BLE Central 设计说明

发布 ***1.0.0***

2018 年 **11** 月 **02** 日

目录

CHAPTER 1

概述

1.1 文档目的

通过一个简单的示例介绍 BLE Central 的编写方法，供用户学习参考。

1.2 术语说明

表 1.1: 术语说明

术语	说明
BLE	蓝牙低功耗技术
GAP	通用访问配置文件
GATT	通用属性配置文件
PROFILE	本文特指通用属性配置文件
CENTRAL	中心设备
PERIPHERAL	外围设备
SERVER	服务器
CLIENT	客户端
CCC	客户端特性描述符
API	应用程序编程接口
RMC	遥控器

1.3 参考文档

- https://www.bluetooth.com/specifications/gatt

- https://www.bluetooth.com/specifications/bluetooth-core-specification

1.4 版本历史

表 1.2: 版本历史

日期	版本	注释	作者
2018-08-24	1.0	初始版本	ZS110A 项目组

CHAPTER 2

BLE Central 概述

2.1 BLE Central 简介

根据 GAP (通用访问配置文件) 中的定义，主动发起连接的设备称为 Central，作为链路层的主设备, 接受连接的设备称为 Peripheral，作为链路层的从设备。

在 GATT 中定义了两种角色：Server 和 Client。GATT Server 是向 GATT Client 提供数据服务的设备，GATT Client 是从 GATT Server 读写应用数据的设备。GATT 角色不与特定的 GAP 角色相关联。

但在多数应用中，Central 会实现 GATT Client 角色，Peripheral 会实现 GATT Server 角色，下面以该应用场景描述 Central-Peripheral 之间通信模型，如图所示：

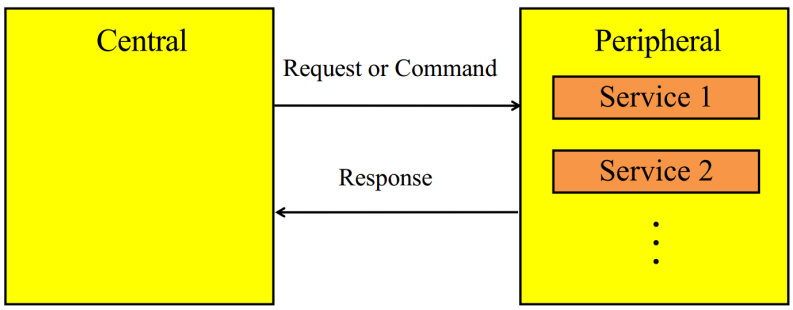

图 2.1: Central-Peripheral 通信模型

更多信息请登录 Bluetooth 官网:

- 核心规格: https://www.bluetooth.com/specifications/bluetooth-core-specification
- GATT 规格: https://www.bluetooth.com/specifications/gatt

CHAPTER 3

BLE Central 编写

对于所有的 BLE Central，它们几乎都遵循同一流程:

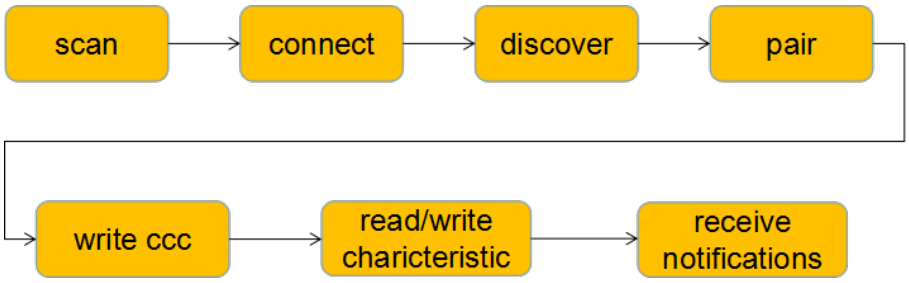

图 3.1: BLE Central 流程

下面以 sample\bluetooth\central_rmc 示例为例，说明 BLE Central 的程序流程。

3.1 central_rmc 示例简介

central_rmc 是 BLE Central 的一个示例，用于连接遥控器 (工程路径:samples\voice_rcu\keil5\peripheral_rmc)。它通过匹配广播数据包和扫描回应数据包中的设备名，对远端设备发起连接。连接成功后，发现服务，使能 notify, 最后接收并打印 notification 的内容。

3.2 central_rmc 程序流程

central_rmc 几乎和一般 BLE Central 的流程一致，如图所示:

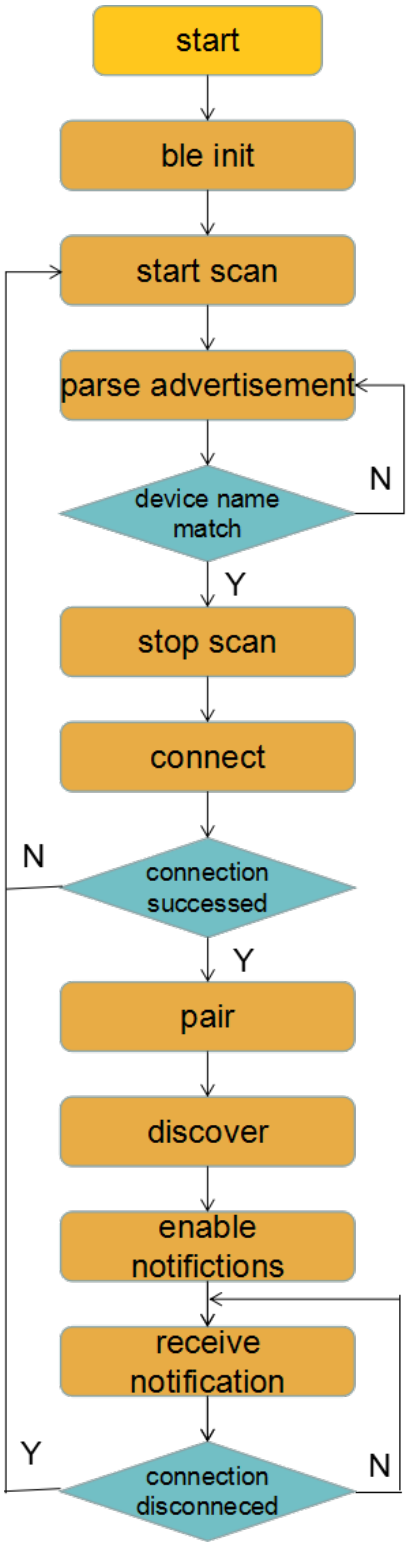

图 3.2: central_rmc 程序流程

在 central_rmc 中，首先调用 bt_enable 完成蓝牙协议栈的初始化，随后调用 start_scan 启动扫描。当空中接口扫描到广播包时，扫描回调函数 device_found 会被调用，用来解析扫描到的广播数据和扫描回应数据，当设备名和 RMC_NAME 相同时，它会先停止扫描，随后发起连接。连接成功后 connected 函数会被调用，它会发送一个类型为 MSG_CONNECTED 的异步消息，主函数接收到该消息后会调用 start_discover

启动发现。当发现到服务时 discover_func 会被调用，发现完成后 discover_func 会发送类型为 MSG_DISCOVER_COMPLETE 异步消息，主函数接收到该消息后会调用 nofity_enable 开启所有通知，并将所有开启的通知注册进协议栈，至此所有的命令和动作都执行完毕。当远端设备有通知上报时，通知回调函数 notify_func 会被调用，打印上报的数据。

注意: 为保证所有请求/命令的同步，务必不要在回调函数中执行 BLE 命令或请求，须采用异步消息的方式在主函数中进行操作。

3.3 central_rmc 配置

1. 目标设备名称配置

   central_rmc 根据广播中的设备名对远端设备发起连接，修改方法:

```
#define RMC_NAME "BLE_YT_RMC"
```

2. 扫描参数配置

   默认采用主动扫描方式，type 等于 BT_HCI_LE_SCAN_ACTIVE，该扫描方式可接收扫描回应，但不可接收定向广播包，相反的有被动扫描，type 等于 BT_HCI_LE_SCAN_PASSIVE，被动扫描无扫描回应，可以收到定向广播包。filter_dup 用于过滤重复广播包，当值为 BT_HCI_LE_SCAN_FILTER_DUP_DISABLE 时不进行过滤，当值为 BT_HCI_LE_SCAN_FILTER_DUP_ENABLE 时使能过滤，应用将收不到相同广播包。

```
struct bt_le_scan_param scan_param = {
        .type = BT_HCI_LE_SCAN_ACTIVE,
        .filter_dup = BT_HCI_LE_SCAN_FILTER_DUP_DISABLE,
        .interval = BT_GAP_SCAN_FAST_INTERVAL,
        .window = BT_GAP_SCAN_FAST_WINDOW
};
```

3. 安全等级配置

```
#define RMC_SECURITY_LEVEL BT_SECURITY_MEDIUM
```

4. 发现参数配置

   发现过程分为两种: 根据 UUID 发现服务/特性和发现所有服务/特性。发现的类型有 5 种: 主服务、次要服务、包含服务、特性和描述符。当 uuid 为 NULL 时发现所有服务/特性，为具体 UUID 时则发现特定服务/特性。start_handle 和 end_handle 用于限定发现服务/特性的 handle 的范围。

```
void start_discover(void)
{
        int err;
        if (!default_conn) {
```

(continues on next page)

(续上页)

```
                printk("Not connected\n");
                return;
        }

        discover_params.uuid = NULL;
        discover_params.func = discover_func;
        discover_params.type = BT_GATT_DISCOVER_PRIMARY;
        discover_params.start_handle = 0x0001;
        discover_params.end_handle = 0xffff;
        err = bt_gatt_discover(default_conn, &discover_params);
        if (err)  {
                printk("Discover failed (err %d)\n", err);
        } else {
                printk("Discover pending\n");
        }
}
```

5. 可发现的服务最大个数配置

```
#define SVC_MAX_CNT 8
```

6. 服务的最大特性个数

```
#define CHRC_MAX_CNT 12
```

7. 通知配置

   当远端设备是遥控器时，配置下面宏为 1，当远端设备不是遥控器时，需要将该宏配置为 0。

```
#define SUBSCRIBE_RMC_ONLY 1
```

CHAPTER 4

BLE Central 常用 API

```c
/* 启动扫描 */
int bt_le_scan_start(const struct bt_le_scan_param *param, bt_le_
↪scan_cb_t cb)

/* 停止扫描 */
int bt_le_scan_stop(void)

/* 创建连接 */
struct bt_conn *bt_conn_create_le(const bt_addr_le_t *peer,
                const struct bt_le_conn_param *param)

/* 断开连接 */
int bt_conn_disconnect(struct bt_conn *conn, u8_t reason)

/* 发现 */
int bt_gatt_discover(struct bt_conn *conn, struct bt_gatt_
↪discover_params *params)

/* 配对 */
int bt_conn_security(struct bt_conn *conn, bt_security_t sec)

/* 开启通知 */
int bt_gatt_subscribe(struct bt_conn *conn, struct bt_gatt_
↪subscribe_params *params)

/* 关闭通知 */
int bt_gatt_unsubscribe(struct bt_conn *conn, struct bt_gatt_
↪subscribe_params *params)
```

(continues on next page)

(续上页)

```
/* 写特性值 */
int bt_gatt_write(struct bt_conn *conn, struct bt_gatt_write_
↪params *params)

/* 无回应写特性值 */
int bt_gatt_write_without_response(struct bt_conn *conn, u16_t␣
↪handle,
                   const void *data, u16_t length, bool sign);
/* 读特性值 */
int bt_gatt_read(struct bt_conn *conn, struct bt_gatt_read_
↪params *params)

/* 交换 MTU */
int bt_gatt_exchange_mtu(struct bt_conn *conn, struct bt_gatt_
↪exchange_params *params)

/* 更新连接参数 */
int bt_conn_le_param_update(struct bt_conn *conn, const struct␣
↪bt_le_conn_param *param)
```

List of Figures

List of Tables