WM_W800_蓝牙系统架构及 API 描述

V0.5

北京联盛德微电子有限责任公司（winner micro)

地址：北京市海淀区阜成路 67 号银都大厦 1802

电话：+86-10-62161900

公司网址：www.winnermicro.com

文档修改记录

版本	修订时间	修订记录	作者	审核
V0.1	2019/9/25	[C]创建文档	Wangm	
V0.2	2020/7/7	1. 增加设置蓝牙名称 AT 指令 2. 更改示例代码路径	Pengxg	
V0.3	2020/7/8	统一字体	Cuiyc	
V0.4	2020/8/12	增加传统蓝牙功能： 1. 2.6 传统蓝牙音频 2. 2.7 传统蓝牙免提电话 3. 3.4.4 传统蓝牙音频 4. 3.4.5 传统蓝牙免提电话	Pengxg	
V0.5	2020/8/17	增加 API 使用示例章节	Pengxg	

目录

1 引言

1.1 编写目的

本文档用于介绍 W800 蓝牙软件系统，硬件系统及其开发蓝牙应用参考，指导用户学习及理解 w800 的蓝牙开发。

1.2 预期读者

蓝牙应用开发人员，蓝牙协议栈维护人员及测试相关人员

1.3 术语定义

序号	术语/缩略语	说明/定义
1	BT	BlueTooth
2	BLE	Bluetooth Low Energy
3	HCI	Host Controller Interface
4	A2DP	Advanced Audio Distribution Profile
5	HFP	Hands-Free Profile

1.4 参考资料

《W800 芯片手册》

《蓝牙 Core spec4.0 及 4.2》

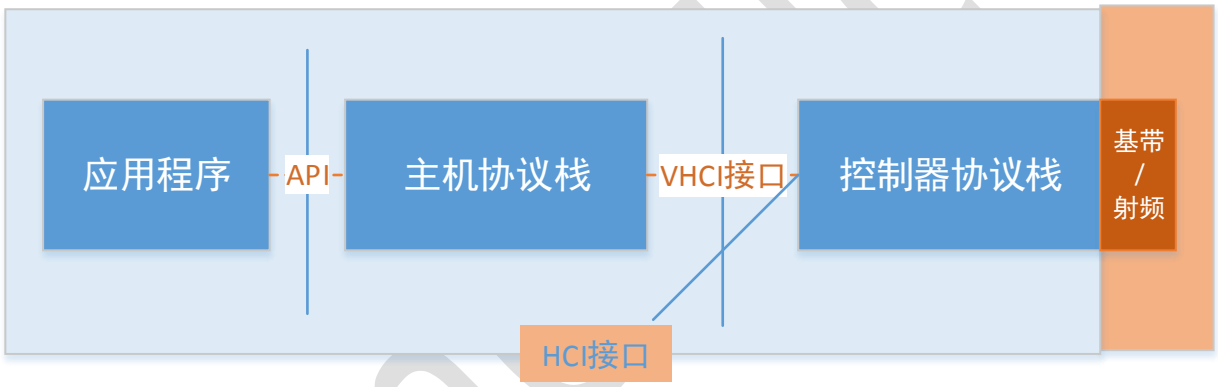
《蓝牙控制器 spec》

2 W800 蓝牙系统

2.1 芯片蓝牙设计框图

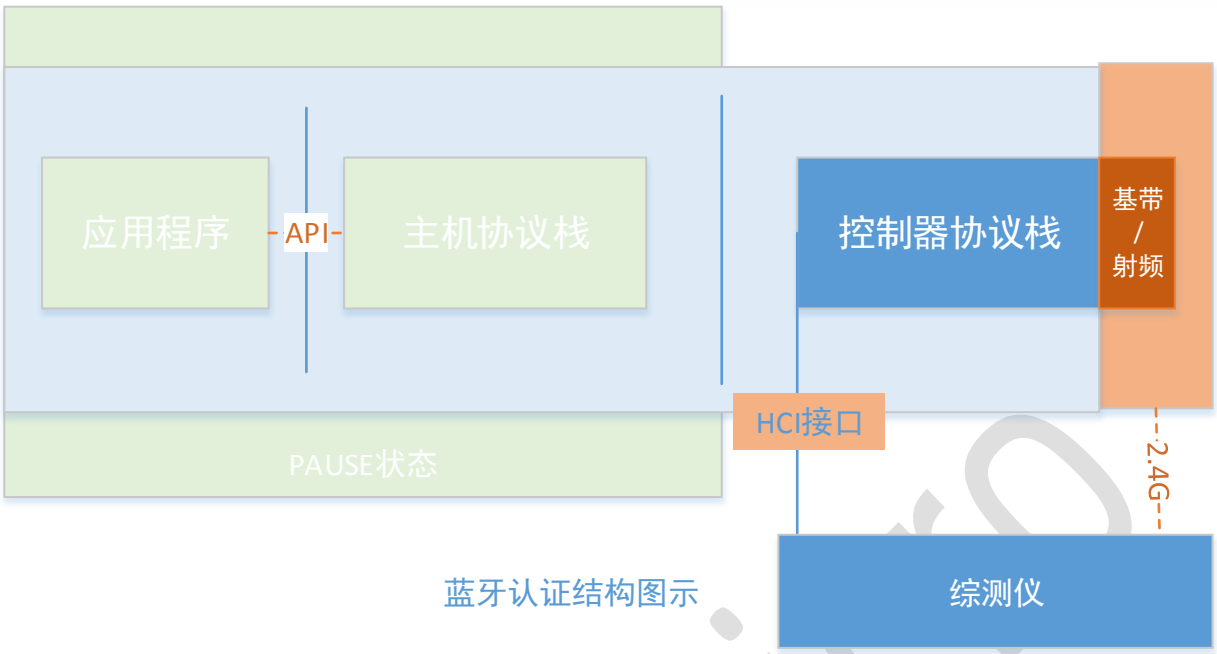
2.2 W800 蓝牙系统框图

W800 蓝牙系统可以分为应用程序部分、主机协议栈、控制器协议栈及蓝牙基带、射频构成。蓝牙的射频部分和 WiFi 系统共用。



蓝牙系统结构图示

认证的 HCI 串口操作指令参见传统蓝牙非信令测试及 BLE 非信令测试文档。具体测试方法如下图所示：



蓝牙认证结构图示

其中 W800 提供可配置的 UART 口，用于 HCI 指令的响应。综测仪通过 UART 口直接控制控制器。此时主机协议栈处于 freeze 状态。

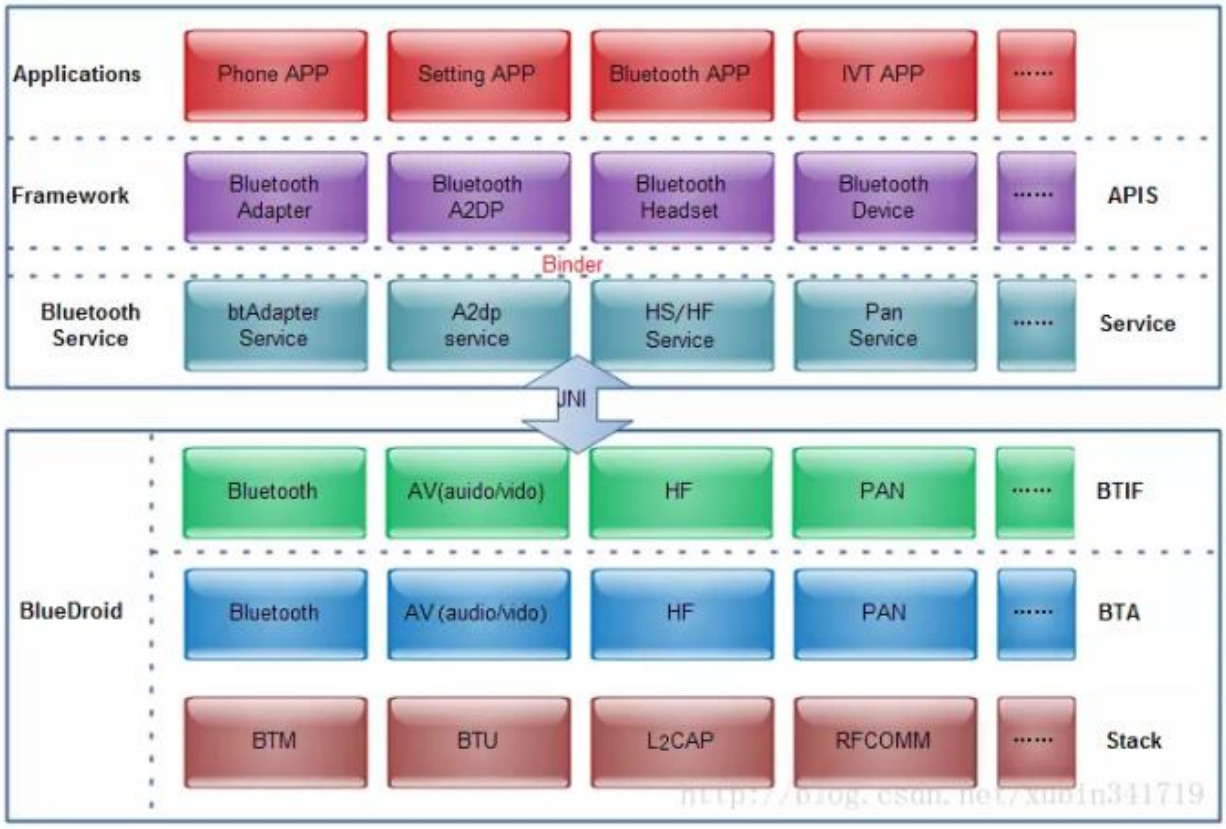
2.3 Bluedroid 介绍

2.3.1 bluedroid

Bluedroid 包含了传统蓝牙和低功耗蓝牙协议栈，同控制器交互采用标准的 HCI 协议。

我们移植了 bluedroid7.0，将内部的 task 替换为我们自己实现的微内核机制运行。

2.3.2 蓝牙 bluedroid 架构图



我们移植后，保留了 STACK,BTA 和简化的 BTIF 三层。用户开发应用程序直接基于 BTIF 层开展。

2.4 各应用层协议描述

Bluedroid 主要分为 3 个部分：BTIF, BTA, Stack。

作为蓝牙核心服务，Bluetooth Stack 模块则由 Bluetooth Application Layer（BTA）和 Bluetooth Embedded System（缩写为 BTE）两大部分组成。

BTE：bluedroid 的内部处理，又可以细分为 BTA，BTU，BTM，HCI 等

2.4.1 BTIF (Bluetooth Profile Interface)

BTIF：Bluetooth Application task(BTA)和 JNI 层之间从当媒介（网上也说胶水层）。

对上层 JNI 提供所有 profile 功能行的接口。该层还存在 Bluetooth Interface Instance,所有 Profile 操作接口注册在其中（GAP, AV, DM, PAN, HF,HH, HL, Storage, Sockets）。Client 应用通过 Instance 来操作 Profile

2.4.2 BTA （Bluetooth Appication）

BTA：蓝牙应用层。指 bluedroid 中对各个 profile 实现和处理。上层下来的请求经过 BTA 层，通过消息发送的方式将请求传到 BTA 层中处理。

所有 BTA 消息送到 BTU_TASK 中，由 bta_sys_event 来处理；如果是 Gatt 相关的消息，由 bta_gatt_hdl_event 处理。

Stack：实现蓝牙底层操作。

2.4.3 BTU （Bluetooth Upper Layer）

BTU：承接 BTA 和 HCI

2.4.4 BTM（Bluetooth Manager ）

BTM：Bluedroid 中的管理层。蓝牙配对和链路管理

2.4.5 HCI

HCI：读取和写入数据到蓝牙 HW。主机与 BT 控制器之间的接口。

2.4.6 GKI 模块

内核统一接口。该层是一个适配层，适配了 OS 相关的进程、内存相关的管理，还可以用于线程间传递消息 。主要通过变量 gki_cb 实现对进程的统一管理。GKI 模块在 Bluedroid 中主要用于线程间通信。

2.4.7 bluedroid 协议栈消息传递和处理

蓝牙协议栈里通信通过**消息队列**完成。

2.5 BLE 介绍

BLE 根据需要提供短数据包，然后关闭链路，BLE 低功耗的原因之一。相对于常规蓝牙的传统配对方式，BLE 设备尽在需要收发信息时才进行链接。

BLE 通信方式极其严密。设备显示收发数据的服务，后者包含称之为特征的内容，用于

定义可共享的数据。特征可包含描述符，帮助 定义数据。

大多数 BLE API 都支持搜索本地设备和发现有关设备的服务、特征和描述符。

2.5.1 ATT

ATT 是专门针对 BLE 设备而设计的优化型协议。ATT 通过发送字节尽可能少的数据。所有属性均带有通用唯一标识符（UUID），后者为标准的 128 位字符串 ID，以唯一的方式识别信息。ATT 传输的属性被格式化为特征和服务。

- 特征（Characteristic）：包含一个单独数据以及 0 个或多个描述符以描述特征值。
- 描述符（Descriptor）：描述符制定了属性，可以描述特征值。可读的描述如可注明单位或测量，或定义可以接受的数值范围
- 服务（Service）：服务指特征的集合。例如一个 service 叫做“Heart Rate Monitor”，它可能包含多个 Characteristics，其中可能包含一个叫做“heart rate measurement"的 Characteristic。

2.5.2 GATT

GATT 配置文件是关于通过蓝牙低功耗链路收发短数据（称为属性）的通用规范。当前 BLE 应用配置文件均以 GATT 为基础。SIG 对 BLE 设备的配置文件数量进行了预定义。这些配置文件是关于描述设备使用方法的规范。

2.5.3 GAP

定义了设备如何发现，建立连接，实现绑定。

2.5.4 SM（Security Manager）

负责 BLE 通信中安全。

2.5.5 中心设备和外围设备

Central 与 peripheral；

GATT server 与 GATT client；

GAP 用于外设设备与中心设备，每个设备可以充当多种角色，同一时间只能充当一种角色。

2.6 传统蓝牙音频

A2DP 定义了传统蓝牙音频传输规范，描述了立体声音频如何从媒体输出(source)传输至输入(sink)。

A2DP 定义了音频设备的两个角色：输出和输入。

- 输出(SRC)-设备在将数字化音频流传输至微微网的输出时则作为输出设备。
- 输入(SNK)-设备在输入来自同一微微网中 SRC 的数字化音频流时则作为输入设备。

A2DP 定义了在 ACL 信道实现高品质音频内容的单声道或立体声分发协议和程序。基带、LMP、L2CAP 和 SDP 是蓝牙核心规格中定义的蓝牙协议。AVDTP 包括一个用于沟通串流参数的信令实体以及一个处理串流的传输实体.

AVRC/AVRC CTRL 定义了音视频控制传输协议，描述了输出和输出端音视频播放控制规范。

2.7 传统蓝牙免提电话

HFP 定义了传统蓝牙免提电话功能，描述了免提设备如何使用网关设备拨打和接听电话。

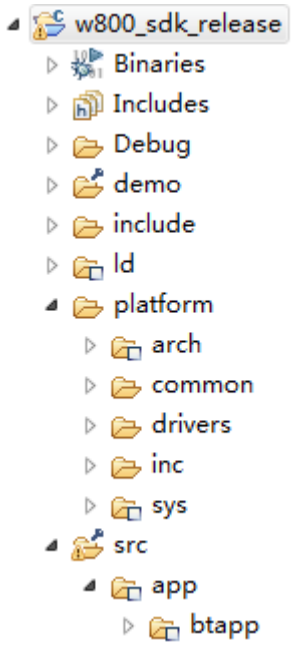
HFP 定义了音频网关(AG)和免提组件(HF)两个角色：

- 音频网关(AG) – 该设备为音频（特别是手机）的输入／输出网关。
- 免提组件(HF) – 该设备作为音频网关的远程音频输入／输出机制，并可提供若干遥控

功能。

2.8 源码框架描述

2.8.1 蓝牙系统软件代码位置



Btapp 目录即蓝牙示例代码，用户可以参考或基于此代码进行二次开发。

应用程序文件列表：

No	应用程序模块	说明
1	wm_bt_app.c	主机协议栈主程序入口
2	wm_ble_dm.c	设备管理模块
3	wm_ble_server.c	BLE server 应用管理模块,负责各 prof 注册及事件的分发处理。
4	wm_ble_server_wifi_prof.c	BLE 配网服务通讯模块
5	wm_ble_server_wifi_app.c	BLE 配网应用协议处理模块
6	wm_ble_server_demo_prof.c	BLE demo prof 示例，结合 AT 指令给出具体应用示例。

7	wm_ble_client.c	BLE client 应用管理模块，负责各应用注册及事件的分发处理
8	wm_ble_client_huawei.c	BLE client 示例应用，实现连接华为手机，读取某个 characteristic 数据
9	Wm_ble_client_demo.c	BLE client demo 示例,结合 AT 指令给出 client 端具体应用示例
10	wm_bt_audio_sink.c	传统蓝牙 audio sink demo 应用示例
11	wm_bt_hfp_hsp.c	传统蓝牙免提电话应用示例

涉及到头文件如下

序号	头文件名称	描述
1	wm_bt.h	蓝牙系统主机和控制器 api 定义
2	wm_ble.h	蓝牙系统设备管理 api 定义
3	wm_ble_gatt.h	蓝牙系统 GATT api 定义
4	wm_bt_def.h	蓝牙系统数据结构定义
5	Wm_bt_av.h	传统蓝牙 Audio sink、source 的 api 定义
6	Wm_bt_hf_client.h	传统蓝牙免提电话 client 端 api 定义

3 API 描述

3.1 蓝牙系统 API

No	API 名称	描述
1	tls_bt_status_t tls_bt_enable(tls_bt_host_callback_t *scb, tls_bt_hci_if_t *uart, tls_bt_log_level_t log_level)	运行蓝牙系统，该函数会依次使能主机协议炸和控制器协议栈。
2	tls_bt_status_t tls_bt_disable(void)	停止蓝牙系统，改函数会依次注销主机协议栈和控制器协议栈。

3	Tls_bt_status_t
	tls_bt_host_cleanup()
		清理 host 系统，如释放资源，注销 task。
		注意：此函数必须等待 tls_bt_disable 调用后，并且要等待 bt_adapter_state 改变为 OFF 后，方可调用。
		tls_bt_disable();
		do
		{
		hal_system_sleep(100);
		}while(adapter_state != WM_BT_STATE_OFF);
		tls_bt_host_cleanup();

3.2 主机端 API

主机端 API 描述了主机协议栈启动及注销等功能

No	API 名称	描述
1	tls_bt_status_t tls_bt_host_enable(tls_bt_host_callback_t *p_callback, tls_bt_log_level_t log_level)	初始化主机端协议栈，分配内存及创建任务等
2	tls_bt_status_t tls_bt_host_disable()	注销主机协议栈
3	tls_bt_status_t tls_bt_pin_reply(const tls_bt_addr_t *bd_addr, uint8_t accept, uint8_t pin_len, tls_bt_pin_code_t *pin_code)	答复 BLE 配对的 pin 码
4	tls_bt_status_t tls_bt_ssp_reply(const tls_bt_addr_t *bd_addr, tls_bt_ssp_variant_t variant uint8_t accept, uint32_t passkey)	回复配对响应
5	tls_bt_status_t tls_bt_set_adapter_property(const tls_bt_property_t *property)	设置 adapter 属性值
6	tls_bt_status_t tls_bt_get_adapter_property(	读取 adapter 属性值

	tls_bt_property_type_t type)	
7	tls_bt_status_t tls_bt_start_discovery()	传统蓝牙扫描
8	tls_bt_status_t tls_bt_cancel_discovery()	停止扫描
9	tls_bt_status_t tls_bt_create_bond(const tls_bt_addr_t *bd_addr, int transport)	发起配对操作
10	tls_bt_status_t tls_bt_cancel_bond(const tls_bt_addr_t *bd_addr)	取消配对操作
11	tls_bt_status_t tls_bt_remove_bond(const tls_bt_addr_t *bd_addr)	删除配对信息

3.3 控制器端 API

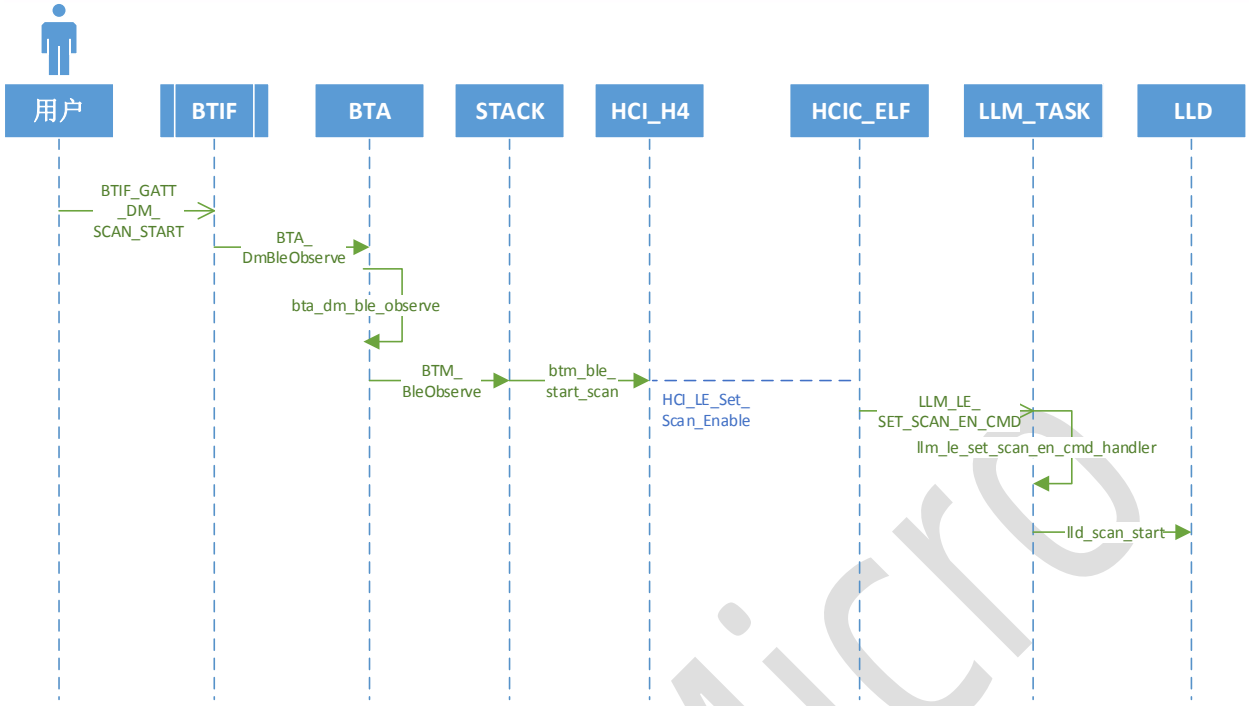
No	API 名称	描述
1	tls_bt_status_t tls_bt_ctrl_enable(tls_bt_hci_if_t *p_hci_if, tls_bt_log_level_t log_level)	初始化控制器端协议栈，分配内存及创建任务等
2	tls_bt_status_t tls_bt_ctrl_disable(void);	注销控制器协议栈
3	tls_bt_status_t tls_ble_set_tx_power(tls_ble_power_type_t power_type, int8_t power_level);	设置 BLE 发射功率索引
4	int8_t tls_ble_get_tx_power(uint8_t power_type);	读取指定工作类型的发送功率索引
5	tls_bt_status_t tls_bredr_set_tx_power(int8_t min_power_level, int8_t max_power_level);	设置传统蓝牙工作时功率索引范围值
6	tls_bt_status_t tls_bredr_get_tx_power(int8_t*min_power_level, int8_t* max_power_level);	读取传统蓝牙工作的发射功率的索引范围值
7	tls_bt_status_t tls_bredr_sco_datapath_set(	设置传统蓝牙 sco 链路的输出路径

	wm_sco_data_path_t data_path);	
8	tls_bt_ctrl_status_t tls_bt_controller_get_status(void);	读取控制器目前的状态，
9	bool wm_bt_vuart_host_check_send_available(void);	用于判断主机能否向控制器发送指令
10	tls_bt_status_t tls_bt_vuart_host_send_packet (uint8_t *data, uint16_t len);	主机协议栈向控制器发送数据接口
11	tls_bt_status_t tls_bt_ctrl_if_register (const tls_bt_host_if_t *p_host_if);	注册控制器数据发送接口，即主机协议栈接收数据接口
12	tls_bt_status_t tls_bt_ctrl_sleep (bool enable);	是否运行控制器在空闲时进入 sleep 模式
13	bool tls_bt_ctrl_is_sleep (void);	读取控制器是否处于 sleep 模式
14	tls_bt_status_t tls_bt_ctrl_wakeup(void)	退出 sleep 模式

3.4 应用层协议 API

3.4.1 设备管理

设备管理层承担控制器的通用设置，如广播，扫描，设备名称修改等功能,下图给出了扫描指令的依次调用关系。

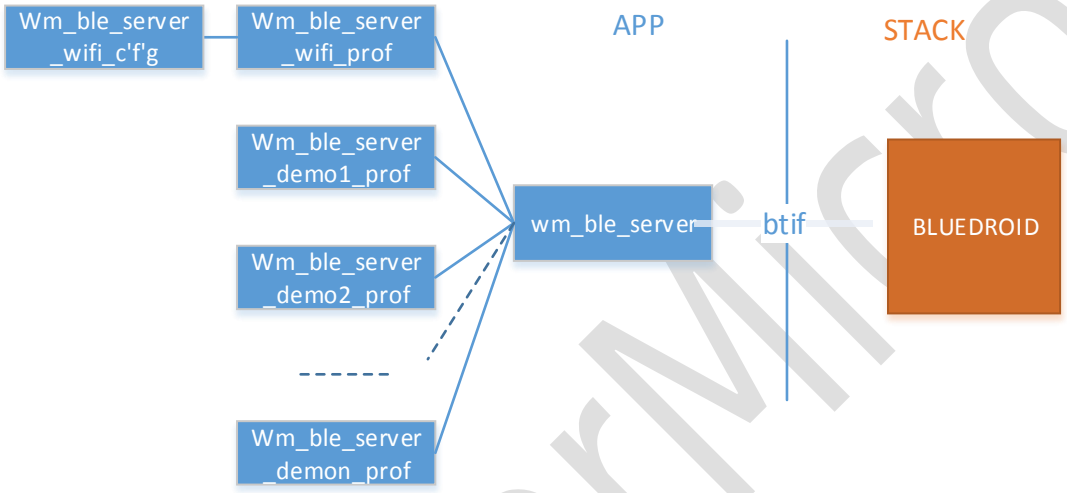


3.4.1.1 设备管理层 API 描述

No	API 名称	描述
1.	tls_bt_status_t tls_ble_adv(bool start);	开始/停止广播
2.	tls_bt_status_t tls_ble_set_adv_data(tls_ble_dm_adv_data_t *data)	设置广播内容
3.	tls_bt_status_t tls_ble_set_adv_param (tls_ble_dm_adv_param_t *param)	设置广播参数
4.	tls_bt_status_t tls_ble_set_scan_param(int window, int interval);	设置扫描参数
5.	tls_bt_status_t tls_ble_scan(bool start);	启动/停止扫描
6	tls_bt_status_t tls_dm_evt_triger (int id,	注册异步处理事件

	tls_ble_dm_triger_callback_t *p_callback)	

3.4.2 BLE server



BLE server应用关系图示

BLE server 承担 slave 角色，wm_ble_server 模块提供了用户程序开发的接口描述，基于该模块，用户可以开发自己的应用程序，如 wm_ble_server_demo1_prof,等等。其中基于 BLE 的 wifi 配网便是一个具体应用，上图中，wm_ble_server_wifi_app 模块用来处理具体配网协议处理，wm_ble_server_wifi_prof 承担 service/character/descriptor 的逻辑处理。

3.4.2.1 BLE server api 描述

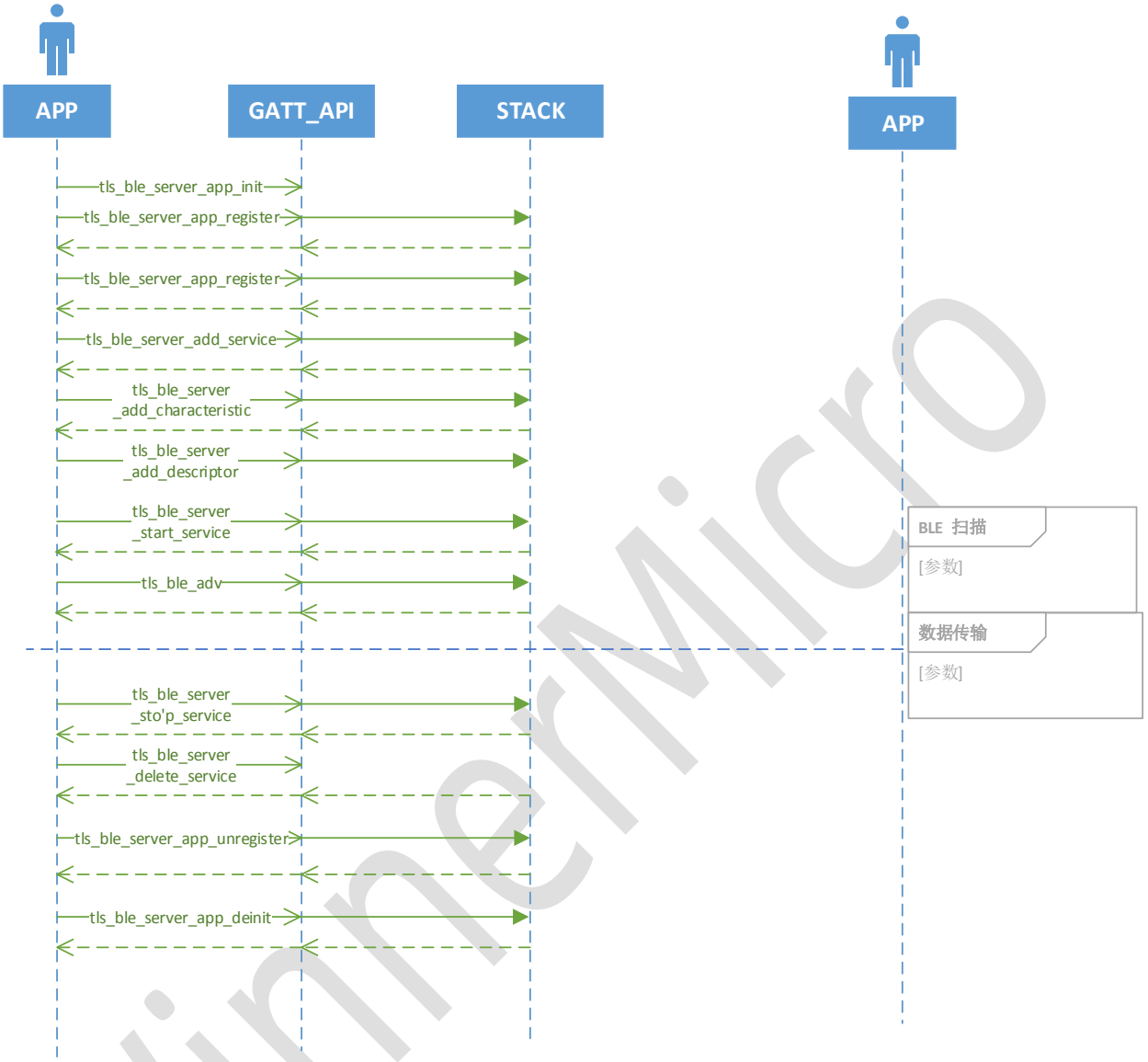
No	API 名称	描述
1	tls_bt_status_t	向 btif 层注册该层的事件响应函数.该函数在主

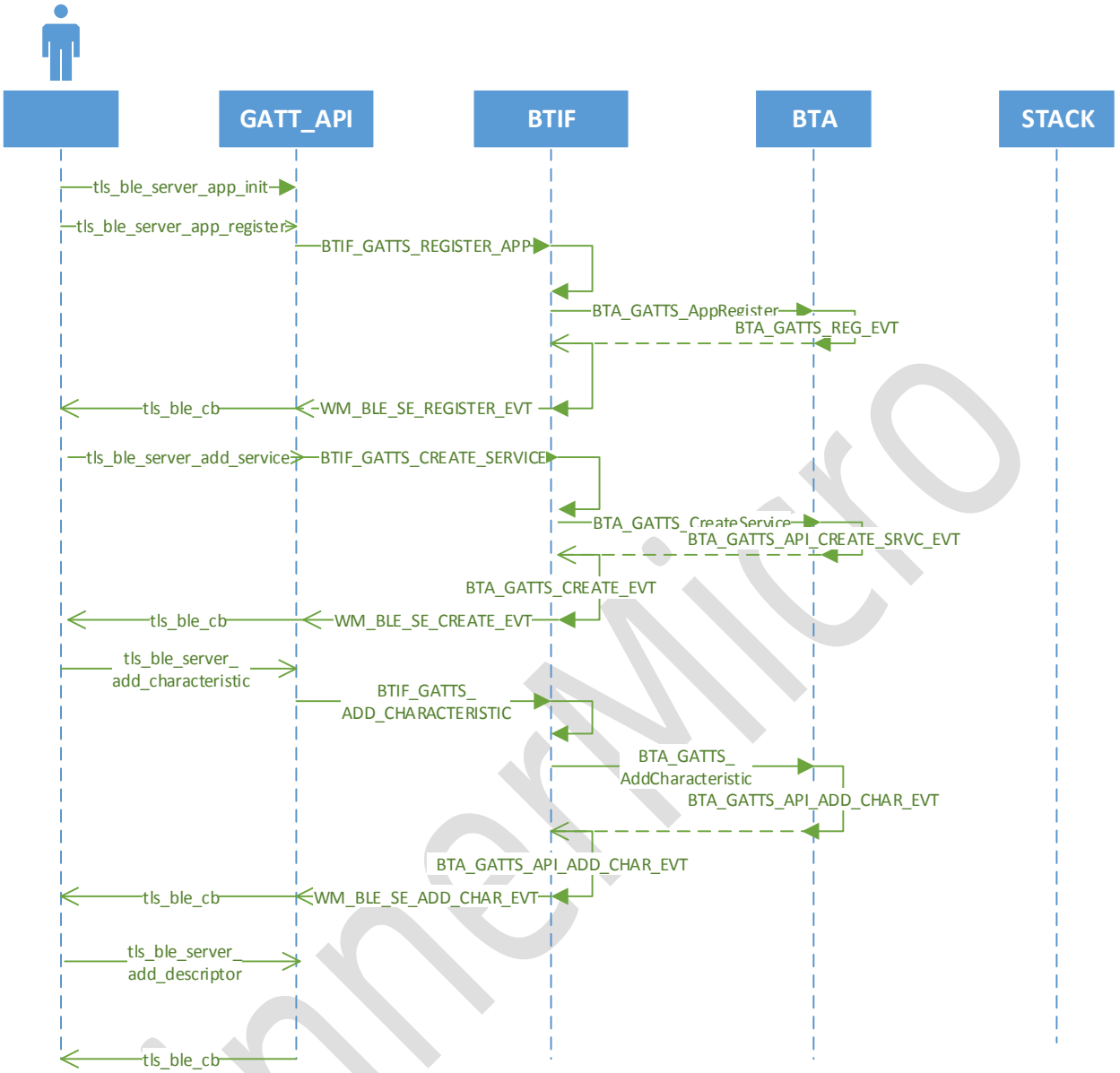
	tls_ble_server_app_init (tls_ble_callback_t *p_callback)	机协议栈启动后调用。
2	tls_bt_status_t tls_ble_server_app_deinit();	向 btif 层注销响应函数
3	tls_bt_status_t tls_ble_server_app_register (tls_bt_uuid_t *uuid)	向 btif 层注册一个指定 UUID 的 app server
4	tls_bt_status_t tls_ble_server_app_unregister(uint8_t server_if);	注销 3 中注册的 app server
5	tls_bt_status_t tls _ble_server_add_service(int server_if, int inst_id, int primay, uint16_t uuid, int num_handles)	向注册的 server 添加服务
6	tls_bt_status_t tls _ble_server_add_characteristic(int server_if, int service_handle, uint16_t handle, int properties, int permission)	向某个指定的服务添加特征值
7	tls_bt_status_t	向某个指定的服务添加特征值的描述

	tls _ble_server_add_descriptor(int server_if, int service_handle, uint16_t uuid, int permissions)	
8	tls_bt_status_t tls _ble_server_start_service(int server_if, int service_handle, int transport)	运行添加的服务
9	tls_bt_status_t tls _ble_server_stop_service (int server_if, int service_handle)	停止服务
10	tls_bt_status_t tls _ble_server_delete_service (int server_if, int service_handle)	删除添加的服务
12	tls_bt_status_t tls_ble_server_connect(int server_if, const bt_bdaddr_t *bd_addr, uint8_t is_direct, int transport)	连接 client 操作[预留]
13	tls_bt_status_t tls_ble_server_disconnect(int server_if, const bt_bdaddr_t *bd_addr,	断开同 client 的连接

	int conn_id)	
14	tls_bt_status_t tls _ble_server_send_indication(int server_if, int attribute_handle, int conn_id, int len, int confirm, char *p_value)	发送 Indication，Notification 消息
15	tls_bt_status_t tls_ble_server_send_response(int conn_id, int trans_id, int offset, int attr_handle, int auth_req, uint8_t *value, int length)	发送读/写的响应

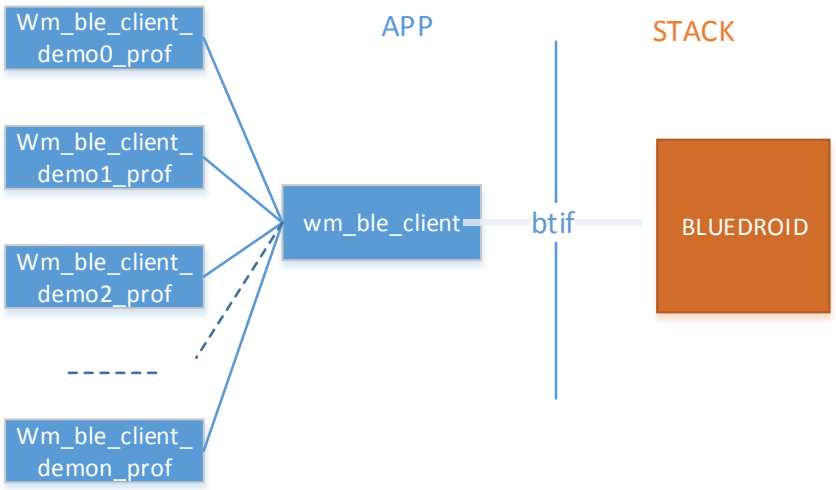
3.4.2.2 BLE server 创建/注销流程描述





3.4.3 BLE client

BLE client 承担 master 角色，即主动发起扫描，连接，通讯等应用。wm_ble_client 模块提供了用户程序开发的接口描述，基于该模块，用户可以开发自己的应用程序。



BLE client应用结构关系图示

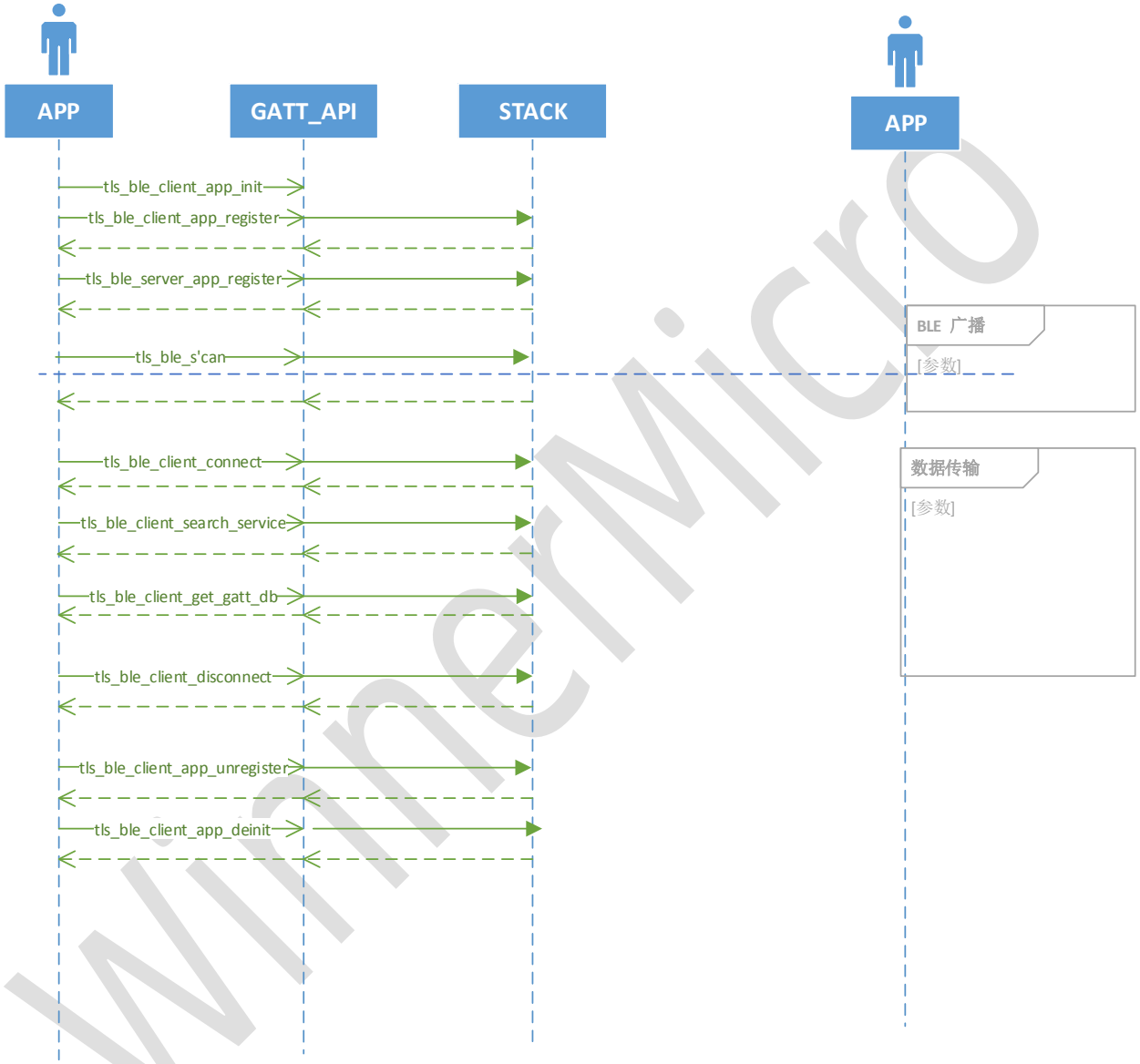
3.4.3.1 BLE client api 描述

No	API 名称	描述
1.	tls_bt_status_t tls_ble_client_app_init(tls_ble_callback_t *p_callback)	向 btif 层注册 client 侧的消息响应函数
2.	tls_bt_status_t tls_ble_client_app_deinit(void)	注销注册的响应函数
3	tls_bt_status_t tls_ble_client_app_register (tls_bt_uuid_t *uuid)	向 btif 层注册指定的 client
4	tls_bt_status_t tls_ble_client_unregister_client(int client_if)	注销指定的 client
5	tls_bt_status_t tls_ble_client_connect (int client_if, const bt_bdaddr_t *bd_addr,	连接某个指定的 server

	uint8_t is_direct, int transport)	
6	tls_bt_status_t tls_ble_client_disconnect(int client_if, const bt_bdaddr_t *bd_addr, int conn_id)	断开同 server 端的连接
7	tls_bt_status_t tls_ble_client_listen (int client_if, uint8_t start)	监听来自 server 端的连接
8	tls_bt_status_t tls_ble_client_refresh (int client_if, const bt_bdaddr_t *bd_addr)	刷新指定的 server 的属性值列表
9	tls_bt_status_t tls_ble_client_search_service(int conn_id, uint16_t filter_uuid)	扫描 server 端属性值列表
10	tls_bt_status_t tls_ble_client_write_characteristic(int conn_id, uint16_t handle, int write_type, int len, int auth_req, char *p_value)	向某个指定的特征值执行写操作
11	tls_bt_status_t tls_ble_client_read_characteristic(int conn_id, uint16_t handle,int auth_req)	读取某个指定的特征值
12	tls_bt_status_t	读取某个指定的特征值描述

	tls_ble_client_read_descriptor (int conn_id, uint16_t handle, int auth_req)	
13	tls_bt_status_t tls_ble_client_write_descriptor (int conn_id, uint16_t handle, int write_type, int len, int auth_req, char *p_value)	向某个指定的特征值描述执行写操作
14	tls_bt_status_t tls_ble_client_execute_write (int conn_id, int execute)	该指令执行 prepare 写操作
15	tls_bt_status_t tls_ble_client_register_for_notification (int client_if, const bt_bdaddr_t *bd_addr, uint16_t handle)	注册某个指定句柄的消息响应函数
16	tls_bt_status_t tls_ble_client_deregister_for_notification (int client_if, const bt_bdaddr_t *bd_addr, uint16_t handle)	注销指定的消息响应函数
17	tls_bt_status_t tls_ble_client_configure_mtu(int conn_id, int mtu)	设置某个连接的最大传输单元值
18	tls_bt_status_t tls_ble_client_get_gatt_db (int conn_id)	读取指定连接 id 的属性值列表

3.4.3.2 BLE client 创建/注销流程描述



3.4.4 传统蓝牙音频

传统蓝牙音频 API 提供了音频传输及播放控制接口，目前发布的版本仅支持 sink 功能。

No	API 名称	描述
1.	tls_bt_status_t tls_bt_av_sink_init (	向 btif 层注册 Audiosink 消息响应函数，初始化 sink 功

	tls_bt_a2dp_sink_callback_t callback)	能
2.	tls_bt_status_t tls_bt_av_sink_connect_src (tls_bt_addr_t *bd_addr)	创建同 source 侧连接
3	tls_bt_status_t tls_bt_av_sink_disconnect(tls_bt_addr_t *bd_addr)	断开同 source 侧连接
4	tls_bt_status_t tls_bt_av_sink_deinit(void);	注销 Audio sink 功能
5	tls_bt_status_t tls_bt_av_src_init(tls_bt_a2dp_src_callback_t callback)	向 btif 层注册 Audiosink 消息响应函数，初始化 source 功能（暂未开放）
6	tls_bt_status_t tls_bt_av_src_connect_sink(tls_bt_addr_t *bd_addr)	创建同 sink 侧的连接（暂未开放）
7	tls_bt_status_t tls_bt_av_src_disconnect(tls_bt_addr_t *bd_addr)	断开同 sink 侧的连接（暂未开放）
8	tls_bt_status_t tls_bt_av_src_deinit(void)	注销 souce 功能（暂未开放）
9	tls_bt_status_t tls_btrc_init(tls_btrc_callback_t callback)	初始化 AVRC 功能
10	tls_bt_status_t tls_btrc_get_play_status_rsp(tls_btrc_play_status_t play_status, uint32_t song_len, uint32_t song_pos)	发送 GetPlayStatus 的响应，指定曲目的播放状态，曲目时间及已经播放的时间信息
11	tls_bt_status_t tls_btrc_get_element_attr_rsp(uint8_t num_attr,	返回当前曲目的元素信息

	tls_btrc_element_attr_val_t *p_attrs)	
12	tls_bt_status_t tls_btrc_register_notification_rsp(tls_btrc_event_id_t event_id, tls_btrc_notification_type_ttype, tls_btrc_register_notification_t *p_param)	注册感兴趣的事件，注册后可以接收到具体的事件通知
13	tls_bt_status_t tls_btrc_set_volume(uint8_t volume)	设置对方播放音量，注意该音量为绝对值[0~127]
14	tls_bt_status_t tls_btrc_deinit(void)	注销 AVRC 功能
15	tls_bt_status_t tls_btrc_ctrl_init(tls_btrc_ctrl_callback_t callback)	注册 AVRC Ctrl 功能
16	tls_bt_status_t tls_btrc_ctrl_send_passthrough_cmd(tls_bt_addr_t *bd_addr, uint8_t key_code, uint8_t key_state);	向 souce 侧发送具体的控制指令
17	tls_bt_status_t tls_btrc_ctrl_change_player_app_setting(tls_bt_addr_t *bd_addr, uint8_t num_attrib, uint8_t *attrib_ids, uint8_t *attrib_vals)	更改播放器的配置参数
18	tls_bt_status_t tls_btrc_ctrl_set_volume_rsp(tls_bt_addr_t *bd_addr, uint8_t abs_vol, uint8_t label)	发送设置绝对音量指令的响应
19	tls_bt_status_t	接收到音量变化通知后，发

	tls_btrc_ctrl_volume_change_notification_rsp(tls_bt_addr_t *bd_addr, tls_btrc_notification_type_t rsp_type, uint8_t abs_vol, uint8_t label)	送响应
20	tls_bt_status_t tls_btrc_ctrl_deinit(void);	注销 avrc ctrl 功能

3.4.5 传统蓝牙免提电话

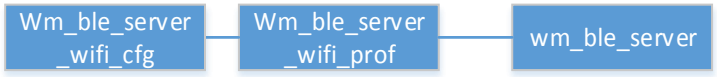
传统蓝牙免提电话 API 提供了免提电话 client 端功能接口。

No	API 名称	描述
1.	tls_bt_status_t tls_bt_hf_client_init(tls_bthf_client_callback_t callback)	注册并使能 hfp client 功能
2.	tls_bt_status_t tls_bt_hf_client_connect(tls_bt_addr_t *bd_addr)	创建同音频网关侧（Audio Gateway）链路连接
3	tls_bt_status_t tls_bt_hf_client_disconnect(tls_bt_addr_t *bd_addr)	断开同音频网关侧链路连接
4	tls_bt_status_t tls_bt_hf_client_connect_audio(tls_bt_addr_t *bd_addr)	创建同音频网关侧（Audio Gateway）音频连接
5	tls_bt_status_t tls_bt_hf_client_disconnect_audio(tls_bt_addr_t *bd_addr)	断开同音频网关侧（Audio Gateway）音频连接
6	tls_bt_status_t tls_bt_hf_client_start_voice_recognition(void)	开始语音识别
7	tls_bt_status_t tls_bt_hf_client_stop_voice_recognition(void)	停止语音识别

8	tls_bt_status_t tls_bt_hf_client_volume_control(tls_bthf_client_volume_type_t type, int volume)	控制 MIC 或者 speaker 音量
9	tls_bt_status_t tls_bt_hf_client_dial(const char *number)	拨号
10	tls_bt_status_t tls_bt_hf_client_dial_memory(int location)	拨打指定位置的号码
11	tls_bt_status_t tls_bt_hf_client_handle_call_action(tls_bthf_client_call_action_t action, int idx)	处理特定通话过程中响应
12	tls_bt_status_t tls_bt_hf_client_query_current_calls(void)	查询通话列表
13	tls_bt_status_t tls_bt_hf_client_query_current_operator_name(void)	查询当前操作者名称
14	tls_bt_status_t tls_bt_hf_client_retrieve_subscriber_info(void)	查询订阅的号码信息
15	tls_bt_status_t tls_bt_hf_client_send_dtmf(char code)	发送号码键的 dtmf 值
16	tls_bt_status_t tls_bt_hf_client_request_last_voice_tag_number(void)	请求音频网关识别到的数字号码
17	tls_bt_status_t tls_bt_hf_client_send_at_cmd(int cmd, int val1, int val2, const char *arg)	向 AudioGateway 发送特定的 AT 指令
18	tls_bt_status_t tls_bt_hf_client_send_audio(tls_bt_addr_t *bd_addr, uint8_t *p_data, uint16_t length)	向 AutioGateway 发送语音数据（具体接口实现依赖音频模块实现）

19	tls_bt_status_t tls_bt_hf_client_deinit(void)	注销免提电话 client 功能

3.5 蓝牙配网 API



BLE wifi快速配网应用关系图示

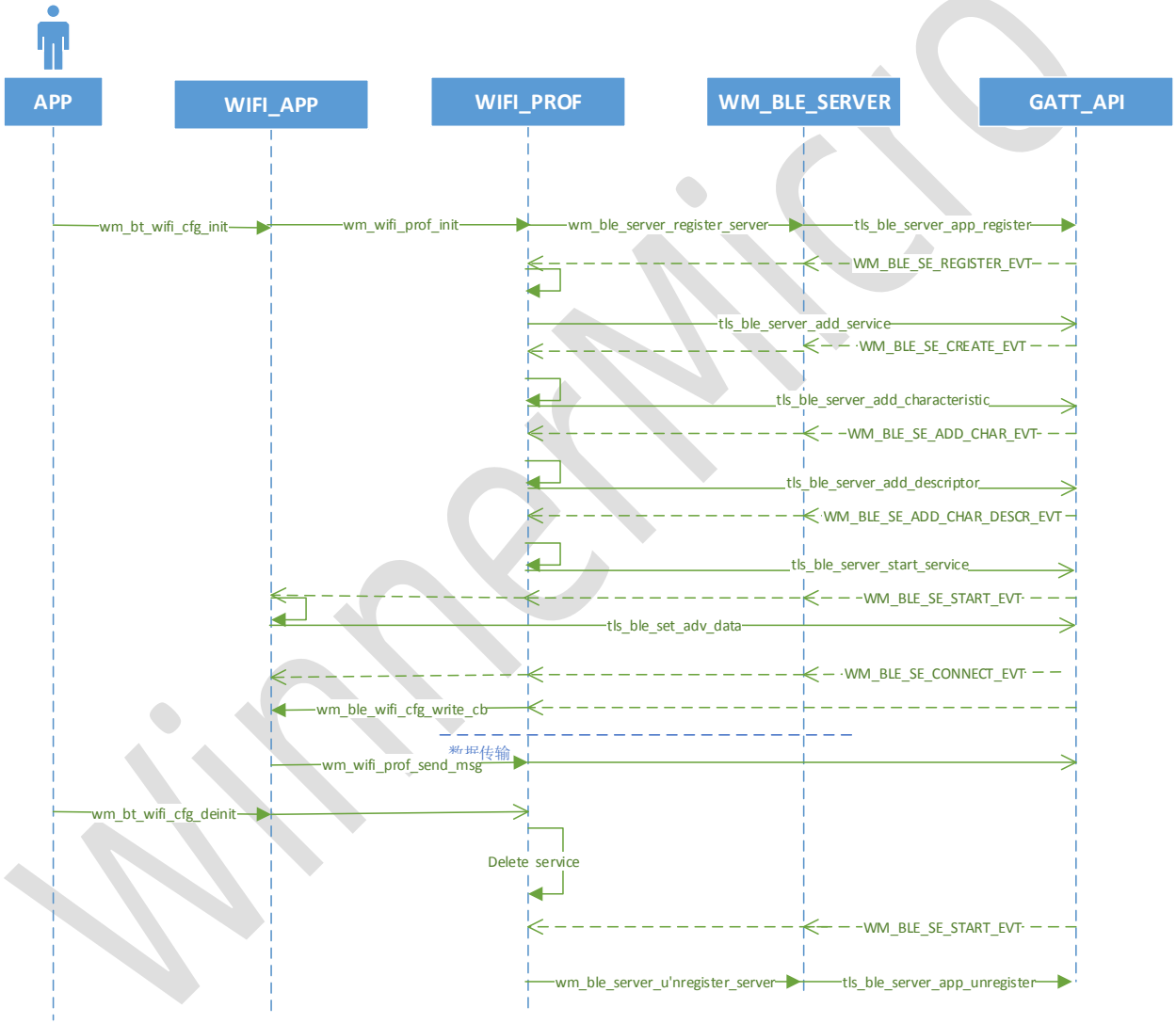
BLE 快速配网，作为 BLE server 的一个具体应用，其关系图示如上所述。Wm_ble_server_wifi_prof 承担 service 功能，即数据的传输处理，wm_ble_server_wifi_cfg 处理具体通讯协议处理。这样的层次结构使得应用处理与具体传输层独立开来，逻辑层次调用更加清晰化。

这部分 API 相对简单，如下：

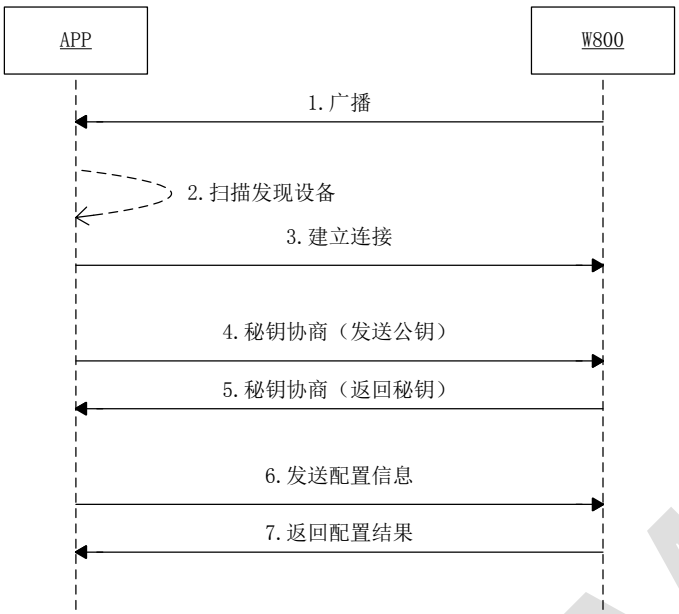
No	API 名称	描述
1	tls_bt_status_t tls_bt_enable(tls_bt_host_callback_t *scb, tls_bt_hci_if_t *uart, tls_bt_log_level_t log_level)	运行蓝牙系统，该函数会依次使能主机协议炸和控制器协议栈。
2	tls_wifi_set_oneshot_flag(flag) flag 0: closed oneshot 1: UDP（broadcast+multicast） 2: AP+socket 3: AP+WEBSERVER	启动/停止配网 注意：1，配网成功后，BLE 的配网 service 会自动退出，广播关闭。如需再次配网请再次调用此 API 即可。

	4: BT	2，如配网失败，用户可以再次配置
3	参见 3.1 蓝牙系统注销描述	

3.5.1 软件模块调用关系



3.5.2 应用流程示例



3.5.3 BLE 配网 Service

Service 定义：

Service uuid： 0x1824

特征值 uuid：0x2ABC Write & Indication

特征值描述 uuid: 2902

Write： BleWiFi（手机 APP -> W800）Characteristic UUID：0x2ABC
Indication：BleWiFi（W800 -> 手机 APP）Characteristic UUID：0x2ABC

3.6 用户实现自己的配网 service

参考 wm_ble_server_demo_prof.c 示例，添加自定义的 service。

4 API 使用示例

W800 蓝牙功能加电后，默认是不使能的。如果用户想默认使用蓝牙，可以参考如下说明。

4.1 蓝牙系统使能（退出）

步骤 1，在 tls_bt_entry()函数中调用打开蓝牙功能，关闭蓝牙系统调用 demo_bt_destroy;

```
void tls_bt_entry()
{
    //demo_bt_enable(); //turn on bluetooth system;
}

void tls_bt_exit()
{
    //demo_bt_destroy(); //turn off bluetooth system;
}
```

步骤 2，蓝牙功能打开成功后，如下回调函数会被调用，用户添加自己的应用程序；

```
void app_adapter_state_changed_callback(tls_bt_state_t status)
{
    tls_bt_property_t btp;
    tls_bt_host_msg_t msg;
    msg.adapter_state_change.status = status;
    TLS_BT_APPL_TRACE_DEBUG("adapter status = %s\r\n", status==WM_BT_STATE_ON?"bt_state_on":"bt_state_off");

    bt_adapter_state = status;

    #if (TLS_CONFIG_BLE == CFG_ON)

    if(status == WM_BT_STATE_ON)
    {
        TLS_BT_APPL_TRACE_VERBOSE("init base application\r\n");
        /* those funtions should be called basicly*/
        wm_ble_dm_init();
        wm_ble_client_init();
        wm_ble_server_init();

        //at here , user run their own applications;

        //application_run();
    }else
    {
        TLS_BT_APPL_TRACE_VERBOSE("deinit base application\r\n");
        wm_ble_dm_deinit();
        wm_ble_client_deinit();
        wm_ble_server_deinit();

        //here, user may free their application;

        //application_stop();
    }

    #endif
```

4.2 运行（退出）demo service

在 4.1 节中步骤 2 标记的位置处，调用 wm_ble_server_api_demo_init();

在 4.1 节中步骤 2 标记的位置处，调用 wm_ble_server_api_demo_deinit()；应用程序的退出功能，会在蓝牙系统退出时，自动释放。当然，蓝牙系统在运行时，用户也可以随时退出自己的应用程序。

4.3 开启广播

4.3.1 默认广播数据配置

```
static void ble_server_adv_enable_cb(uint8_t triger_id)
{
    tls_ble_adv(true);
}
static void ble_server_cfg_and_enable_adv()
{
    tls_ble_dm_adv_data_t data;
    uint8_t adv_data[] = {0x0C, 0x07, 0x00, 0x10};

    memset(&data, 0, sizeof(data));
    data.set_scan_rsp = false;
    data.include_name = true;
    data.manufacturer_len = 4;
    memcpy(data.manufacturer_data, adv_data, 4);

    /*configure the user specific data, 0xFF field*/
    tls_ble_set_adv_data(&data);

    /*enable advertisement*/
    tls_dm_evt_triger(0, ble_server_adv_enable_cb);

}
```

4.3.2 用户自定义广播数据设置

```
static void ble_server_adv_enable_cb(uint8_t triger_id)
{
    tls_ble_adv(true);
}
static void ble_server_cfg_and_enable_adv()
{
    tls_ble_dm_adv_data_t data;

    uint8_t adv_data[] = {0x03, 0x09, ' A' , 'B'};  //用户自己填充广播数据内容
    memset(&data, 0, sizeof(data));
    data.set_scan_rsp = false;
    data.pure_data = true;      //标记用户自己填充广播数据；数据内容为 adv_data
    data.manufacturer_len = 4;  //用户自己填充广播数据的长度；
    memcpy(data.manufacturer_data, adv_data, 4);

    /*configure the user specific data*/
    tls_ble_set_adv_data(&data);

    /*enable advertisement*/
    tls_dm_evt_triger(0, ble_server_adv_enable_cb);

}
```

5 蓝牙 AT 指令

5.1 蓝牙 AT 指令简述

通过蓝牙 AT 指令可以控制蓝牙系统，蓝牙 AT 指令共分为 4 类。主机、控制器部分用来配置主机协议栈和控制器协议栈，应用层部分用来配置蓝牙应用程序，测试部分用来配置蓝牙

认证功能（该部分部分包含了应用层）。

蓝牙 AT 指令中的缩写含义为：

缩写	含义
CTRL	CONTROLLER
BLESC	BLE SERVICE
BLESV	BLE SERVER
BLEC	BLE CLIENT
POW	POWER
STS	STATUS
DES	DESTORY
PRM	PARAM
FLT	FILTER
CT	CREATE
CH	CHARACTERISTIC
STT	START
STP	STOP
DEL	DELETE
DIS	DISCONNECT
SND	SEND
IND	INDICATION
CONN	CONNECT

NTY	NOTIFICATION
ACC	ACCESS
TEST	TESTMODE
EN	ENABLE
GS	GETSTATUS
TPS	TXPOWERSET
TPG	TXPOWERGET

5.2 蓝牙系统 AT 指令

5.2.1.1 AT+BTEN

功能：

使能蓝牙系统。

格式（ASCII）：

AT+BTEN=<uart_no>,<log_level><CR> +OK=<status>,<adapter_status><CR><LF><CR><LF>

参数：

uart_no：串口索引号，定义如下：

值	含 义
1	uart1 目前版本只支持 UART1

Log_level：日志输出等级，定义如下：

值	含 义
0	关闭 log 输出

1	输出 error 级别的 log
2	输出 warn 级别的 log
3	输出 api 级别的 log
4	输出 event 级别的 log
5	输出 debug 级别的 log
6	输出 verbose 级别的 log

返回：

status：指令响应结果

值	含 义
0	成功
Others>1	失败

adapter_status：指令响应结果

值	含 义
1	控制器运行
0	控制器停止

5.2.1.2 AT+BTDES

功能：

停止并注销蓝牙系统。

格式（ASCII）：

AT+BTDES<CR>

+OK=<status>,<adapter_status><CR><LF><CR><LF>

参数：

参见 BTEN 参数描述

5.3 蓝牙主机协议栈 AT 指令

5.3.1.1 AT+BTCFGHOST

功能：

初始化并启动或停止并注销主机协议栈。注意启动主机协议栈之前，必须先启动控制器协议栈。

格式（ASCII）：

AT+BTCFGHOST=[cmd]<CR> +OK<CR><LF><CR><LF>

参数：

cmd：主机协议栈控制命令，定义如下：

值	含 义
0	停止并注销主机协议栈
1	初始化并启动主机协议栈

5.4 蓝牙控制器协议栈 AT 指令

5.4.1.1 AT+BTCTRLEN

功能：

初始化并启动控制器协议栈。

格式（ASCII）：

AT+BTCTRLEN=<uart_no>,<log_level><CR> +OK<CR><LF><CR><LF>

参数：

uart_no：串口索引号，定义如下：

值	含 义
1	uart1，目前版本只支持 UART1

Log_level：日志输出等级，定义如下：

值	含 义
0	关闭 log 输出
1	输出 error 级别的 log

5.4.1.2 AT+BTCTRLDES

功能：

停止并注销控制器协议栈。

格式（ASCII）：

AT+BTCTRLDES<CR> +OK<CR><LF><CR><LF>

参数：

无。

5.4.1.3 AT+BTCTRLGS

功能：

获取控制状态。

格式（ASCII）：

AT+BTCTRLGS<CR> +OK=<status><CR><LF><CR><LF>

参数：

status：控制状态，返回格式定义如下：

```
TLS_BT_CTRL_IDLE        =                   (1<<0),
TLS_BT_CTRL_ENABLED     =                   (1<<1),
TLS_BT_CTRL_SLEEPING =                      (1<<2),
TLS_BT_CTRL_BLE_ROLE_MASTER =               (1<<3),
TLS_BT_CTRL_BLE_ROLE_SLAVE =                (1<<4),
TLS_BT_CTRL_BLE_ROLE_END =                  (1<<5),
TLS_BT_CTRL_BLE_STATE_IDLE =                (1<<6),
TLS_BT_CTRL_BLE_STATE_ADVERTISING = (1<<7),
TLS_BT_CTRL_BLE_STATE_SCANNING =        (1<<8),
TLS_BT_CTRL_BLE_STATE_INITIATING =      (1<<9),
TLS_BT_CTRL_BLE_STATE_STOPPING =        (1<<10),
TLS_BT_CTRL_BLE_STATE_TESTING =         (1<<11),
```

5.4.1.4 AT+BTSLEEP

功能：

设置控制器空闲时 sleep 模式。当前版本暂不支持

格式（ASCII）：

AT+BTSLEEP=<cmd><CR> +OK<CR><LF><CR><LF>

参数：

cmd：控制命令，定义如下：

值	含义
0	禁止控制器进入sleep
1	允许控制器进入sleep

5.4.1.5 AT+BLETPS

功能：

配置 BLE 特定类型下发送功率。当前版本只支持默认功率设置

格式（ASCII）：

AT+BLETPS=<type>,<level><CR> +OK<CR><LF><CR><LF>

参数：

type：ble类型，定义如下：

值	含义

0	特定的连接handle
1	特定的连接handle
2	特定的连接handle
3	特定的连接handle
4	特定的连接handle
5	特定的连接handle
6	特定的连接handle
7	特定的连接handle
8	特定的连接handle
9	广播
10	扫描
11	默认功率

level：功率索引值。

值	含义 dBm
1	1
2	4
3	7
4	10
5	13

5.4.1.6 AT+BLETPG

功能：

获取 BLE 特定类型。当前版本只支持默认功率获取

格式（ASCII）：

AT+BLETPG=?<type><CR> +OK=<level><CR><LF><CR><LF>

参数：

type：ble类型，定义如下：

值	含义
0	特定的连接handle
1	特定的连接handle
2	特定的连接handle
3	特定的连接handle
4	特定的连接handle
5	特定的连接handle
6	特定的连接handle
7	特定的连接handle
8	特定的连接handle
9	广播
10	扫描
11	默认功率

level：功率索引值。参见 4.4.1.5

5.4.1.7 AT+BTTXPOW

功能：

设置/查询发送功率索引。

格式（ASCII）：

AT+BTTXPOW=[?]<min>,<max><CR> +OK=<min>,<max><CR><LF><CR><LF>

参数：

min：功率最小值，最小取值为1，表示最小功率为1dBm，每次增加步长为3db。

max：功率最大值，最大取值为5，表示最大功率为13dBm，每次减小步长为3db。

5.4.1.8 AT+BTSCOPATH

功能：

指定 sco 链路输出路径。当前版本暂不支持

格式（ASCII）：

AT+BTSCOPATH=[?]<path><CR> +OK<CR><LF><CR><LF>

参数：

path：输出路径，定义如下：

值	含义
0	PCM over HCM
1	Internal Interface

5.4.1.9 AT+BTTEST

功能：

设置蓝牙测试模式。

格式（ASCII）：

AT+BTTEST=<mode><CR> +OK<CR><LF><CR><LF>

参数：

mode：测试模式，定义如下：

值	含义
0	退出蓝牙测试模式
1	进入蓝牙测试模式

5.5 蓝牙应用层 AT 指令

蓝牙应用层分为设备管理、BLE server 和 BLE client 三部分。

5.5.1 设备管理 AT 指令

5.5.1.1 AT+BLEADV

功能：

控制 BLE 广播发送和停止。

格式（ASCII）：

AT+BLEADV=<mode><CR> +OK<CR><LF><CR><LF>

参数：

mode：控制模式，定义如下：

值	含义
0	停止BLE广播
1	启动BLE广播

5.5.1.2 AT+BLEADATA

功能：

配置 BLE 广播内容。注意，修改广播内容后，需重新使能广播生效

格式（ASCII）：

AT+BLEADATA=<include_name><data><CR> +OK<CR><LF><CR><LF>

参数：

include_name:1，广播包内包含设备名称；否则不含名称

data：广播内容，该内容将以 FF 字段 paylaod 广播，为 HEX 格式。最大长度为 24 个字符，相当于 16 进制 12 个字节。

例如，设置广播数据为 0x11 0x22 0x33

0x44 0x55，则设置指令为：AT+BLEADVDATA=1122334455。

5.5.1.3 AT+BLEAPRM

功能：

配置 BLE 广播参数。目前版本只支持 adv_int_min, adv_int_max 参数设置

格式（ASCII）：

AT+BLEAPRM=<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<channel_map>,[adv_filter_policy],[peer_addr_type],[peer_addr]<CR> +OK=<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<channel_map>,<adv_filter_policy>,<peer_addr_type>,<peer_addr><CR><LF><CR><LF>

参数：

adv_int_min：最小广播间隔，取值范围：0x0020 ~ 0x4000。

adv_int_max：最大广播间隔，取值范围：0x0020 ~ 0x4000。

adv_type：广播类型，定义如下：

值	含义
0	ADV_TYPE_IND
1	ADV_TYPE_DIRECT_IND_HIGH
2	ADV_TYPE_SCAN_IND
3	ADV_TYPE_NONCONN_IND

own_addr_type：BLE地址类型，定义如下：

值	含义
0	BLE_ADDR_TYPE_PUBLIC

1	BLE_ADDR_TYPE_RANDOM

channel_map：广播信道，定义如下：

值	含义
1	ADV_CHNL_37
2	ADV_CHNL_38
4	ADV_CHNL_39
7	ADV_CHNL_ALL

adv_filter_policy：过滤器，定义如下：

值	含义
0	ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
1	ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
2	ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
3	ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST

peer_addr_type：对方BLE 地址类型，定义如下：

值	含义
0	PUBLIC
1	RANDOM

peer_addr：对方 BLE 地址。

5.5.1.4 AT+BLESCPRM

功能：

配置 BLE 扫描参数。

格式（ASCII）：

AT+BLESCPRM=<window>,<interval><CR>
+OK<CR><LF><CR><LF>

参数：

windows：扫描窗口。[0x0004，0x4000],填写16进制格式，如10,FF等

interval：扫描间隔。[0x0004，0x4000]

interval的值应大于等于windows，当interval等于windows时，意味控制器始终处于扫描状态，即扫描窗口一直处于打开状态。

5.5.1.5 AT+BLESCFLT

功能：

配置扫描过滤参数。

格式（ASCII）：

AT+BLESCFLT=<filter><CR>
+OK<CR><LF><CR><LF>

参数：

filter： 过滤参数，用法暂且不祥，

注意：该指令暂不支持。

5.5.1.6 AT+BLESCAN

功能：

启动或停止扫描。

格式（ASCII）：

AT+BLESCAN=<mode><CR> +OK<CR><LF><CR><LF>

参数：

mode：操作模式，定义如下：

值	含义
0	停止扫描
1	启动扫描

扫描结果如下图所示：

```
484661B4A304,-93,HUAWEI,020102070948554157 4549
484661B4A304,-93,HUAWEI,020102070948554157 4549
484661B4A304,-97,HUAWEI,020102070948554157 4549
484661B4A304,-90,HUAWEI,020102070948554157 4549
7438B770B0E9,-83,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
6130DE163F82,-103,02011A020A0C0AFF4C001005511C041B92
6130DE163F82,-102,02011A020A0C0AFF4C001005511C041B92
484661B4A304,-91,HUAWEI,020102070948554157 4549
7438B770B0E9,-85,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
7438B770B0E9,-88,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
```

5.5.1.7 AT+BTSNAME

功能：

设置蓝牙名称。

格式（ASCII）：

AT+BTSNAME=[!]<name><CR> +OK,<name><CR><LF><CR><LF> 保存至 flash 时返回 +OK,<CR><LF><CR><LF> 不保存 flash 时返回

参数：

Name 蓝牙名称，ASCII 串。最大长度 18 字节。

5.5.1.8 AT+BTGNAME

功能：

获取蓝牙名称。

格式（ASCII）：

AT+BTGNAME= +OK,<name><CR><LF><CR><LF>

参数：

Name 蓝牙名称，ASCII 串。最大长度 18 字节。

5.5.2 BLE server AT 指令

5.5.2.1 AT+BLECTSV

功能：

创建 server。

格式（ASCII）：

AT+BLECTSV=<uuid><CR> +OK=<status><server_if><CR><LF><CR><LF>

参数：

uuid：唯一id，双字节。

status：指令执行结果，0成功。其他，错误。

server_if：server接口索引号。

注意：w800 最多支持 7 个 gatt apps。这 7 个包括 server 和 client。目前分配情况为：server 支持 3 个，client 支持 4 个。

uuid 定义：https://www.bluetooth.com/specifications/assigned-numbers/

5.5.2.2 AT+BLEADDSC

功能：

向 server 添加服务。

格式（ASCII）：

AT+BLEADDSC=<server_if>,<inst_id>,<uuid>,<num_handles><CR>
+OK=<status><server_if><service_handle><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

inst_id：默认值是1。

uuid：此服务的uuid。

num_handles：默认值是5。

service_handle:该服务值的句柄。

注意：w800支持最多8个service。注意TDS（uuid为0x1824为wifi配网专用）。用户不可以使用此uuid。

对于handles的定义：用户创建一个服务，分配一个handle值。用户每添加一个

character分配2个handle，用户每添加一个描述分配一个handle。

5.5.2.3 AT+BLEADDCH

功能：

向服务添加 characteristic 值。

格式（ASCII）：

AT+BLEADDCH=<server_if>,<service_handle>,<uuid>,<prop>,<perm><CR>
+OK=<status><server_if><service_handle><char_handle><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

service_handle：添加服务返回的句柄。

uuid：唯一id，取值格式不祥。

properties：加密授权描述，16进制格式，参见具体定义值4.5.5.3。

permissions：读写属性,16进制格式参见具体定义值4.5.5.3。

status：指令执行结果，0成功。其他，错误。

5.5.2.4 AT+BLEADESC

功能：

向服务添加描述值。

格式（ASCII）：

AT+BLEADESC=<server_if>,<service_handle>,<uuid>,<perm><CR>
+OK=<status><server_if><service_handle><desc_handle><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

service_handle：添加服务返回的句柄。

uuid：此描述服务的uuid。

permissions：读写属性,16进制格式参见具体定义值4.5.5.3。

status：指令执行结果，0成功。其他，错误。

desc_handle:该描述服务的句柄。

5.5.2.5 AT+BLESTTSC

功能：

启动服务。

格式（ASCII）：

AT+BLESTTSC=<server_if>,<service_handle>,<tran_type><CR>
+OK=<status><server_if><service_handle><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

service_handle：添加服务返回的句柄。

tran_type：BLE传输类型，默认值为2。

status：指令执行结果，0成功。其他，错误。

5.5.2.6 AT+BLESTPSC

功能：

停止服务。

格式（ASCII）：

AT+BLESTPSC=<server_if>,<service_handle><CR>
+OK=<status><server_if><service_handle><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

service_handle：添加服务返回的句柄。

status：指令执行结果，0成功。其他，错误。

5.5.2.7 AT+BLEDELSC

功能：

删除服务。

格式（ASCII）：

AT+BLEDELSC=<server_if>,<service_handle><CR>
+OK=<status><server_if><service_handle><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

service_handle：添加服务返回的句柄。

status：指令执行结果，0成功。其他，错误。

5.5.2.8 AT+BLEDESSV

功能：

注销 demo server。

格式（ASCII）：

AT+BLEDESSV=<server_if><CR> +OK=<status><server_if><CR><LF><CR><LF>

参数：

server_if：创建时的返回值。

status：指令执行结果，0成功。其他，错误。

5.5.2.9 AT+BLESCONN

功能：

连接 client。该功能暂不支持

格式（ASCII）：

AT+BLESCONN=[!?]<server_if>,<addr><CR> +OK=<conn_id><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

addr：client的蓝牙mac地址。

conn_id：连接id。

5.5.2.10 AT+BLESVDIS

功能：

断开连接的 client。

格式（ASCII）：

AT+BLESVDIS=<server_if>,<addr>,<conn_id><CR>
+OK=<status><server_if><conn_indication><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

addr：client的蓝牙mac地址。

conn_indication：1,处于连接状态，0连接断开状态

status：指令执行结果，0成功。其他，错误。

5.5.2.11 AT+BLESIND

功能：

发送 indication。

格式（ASCII）：

AT+BLESIND=<server_if><conn_id><attr_handle><data><CR>
+OK=<status><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

conn_id：创建连接时的id号。

attr_handle:创建特征值时的返回值

data：用户输入的字符串。

status：指令执行结果，0成功。其他，错误。

5.5.2.12 AT+BLESRSP

功能：

读写操作返回值。

格式（ASCII）：

AT+BLESRSP=<server_if><conn_id><attr_handle><data><CR> +OK=<status><CR><LF><CR><LF>

参数：

server_if：创建server返回的接口号。

conn_id：创建连接时的id号。

attr_handle:创建特征值时的返回值

data：用户输入的字符串。

status：指令执行结果，0成功。其他，错误。

5.5.3 BLE client AT 指令

5.5.3.1 AT+BLECCT

功能：

创建指定 uuid 的 client。

格式（ASCII）：

AT+BLECCT=<uuid><CR> +OK=<status><client_if><CR><LF><CR><LF>

参数：

uuid：唯一id。

client_if：创建client返回的接口号。

status：指令执行结果，0成功。其他，错误。

注意：w800 最多支持 7 个 gatt apps。这 7 个包括 server 和 client。目前分配情况为：server 支持 3 个，client 支持 4 个。

5.5.3.2 AT+BLECCONN

功能：

连接 server。

格式（ASCII）：

AT+BLECCONN=<client_if>,<addr><CR> +OK=<status><client_if><conn_id><CR><LF><CR><LF>

参数：

client_if：创建client返回的接口号。

addr：server的蓝牙mac地址。

conn_id：连接id。

status：指令执行结果，0成功。其他，错误。

5.5.3.3 AT+BLECSSC

功能：

扫描 server 的 service 列表。

格式（ASCII）：

AT+BLECSSC=<conn_id><CR> +OK=<status><CR><LF><CR><LF>

参数：

conn_id：连接client时返回的id。

status：指令执行结果，0成功。其他，错误。

5.5.3.4 AT+BLECGDB

功能：

返回 service 列表。

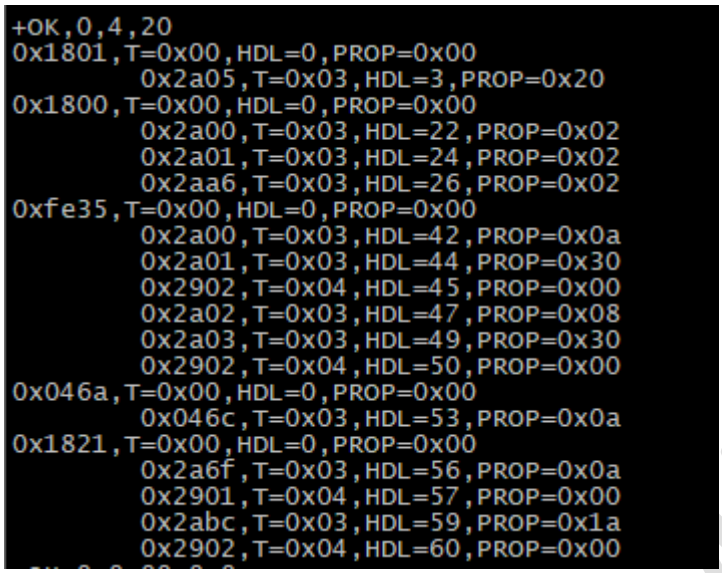
格式（ASCII）：

AT+BLECGDB=<conn_id><CR> +OK=<list><CR><LF><CR><LF>

参数：

conn_id：连接client时返回的id。

list：service 列表:



5.5.3.5 AT+BLECRNTY

功能：

注册响应 server notification 的事件。

格式（ASCII）：

AT+BLECRNTY=<client_if>,<addr>,<attr_handle><CR>
+OK=<status><conn_id><attr_handle><register_or_not><CR><LF><CR><LF>

参数：

client_if：创建client返回的接口号。

addr：mac地址。

attr_handle：service 列表中具有 notification 的 charactertisc handle 值。

conn_id:创建连接时的返回值

rcegister_or_not:表明注册或者是注销。

5.5.3.6 AT+BLECDNTY

功能：

注销已注册的 notification 响应事件。

格式（ASCII）：

AT+BLECDNTY=<client_if>,<addr>,<attr_handle><CR>
+OK=<status><conn_id><attr_handle><register_or_not><CR><LF><CR><LF>

参数：

client_if：创建client返回的接口号。

addr：mac地址。

attr_handle：service 列表中具有 notification 的 charactertisc handle 值。

conn_id:创建连接时的返回值

register_or_not:表明注册或者是注销。

5.5.3.7 AT+BLECACH

功能：

读写 charactertisc。

格式（ASCII）：

AT+BLECACH=<mode>,<conn_id>,<handle>,<auth_req>,[data]<CR>
写操作+OK=<status><conn_id><CR><LF><CR><LF>
读操作+OK=<status><conn_id><length><data><CR><LF><CR><LF>

参数：

mode：操作模式，定义如下：

值	含义
0	写操作
1	读操作

conn_id：连接client时返回的id。

handle：读写特征值的句柄。

auth_req：默认为0。

data：要写入的数据，字符串格式，只对写操作有效。

5.5.3.8 AT+BLECDIS

功能：

断开连接。

格式（ASCII）：

```
AT+BLECDIS=<client_if>,<addr>,<conn_id><CR>

+OK=<status><client_if><conn_id><reason><CR><LF><CR><LF>
```

参数：

client_if：创建client返回的接口号。

addr：mac地址。

conn_id：连接 client 时返回的 id。

reason:如果此命令由 800 发起的，reason 值一直为 0；

如果由 APP 侧发起，参见 reason 码定义,4.5.5.2

5.5.3.9 AT+BLECDES

功能：

注销 client,。

格式（ASCII）：

AT+BLECDES=<client_if><CR> +OK=<status><client_if><CR><LF><CR><LF>

参数：

client_if：创建时分配的接口值。

5.5.4 BLE 配网 AT 指令

5.5.4.1 AT+ONESHOT

功能：

启动或停止配网服务。

格式（ASCII）：

AT+ONESHOT=<mode><CR> +OK=<mode><CR><LF><CR><LF>

参数：

mode：操作模式，定义如下：

值	含义
0	停止配网
1	启动UDP配网
2	启动SoftAP+Socket配网

3	启动SoftAP+WebServer配网
4	启动蓝牙配网

注意：

启动蓝牙配网后，用户可以使用手机 APP 进行配置 WiFi 信息。配网成功后，配网服务自动注销，蓝牙关闭广播。如需再次配网请再次启动蓝牙配网。

5.5.5 传统蓝牙音频 AT 指令

待添加

5.5.6 传统蓝牙免提电话 AT 指令

待添加

5.5.7 状态码定义：

5.5.7.1 GATT Status 定义

BTA_GATT_OK	0x0000
BTA_GATT_INVALID_HANDLE	0x0001
BTA_GATT_READ_NOT_PERMIT	0x0002
BTA_GATT_WRITE_NOT_PERMIT	0x0003
BTA_GATT_INVALID_PDU	0x0004
BTA_GATT_INSUF_AUTHENTICATION	0x0005

BTA_GATT_REQ_NOT_SUPPORTED	0x0006
BTA_GATT_INVALID_OFFSET	0x0007
BTA_GATT_INSUF_AUTHORIZATION	0x0008
BTA_GATT_PREPARE_Q_FULL	0x0009
BTA_GATT_NOT_FOUND	0x000A
BTA_GATT_NOT_LONG	0x000B
BTA_GATT_INSUF_KEY_SIZE	0x000C
BTA_GATT_INVALID_ATTR_LEN	0x000D
BTA_GATT_ERR_UNLIKELY	0x000E
BTA_GATT_INSUF_ENCRYPTION	0x000F
BTA_GATT_UNSUPPORT_GRP_TYPE	0x0010
BTA_GATT_INSUF_RESOURCE	0x0011
BTA_GATT_NO_RESOURCES	0x80
BTA_GATT_INTERNAL_ERROR	0x81
BTA_GATT_WRONG_STATE	0x82
BTA_GATT_DB_FULL	0x83
BTA_GATT_BUSY	0x84
BTA_GATT_ERROR	0x85
BTA_GATT_CMD_STARTED	0x86
BTA_GATT_ILLEGAL_PARAMETER	0x87
BTA_GATT_PENDING	0x88
BTA_GATT_AUTH_FAIL	0x89

BTA_GATT_MORE	0x8a
BTA_GATT_INVALID_CFG	0x8b
BTA_GATT_SERVICE_STARTED	0x8c
BTA_GATT_ENCRYPED_NO_MITM	0x8d
BTA_GATT_NOT_ENCRYPTED	0x8e
BTA_GATT_CONGESTED	0x8f
BTA_GATT_DUP_REG	0x90
BTA_GATT_ALREADY_OPEN	0x91
BTA_GATT_CANCEL	0x92
BTA_GATT_CCC_CFG_ERR	0xFD
BTA_GATT_PRC_IN_PROGRESS	0xFE
BTA_GATT_OUT_OF_RANGE	0xFF

5.5.7.2 Reason 码定义：

Success	0x00
Unknown HCI Command	0x01
Unknown Connection Identifier	0x02
Hardware Failure	0x03
Page Timeout	0x04
Authentication Failure	0x05
PIN or Key Missing	0x06

Memory Capacity Exceeded	0x07
Connection Timeout	0x08
Connection Limit Exceeded	0x09
Synchronous Connection Limit To A Device Exceeded	0x0a
ACL Connection Already Exists	0x0b
Command Disallowed	0x0c
Connection Rejected due to Limited Resources	0x0d
Connection Rejected Due To Security Reasons	0x0e
Connection Rejected due to Unacceptable BD_ADDR	0x0f
Connection Accept Timeout Exceeded	0x10
Unsupported Feature or Parameter Value	0x11
Invalid HCI Command Parameters	0x12
Remote User Terminated Connection	0x13
Remote Device Terminated Connection due to Low Resources	0x14
Remote Device Terminated Connection due to Power Off	0x15
Connection Terminated By Local Host	0x16
Repeated Attempts	0x17
Pairing Not Allowed	0x18

Unknown LMP PDU	0x19
Unsupported Remote Feature / Unsupported LMP Feature	0x1a
SCO Offset Rejected	0x1b
SCO Interval Rejected	0x1c
SCO Air Mode Rejected	0x1d
Invalid LMP Parameters / Invalid LL Parameters	0x1e
Unspecified Error	0x1f
Unsupported LMP Parameter Value / Unsupported LL Parameter Value	0x20
Role Change Not Allowed	0x21
LMP Response Timeout / LL Response Timeout	0x22
LMP Error Transaction Collision	0x23
LMP PDU Not Allowed	0x24
Encryption Mode Not Acceptable	0x25
Link Key cannot be Changed	0x26
Requested QoS Not Supported	0x27
Instant Passed	0x28
Pairing With Unit Key Not Supported	0x29
Different Transaction Collision	0x2a
Reserved	0x2b
QoS Unacceptable Parameter	0x2c

QoS Rejected	0x2d
Channel Classification Not Supported	0x2e
Insufficient Security	0x2f
Parameter Out Of Mandatory Range	0x30
Reserved	0x31
Role Switch Pending	0x32
Reserved	0x33
Reserved Slot Violation	0x34
Role Switch Failed	0x35
Extended Inquiry Response Too Large	0x36
Secure Simple Pairing Not Supported By Host	0x37
Host Busy – Pairing	0x38
Connection Rejected due to No Suitable Channel Found	0x39
Controller Busy	0x3a
Unacceptable Connection Parameters	0x3b
Directed Advertising Timeout	0x3c
Connection Terminated due to MIC Failure	0x3d
Connection Failed to be Established	0x3e
MAC Connection Failed	0x3f

5.5.7.3 Permissions and properties 定义

/** Attribute permissions */

```
#define WM_GATT_PERM_READ                   (1 << 0) /**< bit 0 -  0x0001 */
#define WM_GATT_PERM_READ_ENCRYPTED         (1 << 1) /**< bit 1 -  0x0002 */
#define WM_GATT_PERM_READ_ENC_MITM          (1 << 2) /**< bit 2 -  0x0004 */
#define WM_GATT_PERM_WRITE                  (1 << 4) /**< bit 4 -  0x0010 */
#define WM_GATT_PERM_WRITE_ENCRYPTED        (1 << 5) /**< bit 5 -  0x0020 */
#define WM_GATT_PERM_WRITE_ENC_MITM         (1 << 6) /**< bit 6 -  0x0040 */
#define WM_GATT_PERM_WRITE_SIGNED           (1 << 7) /**< bit 7 -  0x0080 */
#define WM_GATT_PERM_WRITE_SIGNED_MITM (1 << 8) /**< bit 8 -  0x0100 */

/** definition of characteristic properties */
#define WM_GATT_CHAR_PROP_BIT_BROADCAST     (1 << 0)    /**< 0x01 */
#define WM_GATT_CHAR_PROP_BIT_READ          (1 << 1)    /**< 0x02 */
#define WM_GATT_CHAR_PROP_BIT_WRITE_NR      (1 << 2)    /**< 0x04 */
#define WM_GATT_CHAR_PROP_BIT_WRITE         (1 << 3)    /**< 0x08 */
#define WM_GATT_CHAR_PROP_BIT_NOTIFY        (1 << 4)    /**< 0x10 */
#define WM_GATT_CHAR_PROP_BIT_INDICATE      (1 << 5)    /**< 0x20 */
#define WM_GATT_CHAR_PROP_BIT_AUTH          (1 << 6)    /**< 0x40 */
#define WM_GATT_CHAR_PROP_BIT_EXT_PROP      (1 << 7)    /**< 0x80 */
```

6 蓝牙 AT 指令操作示例

本章节结合具体示例，给出了蓝牙 AT 指令具体操作规范。黑色截图即 AT 指令的响应。

6.1 蓝牙系统使能与退出

6.1.1 使能蓝牙系统

AT+BTEN=1,0

```
+OK=0,1
```

6.1.2 退出蓝牙系统

AT+BTDES

```
+OK=0,0
```

6.2 使能 WIFI 配网服务

6.2.1 开启蓝牙功能，使能配网服务

AT+BTEN=1,0 //使能蓝牙系统

AT+ONESHOT=4 //开启配网服务

此时可以用 APP 进行配网操作；注意配网成功后，系统会自动注销配网服务。

```
+OK=0,1

+OK
```

6.2.2 退出 WIFI 配网服务注销蓝牙系统

AT+ONESHOT=0 //退出配网服务

AT+BTDES //退出蓝牙系统

6.3 BLE server 操作示例

本章节描述了通过 AT 指令创建 BLE server 和手机端 Nrf connect APP 进行交互操作。

6.3.1 使能蓝牙系统

AT+BTEN=1,0

```
+OK=0,1
```

6.3.2 创建 server

AT+BLECTSV=9999 //创建 uuid 为 9999 的 server

```
+OK=0,4
```

6.3.3 添加服务

AT+BLEADDSC=4,1,1826,5 //添加 uuid 为 1826 的服务

//1824uuid，为蓝牙配网专用。

```
+OK=0,4,40
```

6.3.4 添加特征值

AT+BLEADDCH=4,40,2abc,28,11 //添加 uuid 为 2abc 的特性值

```
+OK=0,4,40,42
```

6.3.5 添加特征值描述

AT+BLEADESC=4,40,2902,11 //添加 uuid 为 2902 的特性值描述

```
+OK=0,4,40,43
```

6.3.6 开启服务

AT+BLESTTSC=4,40,2 //开启服务

```
+OK=0,4,40
```

6.3.7 配置广播数据

AT+BLEADATA=1,11223344 //设置广播内容,自动开启广播

//广播内容，包含设备名称，

//FF 字段内容为 0x11223344

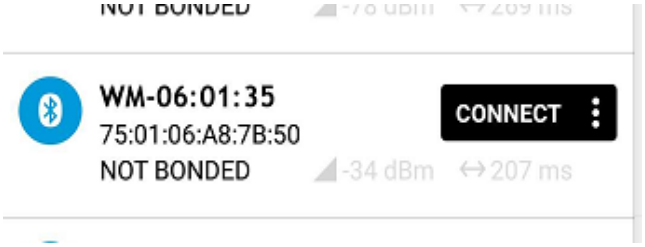
6.3.8 开启广播

AT+BLEADV=1 //使能广播

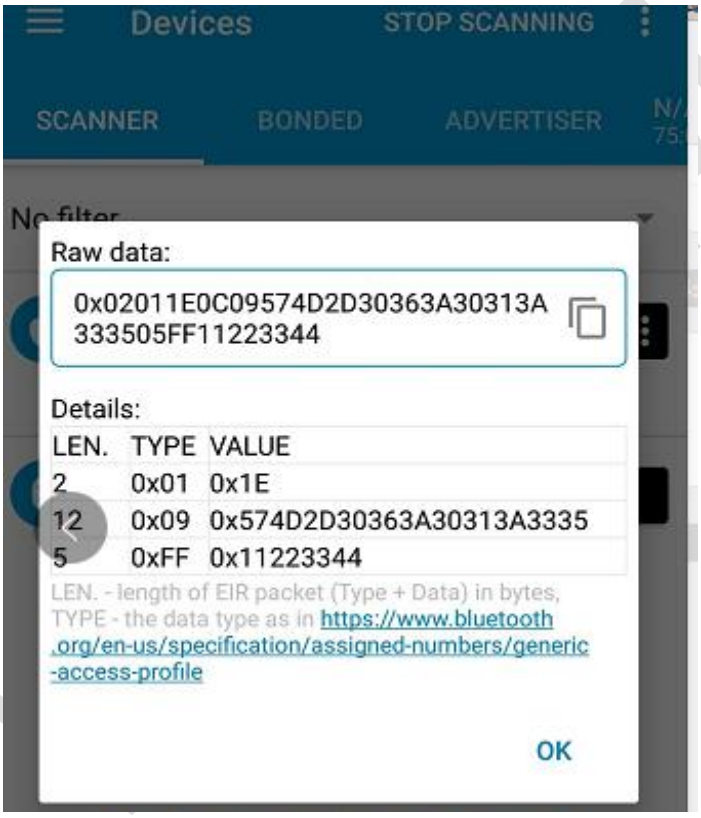
```
+OK
```

6.3.9 手机开始扫描

Nrf connect 扫描结果：



手机端广播数据显示：



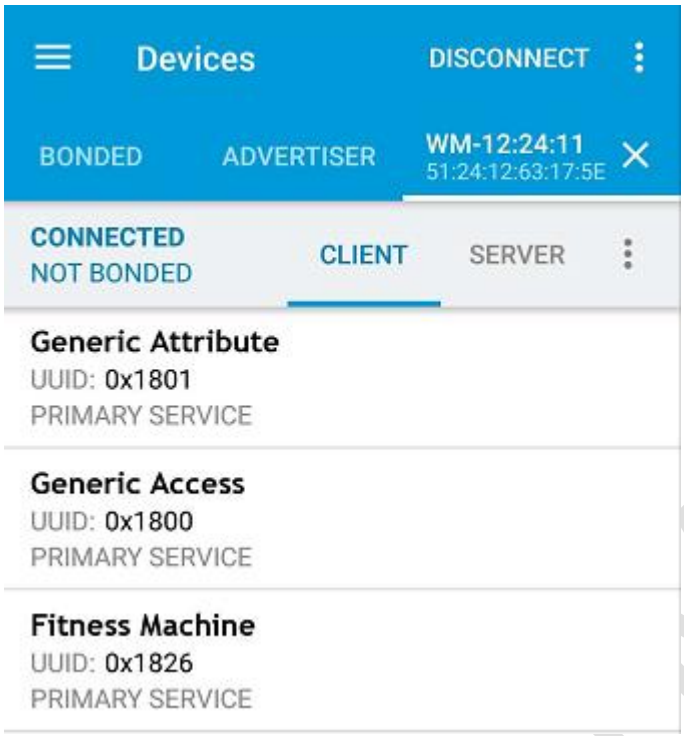
6.3.10 手机侧发起连接

点击 CONNECT 按钮，此时 w800 会显示：

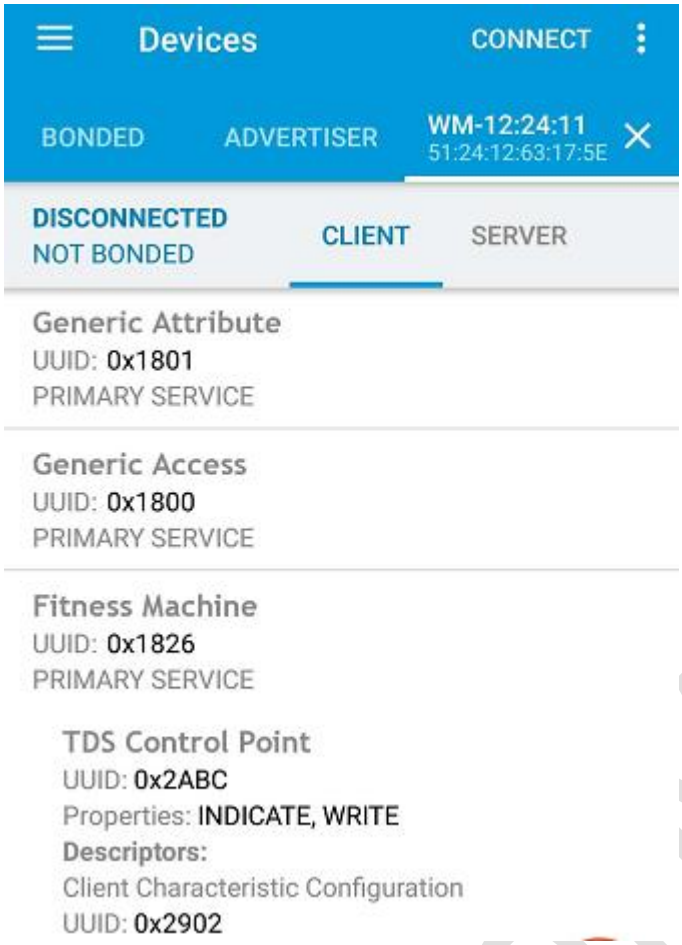
```
+OK=0,4,4,1,60C25D5A4CC1
```

即手机的 MAC 地址 60C25D5A4CC1 连接成功。连接成功后，手机侧可以看到我们创建的服务描述。

下图中，UUID:0x1826 即为我们创建的服务。

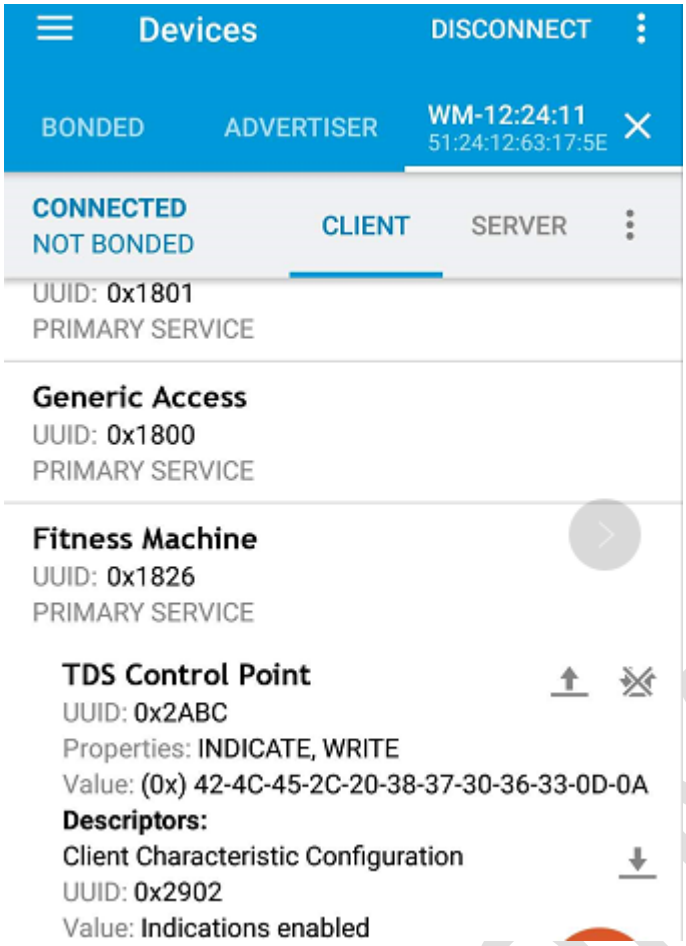


单击 Transport Discovery 即可看到我们创建的特性值及描述



6.3.11 手机侧使能 Indication 功能

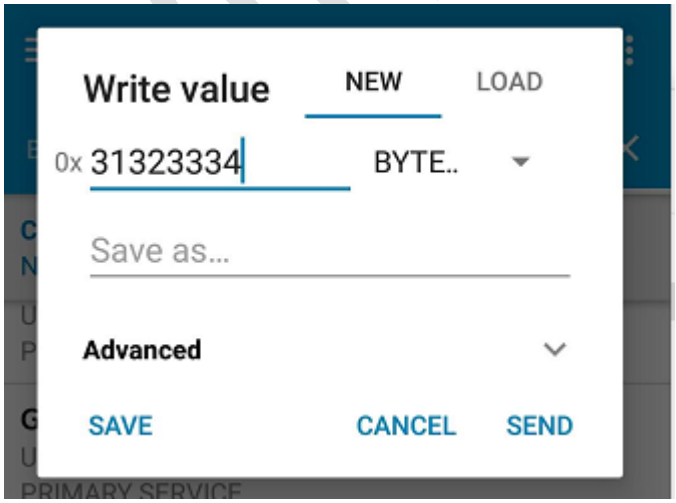
点击 0x2abc 右侧的上下箭头，代表使能 indicate 操作，此后，W800 会每隔 2S 发送字符串给手机，显示如下：BLE，当前系统时间，下面显示的 HEX 格式



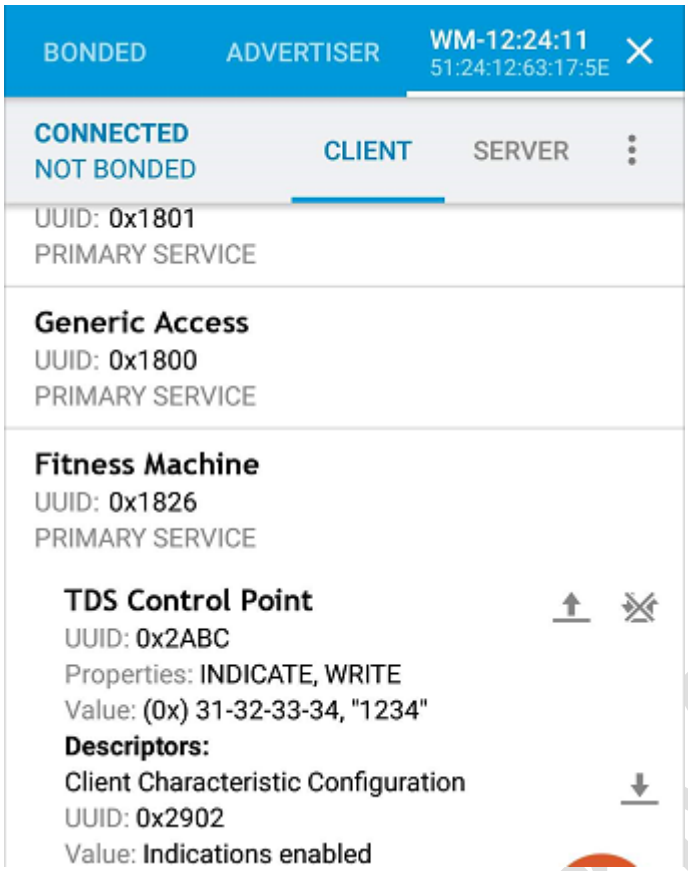
再次点击 0X2ABC 右侧的 上下箭头，此时形状为 ，代表停止 indication 发送。

6.3.12 手机侧写取特征值数据

点击 0X2ABC 右侧向上箭头，代表特性值写操作， W800 会将收到的内容返回回来。



APP 显示收到的返回值：

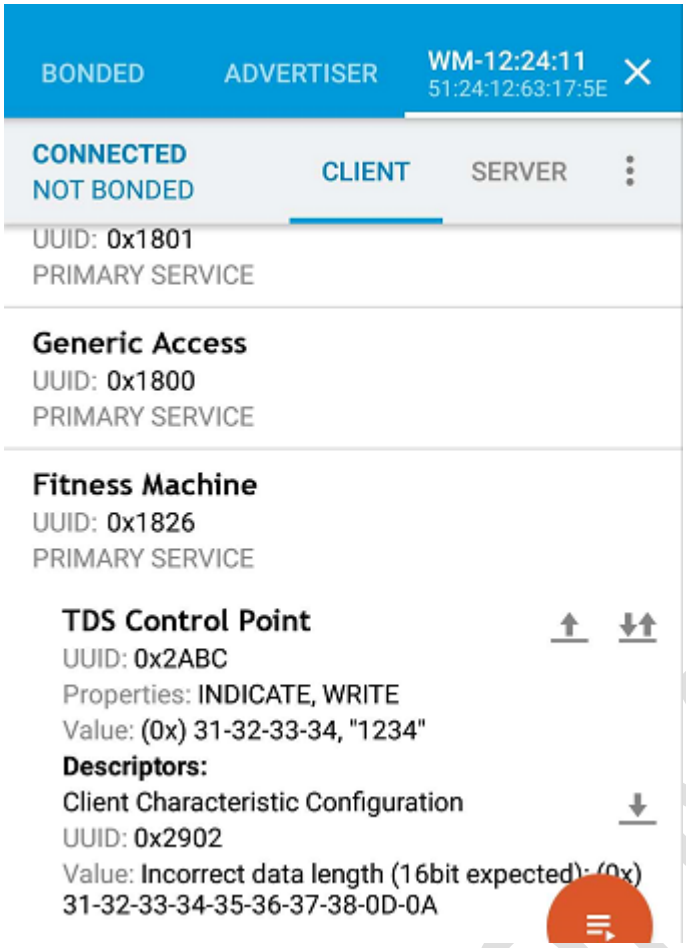


此时 W800 侧会显示收到的内容：

```
+OK=0,4,42,1,4,31323334
+OK=0
```

6.3.13 手机侧读取描述符

点击描述符右侧的读操作，向下的箭头，代表读取描述内容，W800 返回 12345678 字符串：



6.3.14 断开与手机的连接

AT+BLESVDIS=4,047EB5A65FCB,4 //断开与 server 的连接，

//client_if 4，地址 047EB5A65FCB，ID 为 4

```
+OK=0,4,0
```

6.3.15 停止服务

AT+BLESTPSC=4,40 //停止句柄为 40 的服务

```
+OK=0,4,40
```

6.3.16 删除服务

AT+BLEDELSC=4,40

```
+OK=0,4,40
```

6.3.17 注销 server

AT+BLEDESSV=4 //注销 client_if 为 4 的 server

```
+OK=0,4
```

6.3.18 注销蓝牙服务

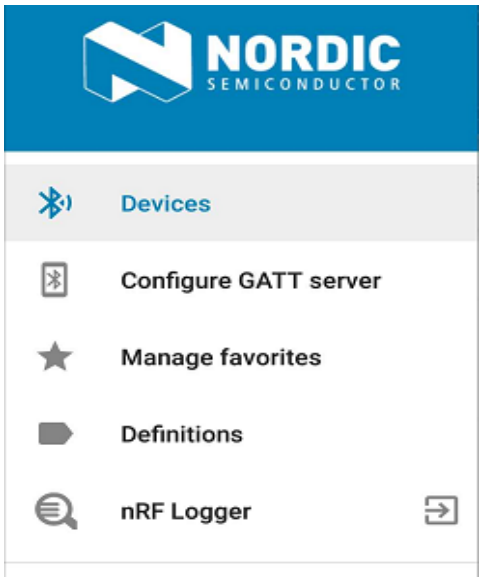
AT+BTDES

6.4 BLE client 操作示例

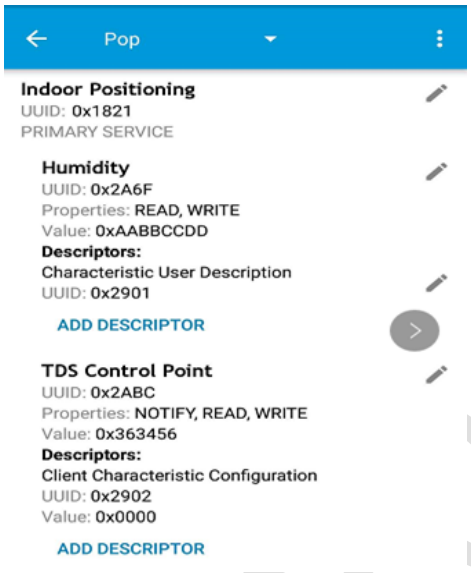
本章节介绍了手机端创建 BLE Server，配置特征值即描述，开启广播。W800 进行扫描，连接，读取特征值操作。手机端依然使用 Nrf connect APP.

6.4.1 手机端创建 server

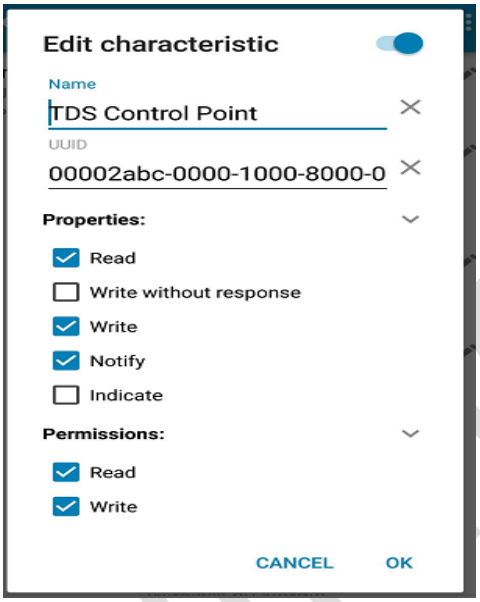
配置 Nrf connect GATT server 流程如下图所示，需要添加服务，配置特征值独写属性及属性值然后开启广播功能：



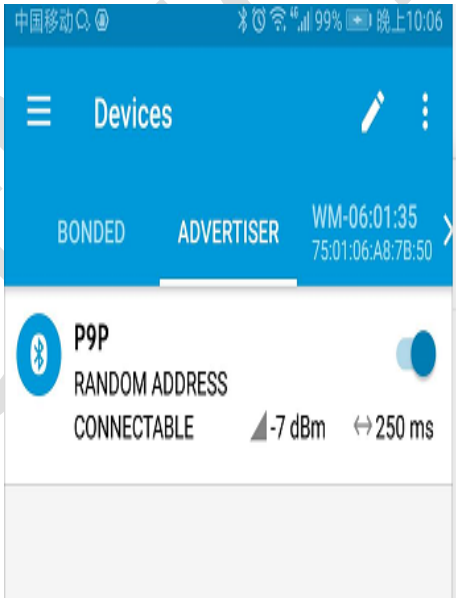
步骤一



步骤二



步骤三



步骤四

6.4.2 W800 使能蓝牙

AT+BTEN=1,0 //使能蓝牙系统

```
+OK=0,1
```

6.4.3 W800 创建 client

AT+BLECCT=8888 //创建 uuid 为 8888 的 client

```
+OK=0,4
```

6.4.4 W800 开启扫描

AT+BLESCAN=1 //开始扫描

```
7438B770B0E9,-89,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
7438B770B0E9,-83,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
71C1608D025C,-93,HUAWEI,020102070948554157 4549
71C1608D025C,-87,HUAWEI,020102070948554157 4549
71C1608D025C,-86,HUAWEI,020102070948554157 4549
```

此时可以看到 设备名称 huawei 即手机发送的广播内容。

6.4.5 W800 停止扫描

AT+BLESCAN=0 //停止扫描

6.4.6 W800 连接 server

AT+BLECCONN=4,71C1608D025C //连接手机

连接错误示例,此时可以重复执行连接操作。

```
+OK=133,4,0
```

连接成功示例:

```
+OK=0,4,4
```

6.4.7 W800 扫描服务列表

AT+BLECSSC=4 //扫描 server 的服务列表

```
+OK=0,4, CMPLT
```

6.4.8 W800 读取服务列表

AT+BLECGDB=4 //读取服务列表

```
+OK=0,4,20
0x1801,T=0x00,HDL=0,PROP=0x00
        0x2a05,T=0x03,HDL=3,PROP=0x20
0x1800,T=0x00,HDL=0,PROP=0x00
        0x2a00,T=0x03,HDL=22,PROP=0x02
        0x2a01,T=0x03,HDL=24,PROP=0x02
        0x2aa6,T=0x03,HDL=26,PROP=0x02
0xfe35,T=0x00,HDL=0,PROP=0x00
        0x2a00,T=0x03,HDL=42,PROP=0x0a
        0x2a01,T=0x03,HDL=44,PROP=0x30
        0x2902,T=0x04,HDL=45,PROP=0x00
        0x2a02,T=0x03,HDL=47,PROP=0x08
        0x2a03,T=0x03,HDL=49,PROP=0x30
        0x2902,T=0x04,HDL=50,PROP=0x00
0x046a,T=0x00,HDL=0,PROP=0x00
        0x046c,T=0x03,HDL=53,PROP=0x0a
0x1821,T=0x00,HDL=0,PROP=0x00
        0x2a6f,T=0x03,HDL=56,PROP=0x0a
        0x2901,T=0x04,HDL=57,PROP=0x00
        0x2abc,T=0x03,HDL=59,PROP=0x1a
        0x2902,T=0x04,HDL=60,PROP=0x00
```

6.4.9 W800 读取特性值

AT+BLECACH=1,4,59,0 //读取感兴趣的句柄值。本例读取的为 59. 此时返回值为：长度为 3，内容为 16 进制字符串 363456

```
+OK=0,4,3,363456
```

6.4.10 W800 断开连接

AT+BLECDIS=4,63573EA5A2F7,4

```
+OK=0,4,4,0
```

6.4.11 W800 注销 client

AT+BLECDES=4

```
+OK=0,4
```

6.4.12 W800 注销蓝牙服务

AT+BTDES

6.5 传统蓝牙音频操作示例

待添加

6.6 传统蓝牙免提电话操作示例

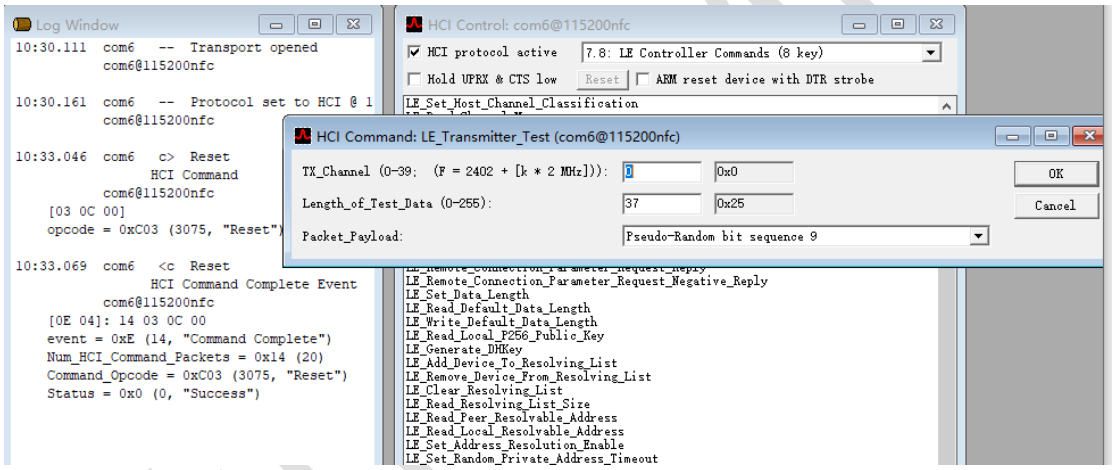
待添加

6.7 W800 测试模式

W800 支持实时进入测试模式，客户可以用于测试 RF 性能及控制器功能测试和认证测试。

6.7.1 W800 进入测试模式

AT+BTTEST=1 //进入蓝牙测试，此时可以用测试工具通过配置的 uart 口直接操作 controller。



6.7.2 W800 退出信令测试

AT+BTTEST=0 //退出测试模式，此时主机协议栈控制 controller.