GR551x BLE Stack用户指南

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

本文档主要介绍了GR551x低功耗蓝牙（Bluetooth Low Energy，BLE）协议栈各层的基本功能，概念以及Application如何使用协议栈提供的API与协议栈进行交互等，旨在帮助开发者更好地使用BLE协议栈中各层的API。

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
- 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	更新了文档页脚版本时间
1.5	2020-05-30	删除“2.4.1.2 ”章节prop参数中的GAP_ADV_PROP_SCANNABLE_BIT不能被置位的描述，删除“3.2.3.1 ”章节初始化通用访问服务所涉及到的接口的描述
1.6	2020-06-30	基于SDK刷新版本

版权所有 © 2020 深圳市汇顶科技股份有限公司

目录

前言 I
1 简介 1
2 通用访问规范（GAP） 2
2.1 GAP角色 2
2.2 GAP访问模式和设备流程 2
2.3 连接 3
2.3.1 连接事件 3
2.3.2 连接参数介绍 4
2.3.3 连接参数更新 5
2.3.4 连接终止 5
2.4 GAP基础流程 5
2.4.1 外围设备 5
2.4.1.1 开启传统广播流程 6
2.4.1.2 开启扩展广播流程 8
2.4.1.3 开启周期性广播流程 11
2.4.2 中央设备 14
2.4.2.1 开启传统扫描流程 14
2.4.2.2 开启扩展扫描流程 15
2.4.2.3 发起传统连接流程 17
2.4.2.4 发起扩展连接流程 18
2.4.2.5 发起周期性广播同步流程 19
3 通用属性规范（GATT） 22
3.1 GATT角色 22
3.2 GATT Profile 层级 22
3.2.1 属性（Attribute） 23
3.2.2 特征（Characteristic） 23
3.2.3 服务（Service） 24
3.2.3.1 通用访问服务 24
3.2.3.2 通用属性服务 25
3.3 属性表 25
3.3.1 属性表定义 26
3.3.2 建立属性表 30
3.4 GATT基础流程 30
3.4.1 客户端 31
3.4.1.1 客户端发起通信 31
3.4.2 服务端 35
3.4.2.1 添加Profile 35

版权所有 © 2020 深圳市汇顶科技股份有限公司

3.4.2.2 Profile的读写回调函数.....35
3.4.2.3 客户端读请求处理.....36
3.4.2.4 客户端写请求处理.....37
3.5 GATT安全.....39
3.5.1 认证.....39
3.5.2 授权.....39
4 安全管理（SM）.....40
4.1 配对.....40
4.1.1 配对方法的选择.....41
4.1.2 配对方法的配置.....42
4.1.2.1 Just Works配对.....43
4.1.2.2 Passkey Entry配对.....43
4.1.2.3 Numeric Comparison 配对.....44
4.1.2.4 关闭配对功能.....44
4.2 绑定.....45
4.2.1 开启绑定功能.....45
4.3 隐私管理.....49
4.3.1 开启隐私管理功能.....49
4.3.2 地址配置说明.....51
5 逻辑链路控制和适配协议（L2CAP）.....53
5.1 L2CAP数据包结构.....53
5.2 最大传输单元（MTU）.....54
5.3 L2CAP信道.....54
5.4 面向连接的信道COC.....55
5.4.1 创建COC流程.....56
6 术语和缩略语.....58

版权所有 © 2020 深圳市汇顶科技股份有限公司

1 简介

GR551x低功耗蓝牙（Bluetooth Low Energy，BLE）软件架构由应用层（Application/Profile）、软件开发工具包（SDK）、低功耗蓝牙协议栈（BLE Protocol Stack）组成，如图 1-1所示。

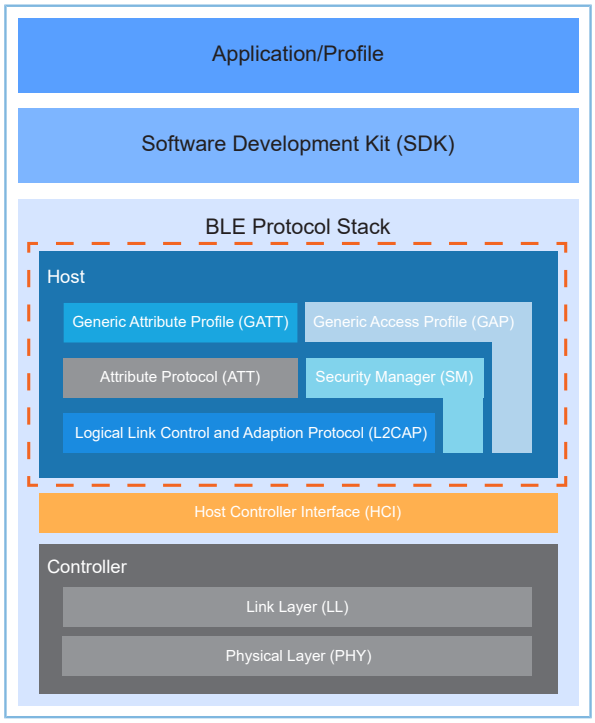

图 1-1 GR551x BLE软件架构

在该软件架构中，应用层通过SDK提供的接口实现与BLE协议栈的交互。开发者可调用BLE协议栈的GAP、GATT、SM、L2CAP的API。

本文档主要介绍这几层的基本功能、概念以及User App与协议栈的交互流程。为帮助开发者快速了解BLE Stack，GR551x SDK为各层还提供了代码示例（`SDK_Folder\projects\ble\ble_basic_example\`，SDK_Folder为GR551x SDK的根目录）。

说明:

更多BLE技术及其协议的相关资料，可访问Bluetooth SIG的官方网站www.bluetooth.com查看。GAP、GATT、SM、L2CAP的规范包含在Bluetooth Core Spec v5.1中。其他BLE应用层Profiles规范可以在GATT Specs页面下载。BLE应用可能会用到的Assigned Numbers，IDs和Codes均列在Assigned Numbers页面。

版权所有 © 2020 深圳市汇顶科技股份有限公司

2 通用访问规范（GAP）

通用访问规范GAP（Generic Access Profile）定义了蓝牙设备之间如何发现以及建立安全/非安全连接。

2.1 GAP角色

GAP定义了四种设备角色：广播者（Broadcaster）、观察者（Observer）、中央设备（Central）、以及外围设备（Peripheral），如图 2-1所示。

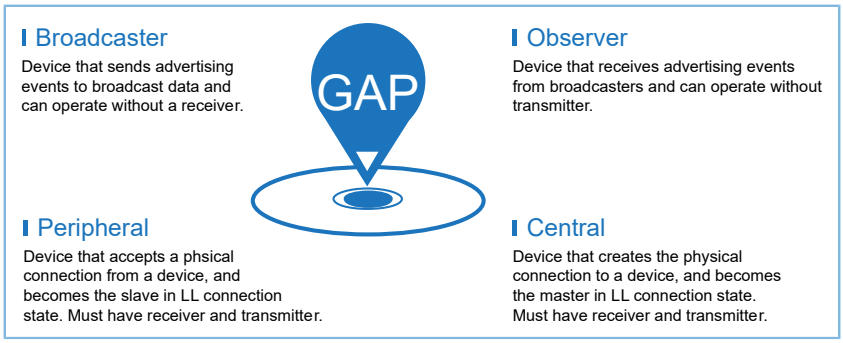

图 2-1 GAP角色

- 广播者（Broadcaster）：以广播事件向外发送广播数据，可以在没有接收机的条件下工作。
- 观察者（Observer）：从广播者设备那里接收到广播数据，可以在没有发射机的条件下工作。
- 中央设备（Central）：向另一个设备发起物理连接，链路层从发起态转换为连接态。必须同时具备接收机和发射机。
- 外围设备（Peripheral）：接收另一个设备发起的物理连接，链路层从广播态转换为连接态。必须同时具备接收机和发射机。

说明:

一个设备可以同时支持多个GAP角色：外围设备可以作为广播者，中央设备可以作为观察者。

2.2 GAP访问模式和设备流程

GAP提供多种访问模式和设备流程，包括：设备发现，连接建立，连接终止，设备参数配置等。

版权所有 © 2020 深圳市汇顶科技股份有限公司

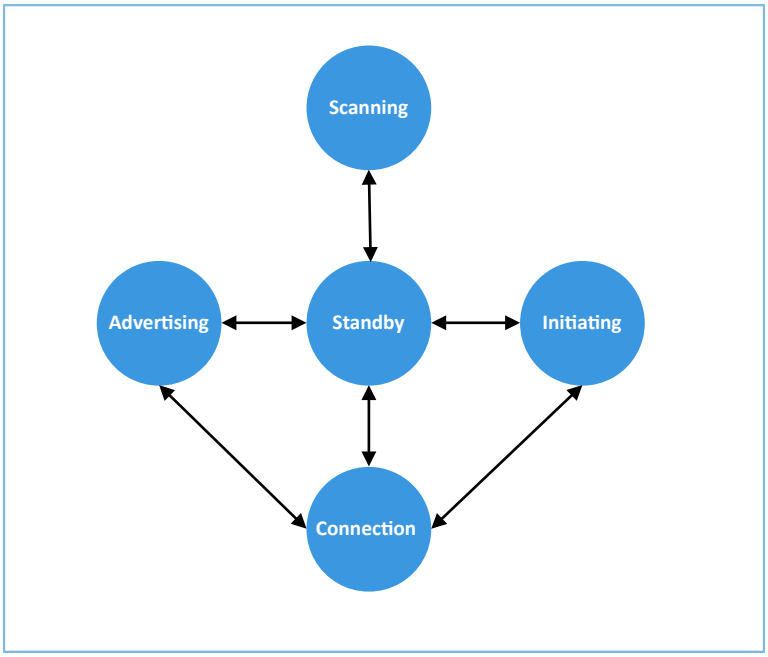

图 2-2 State diagram of the Link Layer state machine

基于给设备设定的角色，图 2-2展示了链路层各种状态间的转换关系，各状态描述如下：

- 就绪态（Standby State）：设备上电后处于初始的待机状态。
- 广播态（Advertising State）：设备向外广播特定的数据，以便让发起连接的设备发现这一广播设备。广播数据中包含广播地址及其他广播信息（比如设备名称等）。
- 扫描态（Scanning State）：设备接收广播数据，并向可扫描广播设备发送扫描请求。广播设备收到扫描请求后会回复一个扫描响应数据。这个过程被称作设备发现（Device Discovery）。
- 发起态（Initiating State）：进入发起态的设备必须指定一个想要对其发起连接的对端设备地址。如果收到的广播包中的广播者地址跟指定的对端地址匹配，发起态设备将向广播设备发送一个连接请求。连接请求数据包中包含一些指定的连接参数信息（具体参考2.3.2 连接参数介绍）。
- 连接态（Connection State）：连接建立的时候，处于广播态的设备将作为Slave转至连接态，处于发起态的设备将作为Master转至为连接态。

2.3 连接

2.3.1 连接事件

连接事件（Connection Event）是指主设备和从设备之间相互发送数据包的过程。

如图 2-3所示，连接事件被一个个的连接间隔分开。从主设备发送数据包开始，每个连接事件可以持续进行，直至主设备或从设备停止响应。在连接事件之外，主从设备之间不发送任何数据包。

版权所有 © 2020 深圳市汇顶科技股份有限公司

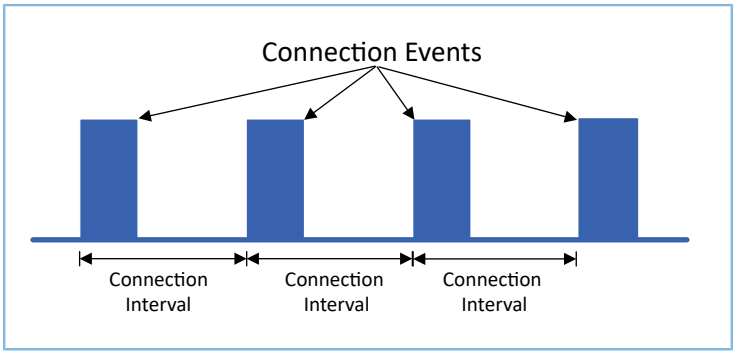

图 2-3 连接事件

2.3.2 连接参数介绍

发起态设备发送的连接请求数据包中包含连接参数信息，并且连接建立之后双方设备也可对连接参数进行修改。

各连接参数具体描述如下：

- 连接间隔（Connection Interval）

  连接间隔决定了主设备和从设备的交互间隔，它是指在一个连接事件的开始到下一个连接事件的开始的时间间隔。

  该参数的单位为1.25 ms，范围是6 ～ 3200，即7.5 ms ～ 4s。

- 从设备延迟（Slave Latency）

  从设备延迟用来指明从设备（外围设备）可以忽略的连接事件的个数。允许从设备在没有数据要发的情况下，跳过一定数目的连接事件，在这些连接事件中不必回复主设备的包，这样就能更加节约功耗。但从设备可以忽略的连接事件个数不应超过设置的“从设备延迟”参数的值。

  该参数的范围为：0 ～ 499。

- 监控超时（Supervision Timeout）

  这个超时时间是指两次成功的连接事件之间的最大可允许的时间。如果在这段时间内都没有一个成功的连接事件，主从设备会断开当前连接。

  该参数的单位是10 ms，范围是10 ～ 3200，即100 ms ～ 32s。

说明:

连接间隔、从设备延迟以及监控超时这三者必须满足如下公式：Supervision Timeout ＞（1 +slaveLatency）*（connection Interval）*2，否则无法建立连接。

这三个连接参数在不同的配置情况下，将对通信速率和功耗产生影响：

- 连接间隔缩短，主设备和从设备通信更加频繁，缩短数据发送周期，增加功耗。
- 连接间隔增长，主设备和从设备通信频率降低，增长数据发送周期，降低功耗。
- 如果从设备延迟设置为0，每次连接事件中都需要回复主设备的包，将提高数据发送速度，增加功耗。

版权所有 © 2020 深圳市汇顶科技股份有限公司

- 从设备延迟加长，将降低数据发送速度，降低功耗。

2.3.3 连接参数更新

连接建立时，主设备通过连接请求数据包发送连接参数。当连接活跃了一段时间后，连接参数可能不再适用于当前应用。为此，主设备可向从设备发送连接参数更新请求（Connection Parameter Update Request），也可直接通知从设备更新参数而不需要进行协商。

另外，从设备可能会根据APP的需要在连接的过程中去更新连接参数，从而向主设备发送连接参数更新请求。对于蓝牙4.1兼容设备，这个请求由链路层来处理；对于蓝牙4.0设备，则由L2CAP层来处理。低功耗蓝牙协议栈会自动选择更新的方法。

无论是由主设备发起的连接参数更新请求还是从设备发起的连接参数更新请求，都只能由主设备发起连接参数更新通知来应用新的连接参数。

说明:

具体的连接参数更新过程可参考Bluetooth Core Spec v5.1中的“6.2 Connection upadate（Vol 6，Part D）”和“6.12 Connection parameters request（Vol 6，Part D）”。

2.3.4 连接终止

连接终止是指断开链路，双方设备从连接态转为就绪态。主设备和从设备都可以主动发起连接终止的通知消息（LL_TERMINATE_IND），当收到对端设备的ACK响应（LL_ACK）之后，双方设备即退出连接的状态，具体流程如图 2-4所示。

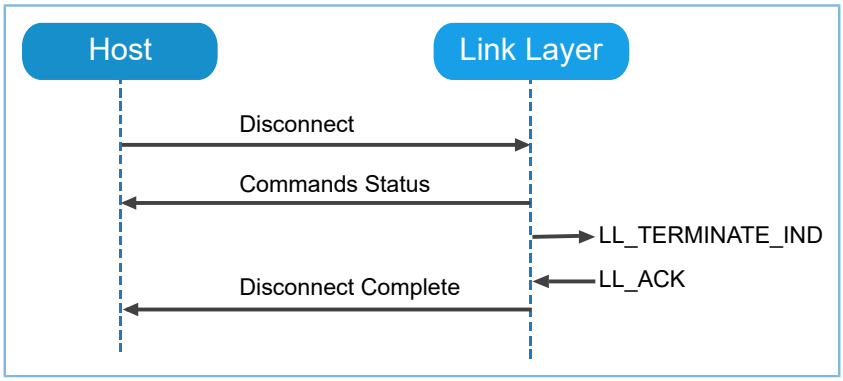

图 2-4 连接终止

另外，监控超时或者MIC（Message Integrity Check，消息完整性检查）检测出错也会导致连接断开。

2.4 GAP基础流程

本节主要以外围设备（Peripheral Role）和中央设备（Central Role）为例介绍GAP的基础流程。

2.4.1 外围设备

外围设备支持开启传统广播流程、扩展广播流程以及周期性广播流程。

版权所有 © 2020 深圳市汇顶科技股份有限公司

2.4.1.1 开启传统广播流程

当设备开启传统广播流程时，APP与BLE Stack之间的交互流程如图 2-5所示。

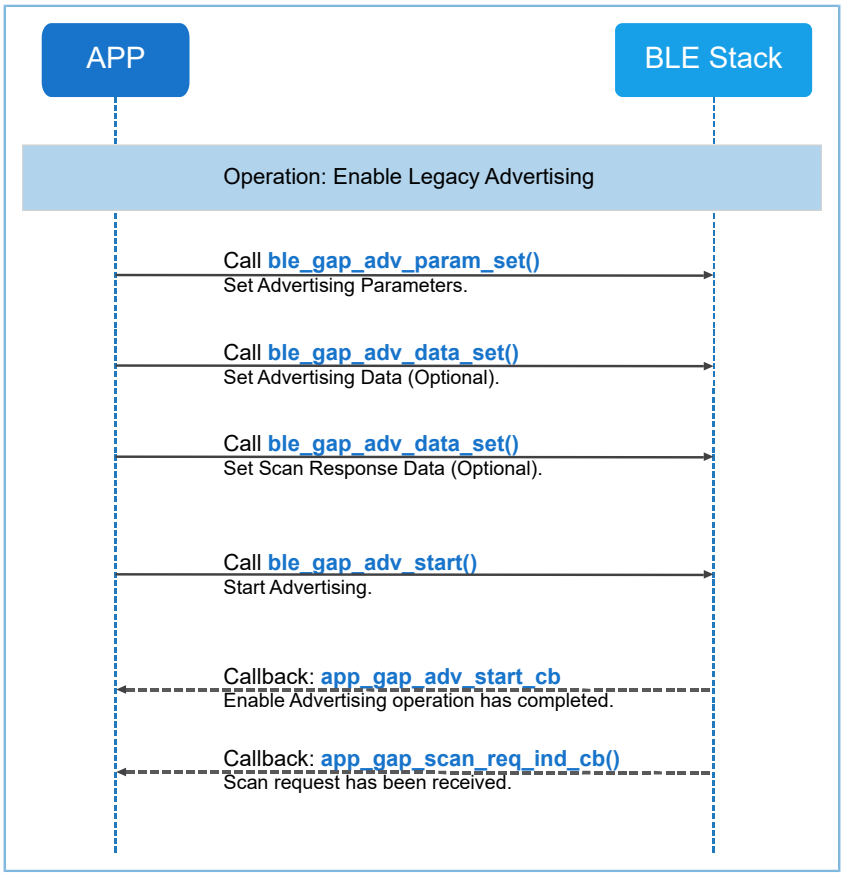

图 2-5 开启传统广播交互流程

开启传统广播流程的具体步骤如下：

说明:

步骤中的代码片段出自于传统广播流程示例ble_app_gap_legacy_adv (`SDK_Folder\projects\ble\ble_basic_example\`)。

---

1. 设置广播参数。

```
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_adv_param_set(0, BLE_GAP_OWN_ADDR_STATIC,
                                   &s_gap_adv_param);
APP_ERROR_CHECK(error_code);
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

- 定向广播的可发现模式只能设置为GAP_DISC_MODE_NON_DISCOVERABLE（不可发现）。
- 发现模式如果为GAP_DISC_MODE_BROADCASTER，则广播模式只能设置为GAP_ADV_TYPE_ADV_NONCONN_IND（不可连接也不可扫描）。
- peer_addr参数只在定向广播或者是controller privacy被使能（即ble_gap_privacy_params_set函数的第二个参数BLE_GAP_PRIV_CFG_PRIV_EN_BIT被置位）的情况下才会使用。
- 代码路径：

  SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_legacy_adv\Src\user\user_app.c

2. 设置广播数据（可选）。

   adv_mode为GAP_ADV_TYPE_ADV_HIGH_DIRECT_IND或GAP_ADV_TYPE_ADV_LOW_DIRECT_IND时，无需设置广播数据，其他情况则需设置。

3. 设置扫描响应数据（可选）。

   adv_mode为GAP_ADV_TYPE_ADV_IND或GAP_ADV_TYPE_ADV_SCAN_IND时，需设置扫描响应数据，其他情况则无需设置。

```
static const uint8_t s_adv_data_set[] =
{
    0x03,
    BLE_GAP_AD_TYPE_COMPLETE_LIST_16_BIT_UUID,
    0x01, 0x00,
};
static const uint8_t s_adv_rsp_data_set[] =
{
    0x0b,
    BLE_GAP_AD_TYPE_COMPLETE_NAME,
    'L', 'e', 'g', 'a', 'c', 'y', '_', 'A', 'D', 'V',
};
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);
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

adv data以及adv rsp data必须按照Bluetooth Core Spec v5.1规定的格式进行组合，即（length, type, data）。其中length所指示的长度为type字段与data字段的总长度。

代码路径：

```
SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_legacy_adv\Src\user
\user_app.c
```

4. 开启广播

   调用ble_gap_adv_start函数开启广播时需要用户传入参数adv_idx，以便指定广播实例索引。支持最多同时建立5个传统广播，所以adv_idx的取值范围是[0，1，2，3，4]。

```
s_gap_adv_time_param.duration = 0;
s_gap_adv_time_param.max_adv_evt = 0;
error_code = ble_gap_adv_start(0, &s_gap_adv_time_param);
APP_ERROR_CHECK(error_code);
```

说明:

代码路径：

```
SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_legacy_adv\Src\user
\user_app.c
```

5. 广播开启成功后调用回调函数app_gap_adv_start_cb。

说明:

如果用户希望更新广播参数，则可以在广播停止后调用ble_gap_adv_data_set和ble_gap_adv_param_set接口重新配置广播数据和广播参数，再调用ble_gap_adv_start来启动广播。

6. 如果广播参数的scan_req_ind_en值为true，那么当收到对端设备发送的扫描请求时，app_gap_scan_req_ind_cb回调函数将被调用。

2.4.1.2 开启扩展广播流程

当设备开启扩展广播流程时，APP与BLE Stack之间的交互流程如图 2-6所示。

版权所有 © 2020 深圳市汇顶科技股份有限公司

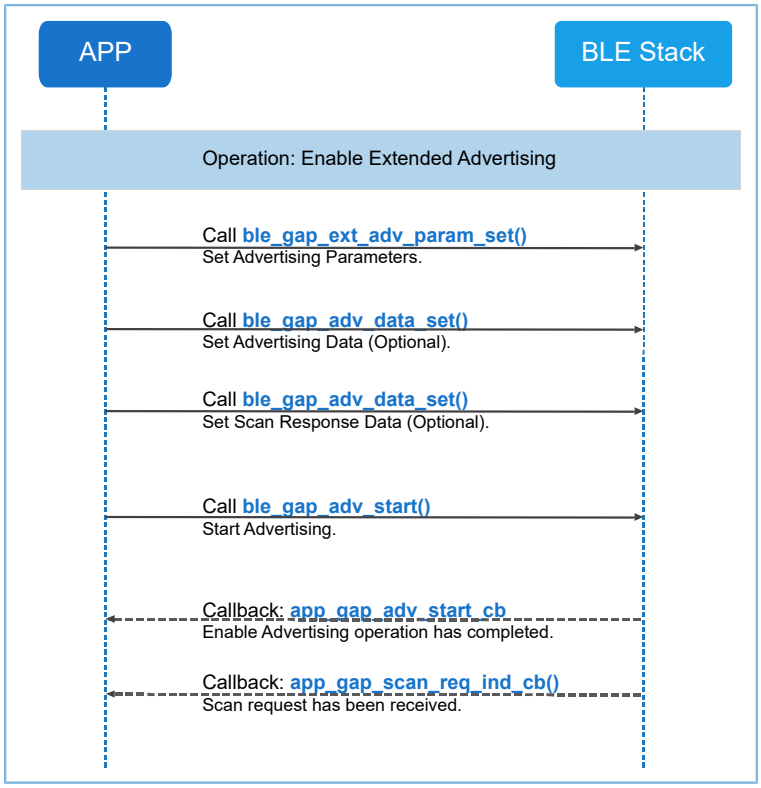

图 2-6 开启扩展广播流程

开启扩展广播流程的具体步骤如下：

说明:

步骤中的代码片段出自于扩展广播流程示例ble_app_gap_extended_adv（SDK_Folder\projects\ble\ble_basic_example\）。

---

1. 设置扩展广播参数。

```
s_gap_adv_param.type = GAP_ADV_TYPE_EXTENDED;
s_gap_adv_param.disc_mode = GAP_DISC_MODE_GEN_DISCOVERABLE;
/* The advertisement shall not be both connectable and scannable, and
    High duty cycle directed advertising cannot be used */
s_gap_adv_param.prop = GAP_ADV_PROP_CONNECTABLE_BIT;
s_gap_adv_param.max_tx_pwr = 0;
s_gap_adv_param.filter_pol = GAP_ADV_ALLOW_SCAN_ANY_CON_ANY;
memset(&s_gap_adv_param.peer_addr, 0, sizeof(gap_bdaddr_t));
s_gap_adv_param.prim_cfg.adv_intv_min = APP_ADV_MIN_INTERVAL;
s_gap_adv_param.prim_cfg.adv_intv_max = APP_ADV_MAX_INTERVAL;
s_gap_adv_param.prim_cfg.chnl_map = GAP_ADV_CHANNEL_37_38_39;
s_gap_adv_param.prim_cfg.phy = GAP_PHY_1MBPS_VALUE;
s_gap_adv_param.second_cfg.max_skip = 0;
s_gap_adv_param.second_cfg.phy = GAP_PHY_1MBPS_VALUE;
s_gap_adv_param.second_cfg.adv_sid = 0x00;
s_gap_adv_param.period_cfg.adv_intv_min = 0;
s_gap_adv_param.period_cfg.adv_intv_max = 0;
error_code = ble_gap_ext_adv_param_set(0, BLE_GAP_OWN_ADDR_STATIC,
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
                                        &s_gap_adv_param);
APP_ERROR_CHECK(error_code);
```

说明:

- 在两种情况下prop参数中GAP_ADV_PROP_DIRECTED_BIT可以被置位，一种情况是：disc_mode参数设置为GAP_DISC_MODE_NON_DISCOVERABLE；一种情况是：disc_mode参数设置为GAP_DISC_MODE_BROADCASTER，并且参数GAP_ADV_PROP_ANONYMOUS_BIT被置位。
- 扩展广播不支持High Duty的定向广播，因此在扩展广播中prop参数的GAP_ADV_PROP_HDC_BIT不能被置位。
- 扩展广播不支持既可扫描又可连接的广播，因此prop参数中GAP_ADV_PROP_CONNECTABLE_BIT和GAP_ADV_PROP_SCANNABLE_BIT不能被同时置位。
- 如果prop参数中GAP_ADV_PROP_ANONYMOUS_BIT被置位，则disc_mode参数只能设置为：GAP_DISC_MODE_NON_DISCOVERABLE或者 GAP_DISC_MODE_BROADCASTER。
- 如果prop参数中GAP_ADV_PROP_ANONYMOUS_BIT被置位，则prop参数中的GAP_ADV_PROP_CONNECTABLE_BIT和GAP_ADV_PROP_SCANNABLE_BIT都不能被置位。
- peer_addr参数只在定向广播或者controller privacy被使能（即ble_gap_privacy_params_set函数的第二个参数 BLE_GAP_PRIV_CFG_PRIV_EN_BIT被置位）的情况下被使用。

  代码路径：`SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_extended_adv\Src\user\user_app.c`

2. 设置广播数据（可选）。如果prop参数中GAP_ADV_PROP_SCANNABLE_BIT被置位，则无需设置。
3. 设置扫描响应数据（可选）。如果prop参数中GAP_ADV_PROP_SCANNABLE_BIT被置位，则需设置。

```
static const uint8_t s_adv_data_set[] =
{
    0x03,
    BLE_GAP_AD_TYPE_COMPLETE_LIST_16_BIT_UUID,
    0x01, 0x00,
};

static const uint8_t s_adv_rsp_data_set[] =
{
    0x0d,
    BLE_GAP_AD_TYPE_COMPLETE_NAME,
    'E', 'x', 't', 'e', 'n', 'd', 'e', 'd', '_', 'A', 'D', 'V',
};
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);
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

adv data以及adv rsp data必须按照Bluetooth Core Spec v5.1规定的格式进行组合，即（length, type, data）。length的长度为type字段与data字段的总长度。

代码路径：

```
SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_extended_adv\Src\user
\user_app.c
```

4. 开启广播。

   调用ble_gap_adv_start函数开启广播时需要用户传入参数adv_idx，以便指定广播实例索引。支持最多同时建立5个扩展广播，所以adv_idx的取值范围是[0，1，2，3，4]。

```
s_gap_adv_time_param.duration = 0;
s_gap_adv_time_param.max_adv_evt = 0;
error_code = ble_gap_adv_start(0, &s_gap_adv_time_param);
APP_ERROR_CHECK(error_code);
```

说明:

代码路径：

```
SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_extended_adv\Src\user
\user_app.c
```

5. 广播开启成功后调用回调函数app_gap_adv_start_cb。

6. 如果广播参数的prop中的GAP_ADV_PROP_SCAN_REQ_NTF_EN_BIT置位，那么当收到对端设备发送过来的扫描请求时，app_gap_scan_req_ind_cb回调函数将被调用。

2.4.1.3 开启周期性广播流程

当设备开启周期性广播流程时，APP与BLE Stack之间的交互流程如图 2-7所示。

版权所有 © 2020 深圳市汇顶科技股份有限公司

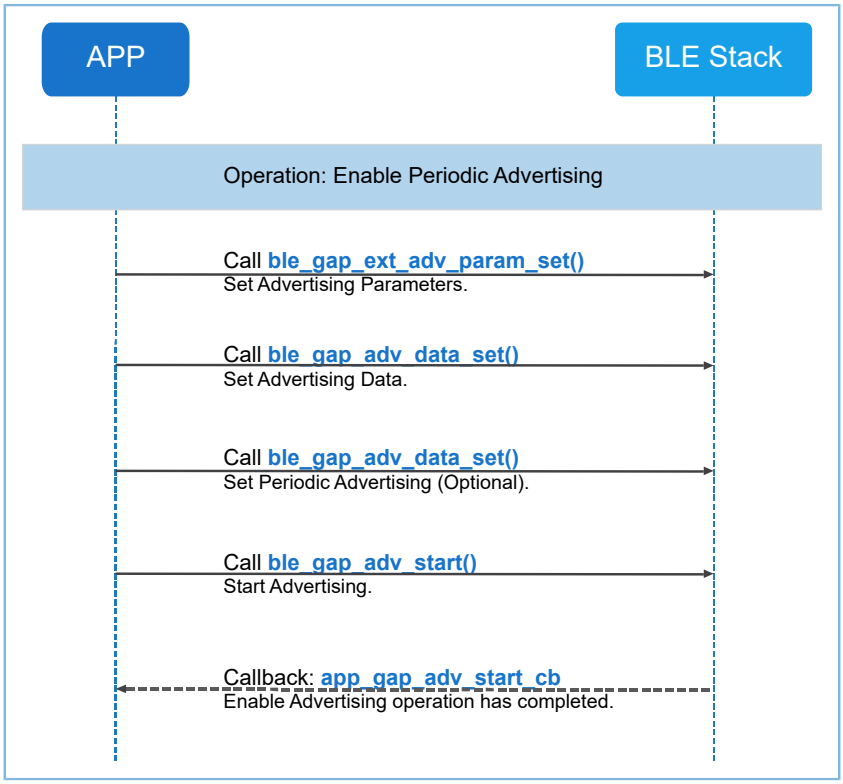

图 2-7 开启周期性广播流程

开启周期性广播流程的具体步骤如下：

说明:

步骤中的代码片段出自于周期性广播流程示例ble_app_gap_periodic_adv（SDK_Folder\projects\ble\ble_basic_example\）。

---

1. 设置广播参数。

```
s_gap_adv_param.type = GAP_ADV_TYPE_PERIODIC;
s_gap_adv_param.disc_mode = GAP_DISC_MODE_GEN_DISCOVERABLE;
/* Connectable, anonymous, scannable, high duty circle bit must be set to 0 */
s_gap_adv_param.prop = 0;
s_gap_adv_param.max_tx_pwr = 0;
s_gap_adv_param.filter_pol = GAP_ADV_ALLOW_SCAN_ANY_CON_ANY;
memset(&s_gap_adv_param.peer_addr, 0, sizeof(gap_bdaddr_t));
s_gap_adv_param.prim_cfg.adv_intv_min = APP_PRIMARY_ADV_MIN_INTERVAL;
s_gap_adv_param.prim_cfg.adv_intv_max = APP_PRIMARY_ADV_MAX_INTERVAL;
s_gap_adv_param.prim_cfg.chnl_map = GAP_ADV_CHANNEL_37_38_39;
s_gap_adv_param.prim_cfg.phy = GAP_PHY_1MBPS_VALUE;
s_gap_adv_param.second_cfg.max_skip = 0;
s_gap_adv_param.second_cfg.phy = GAP_PHY_1MBPS_VALUE;
s_gap_adv_param.second_cfg.adv_sid = 0x00;
s_gap_adv_param.period_cfg.adv_intv_min = APP_PERIODIC_ADV_MIN_INTERVAL;
s_gap_adv_param.period_cfg.adv_intv_max = APP_PERIODIC_ADV_MAX_INTERVAL;
error_code = ble_gap_ext_adv_param_set(0, BLE_GAP_OWN_ADDR_STATIC,
                                       &s_gap_adv_param);
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
APP_ERROR_CHECK(error_code);
```

说明:

- 在配置周期性广播参数过程中，prop参数的GAP_ADV_PROP_CONNECTABLE_BIT，GAP_ADV_PROP_SCANNABLE_BIT，GAP_ADV_PROP_ANONYMOUS_BIT，GAP_ADV_PROP_HDC_BIT都不能被置位。
- peer_addr参数只在定向广播或者controller privacy被使能（即ble_gap_privacy_params_set函数的第二个参数BLE_GAP_PRIV_CFG_PRIV_EN_BIT被置位）的情况下才会使用。
- 代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_periodic_adv\Src\user\user_app.c

2. 设置广播数据。

3. 设置周期性广播数据（可选）。

```
static const uint8_t s_adv_data_set[] =
{
    0x03,
    BLE_GAP_AD_TYPE_COMPLETE_LIST_16_BIT_UUID,
    0x01, 0x00,
    0x0d,
    BLE_GAP_AD_TYPE_COMPLETE_NAME,
    'P', 'e', 'r', 'i', 'o', 'd', 'i', 'c', '_', 'A', 'D', 'V',
};
static const uint8_t s_periodic_adv_data[] =
{
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06
};
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_PER_DATA,
                                  s_periodic_adv_data,sizeof(s_periodic_adv_data));
APP_ERROR_CHECK(error_code);
```

说明:

- adv data以及periodic_adv_data必须按照Bluetooth Core Spec v5.1规定的格式进行组合，即（length, type, data）。其中length所指示的长度为type字段与data字段的总长度。
- 代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_periodic_adv\Src\user\user_app.c

4. 开启广播。调用ble_gap_adv_start函数开启广播时需要用户传入参数adv_idx，以便指定广播实例索引。支持最多同时建立5个周期性广播，所以adv_idx的取值范围是[0，1，2，3，4]。

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
s_gap_adv_time_param.duration = 0;
s_gap_adv_time_param.max_adv_evt = 0;
error_code = ble_gap_adv_start(0, &s_gap_adv_time_param);
APP_ERROR_CHECK(error_code);
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_periodic_adv\Src\user\user_app.c

5. 广播开启成功后调用回调函数app_gap_adv_start_cb。

另外，需要注意的是系统在同一时间最多能同时开启五个广播，含传统广播、扩展广播和周期性广播。

2.4.2 中央设备

中央设备支持开启传统扫描流程、开启扩展扫描流程、发起传统连接流程、发起扩展连接流程以及发起周期性广播同步流程。

2.4.2.1 开启传统扫描流程

当设备开启传统扫描流程时，APP与BLE Stack之间的交互流程如图 2-8所示。

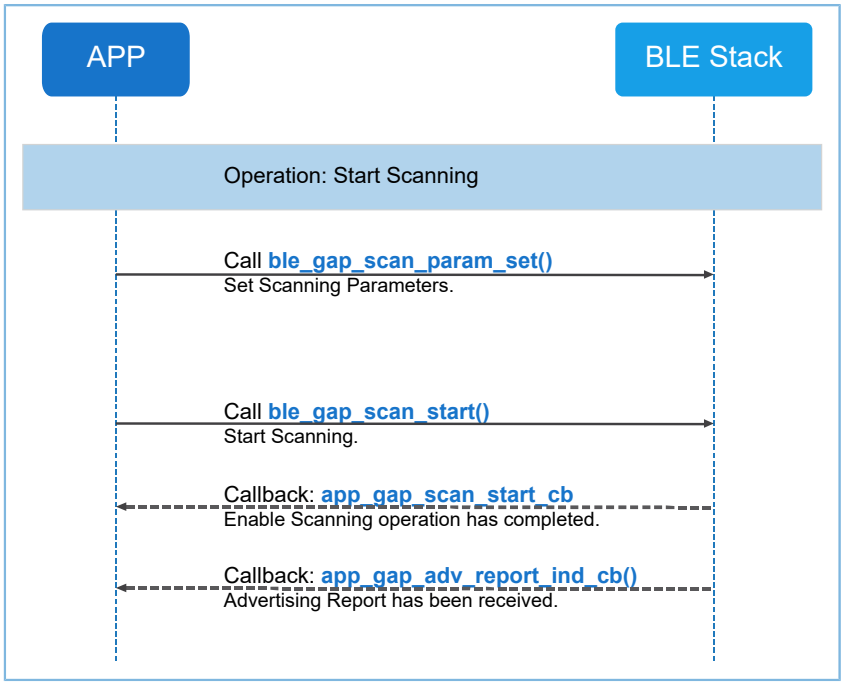

图 2-8 开启传统扫描流程

开启传统扫描流程的具体步骤如下：

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

步骤中的代码片段出自于传统广播流程示例ble_app_gap_legacy_scan（SDK_Folder\projects\ble\ble_basic_example\）。

1. 设置扫描参数

```
s_scan_param.scan_type = GAP_SCAN_ACTIVE;
s_scan_param.scan_mode = GAP_SCAN_OBSERVER_MODE;
s_scan_param.scan_dup_filt = GAP_SCAN_FILT_DUPLIC_DIS;
s_scan_param.use_whitelist = 0;
s_scan_param.interval= APP_SCAN_INTERVAL;
s_scan_param.window= APP_SCAN_WINDOW;
s_scan_param.timeout = 0;
error_code = ble_gap_scan_param_set(BLE_GAP_OWN_ADDR_STATIC,&s_scan_param);
APP_ERROR_CHECK(error_code);
```

说明:

- 参数scan_param.interval不小于scan_param.window。

  若scan_mode=GAP_SCAN_LIM_DISC_MODE或GAP_SCAN_GEN_DISC_MODE，则默认的扫描超时时间为10.24秒。

- 代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_legacy_scan\Src\user\user_app.c

2. 开启扫描

```
error_code = ble_gap_scan_start();
APP_ERROR_CHECK(error_code);
```

3. 扫描开启成功后调用回调函数app_gap_scan_start_cb。

4. 如果收到对端设备发送过来的广播数据或者扫描响应数据，app_gap_adv_report_ind_cb回调函数将被调用。

2.4.2.2 开启扩展扫描流程

当设备开启扩展扫描流程时，APP与BLE Stack之间的交互流程如图 2-9所示。

版权所有 © 2020 深圳市汇顶科技股份有限公司

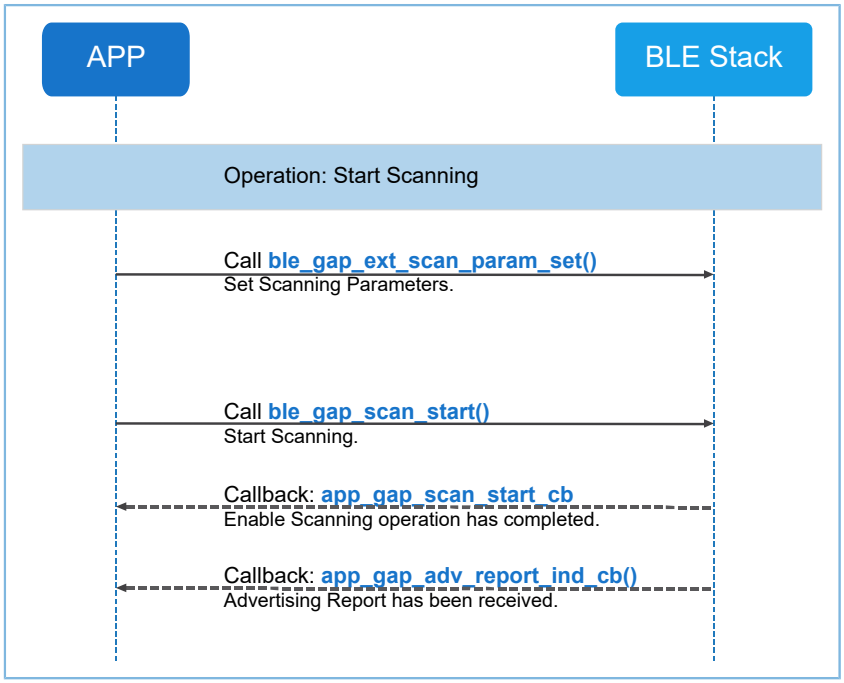

图 2-9 开启扩展扫描流程

开启扩展扫描流程的具体步骤如下：

说明:

步骤中的代码片段出自于扩展扫描流程示例ble_app_gap_extended_scan（SDK_Folder\projects\ble\ble_basic_example\）。

---

1. 设置扩展扫描参数。

```
s_scan_param.type = GAP_EXT_SCAN_TYPE_OBSERVER;
s_scan_param.prop = GAP_SCAN_PROP_PHY_1M_BIT |
                    GAP_SCAN_PROP_FILT_TRUNC_BIT;
s_scan_param.dup_filt_pol = GAP_EXT_DUP_FILT_DIS;
s_scan_param.scan_param_1m.scan_intv = APP_SCAN_INTERVAL;
s_scan_param.scan_param_1m.scan_wd= APP_SCAN_WINDOW;
s_scan_param.duration= 0;
s_scan_param.period= 0;
error_code = ble_gap_ext_scan_param_set(BLE_GAP_OWN_ADDR_STATIC,
                                        &s_scan_param);
APP_ERROR_CHECK(error_code);
```

说明:

- 参数scan_param.scan_param_1m.scan_intv不小于scan_param.scan_param_1m.scan_wd。

  如果scan_param.type = GAP_EXT_SCAN_TYPE_LIM_DISC或者GAP_EXT_SCAN_TYPE_LIM_DISC，则默认的扫描超时时间为10.24秒。

- 代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_extended_scan\Src\user\user_app.c

2. 开启扫描。

```
error_code = ble_gap_scan_start();
APP_ERROR_CHECK(error_code);
```

3. 扫描开启成功后调用回调函数app_gap_scan_start_cb 。

4. 如果收到对端设备发送过来的广播数据或者扫描响应数据，app_gap_adv_report_ind_cb回调函数将被调用。

2.4.2.3 发起传统连接流程

当设备发起传统连接流程时，APP与BLE Stack之间的交互流程如图 2-10所示。

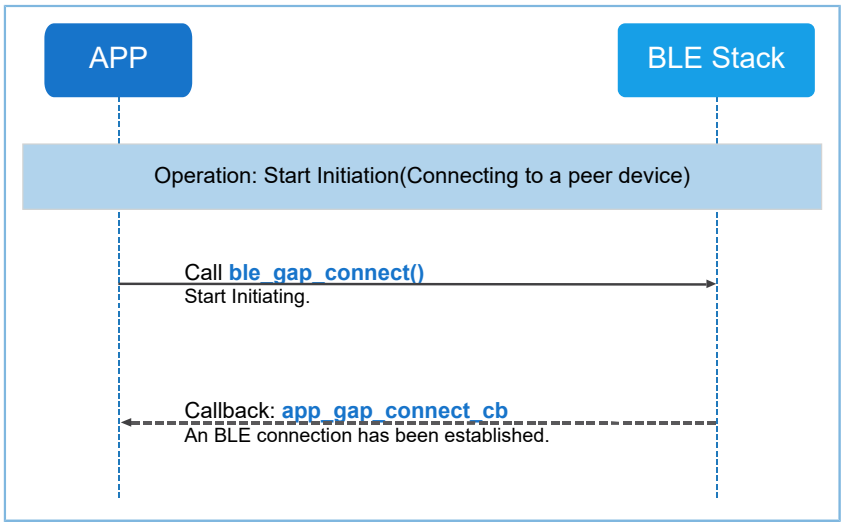

图 2-10 发起传统连接流程

发起传统连接流程的具体步骤如下：

说明:

步骤中的代码片段出自于传统连接流程示例ble_app_gap_legacy_connect（SDK_Folder\projects\ble\ble_basic_example\）。

1. 设置发起连接所需参数。

```
//peer device address
```

```
uint8_t peer_dev_addr[] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xA0};
memcpy(s_conn_param.peer_addr.gap_addr.addr, peer_dev_addr, 6);
s_conn_param.type = GAP_INIT_TYPE_DIRECT_CONN_EST;
//address type is public
s_conn_param.peer_addr.addr_type = 0;
s_conn_param.interval_min = APP_CONNECTION_MIN_INTERVAL;
s_conn_param.interval_max = APP_CONNECTION_MAX_INTERVAL;
s_conn_param.slave_latency = APP_CONNECTION_SLAVE_LATENCY;
s_conn_param.sup_timeout = APP_CONNECTION_MAX_TIMEOUT;
```

说明:

在测试时，m_conn_param.peer_addr需要根据用户的需求配置成实际的连接地址。如果s_conn_param.type配置为GAP_INIT_TYPE_NAME_DISC时，当连接建立后，app_gap_connect_cb不会上报事件给应用层，该Central会自动向peer获取device name。当获取到name后，app_gap_peer_name_ind_cb回调会上报。之后Central内部会终止该连接，终止连接后，app_gap_disconnect_cb也不会上报。

代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_legacy_connect\Src\user\user_app.c

2. 发起连接。

```
error_code = ble_gap_connect(BLE_GAP_OWN_ADDR_STATIC,&s_conn_param);
APP_ERROR_CHECK(error_code);
```

3. 无论连接建立成功或连接创建失败，app_gap_connect_cb回调函数将被调用。

2.4.2.4 发起扩展连接流程

当设备发起扩展连接流程时，APP与BLE Stack之间的交互流程如图 2-11所示。

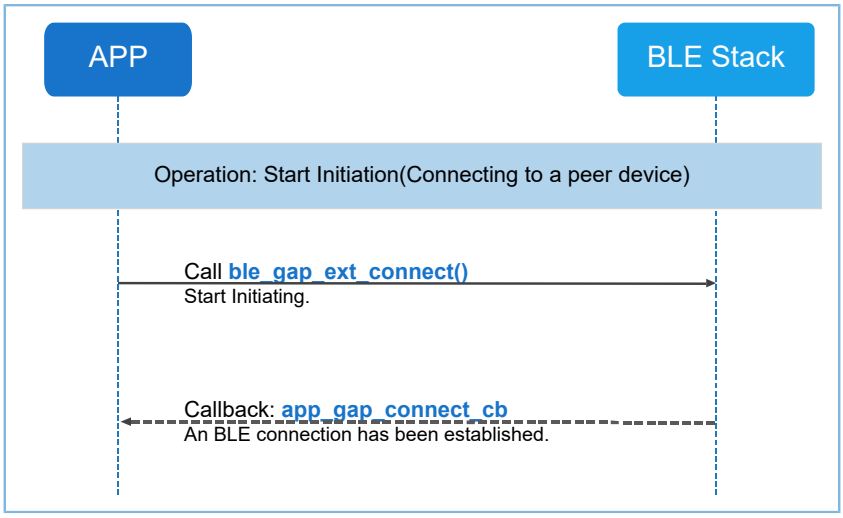

图 2-11 发起扩展连接流程

发起扩展连接流程的具体步骤如下：

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

步骤中的代码片段出自于扩展连接流程示例ble_app_gap_extended_connect（SDK_Folder\projects\ble\ble_basic_example\）。

1. 设置发起连接所需的参数。

```
uint8_t test_addr[] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xA0};
gap_ext_init_param_t ext_conn_param;
memset(&ext_conn_param, 0 , sizeof(ext_conn_param));
ext_conn_param.type = GAP_INIT_TYPE_DIRECT_CONN_EST;
ext_conn_param.prop = GAP_INIT_PROP_1M_BIT;
ext_conn_param.conn_to = 0;
ext_conn_param.scan_param_1m.scan_intv = 15;
ext_conn_param.scan_param_1m.scan_wd = 15;
ext_conn_param.conn_param_1m.conn_intv_min = 6;
ext_conn_param.conn_param_1m.conn_intv_max = 10;
ext_conn_param.conn_param_1m.conn_latency = 1;
ext_conn_param.conn_param_1m.supervision_to = 100;
ext_conn_param.conn_param_1m.ce_len = 0;
ext_conn_param.peer_addr.addr_type = 0;
memcpy(ext_conn_param.peer_addr.gap_addr.addr, test_addr, 6);
```

说明:

在测试时ext_conn_param.peer_addr需要根据用户的需求配置成实际的连接地址。如果ext_conn_param.type 配置为GAP_INIT_TYPE_NAME_DISC时，当连接建立后，app_gap_connect_cb不会上报，该Central 会自动向peer获取device name。当获取到name后，app_gap_peer_name_ind_cb回调会上报。之后Central内部会终止该连接，终止连接后，app_gap_disconnect_cb也不会上报。

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_extended_connect\Src\user\user_app.c

2. 发起连接。

```
error_code = ble_gap_ext_connect(BLE_GAP_OWN_ADDR_STATIC,&ext_conn_param);
APP_ERROR_CHECK(error_code);
```

3. 无论连接建立成功或连接创建失败，app_gap_connect_cb回调函数将被调用。

2.4.2.5 发起周期性广播同步流程

当发起周期性广播同步流程时，APP与BLE Stack之间的交互流程如图 2-12所示。

版权所有 © 2020 深圳市汇顶科技股份有限公司

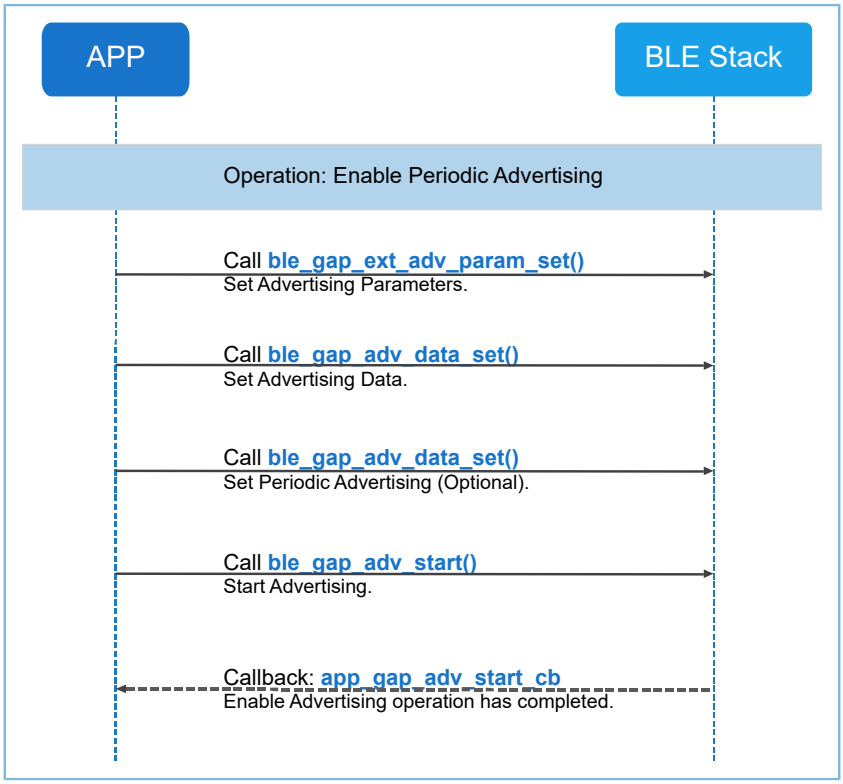

图 2-12 发起周期性广播流程

发起周期性广播同步流程的具体步骤如下：

说明:

步骤中的代码片段出自于周期性广播同步流程示例ble_app_gap_periodic _sync（SDK_Folder\projects\ble\ble_basic_example\）。

---

1. 在开启周期性广播同步之前，需要先进行扩展扫描。所以，首先另外发起一个开启扩展扫描流程，详见2.4.2.2 开启扩展扫描流程。

2. 设置扩展扫描参数。

```
s_scan_param.type = GAP_EXT_SCAN_TYPE_OBSERVER;
s_scan_param.prop = GAP_SCAN_PROP_PHY_1M_BIT |
                    GAP_SCAN_PROP_FILT_TRUNC_BIT;
s_scan_param.dup_filt_pol = GAP_EXT_DUP_FILT_DIS;
s_scan_param.scan_param_1m.scan_intv = APP_SCAN_INTERVAL;
s_scan_param.scan_param_1m.scan_wd= APP_SCAN_WINDOW;
s_scan_param.duration= 0;
s_scan_param.period= 0;
error_code = ble_gap_ext_scan_param_set(BLE_GAP_OWN_ADDR_STATIC,
                                        &s_scan_param);
APP_ERROR_CHECK(error_code);
```

说明:

代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_periodic_sync\Src\user\user_app.c

3. 启动扩展扫描。

```
error_code = ble_gap_scan_start();
APP_ERROR_CHECK(error_code);
```

4. 设置周期性广播同步参数。

   在static void app_gap_adv_report_ind_cb(uint8_t conidx, const gap_ext_adv_report_ind_t *param)回调函数中配置周期性广播同步参数并开启周期性广播同步。

```
s_per_sync_param.skip = 0;
s_per_sync_param.sync_to = APP_SYNC_TIMEOUT;
s_per_sync_param.type = GAP_PER_SYNC_TYPE_GENERAL;
s_per_sync_param.adv_addr.adv_sid = p_adv_report->adv_sid;
memcpy(s_per_sync_param.adv_addr.bd_addr.gap_addr.addr, s_peer_dev_addr, 6);
s_per_sync_param.adv_addr.bd_addr.addr_type = 0;
ble_gap_per_sync_param_set(0, &s_per_sync_param);
```

说明:

在测试过程中，设置peer_dev_addr = 开启周期性广播的设备的地址。

代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_gap_periodic_sync\Src\user_callback\user_gap_callback.c

5. 开始周期性广播同步。调用ble_gap_per_sync_start函数开启周期性同步时需要用户传入参数per_sync_idx。per_sync_idx的取值范围是[0，1，2，3，4]。

```
ble_gap_per_sync_start(0);
```

6. 无论周期性同步建立成功或周期性广播同步创建失败，app_gap_sync_establish_cb回调函数将被调用。

7. 周期性同步建立成功后，接收到的周期性广播数据将通过app_gap_adv_report_ind_cb回调函数上报用户。

版权所有 © 2020 深圳市汇顶科技股份有限公司

3 通用属性规范（GATT）

通用属性规范GATT（Generic Attribute Profile）用于两个连接设备之间的数据通信。在低功耗蓝牙GATT层中，数据以特征（Characteristic）的形式进行传输和存储。

3.1 GATT角色

从GATT的角度来看，当两个设备处于连接状态时，一个设备作为GATT服务端，另一个设备作为GATT客户端。

- GATT客户端：设备发起命令、请求并接收响应、通知和指示。
- GATT服务端：设备接收命令、请求并发出响应、通知和指示。

图 3-1为低功耗蓝牙设备连接的示意图。

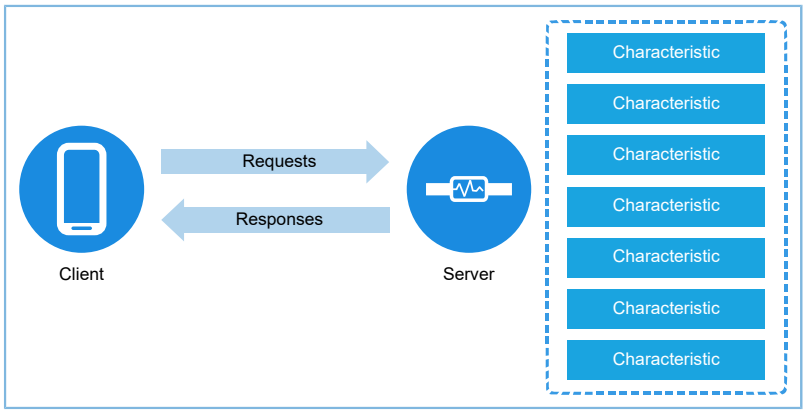

图 3-1 GATT Client 和GATT Server连接示意图

在这个连接当中，外围设备（蓝牙手环）作为GATT 服务端，中央设备（蓝牙手机）作为GATT客户端。这种角色的分配与GAP层设备角色（外围设备、中央设备）无关。一个外围设备既可以充当GATT客户端，也可以充当GATT服务端；一个中央设备同样既可以充当GATT客户端，也可以充当GATT服务端。

3.2 GATT Profile 层级

GATT规定了交换profile文件数据的层级结构，如图 3-2所示：

版权所有 © 2020 深圳市汇顶科技股份有限公司

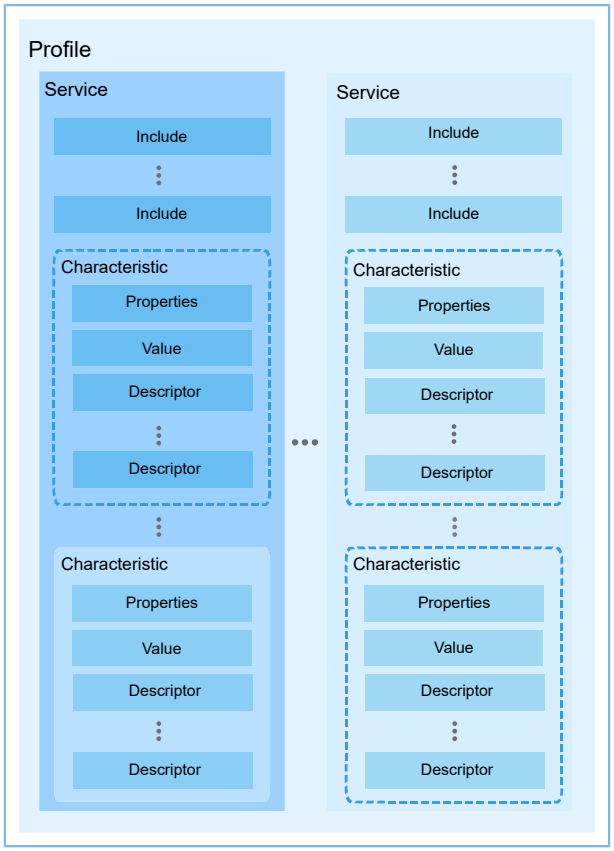

图 3-2 GATT profile 层级

一个profile中可包含一个或多个服务；一个服务可包含一个或者多个特征；一个特征至少包含两个属性，包括特征声明和特征值。

下面章节将具体描述属性、特征和服务。

3.2.1 属性（Attribute）

在低功耗蓝牙中，特征可以看作是一组属性信息的集合，包括特征声明、特征值以及特征描述符。实际上，在低功耗蓝牙GATT层中，数据以属性的形式进行交互。

属性主要包括以下几个部分：

- 属性句柄：句柄是属性在GATT属性表中的索引，每个属性都有唯一的句柄。
- 属性类型：表示数据代表的事物，通常是Bluetooth SIG规定或由用户自定义的UUID（通用唯一标识符）。
- 属性值：属性的数据值。
- 属性权限：规定了GATT客户端设备对属性的访问权限，包括是否能访问和怎样访问。

3.2.2 特征（Characteristic）

一个典型的特征由以下属性组成：

- 特征声明：描述特征的性质、存储位置（句柄）、类型。

版权所有 © 2020 深圳市汇顶科技股份有限公司

- 特征值：特征的数据值。
- 特征描述符：描述特征的附加信息或配置。

3.2.3 服务（Service）

GATT服务是一系列特征的集合，例如，心率服务包括心率测量特征和身体位置特征等。一个profile包含一个或多个服务。

下面列举一些常见的服务：

- 通用访问服务（Generic Access Service）：该服务包括设备及访问信息等，比如设备名称、供应商标志、产品标志等。该服务定义的特征有：设备名称（Device Name）、外观（Appearance）、外设优先连接参数（Peripheral Preferred Connection Parameters）等。
- 通用属性服务（Generic Attribute Service）：该服务主要用于服务端通知所连接的客户端其提供的服务发生了变化。该服务包含Service changed特征。

说明:

上述两个服务在协议栈初始化后，缺省添加到服务端的数据库中。

3.2.3.1 通用访问服务

对具有中央或者外围角色的低功耗蓝牙设备而言，通用访问服务（Generic Access Service）是必备的。通用访问服务主要用于设备发现和连接建立过程。

表 3-1 展示了Generic Access Service在GATT服务端中的存在形式。

表 3-1 通用访问服务的属性表

Handle	UUID	UUID Description	Value	Properties
0x0001	0x2800	Primary Service Declaration	00:18(Generic access)	
0x0002	0x2803	Characteristic Declaration	02:03:00:00:2A	
0x0003	0x2A00	Device Name		0x02
0x0004	0x2803	Characteristic Declaration	02:05:00:01:2A	
0x0005	0x2A01	Appearance		0x02
0x0006	0x2803	Characteristic Declaration	02:07:00:C9:2A	
0x0007	0x2AC9	Resolvable Private Address		0x02

说明:

表 3-1 展示了通用访问服务默认包含的特征，并非其全部内容。

（该服务中部分属性的设置和获取接口存在于

SDK_Folder\components\sdk\ble_sdk_api\ble_gapm.h头文件中。）

设置特征值的接口。

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
int16_t ble_gap_device_name_set(gap_dev_name_write_perm_t write_perm,
                                unit8_t const *p_dev_name, uint16_t length)
void ble_gap_appearance_set(uint16_t appearance)
```

这两个接口函数用来设置通用访问服务中的产品名称（Device Name）、产品外观（Appearance）特征值。

通用访问服务所包含的可选特征，如外设优先连接参数（Peripheral Preferred Connection Parameters）可通过以下接口函数来添加及设置：

```
void ble_gap_ppcp_present_set(bool present_flag)
uint16_t ble_gap_ppcp_set(gap_conn_param_t const *p_conn_params)
uint16_t ble_gap_privacy_params_set(uint16_t renew_dur, bool enable_flag)
```

3.2.3.2 通用属性服务

通用属性服务（Generic Attribute Service ）包含Service Changed特征。Service Changed特征用于GATT服务端通知与其绑定的设备其服务已经发生了更改。断开连接后，若服务发生了变更并且客户端特征配置描述符（CCCD）开启了指示（indication），那么绑定设备再次重连时，GATT服务端将向其GATT 客户端发送服务变更指示（indication）。但GATT客户端无法读取或写入Service Changed特征值。

3.3 属性表

GATT服务端通过属性表来组织发送的数据。

下面以心率服务的属性表为例，展示属性在GATT服务端中的存在形式。

表 3-2 心率服务的属性表

Handle	UUID	UUID Description	Value	Properties
0x002B	0x2800	Primary Service Declaration	0D:18（Heart Rate）	
0x002C	0x2803	Characteristic Declaration	10:2D:00:37:2A	
0x002D	0x2A37	Heart Rate Measurement		0x10
0x002E	0x2902	Client Characteristic Configuration		
0x002F	0x2803	Characteristic Declaration	02:30:00:38:2A	
0x0030	0x2A38	Body Sensor Location		0x02
0x0031	0x2803	Characteristic Declaration	08:32:00:39:2A	
0x0032	0x2A39	Heart Rate Control Point		0x08

心率服务的属性表主要包含以下特征：

版权所有 © 2020 深圳市汇顶科技股份有限公司

- Heart Rate Measurement ---- GATT客户端可以接收该特征值的通知（notify）。
- Body Sensor Location ---- GATT客户端可以对其进行读取操作。
- Heart Rate Control Point ---- GATT客户端可以对其进行写入操作。

下面以属性句柄为索引，对该属性表进行说明：

- 0x002B是Heart Rate服务的声明，该声明的UUID为0x2800，属性值为Heart Rate服务的UUID。
- 0x002C是Heart Rate Measurement特征的声明。这个声明可以被认为是指向Heart Rate Measurement特征值的指针，该声明的UUID为0x2803。声明值包含5个字节，各字节（MSB到LSB）的含义为：
  - 字节0规定了该特征的性质（property）：
    - 0x01 ---- 允许对该特征值进行广播。
    - 0x02 ---- 允许读取特征值。
    - 0x04 ---- 允许写入特征值（没有响应）。
    - 0x08 ---- 允许写入特征值（有响应）。
    - 0x10 ---- 允许向GATT 客户端进行特征值通知（无需确认）。
    - 0x20 ---- 允许向GATT 客户端进行特征值指示（需要确认）。
    - 0x40 ---- 允许对特征值进行签名写操作。
    - 0x80 ---- 存在扩展性质位，该扩展性质位在特征扩展性质描述符中定义。
  - 字节1 - 2表示了该特征值的句柄（handle）。
  - 字节3 - 4表示了该特征值的类型（UUID）。
- 0x002D是Heart Rate Measurement特征的值，该值的UUID为0x2A37。GATT客户端可以收到来自GATT服务端对该特征值的通知。
- 0x002E是Heart Rate Measurement特征的客户端特征配置描述符，该描述符的UUID为0x2902。通过对该属性值的写入，控制GATT服务端是否可以将特征值通知到GATT客户端。
- 0x002F - 0x0032的描述与上述属性的描述类似。

3.3.1 属性表定义

每个Service必须定义一个属性表并通过profile的初始化函数将其传递给协议栈。在`SDK_Folder\components\profiles\hrs\hrs.c`中定义的属性表数组如下：

- static const attm_desc_t hrs_attr_tab[HRS_IDX_NB];

  UUID可以是16 bits也可以是128 bits，128 bits UUID使用attm_desc_128_t类型。开发者可以使用Bluetooth SIG规定的UUID（在*ble_att.h*中定义），也可以使用在profile中自定义的UUID。

  16 bits UUID的数据结构为：

```
/**
 * @brief Service (16bits UUID) description.
```

```
 */
typedef struct
{
    uint16_t uuid;           /**< 16 bits UUID LSB First. */
    uint16_t perm;           /**< Attribute permissions,  see @ref
                                    BLE_GATTS_ATTR_PERM. */
    uint16_t ext_perm;      /**<Attribute extended permissions,  see @ref
                                   BLE_GATTS_ATTR_EXT_PERM. */
    uint16_t max_size;      /**< Attribute max size. */
} attm_desc_t;
```

128 bits UUID的数据结构为：

```
/**
 * @brief Service（128bits UUID） description.
 */
typedef struct
{
    uint8_t uuid[16];           /**< 128 bits UUID LSB First. */
    uint16_t perm;               /**< Attribute permissions,  see @ref
                                        BLE_GATTS_ATTR_PERM. */
    uint16_t ext_perm;           /**<Attribute extended permissions,  see @ref
                                      BLE_GATTS_ATTR_EXT_PERM. */
    uint16_t max_size;          /**< Attribute max size. */
} attm_desc_128_t;
```

- perm权限

  perm权限规定了对端设备GATT客户端是否可以以及如何访问服务端存放的属性值，具体的权限定义如下：

```
/**< Default Read permission. No encrypt*/
#define READ_PERM_UNSEC      (READ << 8)
/**< Read permission set with authenticate level *
#define READ_PERM(sec_level)  (READ << 8 | ((sec_level &
                               SEC_LEVEL_MASK) << READ_POS))

/**< Default Write Permission. No encrypt */
#define WRITE_REQ_PERM_UNSEC  (WRITE_REQ << 8)

/**< Write permission set with authenticate level */
#define WRITE_REQ_PERM(sec_level) (WRITE_REQ << 8 | ((sec_level &
                                   SEC_LEVEL_MASK) << WRITE_POS))

/**< Default Write without Response Permission. No encrypt */
#define WRITE_CMD_PERM_UNSEC  (WRITE_CMD << 8)

/**<  Write without Response permission set with authenticate level */
#define WRITE_CMD_PERM(sec_level)  (WRITE_CMD << 8 | ((sec_level &
                                    SEC_LEVEL_MASK) << WRITE_POS))
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
/**< Default Authenticated Signed Write Permission. No encrypt */
#define WRITE_SIGNED_PERM_UNSEC   (WRITE_SIGNED << 8)

/**< Authenticated Signed Write permission set with authenticate level */
#define WRITE_SIGNED_PERM(sec_level)  (WRITE_SIGNED << 8 |
                        ((sec_level & SEC_LEVEL_MASK) << WRITE_POS))

/**< Default Indicate Permission. No encrypt */
#define INDICATE_PERM_UNSEC            (INDICATE << 8)

/**< Indicate permission set with authenticate level */
#define INDICATE_PERM(sec_level)   (INDICATE << 8 | ((sec_level &
                                    SEC_LEVEL_MASK) << INDICATE_POS))

/**< Default Notify Permission. No encrypt */
#define NOTIFY_PERM_UNSEC                (NOTIFY << 8))

/**< Notify permission set with authenticate level */
#define NOTIFY_PERM(sec_level)       (NOTIFY << 8 | ((sec_level &
                                     SEC_LEVEL_MASK) << NOTIFY_POS))

/**< Broadcast enable. */
#define BROADCAST_ENABLE                (BROADCAST << 8)

/**< Extended Properties enable. */
#define EXT_PROP_ENABLE                   (EXT_PROP << 8)
```

- ext_perm扩展权限

  ext_perm扩展权限规定了对加密链路加密keysize的要求、属性UUID length的定义以及属性值存放地的定义，代码如下：

```
/**< 16 bytes encryption key size. */
#define ATT_ENC_KEY_SIZE_16      (0x1000)
/**< Attribute UUID length set. See @ref BLE_GATTS_UUID_TYPE */
#define ATT_UUID_TYPE_SET(uuid_len)    (uuid_len << 13)
/**< Value location, means value saved in user space, the profile's
      read/write callback will be called. */
#define ATT_VAL_LOC_USER    (0x8000)
```

- 特征值最大长度

  max_size：若该条属性是特征值，那么该成员定义了特征值的最大长度。

  下面以Heart Rate profile（SDK_Folder\components\profiles\hrs）的例子说明attm_desc_t hrs_attr_tab 各种类型的属性定义。

- Service声明

  Heart rate profile服务属性定义：

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
{
    BLE_ATT_DECL_PRIMARY_SERVICE,            //16bits UUID
    READ_PERM_UNSEC,                         //permission
    0,                                       //ext_permission
    0                                        //max size
}
```

UUID被设置为Bluetooth SIG定义的primary service UUID（0x2800）。GATT客户端必须读取该属性，所以其权限被设置为READ_PERM_UNSEC。

- Characteristic声明

  Heart rate profile measurement的特征声明：

```
{
    BLE_ATT_DECL_CHARACTERISTIC,             //16bits UUID
    READ_PERM_UNSEC,                         //permission
    0,                                       //ext_permission
    0                                        //max size
}
```

UUID被设置为Bluetooth SIG定义的特性声明UUID（0x2803）。GATT客户端必须读取该属性，所以其权限被设置为READ_PERM_UNSEC。

- Characteristic值

  Heart rate profile measurement的特征值：

```
{
    BLE_ATT_CHAR_HEART_RATE_MEAS,                //16bits uuid
    READ_PERM_UNSEC|NOTIFY_PERM_UNSEC,           //permission
    ATT_VAL_LOC_USER,                            //ext_permission
    HRS_MEAS_MAX_LEN                             //max size
}
```

UUID被设置为用户自定义的UUID。特征值ext_perm中的最高位若被置位（表示特征值的内存在用户空间分配），此时GATT 客户端读取或者写入该特征值，BLE Stack会调用heart rate profile的读或者写回调函数（如果回调函数存在）。

- 客户端的Characteristic配置

  Heart rate profile measurement的特征值配置：

```
{
    BLE_ATT_DESC_CLIENT_CHAR_CFG,                  //UUID
    READ_PERM_UNSEC| WRITE_REQ_PERM_UNSEC,         //permission
    0,                                             //ext_permission
    0                                              //max size
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
}
```

UUID被设置为Bluetooth SIG定义的client characteristic configuration UUID（0x2902）。GATT 客户端必须读写该属性，所以它应该具有可读可写的权限。由于存在多个连接，所以CCCD的value值也有多个。CCCD value值只允许存放在用户空间，并且该属性的ext_perm最高位会被强制置位。

3.3.2 建立属性表

当设备上电或者重启时，APP需在初始化阶段构建自己的服务列表。每个服务都包含一些属性，并根据自己的功能需求有自己的回调函数。属性表和回调函数被传递给ble_server_prf_add函数并且被保存在协议栈。

属性表的初始化必须在APP初始化函数中进行。例如，在APP初始化函数中，依次调用A_service_init以及B_service_init函数，执行流程如图 3-3所示。

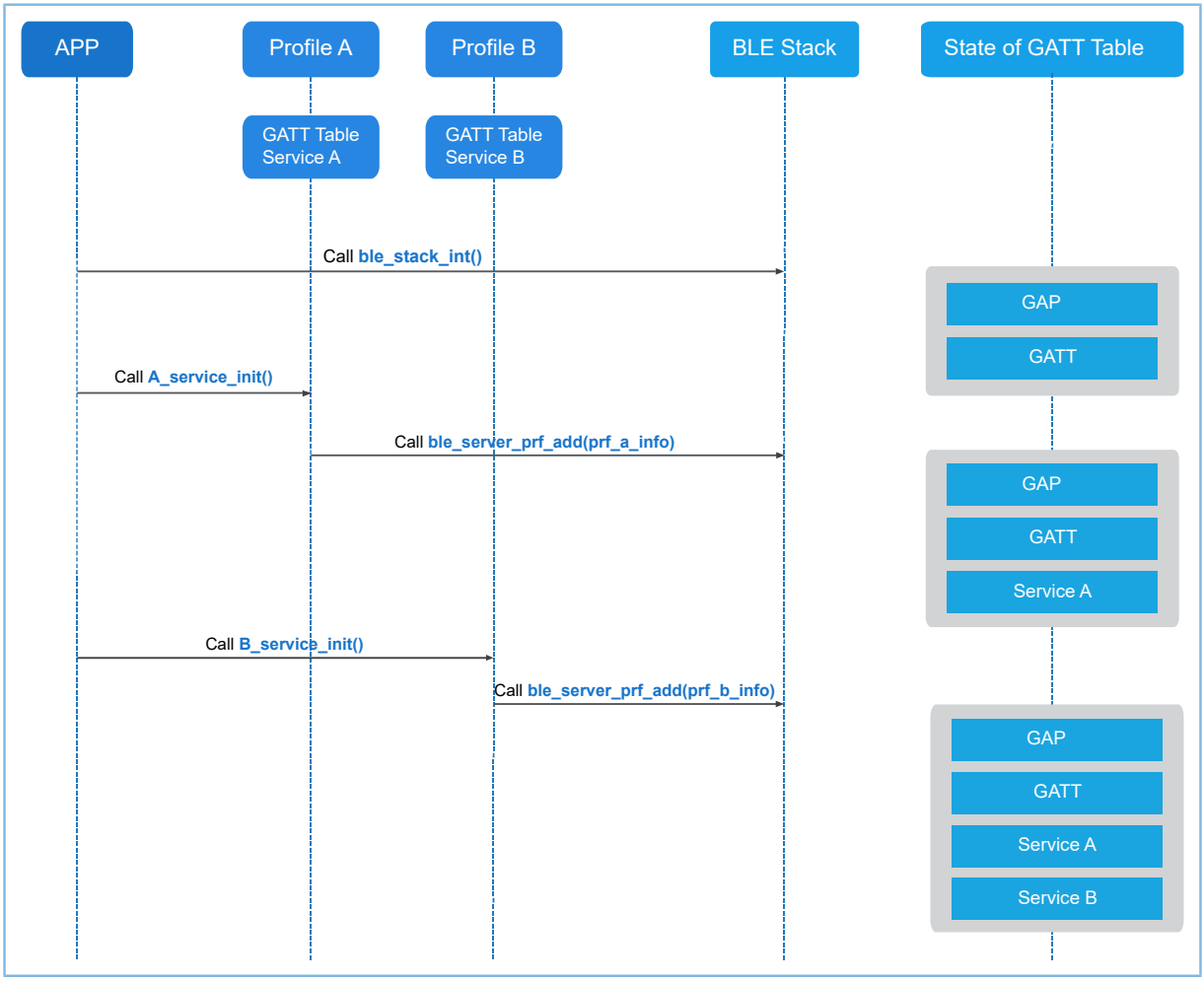

图 3-3 建立属性表流程

3.4 GATT基础流程

GATT定义了若干GATT服务端和客户端之间通信的子过程，下面将分别介绍客户端发起通信和服务端处理请求的流程。

版权所有 © 2020 深圳市汇顶科技股份有限公司

3.4.1 客户端

GATT客户端主要向服务端发起命令和请求并接收服务端的响应、指示和通知。

GATT客户端没有属性表和profile，它主要用于获取属性信息而不是提供属性信息和服务。大多数与GATT层的交互是直接由应用层发起的，在这种情况下可直接使用GATT的API接口。这些API主要用于GATT客户端的应用程序，并且大部分GATT客户端的API被调用之后将通过用户注册的callback函数返回结果，包括读取的属性值、写是否完成状态、指示等。关于API的详细描述，请参考《GR551x API Reference》。

3.4.1.1 客户端发起通信

本节主要描述APP作为GATT客户端发起通信的过程，客户端相关函数声明位于头文件SDK_Folder\components\sdk\ble_sdk_api\ble_gattc.h。

当设备作为GATT客户端发起通信时，APP与BLE Stack之间的交互流程如图 3-4所示。

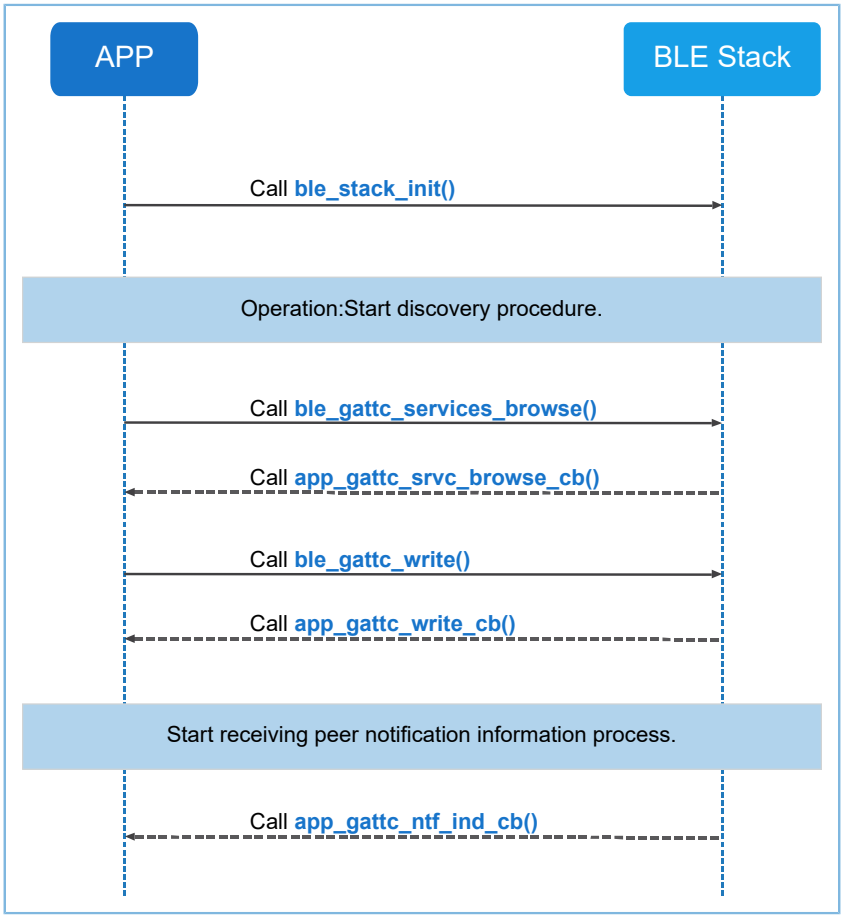

图 3-4 客户端的交互流程示例

GATT客户端发起通信流程的具体步骤如下：

说明:

步骤中的代码片段出自于GATT Client流程示例ble_app_gatt_client（SDK_Folder\projects\ble\ble_basic_example\）。

版权所有 © 2020 深圳市汇顶科技股份有限公司

1. 实现GATT客户端callback函数集合。

   用户需实现的GATT客户端callback函数集合包括app_gattc_callback。

   GATT客户端app_gattc_callback函数集合包含的函数指针成员有：

```
/**@brief GATTC Event callback Structures. */
typedef struct
{
   /**< Primary Service Discovery Response callback. */
   void (*app_gattc_srvc_disc_cb)(uint8_t conn_idx,
                                  uint8_t status,
                                  const ble_gattc_srvc_disc_t * p_prim_srvc_disc);
   /**< Relationship Discovery Response callback. */
   void (*app_gattc_inc_srvc_disc_cb)(uint8_t conn_idx,
                                      uint8_t status,
                                      const ble_gattc_incl_disc_t * p_inc_srvc_disc);
   /**< Characteristic Discovery Response callback. */
   void (*app_gattc_char_disc_cb)(uint8_t conn_idx,
                                  uint8_t status,
                                  const ble_gattc_char_disc_t * p_char_disc);
   /**< Descriptor Discovery Response callback. */
   void (*app_gattc_char_desc_disc_cb)(uint8_t conn_idx,
                                       uint8_t status,
                                       const ble_gattc_char_desc_disc_t *p_char_desc_disc);
   /**< Read Response callback. */
   void (*app_gattc_read_cb)(uint8_t conn_idx,
                             uint8_t status,
                             const ble_gattc_read_rsp_t *p_read_rsp);
   /**< Write complete callback. */
   void (*app_gattc_write_cb)(uint8_t conn_idx,
                              uint8_t status,
                              uint16_t handle);
   /**< Handle Value Notification/Indication Event callback. */
   void (*app_gattc_ntf_ind_cb)(uint8_t conn_idx,
                                const ble_gattc_ntf_ind_t *p_ntf_ind);
   /**< Service found callback during browsing procedure. */
   void app_gattc_srvc_browse_cb(uint8_t conn_idx, uint8_t status,
                                 const ble_gattc_browse_srvc_t *p_browse_srvc);
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

- 用于发现对端主要服务（Primary Service）的API为ble_gattc_primary_services_discover()。一旦对端的Primary Service被发现，将会调用app_gattc_srvc_disc_cb。
- 用于发现对端的包含服务关系（Included Service）的API为ble_gattc_included_services_discover()。一旦对端的Included service被发现，将会调用app_gattc_inc_srvc_disc_cb。
- 用于发现对端特征（Characteristic）的API为ble_gattc_char_discover()。一旦对端的特征声明被发现，将会调用app_gattc_char_disc_cb。
- 用于发现对端的特征描述符（Characteristic Descriptor）的API为ble_gattc_char_desc_discover()。一旦对端的特征描述符被发现，将会调用app_gattc_char_desc_disc_cb。
- 用于读取对端属性值的API为ble_gattc_read()，ble_gattc_read_by_uuid()以及ble_gattc_read_multiple()。一旦对端的属性值读取完成，将会调用app_gattc_read_cb。
- 一旦本端收到对端的通知和指示，将会调用app_gattc_ntf_ind_cb。
- 用于发现对端某个服务或者所有服务所包含的属性的API为ble_gattc_services_browse()。一旦对端服务的所有属性被发现，将会调用app_gattc_srvc_browse_cb。

2. 注册GATT客户端callback函数。

```
static app_callback_t s_app_ble_callback =
{
    .app_ble_init_cmp_callback = ble_init_cmp_callback,
    .app_gap_callbacks         = &app_gap_callbacks,
    .app_gatt_common_callback  = NULL,
    .app_gattc_callback        = &app_gattc_callback,
};
```

说明:

代码路径：

```
SDK_Folder\projects\ble\ble_basic_example\ble_app_gatt_client\Src\user\main.c
```

3. 执行GATT客户端write过程。

   过滤发现特征中的特征描述属性CCCD，并通过写入对端的CCCD特征配置值使能通知功能。

```
//write cccd to enable notification for peer server
uint16_t cccd_value = 0x0001;
if ((BLE_GATTC_BROWSE_ATTR_DESC  ==
     p_browse_srvc->info[fnd_att].attr_type) &&
     (BLE_ATT_DESC_CLIENT_CHAR_CFG == *(uint16_t *)
     (p_browse_srvc->info[fnd_att].attr.uuid)))
{
    APP_LOG_DEBUG("[%s] Char Description: attr handle = %04X\n",
                  __func__, (p_browse_srvc->start_hdl + fnd_att + 1));
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
    if (ble_gattc_write(conn_idx, p_browse_srvc->start_hdl + fnd_att + 1, 0,
                        sizeof(uint16_t),
                        (uint8_t *)&cccd_value) == SDK_SUCCESS)
    {
        APP_LOG_DEBUG("[%s] Send write cccd value command!\n", __func__);
    }
}
```

4. APP接收和处理服务端的响应和通知。

   GATT客户端收到服务端发送的ATT_WRITE_RSP 数据包之后将回调用户注册的app_gattc_write_cb函数。

```
static void app_gattc_write_cb(uint8_t conn_idx, uint8_t status, uint16_t handle)
{
    APP_LOG_DEBUG("[%s]GATT Client Write Completed!", __func__);
}
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gatt_client\Src\user_callback\user_gattc_callback.c

此外GATT客户端还将接收到服务端发送的通知，通过用户注册的app_gattc_ntf_ind_cb函数返回通知的内容。

```
static void app_gattc_ntf_ind_cb(uint8_t conn_idx, const ble_gattc_ntf_ind_t *p_ntf_ind)
{
    APP_LOG_DEBUG("[%s]enter!", __func__);

    char *notify_indicate[2] =
    {
        "GATTC_OP_NOTIFICATION",
        "GATTC_OP_INDICATION",
    };

    if (BLE_GATT_NOTIFICATION == p_ntf_ind->type)
    {
        APP_LOG_DEBUG("[%s]type = %s, ", __func__, notify_indicate[0]);
    }
    else if (BLE_GATT_INDICATION == p_ntf_ind->type)
    {
        APP_LOG_DEBUG("[%s]type = %s, ", __func__, notify_indicate[1]);
    }

    APP_LOG_DEBUG("Attribute handle = %04X, Attribute Value = ", p_ntf_ind->handle);

    for (uint16_t i = 0; i < p_ntf_ind->length; i++)
    {
        if (i == p_ntf_ind->length - 1)
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
        {
            APP_LOG_DEBUG("%02X", p_ntf_ind->p_value[i]);
        }
        else
        {
            APP_LOG_DEBUG("%02X:", p_ntf_ind->p_value[i]);
        }
    }

    /* send confirm pdu */
    if (BLE_GATT_INDICATION == p_ntf_ind->type)
    {
        ble_gattc_indicate_cfm(conn_idx, p_ntf_ind->handle);
    }
}
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_gatt_client\Src\user_callback\user_gattc_callback.c

3.4.2 服务端

GATT服务端负责接收对端设备GATT客户端发送的命令以及请求，并且根据收到的命令以及请求回复对端设备响应，或者向对端设备发送通知或指示。GATT服务端的功能需要和各个profile紧密配合才能够完成。

下面章节将描述如何在GATT服务端添加profile和注册profile回调函数，以及GATT服务端如何处理客户端的读/写请求。

3.4.2.1 添加Profile

在APP初始化阶段需要添加它所支持的profile。每个profile都有一个接口供APP使用来添加该profile。在协议栈中添加的两个profile是：GAP profile与GATT profile。

下面以SDK_Folder\components\profiles\hrs\hrs.c中的接口hrs_service_init(hrs_init_t *p_hrs_init)为例，来说明如何添加profile。

```
memcpy(&s_hrs_env.hrs_init, p_hrs_init, sizeof(hrs_init_t));
return ble_server_prf_add(&hrs_prf_info);
```

调用接口ble_server_prf_add()到协议栈注册profile，profile的初始化函数以及回调函数信息都将由此接口传递给协议栈。

3.4.2.2 Profile的读写回调函数

为了处理对端设备GATT客户端对profile属性的读写，profile需要定义自己的读写回调函数。这些回调函数的注册是通过ble_server_prf_add()完成的。

Profile可以通过发送事件的形式传递信息给应用，profile接收用户注册的事件处理函数，并且这个注册过程是在APP初始化阶段完成的。

版权所有 © 2020 深圳市汇顶科技股份有限公司

例如，在SDK_Folder\components\profiles\hrs\hrs.h中，事件定义如下：

```
/**@brief Heart Rate Service event types. */
typedef enum
{
    HRS_EVT_NOTIFICATION_ENABLED,     /**< Heart Rate value notification has been enabled.*/
    HRS_EVT_NOTIFICATION_DISABLED,    /**< Heart Rate value notification has been disabled.*/
    HRS_EVT_RESET_ENERGY_EXPENDED,  /**< The peer device requests to reset Energy Expended.*/
    HRS_EVT_READ_BODY_SEN_LOCATION,    /**< The peer device read Body Sensor
                                            Location characteristic.*/
} hrs_evt_type_t;
/** @} */

/**
 * @defgroup HRS_STRUCT Structures
 * @{
 */
/**@brief Heart Rate Service event. */
typedef struct
{
    uint8_t         conn_idx;    /**< Index of connection. */
    hrs_evt_type_t  evt_type;    /**< Heart Rate Service event type. */
} hrs_evt_t;
/** @} */

/**
 * @defgroup HRS_TYPEDEF Typedefs
 * @{
 */
/**@brief Heart Rate Service  event handler type. */
typedef void (*hrs_evt_handler_t)(hrs_evt_t *p_evt);
/** @} */
```

3.4.2.3 客户端读请求处理

以Heart rate profile为例，服务端处理客户端读请求的交互流程如图 3-5所示：

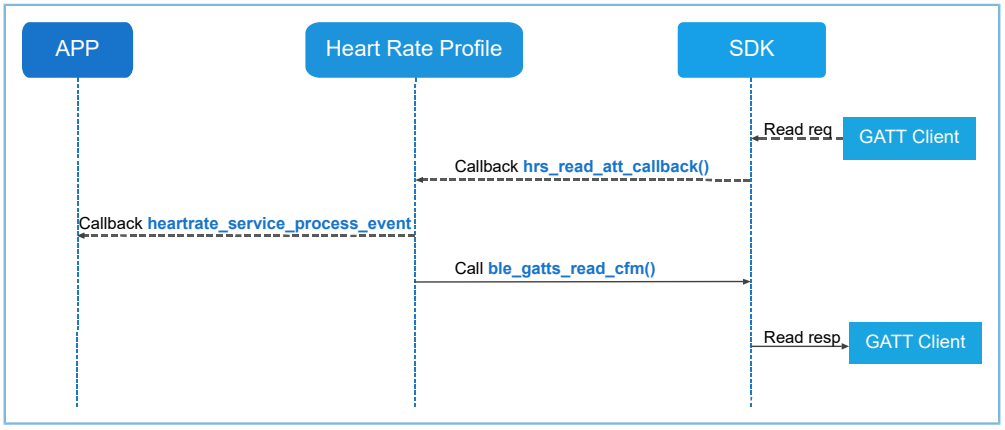

图 3-5 客户端read request处理流程

1. 当收到对端GATT客户端发送的读取某个属性的请求时，协议栈首先会验证该属性是否具有读权限。
2. 如果该属性具有可读权限并且其属性值存放在用户空间，协议栈将调用profile的读回调函数（profile读回调函数必须存在，否则对端会发生读超时的事件）。
3. Profile的读回调函数根据需要调用应用的事件处理函数。
4. Profile的读回调函数调用ble_gatts_read_cfm接口将属性值传递给SDK，经由协议栈发送给GATT 客户端。

   SDK_Folder\components\profiles\hrs\hrs.c中的read callback代码片段如下：

```
switch (tab_index)
{
    case HRS_IDX_HR_MEAS_VAL:
        cfm.length = HRS_MEAS_MAX_LEN;
        fm.value  = m_hrs_env.hr_meas.hr_meas_value;
        break;

    case HRS_IDX_HR_MEAS_NTF_CFG:
        cfm.length = sizeof(uint16_t);
        cfm.value  = (uint8_t *)(&(m_hrs_env.ntf_cfg[conn_idx]));
        break;

    case HRS_IDX_BODY_SENSOR_LOC_VAL:
        if (s_hrs_env.hrs_init.evt_handler)
        {
            evt.conn_idx = conn_idx;
            evt.evt_type = HRS_EVT_READ_BODY_SEN_LOCATION;
            s_hrs_env.hrs_init.evt_handler(&evt);
        }
        cfm.length = sizeof(uint8_t);
        cfm.value  = (uint8_t *)(&s_hrs_env.hrs_init.sensor_loc);
        break;

    default:
        cfm.length = 0;
        cfm.status = BLE_ATT_INVALID_HANDLE;
        break;
}

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

3.4.2.4 客户端写请求处理

以Heart rate profile为例，服务端处理客户端读请求的交互流程如图 3-6所示：

版权所有 © 2020 深圳市汇顶科技股份有限公司

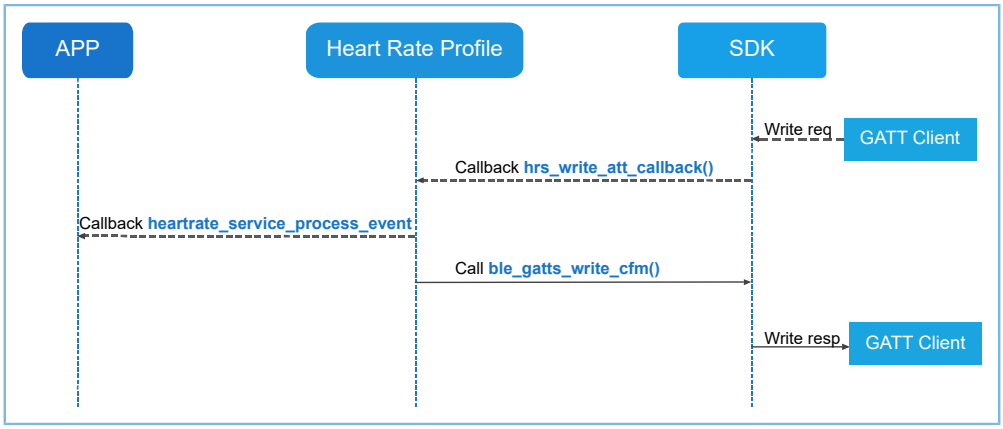

图 3-6 客户端 write request 处理流程

1. 当收到对端GATT客户端发送的写入某个属性的请求时，协议栈会验证该属性是否具有写权限。
2. 如果该属性有写权限，协议栈将调用profile的写回调函数（若profile写回调函数存在）。
3. Profile的写回调函数根据需要调用应用的事件处理函数。事件处理函数将要写入的值保存起来，如果写入的是CCCD值那么profile应该进行notify或者indicate逻辑处理。
4. Profile写回调函数会调用ble_gatts_write_cfm接口将属性值写入操作的执行状态传递给SDK，经由协议栈发送给GATT客户端。

SDK_Folder\components\profiles\hrs\hrs.c中的write callback代码片段如下：

```
switch (tab_index)
{
    case HRS_IDX_HR_MEAS_VAL:
        cfm.length = HRS_MEAS_MAX_LEN;
        cfm.value  = s_hrs_env.hr_meas.hr_meas_value;
        break;

    case HRS_IDX_HR_MEAS_NTF_CFG:
        cfm.length = sizeof(uint16_t);
        cfm.value  = (uint8_t *)(&(s_hrs_env.ntf_cfg[conn_idx]));
        break;

    case HRS_IDX_BODY_SENSOR_LOC_VAL:
        if (s_hrs_env.hrs_init.evt_handler)
        {
            evt.conn_idx = conn_idx;
            evt.evt_type = HRS_EVT_READ_BODY_SEN_LOCATION;
            s_hrs_env.hrs_init.evt_handler(&evt);
        }
        cfm.length = sizeof(uint8_t);
        cfm.value  = (uint8_t *)(&s_hrs_env.hrs_init.sensor_loc);
        break;

    default:
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
        cfm.length = 0;
        cfm.status = BLE_ATT_ERR_INVALID_HANDLE;
        break;
}

ble_gatts_read_cfm(conn_idx, &cfm);
```

3.5 GATT安全

GATT服务端可独立地定义每个特征的权限：可允许任何客户端都能访问某个特征，或者只允许认证或授权的客户端访问某个特征。特征权限通常被定义为更上层Profile规范的一部分。对于自定义的profile而言，用户可自行选择合适的权限。

关于GATT安全的更多信息，请参考Bluetooth Core Spec v5.1中的“8 Security considerations（Vol 3，Part G）”。

3.5.1 认证

对于那些要求认证的特征，客户端需要通过认证配对流程后才能访问。协议栈处理权限的验证和访问控制，无需APP参与处理，仅需要在注册服务的时候，申明相关服务属性的访问认证要求。

例如，在SDK_Folder\components\profiles\hrs\hrs.c中，将服务的Heart Rate Measurement特征值“通知权限”（NOTIFY_PERM_UNSEC）更改为“读取需要认证”（READ _PERM(AUTH)），更改后的代码如下：

```
// HR Measurement Characteristic - Value
[HRS_IDX_HR_MEAS_VAL] = {BLE_ATT_CHAR_HEART_RATE_MEAS,
                         READ_PERM(AUTH),
                         ATT_VAL_LOC_USER,
                         HRS_MEAS_MAX_LEN},
```

当未经认证的客户端尝试读取该特征值时，GATT服务端自动地拒绝该访问并返回BLE_ATT_ERR_INSUFF_AUTHEN(0x05)错误码，不会调用用户定义的read callback函数。

当通过认证的客户端尝试读取该特征值时，读取请求将被传到Profile的read callback函数去处理。

3.5.2 授权

授权是用户控制属性访问的一种权限，即用户决定授权哪些属性可以被访问。属性的授权访问是在应用层由用户实施的。协议栈验证用户在属性表中配置的权限后（假设验证通过），将读写请求转给用户的读写callback函数处理。

如果用户不对客户端的某条属性的读写授权，则需在callback函数中设置授权不足的错误码BLE_ATT_ERR_INSUFF_AUTHOR(0x08)。

版权所有 © 2020 深圳市汇顶科技股份有限公司

4 安全管理（SM）

安全管理（SM），是蓝牙用来进行安全管理的，其定义了配对和密钥分发的实现过程。SM的模块组成如图 4-1所示：

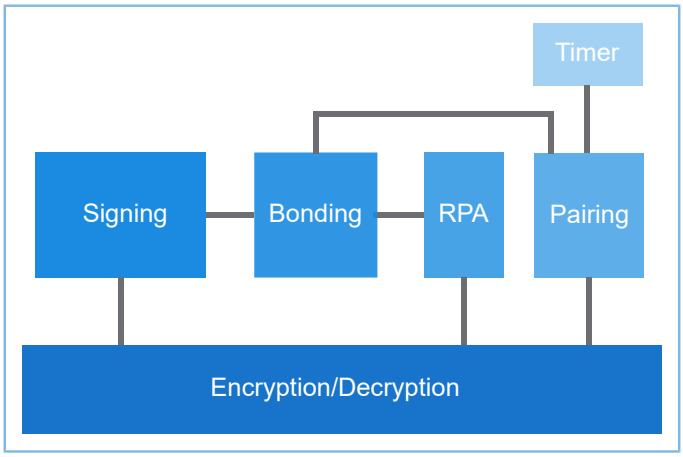

图 4-1 SM 模块结构

4.1 配对

配对过程，是用来建立密钥的，即完成双方密钥协商，就密钥一事达成共同一致的过程。

配对分为3个阶段：

1. 交换配对信息。
2. 生成链路加密密钥。
3. 在加密链路上分发其它指定的密钥信息，并根据设备是否支持绑定决定是否需要在安全数据库中存储分发的密钥信息。

版权所有 © 2020 深圳市汇顶科技股份有限公司

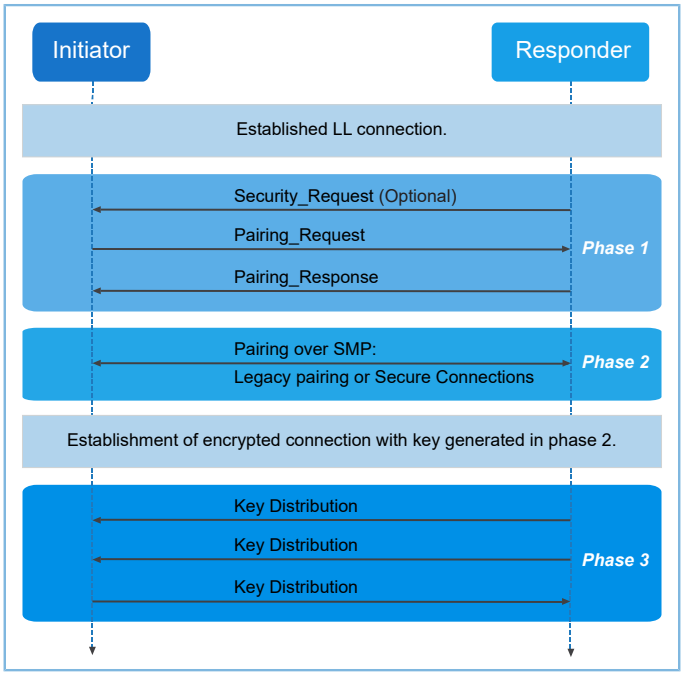

图 4-2 BLE配对过程

4.1.1 配对方法的选择

蓝牙4.2及之后的版本新增了安全性更高的SC（Secure Connections）配对，而4.1及更早版本的配对方法则称为LE（Legacy）配对。二者主要的区别在于SC配对中引入了Elliptic Curve Diffie-Hellman加密算法，而LE配对中则没有。根据设备所支持的安全特性，配对方法可分为如下四种：

- Just Works（Secure Connections or Legacy）
- Passkey Entry（Secure Connections or Legacy）
- Numeric Comparison（Secure Connections）
- Out of Band（Secure Connections or Legacy）

用户可根据以下规则来选择配对方法：

- 如果两个设备都支持SC配对，则采用图 4-3的规则来决定下一步的抉择。

		Initiator			
		OOB Set	**OOB Not Set**	**MITM Set**	**MITM Not Set**
Responder	**OOB Set**	Use OOB	Use OOB		
	OOB Not Set	Use OOB	Check MITM		
	MITM Set			Use IO Capabilities	Use IO Capabilities
	MITM Not Set			Use IO Capabilities	Use Just Works

图 4-3 SC配对规则

- 如果两个设备至少有一个设备不支持SC配对，则采用图 4-4规则来决定下一步的抉择。

版权所有 © 2020 深圳市汇顶科技股份有限公司

		Initiator			
		OOB Set	**OOB Not Set**	**MITM Set**	**MITM Not Set**
Responder	**OOB Set**	Use OOB	Check MITM		
	OOB Not Set	Check MITM	Check MITM		
	MITM Set			Use IO Capabilities	Use IO Capabilities
	MITM Not Set			Use IO Capabilities	Use Just Works

图 **4-4 LE**配对规则

其中IO能力与配对方法的映射关系如图 4-5所示：

	Initiator				
Responder	**DisplayOnly**	**Display YesNo**	**Keyboard Only**	**NoInput NoOutput**	**Keyboard Display**
Display Only	Just Works Unauthenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated
Display YesNo	Just Works Unauthenticated	Just Works (For LE Legacy Pairing) Unauthenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): responder displays, initiator inputs Authenticated
		Numeric Comparison (For LE Secure Connections) Authenticated			Numeric Comparison (For LE Secure Connections) Authenticated
Keyboard Only	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry: initiator and responder inputs Authenticated	Just Works Unauthenticated	Passkey Entry: initiator displays, responder inputs Authenticated
NoInput NoOutput	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated
Keyboard Display	Passkey Entry: initiator displays, responder inputs Authenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated	Passkey Entry: responder displays, initiator inputs Authenticated	Just Works Unauthenticated	Passkey Entry (For LE Legacy Pairing): initiator displays, responder inputs Authenticated
		Numeric Comparison (For LE Secure Connections) Authenticated			Numeric Comparison (For LE Secure Connections) Authenticated

图 **4-5 IO**能力与配对方法的映射关系图

4.1.2 配对方法的配置

本节介绍如何通过配置安全参数来使用不同的配对方法。

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

配对流程示例ble_app_sm_initiator与ble_app_sm_responder（SDK_Folder\projects\ble\ble_basic_example\），默认采用的是Just Works配对方式。其他的配对方式可以通过修改这两个示例工程中的安全参数s_sec_param来实现，具体的参数修改参照对应小节中的代码片段。

4.1.2.1 Just Works配对

如果双方设备都无需带中间人（Man-in-the-middle，MITM）认证，那么就可以使用Just Works配对流程。由于在Just Works 配对过程中无需MITM认证，因此也就无法抵御MITM攻击。Just Works 配对既可以是LE配对也可以是SC配对，用户只需在初始时配置好安全参数，便可启动配对流程，配对过程中无需用户的任何交互。

```
static sec_param_t s_sec_param =
{
    .level = SEC_MODE1_LEVEL1,
    .io_cap = IO_DISPLAY_ONLY,
    .oob = false,
    .auth = AUTH_NONE,
    .key_size = 16,
    .ikey_dist = KDIST_ENCKEY,
    .rkey_dist = KDIST_ENCKEY,
};
ble_sec_params_set(&s_sec_param);
```

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user\user_app.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user\user_app.c

4.1.2.2 Passkey Entry配对

Passkey Entry配对流程支持MITM认证，既可用于LE配对，也可用于SC配对。用户可以通过配置如下的安全参数启动Passkey Entry配对流程，并且在配对过程中需要用户输入6位十进制数的密码。

```
static sec_param_t s_sec_param =
{
    .level = SEC_MODE1_LEVEL1,
    .io_cap = IO_KEYBOARD_ONLY,
    .oob = false,
    .auth = AUTH_MITM,
    .key_size = 16,
    .ikey_dist = KDIST_ENCKEY,
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
    .rkey_dist = KDIST_ENCKEY,
};
ble_sec_params_set（&s_sec_param）;
```

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user\user_app.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user\user_app.c

4.1.2.3 Numeric Comparison 配对

Numeric Comparison配对流程只用于SC配对。如果双方设备都支持SC配对，IO能力都具有显示和输入功能，且需带MITM认证，则可以使用Numeric Comparison配对流程。具体的安全参数配置可参考如下：

```
static sec_param_t s_sec_param =
{
    .level = SEC_MODE1_LEVEL1,
    .io_cap = IO_DISPLAY_YES_NO,
    .oob = false,
    .auth = AUTH_BOND | AUTH_MITM | AUTH_SEC_CON,
    .key_size = 16,
    .ikey_dist = KDIST_ENCKEY,
    .rkey_dist = KDIST_ENCKEY,
};
ble_sec_params_set（&s_sec_param）;
```

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user\user_app.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user\user_app.c

4.1.2.4 关闭配对功能

用户可以通过SDK接口设置pair的开关为false来关闭配对功能。如果关闭了配对功能，则协议栈会拒绝所收到的任何配对请求数据包。

```
ble_gap_pair_enable_set（false）;
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user\user_app.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user\user_app.c

4.2 绑定

绑定的密钥信息用于实现重连之后对链路进行再次加密、数据的签名认证以及地址解析的功能。重连之后再次加密时，如果是绑定过的设备则会直接使用绑定的密钥信息进行链路加密，如果是未绑定的设备则会重新发起配对流程。

4.2.1 开启绑定功能

开启绑定功能时，APP与BLE Stack之间的交互流程如图 4-6所示。

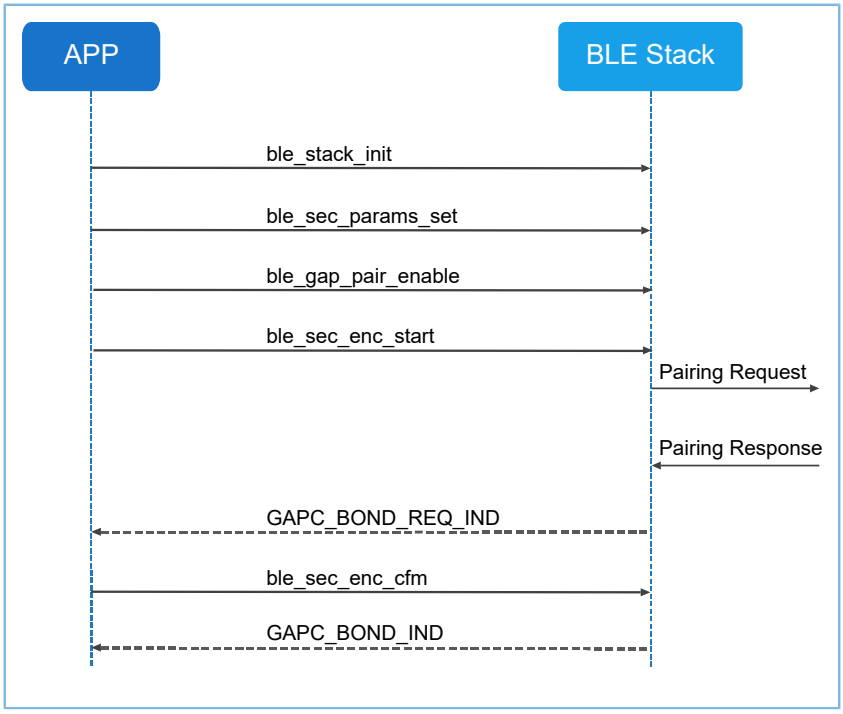

图 4-6 绑定交互流程

开启绑定功能的具体步骤如下：

说明:

步骤中的代码片段出自于配对流程示例ble_app_sm_initiator和ble_app_simple_sm_responder （SDK_Folder\projects\ble\ble_basic_example\）。

1. 配置安全参数，包括设备的IO能力，是否绑定，是否支持MITM，以及所需要分发的key等参数。

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
//set the default security parameters.
static sec_param_t s_sec_param =
{
    .level = SEC_MODE1_LEVEL1,
    .io_cap = IO_DISPLAY_ONLY,
    .oob = false,
    .auth = AUTH_BOND,
    .key_size = 16,
    .ikey_dist = KDIST_ENCKEY,
    .rkey_dist = KDIST_ENCKEY,
};
ble_sec_params_set（&s_sec_param）;
```

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user\user_app.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user\user_app.c

2. 开启配对的功能。

```
ble_gap_pair_enable（true）;
```

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user_callback\user_sm_callback.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user_callback\user_sm_callback.c

3. 实现并注册相关callback函数。

(1) 实现callback函数。

```
static void app_sec_rcv_enc_req_cb(uint8_t conn_idx, sec_enc_req_t *p_enc_req)
{
    APP_LOG_DEBUG("rcv enc req cb\n");

    const uint32_t tk = 123456;
    sec_cfm_enc_t cfm_enc;

    memset((uint8_t *)&cfm_enc, 0, sizeof(sec_cfm_enc_t));

    if (NULL == p_enc_req)
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
    {
        return;
    }
   switch (p_enc_req->req_type)
   {
       case PAIR_REQ:
       {
           APP_LOG_DEBUG("pair req\n");
           cfm_enc.req_type = PAIR_REQ;
           cfm_enc.accept = true;
           break;
        }
        case TK_REQ:
        {
            APP_LOG_DEBUG("tk req\n");
            cfm_enc.req_type = TK_REQ;
            cfm_enc.accept = true;
            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;
        }
        case OOB_REQ:
        {
            APP_LOG_DEBUG("oob req\n");
            break;
        }
        case NC_REQ:
        {
            APP_LOG_DEBUG("nc req\n");
            uint32_t num = *(uint32_t *)(p_enc_req->data.nc_data.value);
            APP_LOG_DEBUG("num=%d\n", num);
            cfm_enc.req_type = NC_REQ;
            cfm_enc.accept = true;
            break;
        }
        default:
            break;
    }
    ble_sec_enc_cfm(conn_idx, &cfm_enc);
}
......
    const sec_cb_fun_t app_sec_callback = {
    .app_sec_enc_req_cb = app_sec_enc_req_cb,
    .app_sec_enc_ind_cb = app_sec_enc_ind_cb,
    .app_sec_keypress_notify_cb = NULL
};
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user_callback\user_sm_callback.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user_callback\user_sm_callback.c

(2) 注册callback函数。

```
static app_callback_t m_app_ble_callback =
{
    .app_ble_init_cmp_callback = ble_init_cmp_callback,
    .app_gap_callbacks         = &app_gap_callbacks,
    .app_sec_callback          = &app_sec_callback,
};
// Initialize ble stack.
ble_stack_init(&s_app_ble_callback, &heaps_table);
```

说明:

代码路径：

- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user\main.c
- SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_responder\Src\user\main.c

4. 在连接完成的callback中启动链路加密。

```
static void app_gap_connect_cb(uint8_t conn_idx, uint8_t status,
                               const gap_conn_cmp_t *p_conn_param)
{
    APP_LOG_DEBUG("Enter connect complete cb, conidx=%d\n", conn_idx);
    ble_sec_enc_start(conn_idx);
}
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_sm_initiator\Src\user_callback\user_gap_callback.c

版权所有 © 2020 深圳市汇顶科技股份有限公司

4.3 隐私管理

BLE中的隐私管理，是指已认证设备可以跟踪识别目标设备，而其他非认证设备无法跟踪识别目标设备。隐私管理使得已认证的设备可以正常地与目标设备进行连接和通讯，同时防止其他非认证设备、恶意破坏设备对目标设备的跟踪。

4.3.1 开启隐私管理功能

系统在初始化时会自动加载绑定的地址解析列表并配置给协议栈。用户需开启隐私管理功能并设置地址更新时间；在广播参数中设置对端身份地址，以便协议栈根据该身份地址寻找对应的解析列表，从而生成RPA地址。

开启隐私管理功能的具体步骤如下：

说明:

步骤中的代码片段出自于GR551x SDK的隐私流程示例ble_app_privacy_slave和ble_app_privacy_master（SDK_Folder\projects\ble\ble_basic_example\）。

---

1. 主设备与从设备进行连接并绑定，绑定过程中需要交换IRK以及身份地址信息，绑定之后主设备与从设备断开连接。绑定的具体流程请参考4.2 绑定。
2. 从设备隐私配置。

   (1) 设置隐私参数，RPA地址刷新时间设为150s。

```
ble_gap_privacy_params_set(150, true);
```

   (2) 从解析列表中获取对端的身份地址，比如解析列表中的第一个设备。

```
// get bond list
bond_dev_list_t bond_list;
memset(&bond_list, 0, sizeof(bond_dev_list_t));
ble_gap_bond_devs_get(&bond_list);

APP_LOG_DEBUG("bond list size = %d\n", bond_list.num);
APP_LOG_DEBUG("addr_type = %d\n", bond_list.items[0].addr_type);
for (uint8_t i = 0; i < 6; i++)
{
    APP_LOG_DEBUG("addr[%d] = 0x%x ", i,
                  bond_list.items[0].gap_addr.addr[i]);
}
APP_LOG_DEBUG("\n");
```

   (3) 在广播参数中设置对端的身份地址。

```
// set peer identity addr
memcpy(g_gap_adv_param.peer_addr.gap_addr.addr,
       bond_list.items[0].gap_addr.addr, 6);
g_gap_adv_param.peer_addr.addr_type = bond_list.items[0].addr_type;
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

(4) 重新设置广播参数，并开启广播。

```
// set adv param and start adv again
ble_gap_adv_param_set(0, BLE_GAP_OWN_ADDR_STATIC, &g_gap_adv_param);
ble_gap_adv_data_set(0, BLE_GAP_ADV_DATA_TYPE_DATA, s_adv_data_set,
                     sizeof(s_adv_data_set));
ble_gap_adv_data_set(0, BLE_GAP_ADV_DATA_TYPE_SCAN_RSP,
                     s_adv_rsp_data_set, sizeof(s_adv_rsp_data_set));
ble_gap_adv_start(0, &g_gap_adv_time_param);
```

说明:

(1) ～ (4)的代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_privacy_slave\Src\user_callback\user_gap_callback.c

3. 主设备隐私配置。

(1) 从解析列表中获取对端的身份地址，比如解析列表中的第一个设备。

```
// get bond list
bond_dev_list_t bond_list;
memset(&bond_list, 0, sizeof(bond_dev_list_t));
ble_gap_bond_devs_get(&bond_list);
APP_LOG_DEBUG("bond list size = %d\n", bond_list.num);
APP_LOG_DEBUG("addr_type = %d\n", bond_list.items[0].addr_type);
for (uint8_t i = 0; i < 6; i++)
{
    APP_LOG_DEBUG("addr[%d] = 0x%x ", i,
    bond_list.items[0].gap_addr.addr[i]);
}
APP_LOG_DEBUG("\n");
// save peer identity addr
memcpy(g_peer_iden_addr.gap_addr.addr,
        bond_list.items[0].gap_addr.addr, 6);
g_peer_iden_addr.addr_type = bond_list.items[0].addr_type;
```

说明:

代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_privacy_master\Src\user_callback\user_sm_callback.c

(2) 设置扫描参数，并开启隐私功能。

```
// set privacy params
ble_gap_privacy_params_set(150, true);

// start scan
gap_scan_param_t scan_param;
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

```
scan_param.scan_type = GAP_SCAN_ACTIVE;
scan_param.scan_mode = GAP_SCAN_GEN_DISC_MODE;
scan_param.scan_dup_filt = GAP_SCAN_FILT_DUPLIC_EN;
scan_param.use_whitelist = 1;
scan_param.interval= 15;
scan_param.window= 15;
scan_param.timeout = 0;
ble_gap_scan_param_set(BLE_GAP_OWN_ADDR_STATIC, &scan_param);
```

(3) 开启扫描。

```
ble_gap_scan_start();
```

(4) 在广播回调中设置连接参数并发起连接请求。

```
static void app_gap_adv_report_ind_cb(const gap_ext_adv_report_ind_t  *p_adv_report)
{
    if (memcmp(p_adv_report->broadcaster_addr.gap_addr.addr,
        g_peer_iden_addr.gap_addr.addr, 6) == 0)
    {
        APP_LOG_DEBUG("scan success\n");
        // connect peer device again
        gap_init_param_t conn_param;
        memcpy(conn_param.peer_addr.gap_addr.addr,
               g_peer_iden_addr.gap_addr.addr, 6);
        conn_param.peer_addr.addr_type = g_peer_iden_addr.addr_type;
        conn_param.type = GAP_INIT_TYPE_DIRECT_CONN_EST;
        conn_param.interval_min = 6;
        conn_param.interval_max = 10;
        conn_param.slave_latency = 1;
        conn_param.sup_timeout = 100;
        ble_gap_connect(BLE_GAP_OWN_ADDR_STATIC, &conn_param);
    }
}
```

说明:

(2) ～ (4)的代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_privacy_master\Src\user_callback\user_gap_callback.c

4.3.2 地址配置说明

当开启广播、扫描、建立连接时，根据不同的配置需求，controller可能会使用不同的地址发送空口数据包。下面将以广播为例，介绍如何配置空口上的使用地址。

1. 使用ble_gap_addr_set接口可以设置设备的身份地址为public地址或者static地址。如果用户没有调用过该接口，并且eFuse/NVDS也不存在默认的public地址，那么默认会根据芯片的chip UUID产生static地址作为设备的身份地址。

版权所有 © 2020 深圳市汇顶科技股份有限公司

2. ble_gap_adv_param_set接口的传参*own_addr_type*可配置host层设置给controller的地址。

   (1) BLE_GAP_OWN_ADDR_STATIC：设置身份地址给controller，即步骤1中所配置的身份地址。

   (2) BLE_GAP_OWN_ADDR_GEN_RSLV：host层产生的rpa地址设置给controller。

   (3) BLE_GAP_OWN_ADDR_GEN_NON_RSLV：host层产生的non-rpa地址设置给controller。

3. ble_gap_privacy_params_set的传参*enable_flag*可配置是否开启隐私。

   (1) 如果没有开启隐私，则空口使用host层配置给controller的地址，即步骤2中所设置的地址。

   (2) 如果开启了隐私，controller通过广播参数所传下来的peer addr信息查找解析列表。如果查找失败，空口则直接使用host层配置给controller的地址。

   (3) 如果开启了隐私，并且controller通过广播参数所传下来的peer addr信息查找解析列表，如果查找成功，则使用controller根据peer addr的IRK信息自动生成的rpa地址。

版权所有 © 2020 深圳市汇顶科技股份有限公司

5 逻辑链路控制和适配协议（L2CAP）

逻辑链路控制和适配协议（L2CAP），是蓝牙系统中的核心协议，主要负责ACL（Asynchronous Connectionless）数据的收发、重组、拆包。此外，还支持通过L2CAP发送信令数据包来创建面向连接的信道COC（Connection Oriented Channel），L2CAP的结构框图如图 5-1所示：

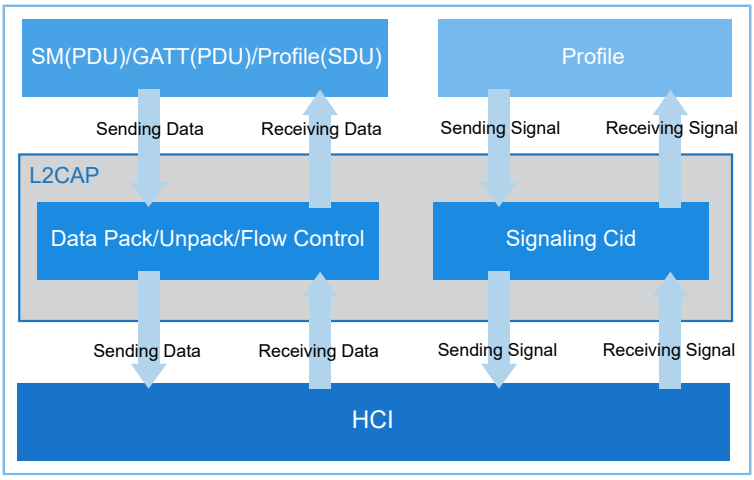

图 5-1 L2CAP结构框图

5.1 L2CAP数据包结构

服务数据单元SDU（Service Data Unit）是从高层信息单元传送到底层协议的，是针对于应用层的数据包，主要应用于通过COC创建动态信道的服务中。协议数据单元PDU（Protocol Data Unit）是指L2CAP层的数据包，一个SDU在L2CAP层可以拆分为一个或多个PDU。每个L2CAP层PDU数据包的有效载荷前端都包含一个32-bit的报头，那么数据包的长度信息必须包含在报头中，以便判断数据包的结束。

PDU数据包结构如图 5-2所示：

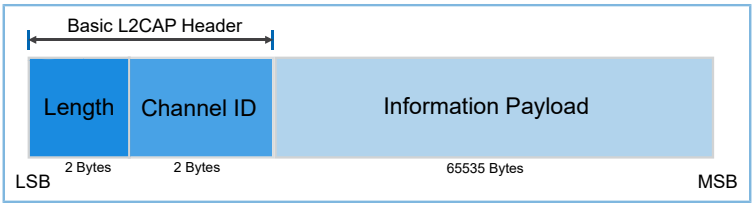

图 5-2 PDU数据包结构

SDU数据包结构如图 5-3所示：

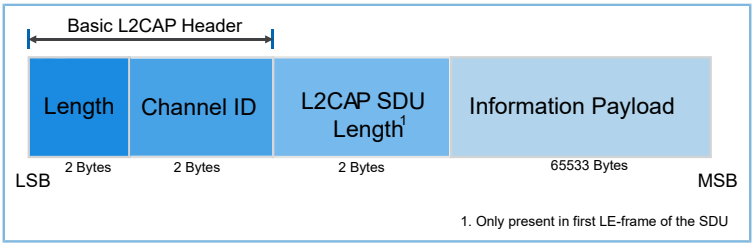

图 5-3 SDU数据包结构

报头包括2字节的长度字段和2字节的信道标识符。长度字段表示报头后的信息载荷字节数。需要注意的是，SDU的第一帧数据包报头之后的两个字节表示SDU有效载荷的长度。

版权所有 © 2020 深圳市汇顶科技股份有限公司

5.2 最大传输单元（MTU）

L2CAP负责分发上层协议（SM、GATT）的数据包，在L2CAP层所允许的最大数据包长度称为最大传输单元（Maximum Transmission Unit，MTU）。如果上层协议所传输的数据包过大，则需要在空口上进行拆包。

PDU根据controller的ACL_Data_Packet_Length的大小进行分片（fragment）。其分包示意图如图 5-4所示：

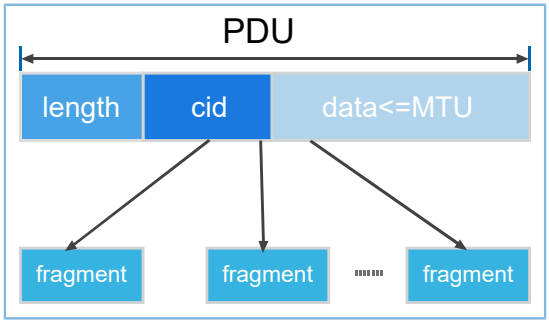

图 5-4 PDU分片示意图

SDU（服务数据单元）首先根据MPS进行分段（segment），然后每一个segment再根据ACL_Data_Packet_Length的大小进行分片（fragment）。其分包示意图如图 5-5所示：

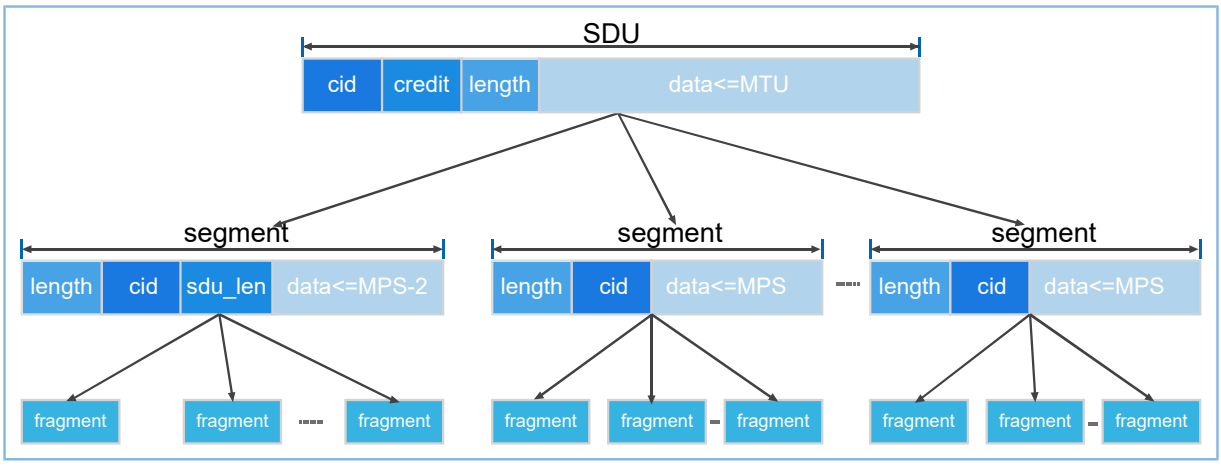

图 5-5 SDU分片示意图

配置MTU和MPS的参考代码如下：

```
// mtu:23～2048， mps:23～mtu， lecb_conn_num: 0x00～0x20
error_code = ble_gap_l2cap_params_set(512, 250, 10);
APP_ERROR_CHECK(error_code);
```

说明:

代码路径：`SDK_Folder\projects\ble\ble_basic_example\ble_app_l2cap_coc_server\Src\user\user_app.c`

5.3 L2CAP信道

低功耗蓝牙既支持固定信道（0x0004 ～ 0x0006），也支持动态创建的面向连接的信道。其中，固定信道指的是两个设备一建立连接就已经存在的、没有任何配置参数的信道。

版权所有 © 2020 深圳市汇顶科技股份有限公司

表 5-1 列出了L2CAP的信道标识符，每个信道标识符为一个16位的数字。信道标识符0x0000为保留标识符，不能被使用。信道标识符0x0001为经典蓝牙信令的固定信道。

表 5-1 L2CAP信道标识符

信道标识符	描述
0x0000	不能使用
0x0001 - 0x0003	保留：日后可能使用
0x0004	属性协议
0x0005	低功耗信令信道
0x0006	安全管理协议
0x0007 - 0x001F	保留：日后可能使用
0x0020 - 0x003E	Bluetooth SIG固定分配的
0x003F	保留：日后可能使用
0x0040 - 0x007F	面向连接信道

5.4 面向连接的信道COC

面向连接的信道COC（Connection Oriented Channel），是L2CAP控制器的一个主要特征。它允许一个LE服务在指定的链路上创建一个专用的信道，服务端和客户端在交换任何数据之前都需要先创建一个COC信道。COC的最大优势是可以通过配置MTU和MPS使得应用层可以发送长包数据，以便提高系统吞吐率，典型的应用为Internet Protocol Support Profile（IPSP）以及Object Transfer Profile（OTP）。

在创建COC信道的过程中，客户端基于指定的PSM（Protocol/Service Multiplexer）发起一个创建COC信道的请求，服务端为了能接受该请求，必须在应用层注册该PSM。任何一个基于未注册的PSM发起的创建COC信道的请求，在服务端都会被直接忽略。PSM可分为固定段和动态分配段，如表 5-2 所示：

表 5-2 PSM分类

范围	类型	描述
0x0001 - 0x007F	Bluetooth SIG所分配的PSM	Bluetooth SIG为已有的标准服务所分配的PSM号
0x0080 - 0x00FF	动态分配的自定义PSM	自定义服务所指定的PSM
0x0100 - 0xFFFF	保留	保留

在注册PSM时还可指定服务的认证权限。

```
gap_lepsm_register_t param;
param.le_psm = 0x25;
param.sec_lvl = 0x00;
param.mks_flag = false;
error_code = ble_gap_lepsm_register(&param);
APP_ERROR_CHECK(error_code);
```

版权所有 © 2020 深圳市汇顶科技股份有限公司

说明:

代码路径：

SDK_Folder\projects\ble\ble_basic_example\ble_app_l2cap_coc_server\Src\user\user_app.c

5.4.1 创建COC流程

低功耗蓝牙协议栈SDK提供API来创建L2CAP COC信道，以在支持此功能的两个设备之间双向传输数据。

创建COC时，APP与BLE Stack之间的交互流程如图 5-6所示。

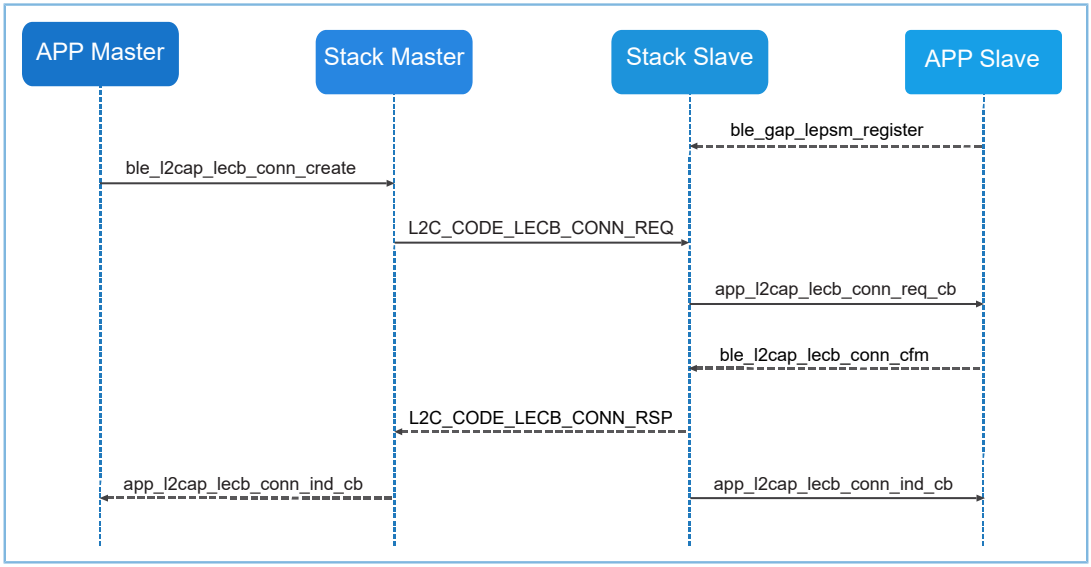

图 5-6 COC创建流程图

创建COC的具体步骤如下：

1. 步骤中的代码片段出自于GR551x SDK的COC创建流程示例ble_app_l2cap_coc_server（SDK_Folder\projects\ble\ble_basic_example\）在初始化时，服务端的应用程序首先向GAP注册PSM。

```
gap_lepsm_register_t param;
param.le_psm = 0x25;
param.sec_lvl = 0x00;
param.mks_flag = false;
error_code = ble_gap_lepsm_register(&param);
APP_ERROR_CHECK(error_code);
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_l2cap_coc_server\Src\user\user_app.c

2. 建立链路连接之后，客户端通过指定的PSM发起一个创建COC信道的请求。

```
lecb_conn_req_t conn_req;
```

```
conn_req.le_psm = psm;
conn_req.local_credits = 0xffff;
conn_req.local_cid = 0;
conn_req.mtu = 512;
conn_req.mps = 230;
ble_l2cap_lecb_conn_create(0, &conn_req);
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_l2cap_coc_client\Src\user_callback\user_gap_callback.c

3. 服务端收到该请求之后转发给应用层。

4. 应用层确认是否接受请求，并且发送创建COC信道的响应消息给对方。如果接受了该连接请求，则表示创建了一个COC信道。

```
static void app_l2cap_lecb_conn_req_cb(uint8_t conn_idx,
                                       lecb_conn_req_ind_t *p_conn_req)
{
    APP_LOG_DEBUG("app rcv lecb con req\n");
    APP_LOG_DEBUG("peer_mtu = %d, peer_mps = %d\n", p_conn_req->peer_mtu,
                  p_conn_req->peer_mps);
    lecb_cfm_conn_t cfm_conn;
    cfm_conn.accept = true;
    cfm_conn.peer_cid = p_conn_req->peer_cid;
    cfm_conn.local_credits = 0xffff;
    cfm_conn.local_cid = 0;
    cfm_conn.mtu = 512;
    cfm_conn.mps = 230;
    ble_l2cap_lecb_conn_cfm(conn_idx, &cfm_conn);
}
```

说明:

代码路径：SDK_Folder\projects\ble\ble_basic_example\ble_app_l2cap_coc_server\Src\user_callback\user_l2cap_callback.c

版权所有 © 2020 深圳市汇顶科技股份有限公司

6 术语和缩略语

表 6-1 术语和缩略语

名称	描述
ACL	Asynchronous Connectionless，无连接的异步连接
ATT	Attribute Protocol，通用接入层
BLE	Bluetooth Low Energy，低功耗蓝牙
CCCD	Client Characteristic Configuration Descriptor，客户端特征配置描述符
COC	Connection Oriented Channel，面向连接的信道
GAP	Generic Access Profile，通用访问规范
GATT	Generic Attribute Profile，通用属性规范
GFSK	Gauss frequency Shift Keying，高斯频移键控
HAL	Hardware Abstract Layer，硬件抽象层
HCI	Host-Controller Interface，主机控制接口
IPSP	Internet Protocol Support Profile，网络协议支持配置文件
L2CAP	Logical Link Control and Adaption Protocol，逻辑链路控制与适配协议
LE	Legacy，传统
LECB	LE Credit Based Connection，基于LE信用的连接
LL	Link Layer，链路层
LSB	Least Significant Bit，最低有效位
MIC	Message Integrity Check，加密消息完整性检查
MITM	Man-in-the-middle，中间人
MSB	Most Significant Bit，最高有效位
MTU	Maximum Transmission Unit，最大传输单元
NVDS	Non-volatile Data Storage，非易失性数据存储
OTP	Object Transfer Profile，对象传输配置文件
PDU	Protocol Data Unit，协议数据单元
PHY	Physical Layer，物理层
SC	Secure Connections，安全连接
SDK	Software Development Kit，软件开发工具包
SDU	Service Data Unit，服务数据单元
SM	Security Manager，安全管理器
SoC	System on Chip，片上系统
UUID	Universally Unique Identifier，通用唯一标识符

版权所有 © 2020 深圳市汇顶科技股份有限公司