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

Copyright© 2018-2026 青风电子社区 ALL rights reserved

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-nRF51822 开发板

2.3 蓝牙 BLE 之 KEY 按键通知

2.3.1 原理分析与讲解：

本节实验将在 BLE LED 应用示例基础上进行，遵循循序渐进的方式，深入和大家对比分析如何修改和添加自己的蓝牙应用。在前一篇建立私有服务 LED 灯的基础的基础上，我们在之前的基础上添加一个特征值属性，这个特征值属性为通知属性，也就是通过从机的按键按下后，通知主机。

这个应用通过一个服务"GATT 层"服务被建立，那么这个子服务应该包括 1 个特性：KEY 的通知特性。加上之前建立 LED 读写任务就实现多个子任务就设置多个特性。

2.3.2 工程项目的建立：

本例工程对比服务篇第一集内容，对比两份文件，几乎没有做修改，只是在 ble_lbs.c 文件中增加了一个私有服务的子服务特性。在这个基础上我们添加按键的私有服务

2.3.2.1 私有服务的实现

关于私有服务的建立，大家可以回去详细阅读上一章内容。本章着重讲解下按键通知服务的实现。

2.3.1 API 设计

我们在样例基础上添加的驱动 ble_lbs.h 头文件实现了各种数据结构、应用需要实现的

Copyright© 2018-2026 青风电子社区 ALL rights reserved

添加一个函数声明 ble_lbs_on_button_change 是用于处理按键变化，并进行反馈，后面再来详细说明：

```
uint32_t ble_lbs_on_button_change(ble_lbs_t * p_lbs, uint8_t
button_state);
```

2.3.2 实现数据结构体

同时在 ble_lbs.h 头文件用到的数据结构 ble_lbs_t 中，添加按键操作才 handler，我们数据结构体如下：

```
typedef struct ble_lbs_s
{
uint16_t                    service_handle;
ble_gatts_char_handles_t    led_char_handles;
ble_gatts_char_handles_t    button_char_handles;
uint8_t                     uuid_type;
uint16_t                    conn_handle;
ble_lbs_led_write_handler_t    led_write_handler;
} ble_lbs_t;
```

重点就是在 ble_lbs_t 结构体中添加 button_char_handles 按键特征值操作句柄，按键特征值句柄属于结构体 ble_gatts_char_handles_t 类型，如下所示：

```
typedef struct
{
  uint16_t            value_handle;
  uint16_t            user_desc_handle;
  uint16_t            cccd_handle;
  uint16_t            sccd_handle;
} ble_gatts_char_handles_t;
```

第一个参数：处理的特征值.

第二个参数：用户描述句柄 ：句柄向用户说明，或 ble_gatt_handle_invalid 不存在。

Copyright© 2018-2026 青风电子社区 ALL rights reserved

第三个参数：客户端的特征描述符配置（CCCD)句柄，或 ble_gatt_handle_invalid 如果不存在。

第四个参数：服务器端的特征描述符配置（SCCD)句柄，或 ble_gatt_handle_invalid 如果不存在。

后面我们具体进入到函数内部看这些结构体特征值如何设置。其中客户端特性配置描述符(Client Characteristic Configuration Descriptor，CCCD)，这个描述符是给任何支持通知或指示功能的特性额外增加的。在 CCCD 中写入“1”使能通知功能，写入“2”使能指示功能，写入“0”同时禁止通知和指示功能。

在 SoftDevice 协议栈中，对任何使能了通知功能或是指示功能的特性，协议栈将自动加入这个类型的描述符。

2.3.3 服务初始化

我们回到主函数，看看在第一章样例基础上哪些函数需要我们修改以及进行添加：

```
int main(void)
{
    leds_init();
    timers_init();
    gpiote_init();
    buttons_init();//外设应用初始化
    ble_stack_init();//协议栈初始化
    gap_params_init();//gap 初始化
    advertising_init();//广播初始化
    services_init();//服务初始化
    conn_params_init();//更新过程初始化

    advertising_start();//广播开始
    // 进去循环，省电模式
        for (;;)
    {
        power_manage();
    }
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
}
```

进入服务初始化函数 services_init()这里需要进行修改，需要自己添加服务，服务声明可以和上一讲的一样，只需要在 ble_lbs_init（）添加按键通知相关的定义和声明。服务函数 ble_lbs_init（）被调用。这个函数主要就是来完成上面定义的结构体 ble_lbs_t 的调用。

```
static void services_init(void)
{
    uint32_t err_code;
    ble_lbs_init_t init;

    init.led_write_handler = led_write_handler;

    err_code = ble_lbs_init(&m_lbs, &init);
    APP_ERROR_CHECK(err_code);
}
```

这个服务函数 ble_lbs_init（）主要添加了 button_char_add(p_lbs, p_lbs_init)按键服务特性声明，如下所示函数代码列出：

```
uint32_t ble_lbs_init(ble_lbs_t * p_lbs, const ble_lbs_init_t *
p_lbs_init)
{
    uint32_t   err_code;
    ble_uuid_t ble_uuid;
    //初始化服务
    p_lbs->conn_handle       = BLE_CONN_HANDLE_INVALID;
    p_lbs->led_write_handler = p_lbs_init->led_write_handler;

    // 添加 UUID 服务
    ble_uuid128_t base_uuid = {LBS_UUID_BASE};
    err_code = sd_ble_uuid_vs_add(&base_uuid, &p_lbs->uuid_type);
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
    if (err_code != NRF_SUCCESS)
    { return err_code;
    }
    ble_uuid.type = p_lbs->uuid_type;
    ble_uuid.uuid = LBS_UUID_SERVICE;

    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
&ble_uuid, &p_lbs->service_handle);
    if (err_code != NRF_SUCCESS)
    {     return err_code;
    }
   //添加按键服务
    err_code = button_char_add(p_lbs, p_lbs_init);
    if (err_code != NRF_SUCCESS)
    {   return err_code;
    }

    err_code = led_char_add(p_lbs, p_lbs_init);
    if (err_code != NRF_SUCCESS)
    {  return err_code;
    }

    return NRF_SUCCESS;
}
```

2.2.3.1 服务 UUID 的添加：

UUID 含义是通用唯一识别码 (Universally Unique Identifier)，这 是一个软件建构的标准。UUID 是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的。通常平台会提供生成的 API。

Copyright© 2018-2026 青风电子社区 ALL rights reserved

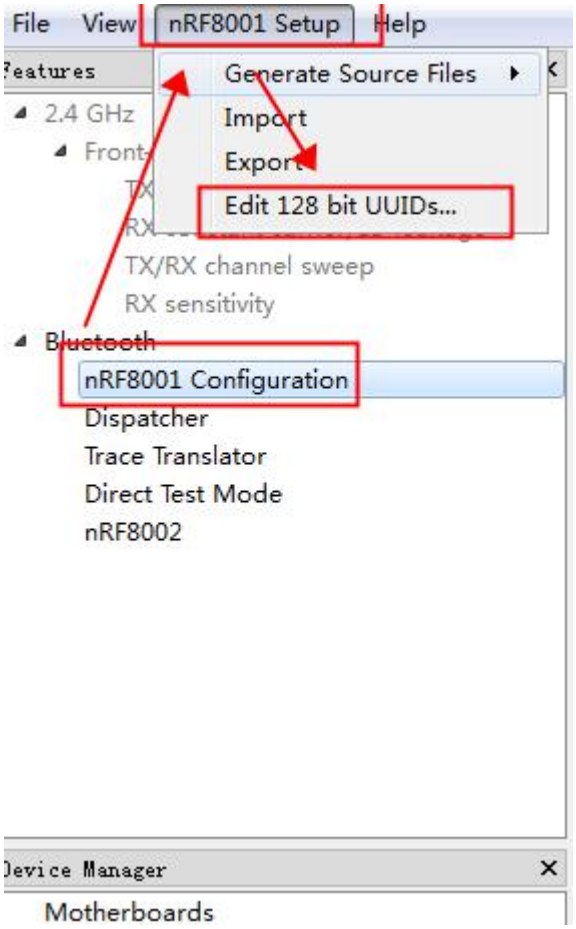
UUID 需要重新设置，因为本服务中将要使用一个定制（私有）的 UUID，以代替蓝牙技术联盟所定义的 UUID。

首先，先定义一个基本 UUID，一种方式是采用 nRFgo Studio 来生成：

1. 打开 nRFgo Studio

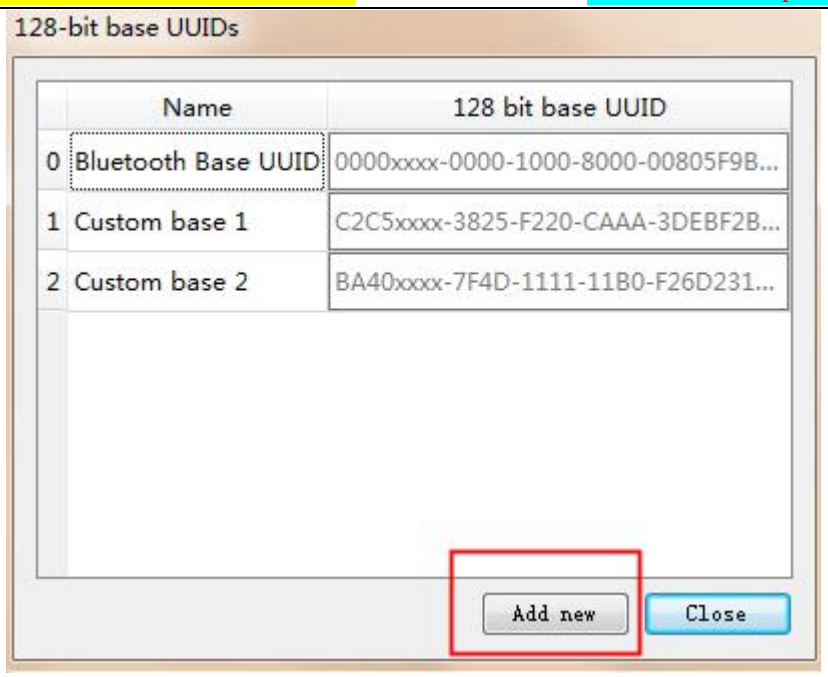
2. 在 nRF8001 Setup 菜单中，选择 Edit 128-bit UUIDs 选项，点击 Add new。

如下图所示：



这就产生了一个随机的 UUID，可以用于你的定制服务中。

Copyright© 2018-2026 青风电子社区 ALL rights reserved



新产生的基本 UUID 必须以数组的形式包含在源代码中，但是只需要在一个地方用到：

为了可读性，在头文件 ble_led.h 中以宏定义的方式添加，连同用于服务和特性的 16 位 UUID 也一起定义：

```
#define LBS_UUID_BASE { 0x23, 0xD1, 0xBC, 0xEA, 0x5F, 0x78,
0x23, 0x15, 0xDE, 0xEF,0x12, 0x12, 0x00, 0x00, 0x00, 0x00 }
#define LBS_UUID_SERVICE        0x1523
#define LBS_UUID_LED_CHAR       0x1525
#define LBS_UUID_BUTTON_CHAR   0x1524
```

添加了按键的 UUID。在服务初始化中：添加基本 UUID 到协议栈列表中，就设置了服务使用这个基本 UUID。在 ble_lbs_init()中只添加一次：

```
ble_uuid128_t base_uuid = LBS_UUID_BASE;
err_code = sd_ble_uuid_vs_add(&base_uuid,
&p_lbs->uuid_type);
if (err_code != NRF_SUCCESS)
{
return err_code;
}
```

以上代码段为加入一个定制的基本 UUID 到协议栈中，并且保存了返回的 UUID 类型。

Copyright© 2018-2026 青风电子社区 ALL rights reserved

当为服务设置 UUID 时，使用这个 UUID 类型，它在 ble_lbs 中 init()中添加服务 GATT 句柄，关于 GATT 原理在前面的章节最前面详细说明了：

```
ble_uuid.type = p_lbs->uuid_type;
ble_uuid.uuid = LBS_UUID_SERVICE;
err_code =
sd_ble_gatts_service_add( BLE_GATTS_SRVC_TYPE_PRIMARY,
&ble_uuid,
&p_lbs->service_handle);
if (err_code != NRF_SUCCESS)
{
return err_code;
}
```

以上代码只是添加了一个空的主服务，所以还必须添加特性按键子服务特性，下面将介绍如何添加特性。

2.2.3.2 按键服务特性的添加：

本服务将要添加 1 个特性，一个是反馈按键的状态。这两个功能需要创建并增加 1 个特性到 ble_led.c 中实现，我们探讨下按键特性设置。

按键特性在有按键状态改变的时候有通知事件，同时也允许对等设备读取这个按键状态。函数代码如下：

```
static uint32_t button_char_add(ble_lbs_t * p_lbs, const ble_lbs_init_t
* p_lbs_init)
{
    ble_gatts_char_md_t char_md;
    ble_gatts_attr_md_t cccd_md;
    ble_gatts_attr_t    attr_char_value;
    ble_uuid_t          ble_uuid;
    ble_gatts_attr_md_t attr_md;

    memset(&cccd_md, 0, sizeof(cccd_md));
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.write_perm);
    cccd_md.vloc = BLE_GATTS_VLOC_STACK;

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

    char_md.char_props.read   = 1;
    char_md.char_props.notify = 1;
    char_md.p_char_user_desc  = NULL;
    char_md.p_char_pf         = NULL;
    char_md.p_user_desc_md    = NULL;
    char_md.p_cccd_md         = &cccd_md;
    char_md.p_sccd_md         = NULL;

    ble_uuid.type = p_lbs->uuid_type;
    ble_uuid.uuid = LBS_UUID_BUTTON_CHAR;
    memset(&attr_md, 0, sizeof(attr_md));
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&attr_md.write_perm
);
    attr_md.vloc        = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth     = 0;
    attr_md.wr_auth     = 0;
    attr_md.vlen        = 0;

    memset(&attr_char_value, 0, sizeof(attr_char_value));
    attr_char_value.p_uuid      = &ble_uuid;
    attr_char_value.p_attr_md   = &attr_md;
    attr_char_value.init_len    = sizeof(uint8_t);
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
    attr_char_value.init_offs     = 0;
    attr_char_value.max_len       = sizeof(uint8_t);
    attr_char_value.p_value       = NULL;

    return sd_ble_gatts_characteristic_add(p_lbs->service_handle,
&char_md,    &attr_char_value,
&p_lbs->button_char_handles);
}
```

初始化中有一个标志设置了 CCCD 的安全模式，它存储在 ble_gap_conn_sec_mode_t 结构体中，这个结构体使用在头文件 ble_gap.h 中定义的宏 BLE_GAP_CONN_SEC_MODE 来设置，根据不同的安全等级定义了不同的宏，你可以根据属性的需要进行选择。

1. 使用宏 BLE_GAP_CONN_SEC_MODE_SET_OPEN 把 CCCD 设置成对任何连接和加密都是可读可写的模式。如果对于按键状态特性我们想让每一个连接都可读但不能写，可以使用 BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS 代替 BLE_GAP_CONN_SEC_MODE_SET_OPEN。

```
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.write_perm);
...
memset(&attr_md, 0, sizeof(attr_md));
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&attr_md.write_perm);
```

设置 UUID 的类型和 UUID 的值：

```
ble_uuid.type = p_lbs->uuid_type;
ble_uuid.uuid = LBS_UUID_BUTTON_CHAR;
```

初始化值不重要，你可以把 p_initial_value 设置为 NULL。确保把返回的特性句柄 button_char_handles 保存在正确的地方，最后调用的函数如下：

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
return ble_gatts_characteristic_add( p_lbs->service_handle,
&char_md,
&attr_char_value,
&p_lbs->button_char_handles );
```

2.2.3.3 增加特性

创建了增加特性的函数之后，你可以在服务初始化的末尾调用它们，如下面的例子：

```
// Add characteristics
 err_code = button_char_add(p_lbs, p_lbs_init);
if (err_code != NRF_SUCCESS)
{
return err_code;
}
return   NRF_SUCCESS;
```

因为任何错误都会导致函数提前退出，因此当你到达函数的末尾时可以认为初始化成功了。

2.2.4 处理协议栈事件

当协议栈需要通知应用程序一些有关它的事情的时候，协议栈事件就发生了，例如当写入特性或是描述符时。对应于本应用，为了让通知功能更好地工作，你需要保存连接句柄，通过这个句柄，你可以在连接事件和断开事件中实现某些操作。

作为 API 的一部分，你可以定义一个函数 ble_lbs_on_ble_evt 用来处理协议栈事件，可以使用简单的 switch-case 语句通过返回事件头部的 id 号来区分不同的事件，并进行不同的处理。

```
void ble_lbs_on_ble_evt(ble_evt_t const * p_ble_evt, void *
p_context)
{
    ble_lbs_t * p_lbs = (ble_lbs_t *)p_context;

    switch (p_ble_evt->header.evt_id)
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
    {
        case BLE_GATTS_EVT_WRITE:
            on_write(p_lbs, p_ble_evt);
            break;

        default:
            // No implementation needed.
            break;
    }
}
```

2.2.5 数据上传函数

你已经添加了一个回调 API 函数让服务知道按键何时被按下，但还没有全部实现，因此你需要从头文件中通过复制来添加，在处理按键按下的时候，你需要给对等设备发送一个通知以告知它新的按键状态。协议栈 SoftDevice API 函数 sd_ble_gatts_hvx 来完成这个事情，它需要连接句柄和结构体 ble_gatts_hvx_params_t 作为输入参数，它管理一个值被通知的整个过程。

在结构体 ble_gatts_hvx_params_t 中，你需要设置为通知模式还是指示模式，用哪一个属性的句柄用来进行通知（本例中使用值的句柄），新的值以及值的长度。方法如下：

```
uint32_t ble_lbs_on_button_change(ble_lbs_t * p_lbs, uint8_t
button_state)
{
ble_gatts_hvx_params_t params;
uint16_t len = sizeof(button_state);
memset(&params, 0, sizeof(params));
params.type = BLE_GATT_HVX_NOTIFICATION;
params.handle = p_lbs->button_char_handles.value_handle;
params.p_data = &button_state;
params.p_len = &len;
return  sd_ble_gatts_hvx(p_lbs->conn_handle, &params);
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
}
```

params.type 为空中属性类型是通知还是指示

params.handle 为本次操作的句柄

params.p_len 为数据长度，注意这个长度需要和之前特征值属性的长度值对应，上传主机的数据不仅仅这一个地方决定。

按照本服务的实现方法，本例子可以应用在其他应用中。

2.3 应用层实现

2.3.1 使用调度

SDK 提供了调度模块，这个模块提供了把事件处理和中断处理转移到 main 函数中进行的机制，它保证了所有的中断处理都能快速被执行。本次调度只有一个 CCCD 使能需要处理。

在 main 函数最开头，添加 BLE_LBS_DEF(m_lbs)回调处理函数，该函数内部调用了协议栈处理函数 ble_lbs_on_ble_evt。该函数前面已经谈过，只有一个 **on_write 写操作。也就是说在从机发通知给主机之前，主机需要向从机写一个 CCCD 使能。手机 APP 操作的时候如下图所示：**

2.3.2 按键处理

Copyright© 2018-2026 青风电子社区 ALL rights reserved

为了完成本应用，你需要定义如何处理按键按下，你可以使用 SDK 提供的 app_button 模块，这个模块将会提供当按键按下和释放时的一个回调函数。

在按键初始化 buttons_init()里，设置你需要的按键，在这个示例中使用了 evaluation kit 开发板上的 button 1。

添加一个新的宏定义，只是为了让代码具有可读性：

```
#define  LEDBUTTON_BUTTON_PIN_NO      BUTTON_1
```

在 buttons_init()里配置按键的引脚，添加按键配置数组如下：

```
static app_button_cfg_t buttons[] =
{
{LEDBUTTON_BUTTON_PIN_NO, false, NRF_GPIO_PIN_PULLUP,
button_event_handler},
};
APP_BUTTON_INIT( buttons, sizeof(buttons) / sizeof(buttons[0]),
BUTTON_DETECTION_DELAY, true);
```

在开发板上的按键是低电平有效，所以第 2 个参数为 false，但它没有外部上拉电阻，因此你需要使用 NRF_GPIO_PIN_PULLUP 来使能内部上拉电阻。完成之后，你就已经完成按键模块的初始化了。

函数 button_event_handler()为按键中断处理函数，当按键按下后，模块将通过 ble_lbs_on_button_chang 上传主机当前的按键状态，因此你可以通过服务 API 函数直接传递这个按键状态：

```
static void button_event_handler(uint8_t pin_no, uint8_t button
action)
{
uint32_t err_code;
switch(pin_no)
{
case LEDBUTTON_BUTTON_PIN_NO:
err_code = ble_lbs_on_button_change(&m_lbs, button_action);
if (err_code != NRF_SUCCESS &&
```

Copyright© 2018-2026 青风电子社区 ALL rights reserved

```
err_code != BLE_ERROR_INVALID_CONN_HANDLE &&
err_code != NRF_ERROR_INVALID_STATE)
{
APP_ERROR_CHECK(err_code);
}
break;
default:
APP_ERROR_HANDLER(pin_no);
break;
}
}
```

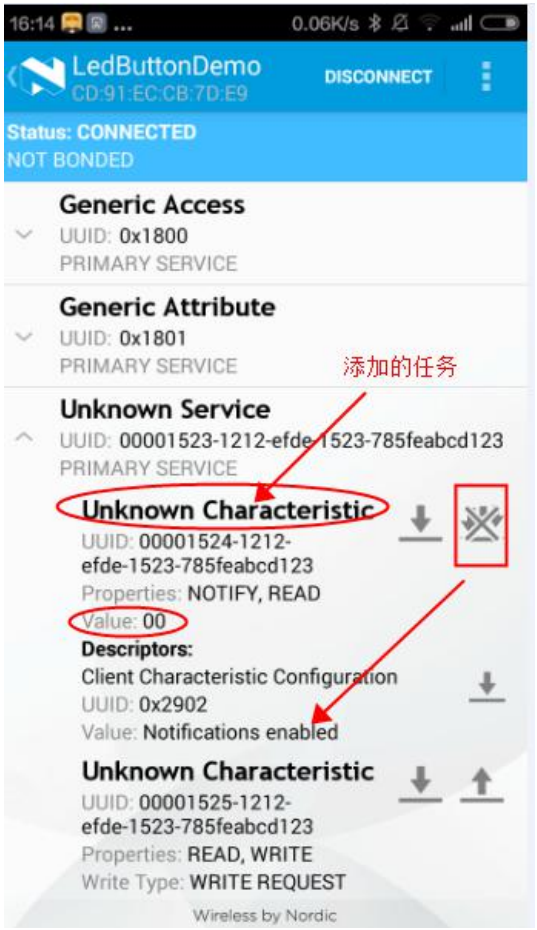
在上面的代码中，我们忽略可能客户端 CCCD 还没有被写入，或者我们没有正确连接所带来的错误。按键状态也最终通过数据上传函数 sd_ble_gatts_hvx 上传给手机（主机）了。

2.5 下载验证：

下载完成应用代码后，按下开发板复位按键运行程序。然后打开 app 如下图所示，发送名字为 Nordic_Blinky 的工程名字，名字可以在程序里随意修改：

Copyright© 2018-2026 青风电子社区 ALL rights reserved

点击连接 connect，连接成功如下图所示：



Copyright© 2018-2026 青风电子社区 ALL rights reserved

显示私有服务，比上一个实验增加了一个，当然大家可以继续增加下去。可以点击展开，查 UUID，按键特性：可读，通知类型。按下按键 button1，通知 Value 值发生变化，如下图所示：

Copyright© 2018-2026 青风电子社区 ALL rights reserved