

青风带你玩蓝牙 nRF52832 系列教程.....	2
-----作者: 青风.....	2
作者: 青风.....	3
出品论坛: www.qfv8.com	3
淘宝店: http://qfv5.taobao.com	3
QQ 技术群: 346518370.....	3
硬件平台: 青云 QY-nRF52832 开发板.....	3
蓝牙广播初始化分析.....	3
1: nRF52832 蓝牙 BLE 广播初始化:	3
1.1 广播参数定义:	4
1.2 广播模式配置:	7
1.3 BLE 广播初始化步骤:	9
2: 不进入 IDLE 无效模式.....	13
3: BLE 广播响应包:	15

青风带你玩蓝牙 nRF52832 系列教程

-----作者: 青风

出品论坛: www.qfv8.com 青风电子社区



作者: 青风

出品论坛: www.qfv8.com

淘宝店: <http://qfv5.taobao.com>

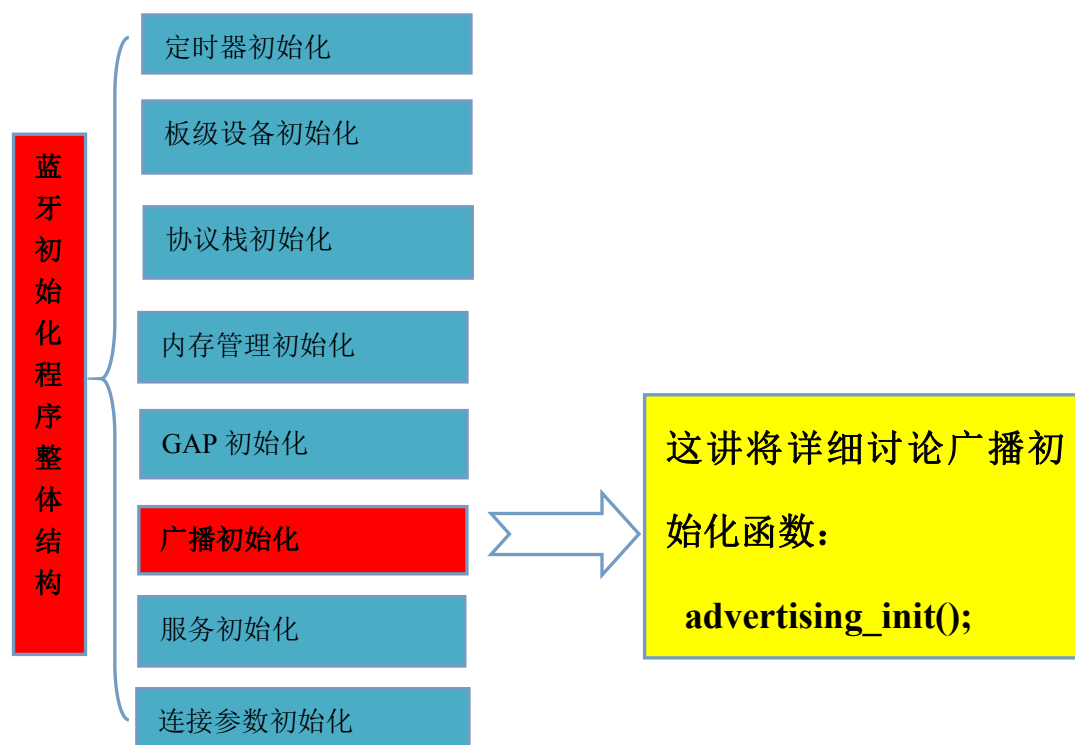
QQ 技术群: 346518370

硬件平台: 青云 QY-nRF52832 开发板

蓝牙广播初始化分析

对应蓝牙广播的初始化, 很多读者无法理解其设置和运行过程, 不知道相关的基础知识, 这一节将详细的进行讨论。

并且通过分析基本原理, 设置一个固定广播模式, 本例在匹配的 SDK11.0 的蓝牙样例的例子基础上就行编写, 使用的协议栈为: s132。



1: nRF52832 蓝牙 BLE 广播初始化:

首先看下蓝牙样例里的广播初始化代码如下所示:

```
static void advertising_init(void)
{
    uint32_t      err_code;
    ble_advertising_init_t  init;
    //
    memset(&advdata, 0, sizeof(advdata));

    init.advdata.name_type = BLE_ADVDATA_FULL_NAME;//广播时的名称显示
    init.advdata.include_appearance = true;//是否需要图标
    init.advdata.flags= BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
                        //蓝牙设备模式
    init.advdata.uuids_complete.uuid_cnt=sizeof(m_adv_uuids)/ sizeof(m_adv_uuids[0]);
                        //UUID
    init.advdata.uuids_complete.p_uuids  = m_adv_uuids;

    init.config.ble_adv_fast_enabled  = true;//广播类型
    init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;//广播间隔
    init.config.ble_adv_fast_timeout  = APP_ADV_DURATION;//广播超时

    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);//初始化广播，导入参数
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising, APP_BLE_CONN_CFG_TAG);
    //设置广播识别号
}
```

广播初始化实际上就是初始化两个结构体，一个是&advdata 广播数据，一个是 &config 选择项，下面我们就具体连对广播初始化进行分析。

1.1 广播参数定义:

&advdata 广播数据定义了广播的基本参数，如下图所示，比如名称类型，最短名称长度，设备模式，发射功率等等，不是所有参数你都需要在初始化函数中进行设置。只需要选择自己需要设置的，比如我们讲发射功率的时候就需要添 p_tx_power_level 参数等。

```
typedef struct
{
    ble_advdata_name_type_t    name_type;          /**< Type of device name. */
    uint8_t                    short_name_len;      /**< Length of short device name (if short type is specified). */
    bool                        include_appearance;  /**< Determines if Appearance shall be included. */
    uint8_t                    flags;               /**< Advertising data Flags field. */
    int8_t *                    p_tx_power_level;   /**< TX Power Level field. */
    ble_advdata_uuid_list_t    uuids_more_available; /**< List of UUIDs in the 'More Available' list. */
    ble_advdata_uuid_list_t    uuids_complete;     /**< List of UUIDs in the 'Complete' list. */
    ble_advdata_uuid_list_t    uuids_solicited;    /**< List of solicited UUIDs. */
    ble_advdata_conn_int_t *    p_slave_conn_int;   /**< Slave Connection Interval Range. */
    ble_advdata_manuf_data_t *  p_manuf_specific_data; /**< Manufacturer specific data. */
    ble_advdata_service_data_t * p_service_data_array; /**< Array of Service data structures. */
    uint8_t                    service_data_count;  /**< Number of Service data structures. */
    bool                        include_ble_device_addr; /**< Determines if LE Bluetooth Device Address shall be included. */
    ble_advdata_le_role_t       le_role;           /**< LE Role field. Included when different from @ref BLE_ADVDA... */
    ble_advdata_tk_value_t *    p_tk_value;        /**< Security Manager TK value field. Included when different fr... */
    uint8_t *                    p_sec_mgr_oob_flags; /**< Security Manager Out Of Band Flags field. Included when dif... */
    ble_gap_lesec_oob_data_t *  p_lesec_data;      /**< LE Secure Connections OOB data. Included when different fro... */
} ble_advdata_t;
```

我们分析一下 BLE 样例里给的一般需要的几个设置项:

1. `init.advdata.name_type = BLE_ADVDATA_FULL_NAME;` //广播时的名称显示
设置广播里, 广播名称类型, 类型就三种:

```
typedef enum
{
    BLE_ADVDATA_NO_NAME,
    BLE_ADVDATA_SHORT_NAME,
    BLE_ADVDATA_FULL_NAME
} ble_advdata_name_type_t;
```

没有名称, 短名称, 全名三种设置。这个可以更具自己需要设置, 具体见《蓝牙笔记: GAP 详解》里讲述了。

2. `init.advdata.include_appearance = true;` 设置是否需要展示图标, 在 GAP 中设置图标, 具体内容见《蓝牙笔记: GAP 详解》里讲述了。

3. `init.advdata.flags= BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;` //蓝牙设备模式

这个设置蓝牙设备的模式, 选项如下:

```
/** @{ */
#define BLE_GAP_ADV_FLAG_LE_LIMITED_DISC_MODE (0x01) /**< LE Limited Discoverable Mode. */
#define BLE_GAP_ADV_FLAG_LE_GENERAL_DISC_MODE (0x02) /**< LE General Discoverable Mode. */
#define BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED (0x04) /**< BR/EDR not supported. */
#define BLE_GAP_ADV_FLAG_LE_BR_EDR_CONTROLLER (0x08) /**< Simultaneous LE and BR/EDR, Controller. */
#define BLE_GAP_ADV_FLAG_LE_BR_EDR_HOST (0x10) /**< Simultaneous LE and BR/EDR, Host. */
#define BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE (BLE_GAP_ADV_FLAG_LE_LIMITED_DISC_MODE | BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED)
#define BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE (BLE_GAP_ADV_FLAG_LE_GENERAL_DISC_MODE | BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED)
/** @} */
```

Value	Description	Bit	Information
0x01	Flags	0	LE Limited Discoverable Mode
		1	LE General Discoverable Mode
		2	BR/EDR Not Supported (i.e. bit 37 of LMP Extended Feature bits Page 0)
		3	Simultaneous LE and BR/EDR to Same Device Capable (Controller) (i.e. bit 49 of LMP Extended Feature bits Page 0)
		4	Simultaneous LE and BR/EDR to Same Device Capable (Host) (i.e. bit 66 of LMP Extended Feature bits Page 1)
		5..7	Reserved

Table 18.1: Flags

蓝牙模式设置通过 flags 来进行标识:

Bit0:LE 有限发现模式

Bit1:LE 普通发现模式

Bit2:不支持 **BR/EDR 模式**

Bit3:同时支持 BLE 和 **BR/EDR 模式 (控制器)**

Bit4:同时支持 BLE 和 **BR/EDR 模式 (主机)**

如上表所示, 蓝牙 nrf52832 实际上设置的蓝牙类型是严格限定的, 其是单模蓝牙, 不支持传统蓝牙的。通常设置两种情况

BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE: **LE 有限发现模式和不支持 BR/EDR 模式。**

BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE: **LE 普通发现模式和不支持 BR/EDR 模式。**

LE 和 BR/EDR 这两个英文解释一下, LE 也就是 BLE 后面两个字 LE, 可以把 BLE 写成: Bluetooth Smart 也就是低功耗蓝牙。LE 有限发现模式和 LE 普通发现模式是有区别的, 有限发现模式下, 设备广播间隔比一般发现模式小, 同时持续时间有限。而普通发现模式则没有时间限制, 如果你需要你的设备一直广播不休眠, 就必须把它设置为普通发现模式。




蓝牙 BR/EDR (蓝牙基本速率/增强数据率), 低功耗是 Bluetooth Smart 的亮点之一。Bluetooth Smart 设备仅靠一颗纽扣电池就能运行数月甚至数年之久。Bluetooth Smart 灵活的配置也让应用能够更好地管理连接间隔 (connection interval), 以优化接收机的工作周期。对于蓝牙 BR/EDR, 由于其数据吞吐量更高, 功耗也会相应增加。

BR/EDR 配置文件包括: 耳机 (HSP)、对象交换 (OBEX)、音频传输 (A2DP)、视频传输 (VDP) 和文件传输 (FTP)。也就是一些大的数据的传输。

经常会有人问我, 如果一个蓝牙耳机兼容蓝牙 4.x, 是否意味着这副耳机是低功耗的呢? Bluetooth Smart 可否用于音频应用?

Bluetooth 4.x 核心规格中有一卷是低功耗控制器, 还有一卷是蓝牙 BR/EDR 控制器。通常情况下, 如果耳机支持 4.x, 则兼容 4.x BR/EDR 规格, 而不兼容低功耗规格或 Bluetooth Smart。可以通过辨认产品包装上的 Bluetooth Smart 商标确认其是否为 Bluetooth Smart 产品。**所以注意 nrf52832 是不是用来传输音频的方案。**

除了这个之外, 蓝牙商标其实有三种, 可用来区分产品所采用的蓝牙类型。制造商会在产品本身或其包装上使用这些商标。

 Bluetooth [®]	Bluetooth BR/EDR
 Bluetooth [®] SMART	Bluetooth Smart
 Bluetooth [®] SMART READY	Bluetooth Smart Ready

最后一种称为: **Bluetooth Smart Ready** 设备可以接收来自其他蓝牙设备的数据、这些数据可以被 **Bluetooth Smart Ready** 设备上的应用转化成有用的信息, 比如智能手机、个人电脑、平板电脑等都是 **Bluetooth Smart Ready** 设备。

```
4. init.advdata.uuids_complete.uuid_cnt=sizeof(m_adv_uuids)/sizeof(m_adv_uuids[0]); //UUID
    advdata.uuids_complete.p_uuids = m_adv_uuids;
```

设置 UUID 类型和 UUID 设备信息服务, 关于 UUID 的设置见《青风带你学蓝牙: 蓝牙 UUID 设置》这篇文章里有详细的介绍, 这里就不重复了。

1.2 广播模式配置:

再来看 `&confing` 选择项, `confing` 为 `ble_adv_modes_config_t` 结构体, 设置了如下图所示的相关参数:

```

119 typedef struct
120 {
121     bool    ble_adv_on_disconnect_disabled;
122     bool    ble_adv_whitelist_enabled;
123     bool    ble_adv_directed_high_duty_enabled;
124     bool    ble_adv_directed_enabled;
125     bool    ble_adv_fast_enabled;
126     bool    ble_adv_slow_enabled;
127     uint32_t ble_adv_directed_interval;
128     uint32_t ble_adv_directed_timeout;
129     uint32_t ble_adv_fast_interval;
130     uint32_t ble_adv_fast_timeout;
131     uint32_t ble_adv_slow_interval;
132     uint32_t ble_adv_slow_timeout;
133     bool    ble_adv_extended_enabled;
134     uint32_t ble_adv_secondary_phy;
135     uint32_t ble_adv_primary_phy;
136     ble_adv_modes_config_t;

```

这个结构体设置了广播的模式配置参数，比如比较多，包括快速广播，慢速广播，定向广播等。
注：广播扩展是蓝牙 5.0 新增加的方式。

程序设置里为： `config.ble_adv_fast_enabled = BLE_ADV_FAST_ENABLED;` //广播类型

◎这个里面关键的问题就是广播类型的选择，广播类型有几种，如下图所示（英文解释具体查看官方 SDK 说明）：

Advertising mode	Behavior
Directed High Duty	After disconnecting, the application immediately attempts to reconnect to the peer that was connected most recently. This advertising mode is very useful to stay connected to one peer and seamlessly recover from accidental disconnects. This is only allowed very briefly since it has a high chance of blocking other wireless traffic. Directed High Duty will only work if extended advertising is disabled. ble_adv_modes_config_t .
Directed	After the initial high duty advertising burst, The application can continue advertising directly to its last peer, but with lower duty cycle compared to "Directed High Duty" above.
Fast	The application rapidly advertises to surrounding devices for a short time.
Slow	The application advertises to surrounding devices, but with a longer advertising interval than in fast advertising mode. This advertising mode conserves power and causes less traffic for other wireless devices that might be present. However, finding a device and connecting to it might take more time in slow advertising mode.
Idle	The application stops advertising.

解释一下：

Direct 高速连接任务模式：利用的就是 ble 中的直连广播，该模式是为了快速重连上刚刚断开的配对设备。比如利用在快速重连上意外断开的设备，已达到无缝恢复的目的。这种模式只有在扩展广播被关闭下才能够使用

Direct 模式：这种方式比高速模式的广播周期要低，为一种定向广播模式。

Fast 模式：就是普通的广播，不过连接间隔我们可以设置的快一点。

Slow 模式：普通广播，连接间隔设置的慢一点

Idle 模式：停止广播。

◎广播间隔 interval: 广播包的广播时间间隔, 不同的广播类型下都有自己对应的广播间隔。

工程中设置为: `init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;`

```
typedef struct
{
    bool    ble_adv_on_disconnect_disabled;
    bool    ble_adv_whitelist_enabled;
    bool    ble_adv_directed_high_duty_enabled;
    bool    ble_adv_directed_enabled;
    bool    ble_adv_fast_enabled;
    bool    ble_adv_slow_enabled;
    uint32_t ble_adv_directed_interval;
    uint32_t ble_adv_directed_timeout;
    uint32_t ble_adv_fast_interval;
    uint32_t ble_adv_fast_timeout;
    uint32_t ble_adv_slow_interval;
    uint32_t ble_adv_slow_timeout;
    bool    ble_adv_extended_enabled;
    uint32_t ble_adv_secondary_phy;
    uint32_t ble_adv_primary_phy;
} ble_adv_modes_config_t;
```

◎广播时间 timeout: 广播的时间, 超过这个时间会发生广播超时处理。

工程中设置为 `init.config.ble_adv_fast_timeout = APP_ADV_DURATION;`

1.3 BLE 广播初始化步骤:

蓝牙样例中, 在广播初始化程序里设置为 `init.config.ble_adv_fast_enabled = true;` 广播类型, 为快速广播。那么开始广播就是直接进入快速广播。整个启动过程交给了主函数中的 `ble_advertising_start` 来实现的:

```
827 int main(void)
828 {
829     bool erase_bonds;
830
831     // Initialize.
832     log_init();
833     timers_init();
834     buttons_leds_init(&erase_bonds);
835     power_management_init();
836     ble_stack_init();
837     gap_params_init();
838     gatt_init();
839     advertising_init();
840     services_init();
841     conn_params_init();
842     peer_manager_init();
843
844     // Start execution.
845     NRF_LOG_INFO("Template example started.");
846     application_timers_start();
847
848     advertising_start(erase_bonds);
849
850     // Enter main loop.
851     for (;;)
852     {
853         idle_state_handle();
854     }
855 }
```

详细看下 ble_advertising_start 函数内部设置, 在给定设置模式后, 会根据你配置模式来启动相应的广播, 如下图所示:

```

319 | switch (p_advertising->adv_mode_current)
320 | {
321 |     case BLE_ADV_MODE_DIRECTED_HIGH_DUTY:
322 |         ret = set_adv_mode_directed_high_duty(p_advertising, &p_advertising->adv_params);
323 |         break;
324 |
325 |     case BLE_ADV_MODE_DIRECTED:
326 |         ret = set_adv_mode_directed(p_advertising, &p_advertising->adv_params);
327 |         break;
328 |
329 |     case BLE_ADV_MODE_FAST:
330 |         ret = set_adv_mode_fast(p_advertising, &p_advertising->adv_params);
331 |         break;
332 |
333 |     case BLE_ADV_MODE_SLOW:
334 |         ret = set_adv_mode_slow(p_advertising, &p_advertising->adv_params);
335 |         break;
336 |
337 |     case BLE_ADV_MODE_IDLE:
338 |         p_advertising->adv_evt = BLE_ADV_EVT_IDLE;
339 |         break;
340 |
341 |     default:
342 |         break;
343 | }

```

根据广播类型设置, 启动快速广播

但是, 我们蓝牙设备不可能一直广播下去, 这样很费电, 我们需要动态的切换广播状态, 这个切换的工作怎么做了?

其实广播事件切换就抛给了广播回调函数, 在 BLE_ADVERTISING_H 文件中如下所示 (关于回调函数我们专门写一篇文章讲述):

```

73 | */
74 | #define BLE_ADVERTISING_DEF(_name)
75 | static ble_advertising_t _name;
76 | NRF_SDH_BLE_OBSERVER(_name ## _ble_obs,
77 |                       BLE_ADV_BLE_OBSERVER_PRIO,
78 |                       ble_advertising_on_ble_evt, &_name);
79 | NRF_SDH_SOC_OBSERVER(_name ## _soc_obs,
80 |                       BLE_ADV_SOC_OBSERVER_PRIO,
81 |                       ble_advertising_on_sys_evt, &_name)
82 |

```

这个是回调函数里必须有的部分, 其中 ble_advertising_on_ble_evt(p_ble_evt) 广播蓝牙回调时间函数是主要作用, 深入到这个函数内部详细看下, 这里有一个广播超时处理。首先蓝牙协议栈 5.0 开始分成了普通定向广播 (Directed Advertising) 和高速定向广播 (HIGH Duty Cycle Directed Advertising), 派发函数有一个广播超时处理函数, 超时时间就是之前设置的广播 TIMEOUT 的时间:

```

670 | void ble_advertising_on_ble_evt(ble_evt_t const * p_ble_evt, void * p_context)
671 | {
672 |     ble_advertising_t * p_advertising = (ble_advertising_t *)p_context;
673 |
674 |     switch (p_ble_evt->header.evt_id)
675 |     {
676 |         case BLE_GAP_EVT_CONNECTED:
677 |             on_connected(p_advertising, p_ble_evt);
678 |             break;
679 |
680 |         // Upon disconnection, whitelist will be activated and direct advertising is started.
681 |         case BLE_GAP_EVT_DISCONNECTED:
682 |             on_disconnected(p_advertising, p_ble_evt);
683 |             break;
684 |
685 |         // Upon terminated advertising (time-out), the next advertising mode is started.
686 |         case BLE_GAP_EVT_ADV_SET_TERMINATED:
687 |             on_terminated(p_advertising, p_ble_evt);
688 |             break;
689 |
690 |         default:
691 |             break;
692 |     }
693 | }

```

gap 连接事件

gap 断开事件

广播超时事件

广播超时处理函数

比如如果我们程序开始设置为快速广播, 那么在超时后进入下一级慢速模式。如果是慢速广播, 超时就进入无效模式。在函数中我们只设置了快速广播超时时间, 其他几种广播超时时间没有设置,

那么就默认为 0，那么整个状态就变为：快速广播超时的 TIMEOUT 时间内没有连接---》进入慢速广播---》立即进入无效模式：

```
static void on_terminated(ble_advertising_t * const p_advertising, ble_evt_t const * p_ble_evt)
{
    ret_code_t ret;

    if (p_ble_evt->header.evt_id != BLE_GAP_EVT_ADV_SET_TERMINATED)
    {
        // Nothing to do.
        return;
    }

    if (
        p_ble_evt->evt.gap_evt.params.adv_set_terminated.reason == BLE_GAP_EVT_ADV_SET_TERMINATED_REASON_TIMEOUT ||
        p_ble_evt->evt.gap_evt.params.adv_set_terminated.reason == BLE_GAP_EVT_ADV_SET_TERMINATED_REASON_LIMIT_REACHED
    )
    {
        // Start advertising in the next mode.
        ret = ble_advertising_start(p_advertising, adv_mode_next_get(p_advertising->adv_mode_current));
        if ((ret != NRF_SUCCESS) && (p_advertising->error_handler != NULL))
        {
            p_advertising->error_handler(ret);
        }
    }
}
```

广播超时

达到限制的广播事件

开始广播下一个广播模式

ble_advertising_start 开始广播函数每次在超时的时候会调用 adv_mode_next_get 函数，这个函数调用一次会把模式自动加 1，如下代码所示：

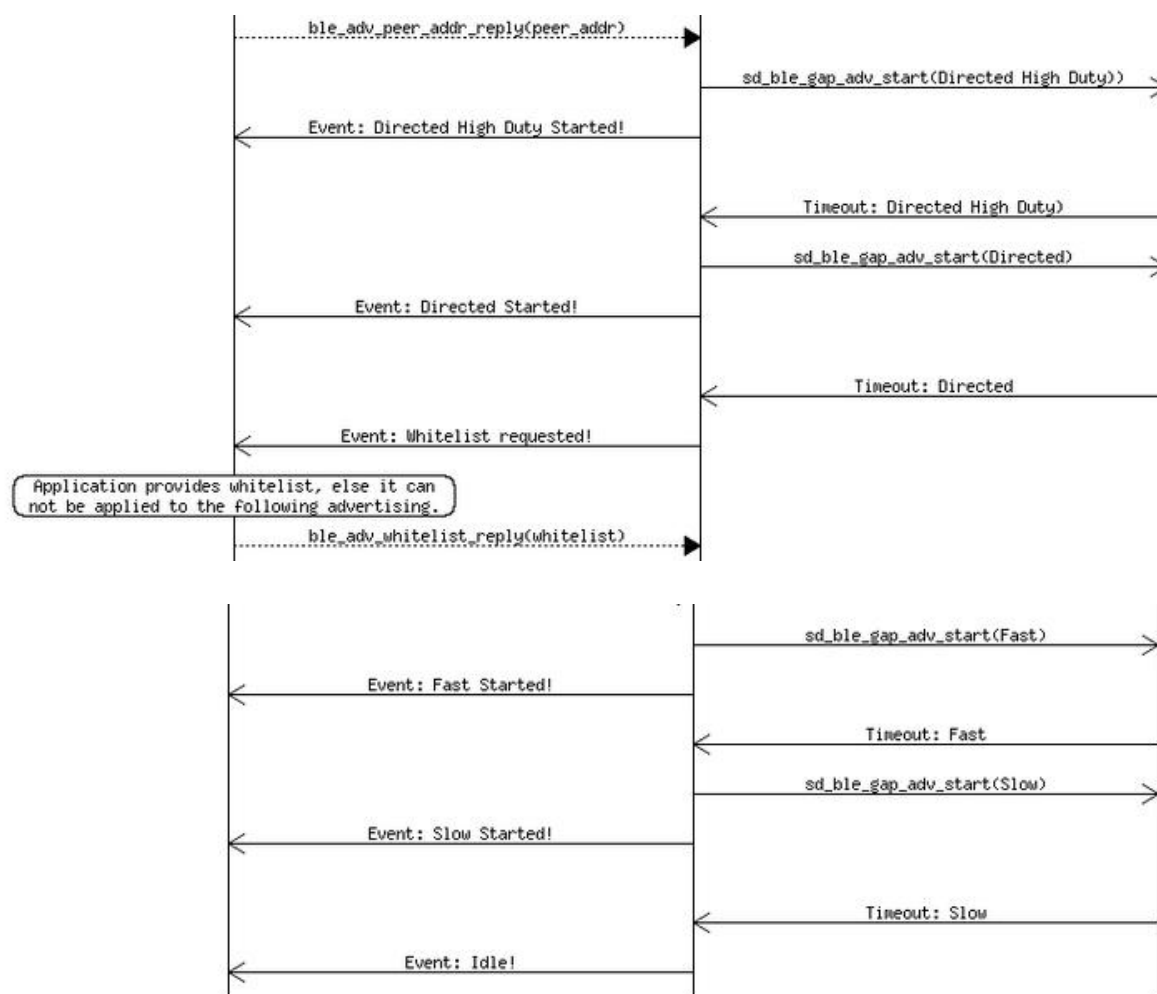
```
85 static ble_adv_mode_t adv_mode_next_get(ble_adv_mode_t adv_mode)
86 {
87     return (ble_adv_mode_t)((adv_mode + 1) % BLE_ADV_MODES);
88 }
89
```

设置广播模式的时候是递进设置的，这四种模式（其实 5 种，定向广播分 2 种）是递进的方式，比如你设置了启动广播时选择 Direct 模式，但是如果你并未在初始化时设置 Direct 模式的相关参数，那么它就会回尝试 Fast 模式，如果初始化时 Fast 模式的相关信息也没设置，就会再尝试 Slow 模式，如果初始化时 Slow 模式相关信息也没设置最后就直接进入到 Idle 模式了。

同样的，广播超时后的超时处理就是选择下一个模式再进行广播，比如你 Fast 模式启动广播成功后，如果超时时间是 3 分钟，3 分钟后，广播超时处理中就是选择尝试 Slow 模式广播。

其实模式的定义只是给出了一个可以直接利用的模块，比如 Fast 模式和 Slow 模式并没有定义所谓的快慢是多少，只是给以一个你可以直接使用的代码模块。比如你的使用场景是希望设备上电后以 30ms 快速广播 20s，如果一直都没有被连接上，30s 后切换成 200ms 的广播 3 分钟以达到减小功耗目的。

整个的切换过程，如下图所示：



其优先级如下：高速定向广播—》普通定向广播—》快速广播—》慢速广播—》无效广播

设置完广播的参数后,通过函数 `err_code = ble_advertising_init(&m_advertising, &init);` //广播参数配置到协议栈中。

最后谈一个问题, 设置广播模式成功后, 设备除了要广播外, 还会执行什么操作? 就是**广播初始化事件派发函数**干的事情了, 在广播初始里设置为:

```
init.evt_handler = on_adv_evt;
```

函数中 `on_adv_evt` 参数为广播处理函数, 设置成功模式后执行对应操作, 蓝牙样例里由于没有慢速广播超时时间, 实际上只需用到了两种状态, 设置的快速广播和无效广播触发的操作:


```

301 L
302 /**@brief Function for handling advertising events.
303 *
304 * @details This function will be called for advertising events which are pa
305 *
306 * @param[in] ble_adv_evt Advertising event.
307 */
308 static void on_adv_evt(ble_adv_evt_t ble_adv_evt) 广播事件处理函数
309 {
310     uint32_t err_code;
311
312     switch (ble_adv_evt)
313     {
314         case BLE_ADV_EVT_FAST: 快速广播模式
315             err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
316             APP_ERROR_CHECK(err_code);
317             break;
318         case BLE_ADV_EVT_IDLE: 无效模式
319             sleep_mode_enter();
320             break;
321         default:
322             break;
323     }
324 }
325
326

```

快速广播开始后, led 灯会闪烁。而无效广播模式会进入休眠模式。所以, 大家会发现广播一段时间后, 广播 LED 灯会停止闪烁, 进入休眠模式函数, 休眠模式函数如下:

```

L */
static void sleep_mode_enter(void)
{
    uint32_t err_code = bsp_indication_set(BSP_INDICATE_IDLE);
    APP_ERROR_CHECK(err_code);

    // Prepare wakeup buttons.
    err_code = bsp_btn_ble_sleep_mode_prepare();
    APP_ERROR_CHECK(err_code);

    // Go to system-off mode (this function will not return; wakeup will cause a reset).
    err_code = sd_power_system_off();
    APP_ERROR_CHECK(err_code);
}

```

设置了唤醒按键后, 系统进入休眠。总结下: 广播初始化中设置了 FAST 模式广播的相关参数, 然后按 FAST 模式启动广播。当广播超时时, 超时时间处理中判断是 FAST 模式超时, 于是再启动 SLOW 模式广播, 但是因为 SLOW 模式广播的相关参数并没有设置, 于是切换成 IDLE 模式, 并且调用了初始化时设置的回调函数。回函数中会设置唤醒按键然后设置深度睡眠。

那么广播初始化基本就讲到这里了。原理清楚了, 那么大家就可以自己进行相关修改。下面验证下, 我们不进入休眠, 一直进行快速广播, 如下修改了?

2: 不进入 IDLE 无效模式

通过上面的分析, 广播模式的切换主要就是广播超时处理, 如果无这个超时时间, 那么就不会切换到下一种模式, 所以, 如何你需要保持一种广播模式不变, 你可以注销这个超时时间, 设置如下:


```
static void advertising_init(void)
{
    ret_code_t          err_code;
    ble_advertising_init_t init;

    memset(&init, 0, sizeof(init));

    init.advdata.name_type          = BLE_ADVDATA_FULL_NAME;
    init.advdata.include_appearance = true;
    init.advdata.flags              = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
    init.advdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    init.advdata.uuids_complete.p_uuids  = m_adv_uuids;
    init.config.ble_adv_fast_enabled     = true;
    init.config.ble_adv_fast_interval    = APP_ADV_INTERVAL;
    init.config.ble_adv_fast_timeout     = APP_ADV_DURATION;

    init.evt_handler = on_adv_evt;

    err_code = ble_advertising_init(&m_advertising, &init);
    APP_ERROR_CHECK(err_code);

    ble_advertising_conn_cfg_tag_set(&m_advertising, APP_BLE_CONN_CFG_TAG);
}
```

把上面的 `APP_ADV_DURATION` 设置为 0 就可以了，然后修改后编译通过，提示 OK。

主要这里还需要特别提醒，蓝牙模式要设置为 **LE 普通发现模式**和**不支持 BR/EDR 模式**，因为有的例子里是有限发现模式。

本实验需要使用抓包器，观察广播信号，广播信号一直广播，不切换模式。



3: BLE 广播响应包:

广播包有两种: 广播包 (Advertising Data) 和 响应包 (Scan Response), 其中广播包是每个设备必须广播的, 而响应包是可选的。关于广播包的数据格式在蓝牙 5.0 协议栈核心第 7.8.7 和 7.8.8 有详细描述。如下所示

7.8.7 LE Set Advertising Data Command 1256

7.8.8 LE Set Scan Response Data Command..... 1257

Advertising_Data_Length: *Size: 1 Octet*

Value	Parameter Description
0x00 – 0x1F	The number of significant octets in the Advertising_Data.

Advertising_Data: *Size: 31 Octets*

Value	Parameter Description
	31 octets of advertising data formatted as defined in [Vol 3] Part C, Section 11.
	All octets zero (default).

Scan_Response_Data_Length:**Size: 1 Octet**

Value	Parameter Description
0x00 – 0x1F	The number of significant octets in the Scan_Response_Data.

Scan_Response_Data:**Size: 31 Octets**

Value	Parameter Description
	31 octets of Scan_Response_Data formatted as defined in [Vol 3] Part C, Section 11.
	All octets zero (default).

每个包都是 31 字节，数据包中分为有效数据（significant）和无效数据（non-significant）两部分。

◎有效数据部分：包含若干个广播数据单元，称为 AD Structure。如图中所示，AD Structure 的组成是：第一个字节是长度值 Len，表示接下来的 Len 个字节是数据部分。数据部分的第一个字节表示数据的类型 AD Type，剩下的 Len - 1 个字节是真正的数据 AD data。其中 AD type 非常关键，决定了 AD Data 的数据代表的是什么和怎么解析，这个在后面会《广播包和数据包分析》中详细讲述。

◎无效数据部分：因为广播包的长度必须是 31 个 byte，如果有效数据部分不到 31 自己，剩下的就用 0 补全。这部分的数据是无效的，解释的时候，忽略即可。

这里我们主要讨论下广播响应包（Scan Response）的设置，**广播响应包**是为了给广播一个额外的 31 字节数据，用于主机为**主动扫描**情况下，反馈数据使用。

比如有些私有任务的 128bit UUID, 我们需要反馈给主机的时候，我们可以采用这种下面这种方式配置，如图所示：

```

244 | * @{ */
245 | #define BLE_GAP_ADV_SET_DATA_SIZE_MAX (31) /**< Maximum data length for an advertising set. */
246 | /**@} */
247 |

```

设置广播响应包和广播包的数据长度，最长为 31 字节：

```

111 | static ble_gap_adv_data_t m_adv_data =
112 | {
113 |     .adv_data =
114 |     {
115 |         .p_data = m_enc_advdata,
116 |         .len = BLE_GAP_ADV_SET_DATA_SIZE_MAX
117 |     },
118 |     .scan_rsp_data =
119 |     {
120 |         .p_data = m_enc_scan_response_data,
121 |         .len = BLE_GAP_ADV_SET_DATA_SIZE_MAX
122 |     }
123 | };

```

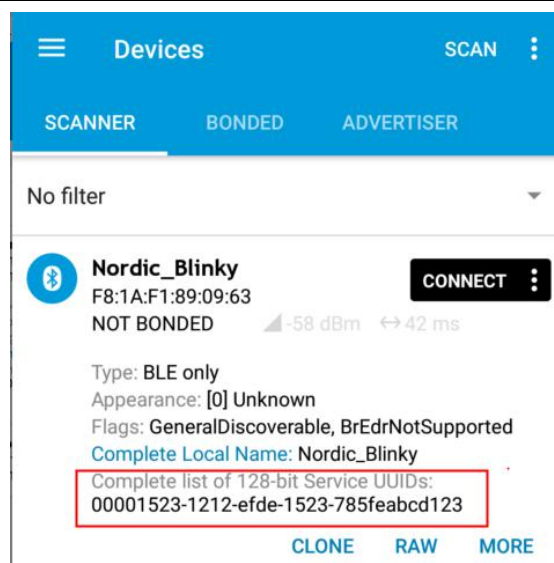
```

209 static void advertising_init(void)
210 {
211     ret_code_t    err_code;
212     ble_advdata_t advdata;
213     ble_advdata_t srdata;
214
215     ble_uuid_t adv_uuids[] = {{LBS_UUID_SERVICE, m_lbs.uuid_type}};
216
217     // Build and set advertising data.
218     memset(&advdata, 0, sizeof(advdata));
219
220     advdata.name_type      = BLE_ADVDATA_FULL_NAME; // 广播名称
221     advdata.include_appearance = true;
222     advdata.flags          = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE; // 广播类型
223
224     memset(&srdata, 0, sizeof(srdata));
225     srdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) / sizeof(adv_uuids[0]);
226     srdata.uuids_complete.p_uuids = adv_uuids; // 广播的UUID
227
228
229
230
231     err_code = ble_advdata_encode(&advdata, m_adv_data.adv_data.p_data, &m_adv_data.adv_data.len);
232     APP_ERROR_CHECK(err_code);
233
234     err_code = ble_advdata_encode(&srdata, m_adv_data.scan_rsp_data.p_data, &m_adv_data.scan_rsp_data.len);
235     APP_ERROR_CHECK(err_code);
236
237     ble_gap_adv_params_t adv_params;
238
239     // Set advertising parameters. 设置广播参数
240     memset(&adv_params, 0, sizeof(adv_params));
241
242     adv_params.primary_phy = BLE_GAP_PHY_1MBPS;
243     adv_params.duration   = APP_ADV_DURATION;
244     adv_params.properties.type = BLE_GAP_ADV_TYPE_CONNECTABLE_SCANNABLE_UNDIRECTED;
245     adv_params.p_peer_addr  = NULL;
246     adv_params.filter_policy = BLE_GAP_ADV_FP_ANY;
247     adv_params.interval     = APP_ADV_INTERVAL;
248
249     err_code = sd_ble_gap_adv_set_configure(&m_adv_handle, &m_adv_data, &adv_params);
250     APP_ERROR_CHECK(err_code);
251 }

```

程序下载后, 使用 nrf connect app 进行扫描, 这个 APP 不点击连接, 可以点击广播信息, 实现主动扫描方式 (用开发板主动扫描方式在主机教程里讲)





用抓捕器抓取如下图所示, 会出现扫描请求包, 请求后出现一个扫描相应包:

P.nbr.	Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header	AdvA	AdvData	CRC	RSSI (dBm)	FCS
408	+41248 =16533986	0x25	0x8E89BED6	ADV_IND	Type TxAdd RxAdd PDU-Length 0 1 0 28	0xF81AF1890963	03 19 00 00 02 01 06 0E 09 4E 6F 72 64 69 63 5F 42 6C 69 6E 6B 79	0x34C587	-37	OK
P.nbr.	Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header	ScanA	AdvA	CRC	RSSI (dBm)	FCS
409	+454 =16534440	0x25	0x8E89BED6	ADV_SCAN_REQ	Type TxAdd RxAdd PDU-Length 3 1 1 12	0x7D7F91BBA71B	0xF81AF1890963	0x286733	-46	OK
P.nbr.	Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header	AdvA	ScanRspData	CRC	RSSI (dBm)	FCS
410	+326 =16534766	0x25	0x8E89BED6	ADV_SCAN_RSP	Type TxAdd RxAdd PDU-Length 4 1 0 24	0xF81AF1890963	11 07 23 D1 BC EA 5F 78 23 15 DE EF 12 12 23 15 00 00	0x4B5EC5	-37	OK
P.nbr.	Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header	AdvA	AdvData	CRC	RSSI (dBm)	FCS
411	+44499 =16579265	0x25	0x8E89BED6	ADV_IND	Type TxAdd RxAdd PDU-Length 0 1 0 28	0xF81AF1890963	03 19 00 00 02 01 06 0E 09 4E 6F 72 64 69 63 5F 42 6C 69 6E 6B 79	0x34C587	-37	OK

回应包 ScanRspData 里 11 和 07 分别表述空白字段和 128bit UUID

0x11	
0x01	Length of this data
0x05	Complete list of 32-bit Service UUIDs
0x01	Length of this data
0x07	Complete list of 128-bit Service UUIDs
0x00	End of Data (Not transmitted over the air)

后面的内容就是该 UUID 的数值了。这样就为了广播一个额外的 31 字节数据来扩展广播内容。