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
4 蓝牙 BLE 之蓝牙串口 3
4.1 工程项目的建立： 3
4.1.1 主函数的建立 4
4.1.2 外设部分初始化 5
4.1.3 服务初始化 6
4.2 蓝牙串口服务头文件的设计 8
4.2.1 串口服务函数库头文件设计： 8
4.2.2 数据结构体设计 9
4.3 蓝牙串口服务的设计： 11
4.3.1 开通 API 应用服务 12
4.3.2 UUID 设置方法 14
4.3.3 串口接收和发送服务添加 16
4.3.4 蓝牙串口服务特性的配置 17
4.3.5 处理协议栈事件 21
4.3.6 处理串口写事件 22
4.3.6 处理串口接收事件 25
4.4 下载验证 28

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

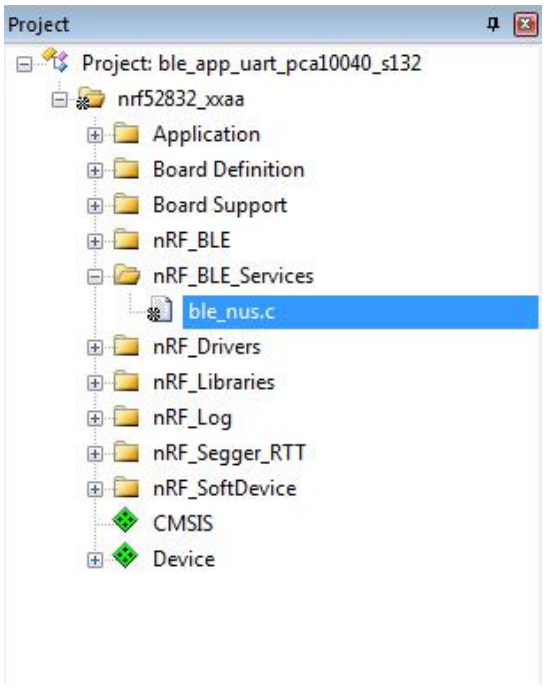
4 蓝牙 BLE 之蓝牙串口

BLE 串口应用示例学习必须在你之前认真解读了前面服务篇的讲详解里的内容为基础后进行。这里特别注意。

本节将结合实例代码，一步一步的通过原理分析和讲解，再次带大家深入到编写蓝牙应用的这个过程中，学习的时候大家一定要多对照理论进行学习，从而深入理解代码，为自己独立编写应用打下基础。

4.1 工程项目的建立：

本例工程对比第一章的工程样例，对比两份工程项目，分析使用了哪些文件，哪些文件未使用，工程目录如下图所示：



对比两个工程数，蓝牙串口需要单独写一个 API 应用文件，开通串口蓝牙服务，也就是工程中的 ble_nus.c 文件。下面的章节重点就是来教大家如何编写这个函数文件。提出两个问题：1：如何开通蓝牙串口服务:2：串口服务 TX 和 RX 数据通道如何搭建的。

4.1.1 主函数的建立

Nrf52832 蓝牙工程的主函数有着一定的通用性，初始化过程类似，需要修改的就是初始化过程中的子函数，先看看本列的主函数流程：

```
int main(void)
{
    uint32_t err_code;
    bool erase_bonds;

    // 初始化
    uart_init();//多出来的串口初始化
    log_init();
    timers_init();
    buttons_leds_init(&erase_bonds);
    power_management_init();
    ble_stack_init();
    gap_params_init();
    gatt_init();
    services_init();
    advertising_init();
    conn_params_init();

    // Start execution.
    printf("\r\nUART started.\r\n");
    NRF_LOG_INFO("Debug logging for UART over RTT started.");
    advertising_start();
```

```
    // Enter main loop.
    for (;;)
    {
        idle_state_handle();
    }
}
```

整个主函数蓝牙框架流程在前面蓝牙程序搭建篇里已经对这个过程进行了讲述，下面我们还是来看看本例有哪些不同，主函数红色标注了多次的部分，多出了串口初始化和串口开始提示，下面看看其他服务函数不同的内容。

4.1.2 外设部分初始化

外设的初始化不涉及蓝牙协议，仅仅是对外设功能的初始化，在前面外设篇我们已经把外设功能和大家学过了一篇，这里大家应该能够熟练的调用和编写相关函数。本例的重点就是初始化串口功能,函数设置如下：

```
static void uart_init(void)
{
    uint32_t                     err_code;
    const app_uart_comm_params_t comm_params =
    {
        .rx_pin_no    = RX_PIN_NUMBER,
        .tx_pin_no    = TX_PIN_NUMBER,
        .rts_pin_no   = RTS_PIN_NUMBER,
        .cts_pin_no   = CTS_PIN_NUMBER,//配置串口管脚
        .flow_control = APP_UART_FLOW_CONTROL_DISABLED,//是否使能流量控制
        .use_parity   = false,//设置串口中断优先级
      #if defined (UART_PRESENT)//是否为异步收发
        .baud_rate    = NRF_UART_BAUDRATE_115200
#else
        .baud_rate    = NRF_UARTE_BAUDRATE_115200//定义串口波特率
```

```
#endif
    };//配置串口参数


    APP_UART_FIFO_INIT( &comm_params,
                        UART_RX_BUF_SIZE,
                        UART_TX_BUF_SIZE,
                        uart_event_handle,
                        APP_IRQ_PRIORITY_LOW,
                        err_code);//带入前面配置的参数，配置串口缓冲，并且初始化串口
    APP_ERROR_CHECK(err_code);
}
```

串口设置里，首先配置串口端口、波特率、流量控制、优先级等串口参数，再使用函数 APP_UART_FIFO_INIT **配置串口缓冲并且初始化串口**。

在函数 APP_UART_FIFO_INIT 中调用了 app_uart_init 初始化函数，把我们之前 comm_params 配置的串口参数传导到设置中，函数具体如下，有 4 个形参，分别为：

1：串口通用参数，2：串口缓冲，3：串口中断事件，4：串口中断优先级。函数如下所示：

```
uint32_t app_uart_init(const app_uart_comm_params_t * p_comm_params,
                       app_uart_buffers_t *           p_buffers,
                       app_uart_event_handler_t       error_handler,
                       app_irq_priority_t             irq_priority);
```

4.1.3 服务初始化

首先是进入服务初始化函数 services_init()这里需要进行修改，在《蓝牙样本工程详解》有说明，当时蓝牙样本例子给出的是一个空的函数，需要自己添加服务。而这个添加的服务也就是我们需要开通的服务。本例里应该是串口蓝牙通信服务。

```
static void services_init(void)
{
    uint32_t           err_code;
    ble_nus_init_t     nus_init;
    nrf_ble_qwr_init_t qwr_init = {0};

     // 初始化写队列空间
     qwr_init.error_handler = nrf_qwr_error_handler;
     err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

    //我们添加的服务
    nus_init.data_handler = nus_data_handler;//蓝牙处理事件
    err_code = ble_nus_init(&m_nus, &nus_init);//添加的蓝牙服务声明
    ****/
    APP_ERROR_CHECK(err_code);
}
```

添加服务的时候，首先来设置在服务初始化结构体和服务结构体中的函数 nus_data_handler 作为串口传输处理句柄，作为回调函数。这个函数用来处理的串口 UART 发送数据和从接收串口端来的数据。没有这个函数，串口数据是无法通过蓝牙串口服务的。这个句柄的书写也是非常简单的，直接调用外设函数，代码如下：

```
void nus_data_handler(ble_nus_t * p_nus, uint8_t * p_data, uint16_t length)
{
    for (int i = 0; i < length; i++)
    {
        simple_uart_put(p_data[i]);
    }
    simple_uart_put('\n');
}
```

然后添加串口蓝牙服务声明 ble_nus_init(&m_nus, &nus_init)。这个函数大家可以自己任意命名。那么有朋友会问，函数内部的形参如何设置？需要怎么编写？这就是 API 需要来实现和完成的工作了。下面将来详细讲解，这个是关键。

4.2 蓝牙串口服务头文件的设计

4.2.1 串口服务函数库头文件设计：

我们在样例基础上添加的驱动 ble_nus .h 头文件实现了各种数据结构、应用需要实现的事件句柄和以下 3 个 API 函数,下面来一一介绍：

函数声明，首先看下面两个函数：

```
函数 1：
uint32_t   ble_nus_init(ble_nus_t * p_nus, const ble_nus_init_t * p_nus_init);
函数 2：
void       ble_nus_on_ble_evt(ble_nus_t * p_nus, ble_evt_t * p_ble_evt);
函数 3：
uint32_t   ble_nus_data_send(ble_nus_t * p_nus,
                             uint8_t   * p_data,
                             uint16_t  * p_length,
                             uint16_t    conn_handle);
```

在串口服务函数文件中实际上建立了 3 个主要的函数：

第一个函数：建立蓝牙串口主服务，再前面服务建立篇里有讨论过，这个串口服务属于私有服务类型。

第二个函数：串口回调函数，在 SKD15 版本开始，Nordic 官方代码省略了专门的回调处理函数，而把服务回调设置在观察处理中，如下图所示，在 ble_nus.h 最开头，设置了当发生了串口蓝牙事件会进行过派发回调，关于派发回调，我们会专门写一篇文章进行描述。

ble_nus.h

```
78 /**@brief   Macro for defining a ble_nus instance.
79  *
80  * @param     _name            Name of the instance.
81  * @param[in] _nus_max_clients Maximum number of NUS clients connected at a
82  * @hideinitializer
83  */
84 #define BLE_NUS_DEF(_name, _nus_max_clients)                \
85     BLE_LINK_CTX_MANAGER_DEF(CONCAT_2(_name, _link_ctx_storage),  \
86                              (_nus_max_clients),            \
87                              sizeof(ble_nus_client_context_t));  \
88     static ble_nus_t _name =                                \
89     {                                                       \
90         .p_link_ctx_storage = &CONCAT_2(_name, _link_ctx_storage) \
91     };                                                      \
92     NRF_SDH_BLE_OBSERVER(_name ## _obs,                     \
93                          BLE_NUS_BLE_OBSERVER_PRIO,         \
94                          ble_nus_on_ble_evt,                \
95                          &_name)
96
```

第三个函数声明 ble_nus_send_string 是用于串口连接状态变化，并进行反馈，后面再来详细说明。

4.2.2 数据结构体设计

ble_nus.h 头文件中有几个头文件需要简单说明下，大家看这上面设置的几个函数里的形参，使用了三个头文件：

一个 ble_nus_s *结构体，一个 ble_nus_evt_t*结构体，一个 ble_nus_init_t *结构体。

◎ ble_nus_evt_t 为常见的事件类型，模块的具体事件的报告：

```
typedef struct
{
    ble_nus_evt_type_t          type;        /* 串口事件类型. */
    ble_nus_t                 * p_nus;       /* 事例指针*/
    uint16_t                    conn_handle; /*连接句柄 */
    ble_nus_client_context_t * p_link_ctx;  /* 上下文链接 */
    union
    {
        ble_nus_evt_rx_data_t rx_data; /*参数 BLE_NUS_EVT_RX_DATA 事件的数据. */
    } params;
} ble_nus_evt_t;
```

◎ ble_nsu_init_t 用于初始化参数，所有的 API 函数使用一个指向服务实例的指针作为第一个输入参数。ble_nus_init_t 专用于服务，此服务不依赖于任何启动或停止。

```
typedef struct
{
    ble_nus_data_handler_t    data_handler;
} ble_nus_init_t;
```

◎ ble_nus_s 用于引用这个服务类型结构，包括 UUID 和一些处理事件，在后面还会用到，定义如下内容：

- UUID 类型
- 服务的句柄
- 串口发送的句柄
- 串口接收的句柄
- 存储链接空间
- 传输数据句柄

服务结构体定义如下：

```
struct ble_nus_s
{
uint8_t       uid_type;
uint16_t      service_handle;
ble_gatts_char_handles_t        tx_handles;
ble_gatts_char_handles_t        rx_handles;
blcm_link_ctx_storage_t * const p_link_ctx_storage;
ble_nus_data_handler_t          data_handler;
};
```

重点就是在 ble_nus_s 结构体中添加 tx_handles 和 rx_handles 特征值操作句柄，这个是串口传输和接收服务需要使用的句柄。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 . */

第三个参数：/**< 客户的特征描述符配置（CCCD)句柄，或 ble_gatt_handle_invalid 如果不存在. */

第四个参数：/**< 服务器的特征描述符配置（SCCD)句柄，或 ble_gatt_handle_invalid 如果不存在. */

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。

注意一个特性至少包含 2 个属性：一个属性用于声明，一个属性用于存放特性的值。所有通过 GATT 服务传输的数据必须映射成一系列的特性，可以把特性中的这些数据看成是一个个捆绑起来的数据，每个特性就是一个自我包容而独立的数据点。

例如，如果几块数据总是一起变化，那么我们可以把它们集中在一个特性里。实际编写会在下面的串口 RX 或者 TX 子服务初始化中，详细讨论串口特征值属性和特征值属性声明。

4.3 蓝牙串口服务的设计：

前面设计完了有函数 ble_nus.h，下面就需要考虑如何来写头文件里设计的几个函数了。下面来一一分析：

确定在 API 主函数里要实现几个功能：

1.开通 API 应用服务；

2.服务特征值的添加；

3.协议栈事件的处理；

4.应用层的实现；

配置实现上面几个功能，用户的应用功能就可以通过蓝牙 BLE 进行实现。

4.3.1 开通 API 应用服务

那么这里我们需要添加自己对于的服务，本例为串口蓝牙服务，见下面函数代码：

```
static void services_init(void)
{
    uint32_t         err_code;
    ble_nus_init_t   nus_init;
    memset(&nus_init, 0, sizeof(nus_init));
////添加的服务：
    nus_init.data_handler = nus_data_handler;
    err_code = ble_nus_init(&m_nus, &nus_init);
    /////////
    APP_ERROR_CHECK(err_code);
}
```

首先设置： 在服务初始化结构体和服务结构体中，nus_data_handler 作为串口发送写处理句柄，作为回调函数。如下所示，在主函数内配置好：

```
void nus_data_handler(ble_nus_t * p_nus, uint8_t * p_data, uint16_t
length)
{
    for (int i = 0; i < length; i++)
    {
        simple_uart_put(p_data[i]);
    }
    simple_uart_put('\n');
```

```
}
```

那么这个开通服务函数 ble_nus_init(&m_nus, &nus_init)主要要实现什么功能？在 API 函数文件内实现本函数。下面来详细探讨一下，首先列出函数代码：

```
uint32_t ble_nus_init(ble_nus_t * p_nus, const ble_nus_init_t *
p_nus_init)
{
    uint32_t         err_code;
    ble_uuid_t       ble_uuid;
    ble_uuid128_t    nus_base_uuid = {0x9E, 0xCA, 0xDC, 0x24, 0x0E, 0xE5,
0xA9, 0xE0,0x93, 0xF3, 0xA3, 0xB5, 0x00, 0x00, 0x40, 0x6E};

    if ((p_nus == NULL) || (p_nus_init == NULL))
    {
        return NRF_ERROR_NULL;
    }

   //初始化服务
    p_nus->conn_handle               = BLE_CONN_HANDLE_INVALID;
    p_nus->data_handler              = p_nus_init->data_handler;
    p_nus->is_notification_enabled   = false;

    // 添加 UUID 服务
    err_code = sd_ble_uuid_vs_add(&nus_base_uuid, &p_nus->uuid_type);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    ble_uuid.type = p_nus->uuid_type;
```

```
    ble_uuid.uuid = BLE_UUID_NUS_SERVICE;

    // 添加 GATT 服务
    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY,
                                        &ble_uuid,
                                        &p_nus->service_handle);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    // 添加 RX Characteristic.
    err_code = rx_char_add(p_nus, p_nus_init);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    //添加 TX Characteristic.
    err_code = tx_char_add(p_nus, p_nus_init);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }
    return NRF_SUCCESS;
}
```

下面来详细分析：

4.3.2 UUID 设置方法

在“GATT 层”中规范定义的所有属性都有一个 UUID 值，UUID 是全球唯一的 128 位的号码，它用来识别不同的特性。

4.3.2.1 蓝牙技术联盟 UUID

蓝牙核心规范制定了两种不同的 UUID，一种是基本的 UUID，一种是代替基本 UUID 的 16 位 UUID。

所有的蓝牙技术联盟定义 UUID 共用了一个基本的 UUID：

0x0000xxxx-0000-1000-8000-00805F9B34FB

为了进一步简化基本 UUID，每一个蓝牙技术联盟定义的属性有一个唯一的 16 位 UUID，以代替上面的基本 UUID 的‘x’部分。例如，心率测量特性使用 0X2A37 作为它的 16 位 UUID，因此它完整的 128 位 UUID 为：

0x0000**2A37**-0000-1000-8000-00805F9B34FB

虽然蓝牙技术联盟使用相同的基本 UUID，但是 16 位的 UUID 足够唯一地识别蓝牙技术联盟所定义的各种属性。

蓝牙技术联盟所用的基本 UUID 不能用于任何定制的属性、服务和特性。对于定制的属性，必须使用另外完整的 128 位 UUID。

4.3.2.2 供应商特定的 UUID

SoftDevice 根据蓝牙技术联盟定义 UUID 类似的方式定义 UUID：先增加一个特定的基本 UUID，再定义一个 16 位的 UUID（类似于一个别名），再加载在基本 UUID 之上。这种采用为所有的定制属性定义一个共用的基本 UUID 的方式使得应用变为更加简单，至少在同一服务中更是如此。

使用软件 nRFgo Studio 非常容易产生一个新的基本 UUID

例如，在 BLE 串口 示例中，采用

0x6E400000-B5A3-F393-E0A9-E50E24DCCA9E 作为基本 UUID。

蓝牙核心规范没有任何规则或是建议如何对加入基本 UUID 的 16 位 UUID 进行分配，因此你可以按照你的意图来任意分配。

例如，在 BLE 串口示例中，0x0001 作为服务的 UUID，0x0002 作为串口 TX 特性的 UUID。0x0003 作为串口 RX 特性的 UUID。

为了可读性，在头文件 ble_nus.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 BLE_UUID_NUS_SERVICE                0x0001
/**< The UUID of the Nordic UART Service. */
```

```
#define BLE_UUID_NUS_TX_CHARACTERISTIC  0x0002
/**< The UUID of the TX Characteristic. */

#define BLE_UUID_NUS_RX_CHARACTERISTIC  0x0003
```

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_nus_init()中只添加一次：

```
ble_uuid128_t    nus_base_uuid = {0x9E, 0xCA, 0xDC, 0x24, 0x0E,
0xE5, 0xA9, 0xE0, 0x93, 0xF3, 0xA3, 0xB5, 0x00, 0x00, 0x40, 0x6E};

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。当为服务设置 UUID 时，使用这个 UUID 类型，它在 ble_nus_init()中添加服务 GATT 句柄，：

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

ble_uuid.uuid = BLE_UUID_NUS_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，这里可以不管这个服务，就让他空着，还必须添加串口发送和接收特性到子服务中，下面将介绍如何添加特性。

4.3.3 串口接收和发送服务添加

大部分的空中操作事件都是采用句柄来进行的，因为句柄能够唯一识别各个属性。如何使用特性依据它的性质，特性的性质包括：

- 写
- 没有回应的写
- 读
- 通知：客户端发给请求给服务器，不需要服务器回复一个响应
- 指示：服务器发给指示给客户端，需要客户端发一个确认给服务器

更多的性质在蓝牙规范中有明确的定义，但以上性质更为常用。

在 ble_nus_init()函数中添加服务串口发送和接收特性：

```
// Add RX Characteristic.
err_code = rx_char_add(p_nus, p_nus_init);
if (err_code != NRF_SUCCESS)
{
    return err_code;
}
// Add TX Characteristic.
err_code = tx_char_add(p_nus, p_nus_init);
if (err_code != NRF_SUCCESS)
{
    return err_code;
}
```

具体到两个添加函数，我们来看看需要设置和编写写什么内容？加入蓝牙的哪些服务？

4.3.4 蓝牙串口服务特性的配置

串口蓝牙服务特征值配置需要两个配置函数，分别配置接收和发送特征值，通过分析代码来了解如何编写特征值配置函数，首先看 RX 的代码如下所示：

```
static uint32_t rx_char_add(ble_nus_t * p_nus, const ble_nus_init_t * p_nus_init)
{
    /**@snippet [Adding proprietary characteristic to S110 SoftDevice]在 S110
```

```
协议栈下添加专用特征值 */
    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));
    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));
    ///GATT 特征值参数组
    char_md.char_props.notify = 1;//空中属性设置为通知使能
    char_md.p_char_user_desc  = NULL;//用户描述符为空
    char_md.p_char_pf         = NULL;
    char_md.p_user_desc_md    = NULL;
    char_md.p_cccd_md         = &cccd_md;//cccd 通知使能描述符
    char_md.p_sccd_md         = NULL;
    ble_uuid.type             = p_nus->uuid_type;//子服务 UUID 类型
    ble_uuid.uuid             = BLE_UUID_NUS_RX_CHARACTERISTIC;//串口 RX 子服务
UUID

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

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);//安全属性
    //属性参数组
```

```
    attr_md.vloc                = BLE_GATTS_VLOC_STACK;//GATT 栈空间
    attr_md.rd_auth             = 0;//读授权
    attr_md.wr_auth             = 0;//写授权
   attr_md.vlen                = 1;//可变长度属性

    memset(&attr_char_value, 0, sizeof(attr_char_value));
    //GATT 属性参数组
    attr_char_value.p_uuid     = &ble_uuid;//UUID
    attr_char_value.p_attr_md = &attr_md;//属性参数组带入
    attr_char_value.init_len   = sizeof(uint8_t);//初始化特征值长度
    attr_char_value.init_offs = 0;//偏移量
    attr_char_value.max_len    = BLE_NUS_MAX_RX_CHAR_LEN;//最大的特征值长度

    return sd_ble_gatts_characteristic_add(p_nus->service_handle,
                                           &char_md,
                                           &attr_char_value,
                                           &p_nus->rx_handles);
}
```

特性至少包含 2 个属性：一个属性用于声明，一个属性用于存放特性的值。所有通过 GATT 服务传输的数据必须映射成一系列的特性，可以把特性中的这些数据看成是一个个捆绑起来的数据。

在上面的代码中 RX 的特性添加中：结构体 char_md 属性用于声明

结构体 attr_char_value 用于存放特性的值

对于以上代码展开说明：

4.3.4.1 CCCD 安全设置

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。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。

```
BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
```

4.3.4.2 特征参数组设置

参数组的值需要写入到特征值函数中，有几个参数主需要配置，这个模板可以用于其他任何的用于协议栈的特征值写入服务应用中：

```
///GATT 特征值参数组

//属性参数组

//GATT 属性参数组
```

一个属性包含句柄、UUID、值，句柄是属性在 GATT 表中的索引，在一个设备中每一个属性的句柄都是唯一的。UUID 包含属性表中数据类型的信息，它是理解属性表中的值的每一个字节的意义的关键信息。在一个 GATT 表中可能有许多属性，这些属性能可能有相同的 UUID。

特别注意这里要设置的内容：

```
//GATT 属性参数组
    attr_char_value.p_uuid    = &ble_uuid;
    attr_char_value.p_attr_md = &attr_md;
    attr_char_value.init_len  = sizeof(uint8_t);
    attr_char_value.init_offs = 0;
    attr_char_value.max_len   = BLE_NUS_MAX_RX_CHAR_LEN;
```

串口接收的长度，初始长度为一个字节，最大长度 BLE_NUS_MAX_RX_CHAR_LEN 定义一下。

4.3.4.3 特征添加

特性可以通过 sd_ble_gatts_characteristic_add()函数进行添加，它有 4 个参数。为了代码清晰，这个函数应该只能出现在服务文件中，而不能出现在应用层中。

第 1 个参数是特性要加入的服务的句柄，第 2 个参数是特性的结构体，它是一个全局变量，它包含了特性可能用到的性质（读，写，通知等）。第 3 个参数是值属性的描述，它包含了它的 UUID，长度和初始值。第 4 个参数是返回的特性和描述符的唯一句柄，这个句柄可以在以后用于识别不同的特性。例如，在写事件中用于识别哪一个特性被写入。

```
sd_ble_gatts_characteristic_add(p_nus->service_handle,
```

```
                                        &char_md,
                                        &attr_char_value,
                                        &p_nus->rx_handles);
```

串口 TX 的设置类似，这里就不再展开。代码请看参考代码。

4.3.5 处理协议栈事件

上面几节内容已经描述了如何设置蓝牙服务，当蓝牙服务设置好后，需要搭建整个串口和蓝牙的数据通道。也就是说要实现下面两个数据流向：

主机蓝牙---》从机接收---》串口发往 PC

PC 串口助手---》从机发送---》主机接收

如何实现这个数据通道的搭建了？下面将详细来讨论下。

首先我们要谈下回调处理函数。当底层协议栈需要通知应用程序一些有关它的事情的时候，就会发生对应的协议栈事件，这个事件会上抛给应用，触发应用执行相应的操作。例如当发生一个 GAP 连接事件 BLE_GAP_EVT_CONNECTED，应用层根据设置在这个事件下发生连接操作。

作为 API 的一部分，你可以定义一个函数 ble_nus_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。我们首先关注下写事件：

```
void ble_nus_on_ble_evt(ble_lbs_t * p_lbs, ble_evt_t * p_ble_evt)
{
    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            on_connect(p_lbs, p_ble_evt);
            break;
        case BLE_GAP_EVT_DISCONNECTED:
            on_disconnect(p_lbs, p_ble_evt);
            break;
        case BLE_GATTS_EVT_WRITE:
            on_write(p_lbs, p_ble_evt);
```

```
            break;
        default:
            break;
    }
}
```

4.3.6 处理串口写事件

当特性被写入的时候，你添加到数据结构的函数指针将会通知到应用层，你可以在on_write()函数中实现这样的功能。

当接收到一个写事件时，验证这个写事件是发生在对应的特性上是一个基本的任务，包括验证数据的长度，回调函数是否已设置。如果所有这些都是正确的，则回调函数将会调用，并且把已经写入的值作为输入参数。因此，on_write()函数的内容将会是这样：

```
static void on_write(ble_nus_t * p_nus, ble_evt_t const * p_ble_evt)
{
    ret_code_t                    err_code;
    ble_nus_evt_t                 evt;
    ble_nus_client_context_t    * p_client;
    ble_gatts_evt_write_t const * p_evt_write =
&p_ble_evt->evt.gatts_evt.params.write;

    err_code = blcm_link_ctx_get(p_nus->p_link_ctx_storage,
                                 p_ble_evt->evt.gatts_evt.conn_handle,
                                 (void *) &p_client);
    if (err_code != NRF_SUCCESS)
    {
     NRF_LOG_ERROR("Link context for 0x%02X connection handle could not be
fetched.",p_ble_evt->evt.gatts_evt.conn_handle);
    }
```

```
memset(&evt, 0, sizeof(ble_nus_evt_t));
evt.p_nus       = p_nus;
evt.conn_handle = p_ble_evt->evt.gatts_evt.conn_handle;
evt.p_link_ctx  = p_client;

if ((p_evt_write->handle == p_nus->tx_handles.cccd_handle) &&
    (p_evt_write->len == 2))
{
    if (p_client != NULL)
    {
        if (ble_srv_is_notification_enabled(p_evt_write->data))
        {
            p_client->is_notification_enabled = true;
            evt.type  = BLE_NUS_EVT_COMM_STARTED;
        }
        else
        {
            p_client->is_notification_enabled = false;
            evt.type  = BLE_NUS_EVT_COMM_STOPPED;
        }
       //当是写通知的时候，则根据通知使能写的值，来置为使能。
        if (p_nus->data_handler != NULL)
        {
            p_nus->data_handler(&evt);
        }

    }
}
```

```
    else if ((p_evt_write->handle == p_nus->rx_handles.value_handle) &&
             (p_nus->data_handler != NULL))
    {
        evt.type                  = BLE_NUS_EVT_RX_DATA;
        evt.params.rx_data.p_data = p_evt_write->data;
        evt.params.rx_data.length = p_evt_write->len;

        p_nus->data_handler(&evt);
    }//如果是 RX 数据写的话，这来把 RX 数据写入指针
    else
    {
        // Do Nothing. This event is not relevant for this service.
    }
}
```

这个函数虽然很长，但是实际上就一个作用，判断是 CCCD 通知使能的写入还是 RX 数据的写入。如果是 RX 数据的写入，则把数据值复制至给结构体中的数据 evt.params.rx_data.p_data，实现了接收主机数据的功能。我们接收了主机数据后，观察者可以通过发送给 PC 的调试助手进行观察,这个操作在串口事件处理函数中执行：

```
static void nus_data_handler(ble_nus_evt_t * p_evt)//串口中断操作
{
    if (p_evt->type == BLE_NUS_EVT_RX_DATA)//如果有 RX 数据
    {
        uint32_t err_code;
        for (uint32_t i = 0; i < p_evt->params.rx_data.length; i++)
        {
            do
            {
              err_code =app_uart_put(p_evt->params.rx_data.p_data[i]);
      ..............
```

```
    ..............
    ..............
     }
}
```

串口事件处理函数当判断有 RX 数据事件，则会把刚才我们从主机接收的数据，通过 app_uart_put 函数发送到 PC 的串口调整助手显示。那么这样就搭建起 RX 接收的数据通道：

主机蓝牙---》从机接收---》串口发往 PC

4.3.6 处理串口接收事件

在前面按键通知的学习中，我们知道协议栈 SoftDevice API 通过函数 sd_ble_gatts_hvx 来实现从机数据上传主机的过程，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，在 ble_nus.h 函数中已该函数为基础，搭建了函数 ble_nus_data_send 对从机数据进行上传，代码如下：

```
uint32_t ble_nus_data_send(ble_nus_t * p_nus,
                           uint8_t    * p_data,
                           uint16_t   * p_length,
                           uint16_t     conn_handle)
{
    ret_code_t                 err_code;
    ble_gatts_hvx_params_t     hvx_params;
    ble_nus_client_context_t * p_client;

    VERIFY_PARAM_NOT_NULL(p_nus);

    err_code = blcm_link_ctx_get(p_nus->p_link_ctx_storage,
conn_handle, (void *) &p_client);
    VERIFY_SUCCESS(err_code);

    if ((conn_handle == BLE_CONN_HANDLE_INVALID) || (p_client
```

```
== NULL))
    {
        return NRF_ERROR_NOT_FOUND;
    }

    if (!p_client->is_notification_enabled)
    {
        return NRF_ERROR_INVALID_STATE;
    }

    if (*p_length > BLE_NUS_MAX_DATA_LEN)
    {
        return NRF_ERROR_INVALID_PARAM;
    }

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

    hvx_params.handle = p_nus->tx_handles.value_handle;
    hvx_params.p_data = p_data;//上传数据
    hvx_params.p_len  = p_length;//数据长度
    hvx_params.type   = BLE_GATT_HVX_NOTIFICATION;//通知类型

    return sd_ble_gatts_hvx(conn_handle, &hvx_params);
}
```

函数中 hvx_params 设置我们上传数据的格式和数据指针、类型。数据上传只有通知和指示类型。指示类型需要主机回应，通知类型不需要，这里我们使用通知类型。数据的指针和数据长度作为 ble_nus_data_send 函数形参，在调用的时候确定。

从机上传到主机的数据，需要通过串口助手发送过来，这个数据发送过来后会触发串口接收中断，接收中断中使用 app_uart_get 获取 PC 发过来的数据，再把这个数据通过 **ble_nus_data_send 函数上传主机，因此**如下代码所示：

```
void UART0_IRQHandler(void)
{
    static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
    static uint8_t index = 0;
    uint32_t err_code;
    UNUSED_VARIABLE(app_uart_get(&data_array[index]));
    index++;
    if ((data_array[index - 1] == '\n') || (index >=
(BLE_NUS_MAX_DATA_LEN - 1)))
    {
        uint16_t length = (uint16_t)index;
       err_code = ble_nus_data_send(&m_nus, data_array, &length,
m_conn_handle);
if (err_code != NRF_ERROR_INVALID_STATE)
        {
            APP_ERROR_CHECK(err_code);
        }

        index = 0;
    }

    /**@snippet [Handling the data received over UART] */
}
```

这个步骤完成了对 TX 通道的搭建：

PC 串口助手---》从机发送---》主机接收

因此串口的接收与发送服务都配置完成，而且大家也知道串口接收和发送的数据在哪里调用了吧。

4.4 下载验证

下载方法和前面服务建立篇相同，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，发现串口，点击连接：

连接成功如下图所示：

发送和接收数据，同时请连接青云 NRF51822 开发板的串口到 PC 机，打开 PC 上的串口调试助手，PC 机的串口调试助手设置波特率 115200，调试助手可以和手机进行互连，注意调制助手打开流控：