */
typedef struct
{
    /**< Sample Service initialization variables. */
    samples_init_t samples_init;
    /**< Service start handle. */
    uint16_t start_hdl;
    /**< TX Character Notification configuration of peer devices. */
    uint16_t tx_ntf_cfg[SAMPLES_CONNECTION_MAX];
    /**< ADD Character Notification configuration of peer devices. */
    uint16_t add_ntf_cfg[SAMPLES_CONNECTION_MAX];
} samples_env_t;
```

3. 添加新特征索引枚举（在*sample_service.c*中添加）。

```
/**@brief Sample Service Attributes Indexes. */
enum samples_attr_idx_t
{
    SAMPLES_IDX_SVC,

    SAMPLES_IDX_TX_CHAR,
    SAMPLES_IDX_TX_VAL,
    SAMPLES_IDX_TX_CFG,
    SAMPLES_IDX_RX_CHAR,
```

```
    SAMPLES_IDX_RX_VAL,
    SAMPLES_IDX_ADD_CHAR,
    SAMPLES_IDX_ADD_VAL,
    SAMPLES_IDX_ADD_CFG,

    SAMPLES_IDX_NB,
};
```

4. 添加新特征应用层返回事件类型（在*sample_service.h*中添加）。

```
/**@brief Sample Service event type. */
typedef enum
{
    SAMPLES_EVT_INVALID,
    SAMPLES_EVT_TX_NOTIFICATION_ENABLED,
    SAMPLES_EVT_TX_NOTIFICATION_DISABLED,
    SAMPLES_EVT_RX_RECEIVE_DATA,
    SAMPLES_EVT_TX_NOTIFY_COMPLETE,
    SAMPLES_EVT_ADD_NOTIFICATION_ENABLED,
    SAMPLES_EVT_ADD_NOTIFICATION_DISABLED,
    SAMPLES_EVT_ADD_RECEIVE_DATA,
    SAMPLES_EVT_ADD_NOTIFY_COMPLETE,
} samples_evt_type_t
```

5. 定义新特征属性描述（在*sample_service.c*中添加）。

   完成后，将生成新特征显示于“Goodix_Tem”连接界面中，参考图 4-2。

```
/**@brief Full SAMPLES Database Description - Used to add attributes into the database. */
static const attm_desc_128_t samples_att_db[SAMPLES_IDX_NB] =
{
    ...
    [SAMPLES_IDX_RX_VAL]    = {SAMPLE_SERVER_RX_UUID,
                               WRITE_CMD_PERM_UNSEC,
                               (ATT_VAL_LOC_USER |
                               ATT_UUID_TYPE_SET(UUID_TYPE_128)),
                               SAMPLES_MAX_DATA_LEN},

    //SAMPLE ADD Characteristic Declaration
    [SAMPLES_IDX_ADD_CHAR] = {ATT_128_CHARACTERISTIC, READ_PERM_UNSEC, 0, 0},
    //SAMPLE ADD Characteristic Value
    [SAMPLES_IDX_ADD_VAL]   = {SAMPLE_SERVER_ADD_UUID,
                               NOTIFY_PERM_UNSEC | WRITE_CMD_PERM_UNSEC,
                               (ATT_VAL_LOC_USER| ATT_UUID_TYPE_SET(UUID_TYPE_128)),
                               SAMPLES_MAX_DATA_LEN},
     //SAMPLE ADD Characteristic - Client Characteristic Configuration Descriptor
     [SAMPLES_IDX_ADD_CFG]   = {ATT_128_CLIENT_CHAR_CFG,
                                READ_PERM_UNSEC | WRITE_REQ_PERM_UNSEC,
                                0,
```

```
                                    0},
};
```

4.2 读写新特征

1. 在samples_read_att_cb()函数中添加读新特征功能。

```
static void samples_read_att_cb(uint8_t conn_idx, const gatts_read_req_cb_t *p_param)
{

    ...
    switch (tab_index)
    {
        case SAMPLES_IDX_TX_CFG:
            cfm.length = sizeof(uint16_t);
            cfm.value = (uint8_t *)(&s_samples_env[i].tx_ntf_cfg[conn_idx]);
            break;

        case SAMPLES_IDX_ADD_CFG:
            cfm.length = sizeof(uint16_t);
            cfm.value = (uint8_t *)(&s_samples_env[i].add_ntf_cfg[conn_idx]);
            break;

        default:
            break;
    }

    ble_gatts_read_cfm(conn_idx, &cfm);
}
```

2. 在samples_write_att_cb()函数中添加写新特征功能。

```
static void samples_write_att_cb(uint8_t conn_idx, const gatts_write_req_cb_t *p_param)
{
    ...
    switch (tab_index)
    {
        case SAMPLES_IDX_RX_VAL:
            event.conn_idx = conn_idx;
            event.evt_type = SAMPLES_EVT_RX_RECEIVE_DATA;
            break;
        case SAMPLES_IDX_TX_CFG:
            cccd_value = le16toh(&p_param->value[0]);
            event.conn_idx = conn_idx;
            event.evt_type = (PRF_CLI_START_NTF == cccd_value) ?
                        SAMPLES_EVT_TX_NOTIFICATION_ENABLED :
                        SAMPLES_EVT_TX_NOTIFICATION_DISABLED;
                        s_samples_env[i].tx_ntf_cfg[conn_idx]= cccd_value;
            break;
        case SAMPLES_IDX_ADD_VAL:
```

```
                event.conn_idx = conn_idx;
                event.evt_type = SAMPLES_EVT_ADD_RECEIVE_DATA;
            break;
        case SAMPLES_IDX_ADD_CFG:
            cccd_value = le16toh(&p_param->value[0]);
            event.conn_idx = conn_idx;
            event.evt_type = (PRF_CLI_START_NTF == cccd_value) ?
                             SAMPLES_EVT_ADD_NOTIFICATION_ENABLED :
                             SAMPLES_EVT_ADD_NOTIFICATION_DISABLED;
                             s_samples_env[i].add_ntf_cfg[conn_idx]= cccd_value;
          break;
       default:
           cfm.status = BLE_ATT_ERR_INVALID_HANDLE;
           break;
    }
    ...
}
```

3. 在samples_cccd_set_cb()函数中添加新增特征值CCCD设置。

```
static void samples_cccd_set_cb(uint8_t conn_idx, uint16_t handle, uint16_t cccd_value)
{
    ...
    switch (tab_index)
    {
        case SAMPLES_IDX_TX_CFG:
            event.conn_idx = conn_idx;
            event.evt_type = (PRF_CLI_START_NTF == cccd_value) ?    \
                        SAMPLES_EVT_TX_NOTIFICATION_ENABLED : \
                        SAMPLES_EVT_TX_NOTIFICATION_DISABLED;
                        s_samples_env[i].tx_ntf_cfg[conn_idx] = cccd_value;
      break;

      case SAMPLES_IDX_ADD_CFG:
            event.conn_idx = conn_idx;
            event.evt_type = (PRF_CLI_START_NTF == cccd_value) ?\
                         SAMPLES_EVT_ADD_NOTIFICATION_ENABLED :\
                         SAMPLES_EVT_ADD_NOTIFICATION_DISABLED;
                         s_samples_env[i].add_ntf_cfg[conn_idx]=cccd_value;
      break;
      default:
          break;
    }
    ...
}
```

4.3 添加新特征Notify函数

请参照以下代码，完成添加新特征Notify函数，并在*sample_service.h*中进行声明。

```
sdk_err_t samples_notify_add_data(uint8_t conn_idx, uint8_t ins_idx,
                                  uint8_t *p_data, uint16_t length)
{
     sdk_err_t  error_code = SDK_ERR_NTF_DISABLED;
     gatts_noti_ind_t send_cmd;

     if (PRF_CLI_START_NTF == s_samples_env[ins_idx].add_ntf_cfg[conn_idx])
     {
         if (ins_idx <= s_samples_ins_cnt)
         {
            // Fill in the parameter structure
             send_cmd.type   = BLE_GATT_NOTIFICATION;
             send_cmd.handle = prf_find_handle_by_idx(SAMPLES_IDX_ADD_VAL,
                               s_samples_env[ins_idx].start_hdl,
                               (uint8_t *)&s_samples_features);
             // pack measured value in database
             send_cmd.length       = length;
             send_cmd.value        = p_data;
             s_now_ins_cnt         = ins_idx;
             s_now_notify_cmp_type = SAMPLES_EVT_ADD_NOTIFY_COMPLETE;
             // send notification to peer device
             error_code = ble_gatts_noti_ind(conn_idx, &send_cmd);
         }
     }
     return error_code;
}
```

4.4 应用自定义Service

为了应用并验证修改后的Sample Service，可以设置开启一个1s的定时器，使十六进制的变量累加，并通过新加的特征（UUID: 0x1B, 0xD7, 0x90, 0xEC, 0xE8, 0xB9, 0x75, 0x80, 0x0A,0x46, 0x44, 0xD3, 0x04, 0x01, 0xED, 0xA6,Properties: Write Without Response/Notify），将值Notify给Master。

4.4.1 添加定时器

在*usr_app.c*中添加定时器，具体操作步骤如下。

1. 添加定时器头文件。

```
#include "user_app.h"
#include "gr55xx_sys.h"
#include "app_log.h"
#include "app_error.h"
#include "sample_service.h"
#include "app_timer.h"
```

2. 添加定义定时器ID变量和累加变量。

```
static app_timer_id_t s_add_timer_id;
static uint16_t s_add_count = 0;
```

3. 在ble_init_cmp_callback()中创建定时器。

```
void ble_init_cmp_callback(void)
{
    ...
    error_code = ble_gap_adv_start(0, &s_gap_adv_time_param);
    APP_ERROR_CHECK(error_code);

    APP_LOG_INFO("Template application example started.");

    app_timer_create(&s_add_timer_id, ATIMER_REPEAT, add_time_out_handler);
}
```

4. 实现Sample Service事件处理逻辑。

说明:

定时器的基础时间是1 ms，因此Timer值设置为1000，即1s。

```
static void sample_envt_process(samples_evt_t *p_evt)
{
    switch(p_evt->evt_type)
    {
        case SAMPLES_EVT_TX_NOTIFICATION_ENABLED:
            break;
        case SAMPLES_EVT_TX_NOTIFICATION_DISABLED:
            break;
        case SAMPLES_EVT_RX_RECEIVE_DATA:
            break;
        case SAMPLES_EVT_TX_NOTIFY_COMPLETE:
            break;
        case SAMPLES_EVT_ADD_NOTIFICATION_ENABLED:
            app_timer_start(s_add_timer_id, 1000, NULL);
            break;
        case SAMPLES_EVT_ADD_NOTIFICATION_DISABLED:
            app_timer_stop (s_add_timer_id);
            break;
        case SAMPLES_EVT_ADD_RECEIVE_DATA:
            break;
        case SAMPLES_EVT_ADD_NOTIFY_COMPLETE:
            break;

        default:break;
   }
}
```

5. 实现定时器超时处理函数。

```
static void add_time_out_handler(void *p_arg)
{
    s_add_count++;
```

```
    samples_notify_add_data(0,0,(uint8_t*)&s_add_count,2);
}
```

4.4.2 测试及验证

完成以上操作后，编译工程，将目标文件下载至开发板。测试及验证步骤如下。

1. 使用GRToolbox APP扫描设备，发现广播名为“Goodix_Tem”的设备。

图 4-1 发现Goodix_Tem设备

2. 点击“Goodix_Tem”选项的“连接”，查看Sample Service特征值中是否包含新添加的特征。

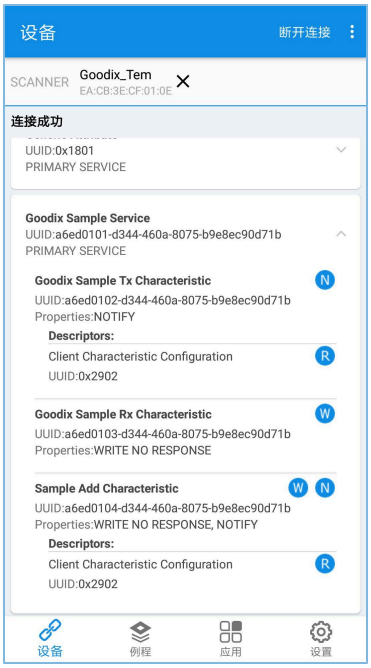

图 4-2 查看自定义Sample Service特征值

若该特征的UUID、属性与添加的新特征相同，说明Sample Service修改成功。

3. 点击“Sample Add Characteristic”右方 图标，在下拉界面中，可观察到Notify特征值每秒递增。

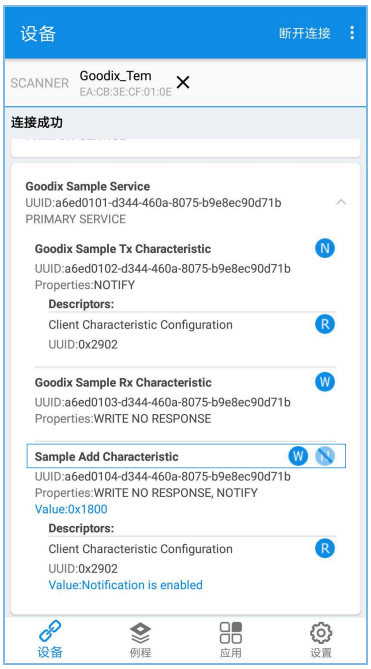

图 4-3 自定义Sample Service应用成功

由此表明，自定义Sample Service应用成功。