N32WB031蓝牙芯片主机和从机通信过程应用笔记V1.0

简 介

本文档介绍 N32WB031 系列 32 位 蓝牙芯片（以下简称 N32WB031）主从通信过程说明,包括单主机扫描过程、从机广播过程，以及主从机之间建立连接和进行数据收发过程。本文档目的在于让使用者能够快速熟悉主从机通信过程，以减少开发前期的准备时间，降低开发难度和出现错误操作的概率，最终达到快速开发量产。

主机下载 rdtsc 例程，从机下载 rdtss 例程，解释主机数据收发时对应的是 rdtsc 例程，解释从机数据收发时对应的程序是 rdtss 例程，建议同时打开主从机例程，对照笔记的解释进行数据传输过程分析和理解。

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路109号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationz.com.cn 邮编：518057

目 录

1. 主机扫描建立

蓝牙初始化

```
/**
 * @brief  main function
 * @param
 * @return
 * @note   Note
 */
int main(void)
{
    //for hold the SWD before sleep
    delay_n_10us(200*1000);

    NS_LOG_INIT();

    app_ble_init();

    NS_LOG_INFO(DEMO_STRING);

    // periph init
    LedInit(LED1_PORT,LED1_PIN);  // power led
    LedInit(LED2_PORT,LED2_PIN);  //connection state
    LedOn(LED1_PORT,LED1_PIN);
    app_usart_dma_enable(ENABLE);
    //init text
    usart_tx_dma_send((uint8_t*)DEMO_STRING, sizeof(DEMO_STRING));

    delay_n_10us(500);
    //disable usart for enter sleep
    app_usart_dma_enable(DISABLE);


    while (1)
    {
        /*schedule all pending events*/
        rwip_schedule();
        ns_sleep();

    }
}
```

进行主机扫描初始化，开启蓝牙扫描，主动扫描搜索目标从机蓝牙设备并发送连接请求进行蓝牙连接

```
void app_ble_init(void)
{

    struct ns_stack_cfg_t app_handler = {0};
    app_handler.ble_msg_handler  = app_ble_msg_handler;
    app_handler.user_msg_handler = app_user_msg_handler;
    //initialization ble stack
    ns_ble_stack_init(&app_handler);

    app_ble_gap_params_init();
    app_ble_sec_init();
    app_ble_scan_init();
    app_ble_prf_init();
    //start scan
    ns_ble_start_scan();
}
```

2. 从机广播建立

初始化蓝牙

```
int main(void)
{
    //for hold the SWD before sleep
    delay_n_10us(200*1000);


    NS_LOG_INIT();

    #if  (CFG_APP_NS_IUS)
    if(CURRENT_APP_START_ADDRESS == NS_APP1_START_ADDRESS){
        NS_LOG_INFO("application 1 start new ...\r\n");
    }else if(CURRENT_APP_START_ADDRESS == NS_APP2_START_ADDRESS){
        NS_LOG_INFO("application 2 start new ...\r\n");
    }
    #endif
    app_ble_init();

    NS_LOG_INFO(DEMO_STRING);

    // periph init
    LedInit(LED1_PORT,LED1_PIN);  // power led
    LedInit(LED2_PORT,LED2_PIN);  //connection state
    LedOn(LED1_PORT,LED1_PIN);
    app_usart_dma_enable(ENABLE);
    //init text
    usart_tx_dma_send((uint8_t*)DEMO_STRING, sizeof(DEMO_STRING));

    delay_n_10us(500);
    //disable usart for enter sleep
    app_usart_dma_enable(DISABLE);


    while (1)
    {
        /*schedule all pending events*/
        rwip_schedule();
        ns_sleep();

    }
```

进行从机广播初始化并开始广播，等待主机发送连接请求进行蓝牙连接

```
void app_ble_init(void)
{
    struct ns_stack_cfg_t app_handler = {0};
    app_handler.ble_msg_handler  = app_ble_msg_handler;
    app_handler.user_msg_handler = app_user_msg_handler;
    //initialization ble stack
    ns_ble_stack_init(&app_handler);

    app_ble_gap_params_init();
    app_ble_sec_init();
    app_ble_adv_init();
    app_ble_prf_init();
    //start adv
    ns_ble_adv_start();
}
```

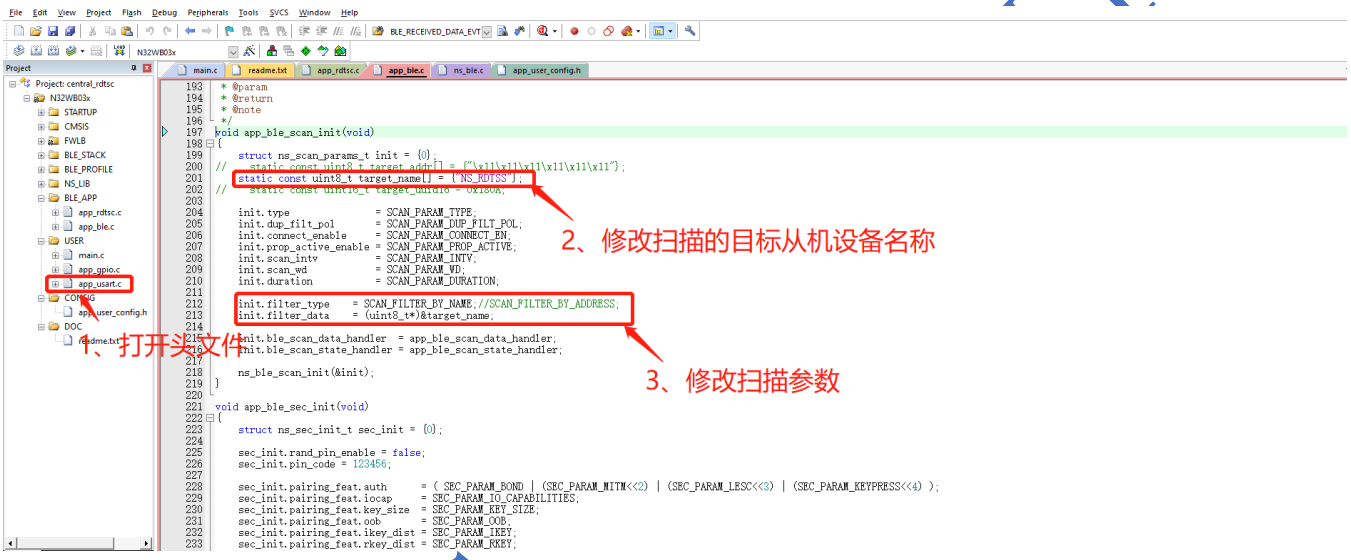
3 .主从机进行连接

主机可以通过扫描收索指定的设备名称或者 MAC 地址的蓝牙从机设备，发送连接请求进行连接。

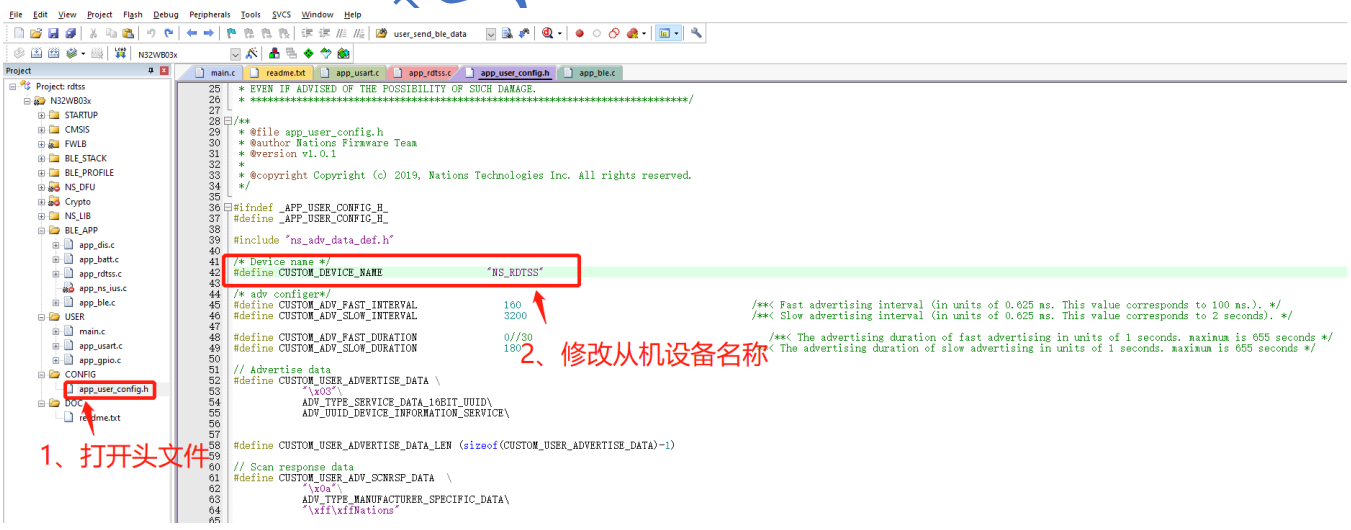
3.1 通过设备名称进行连接

以下代码配置选择设备的名称进行连接，当主机扫描到仪“NS_RDTSS”为设备名的从机时就与之建立连接。主机下载 rdtsc 例程，如下配置目标连接参数，从机下载设备名称为“NS_RDTSS”的 rdtss 例程，当主从机上电后立即建立连接，连接速度很快。

主机扫描参数配置：app_ble_scan_init(void)

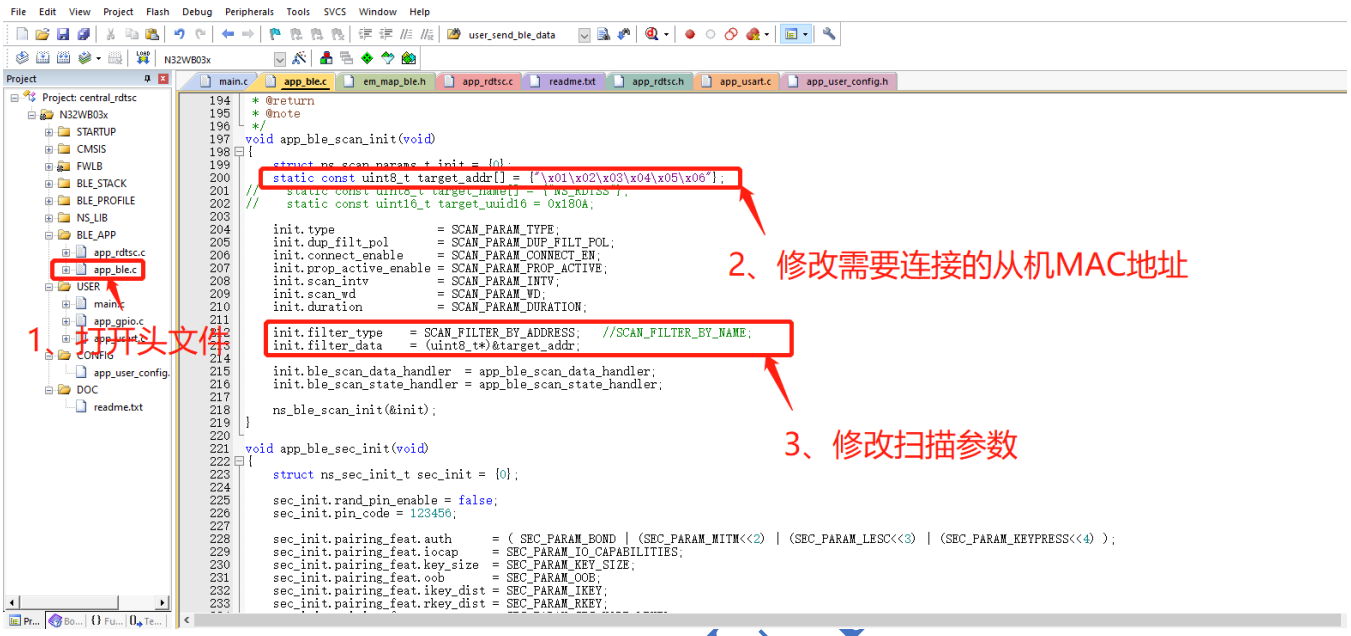


从机设备名称配置：CUSTOM_DEVICE_NAME

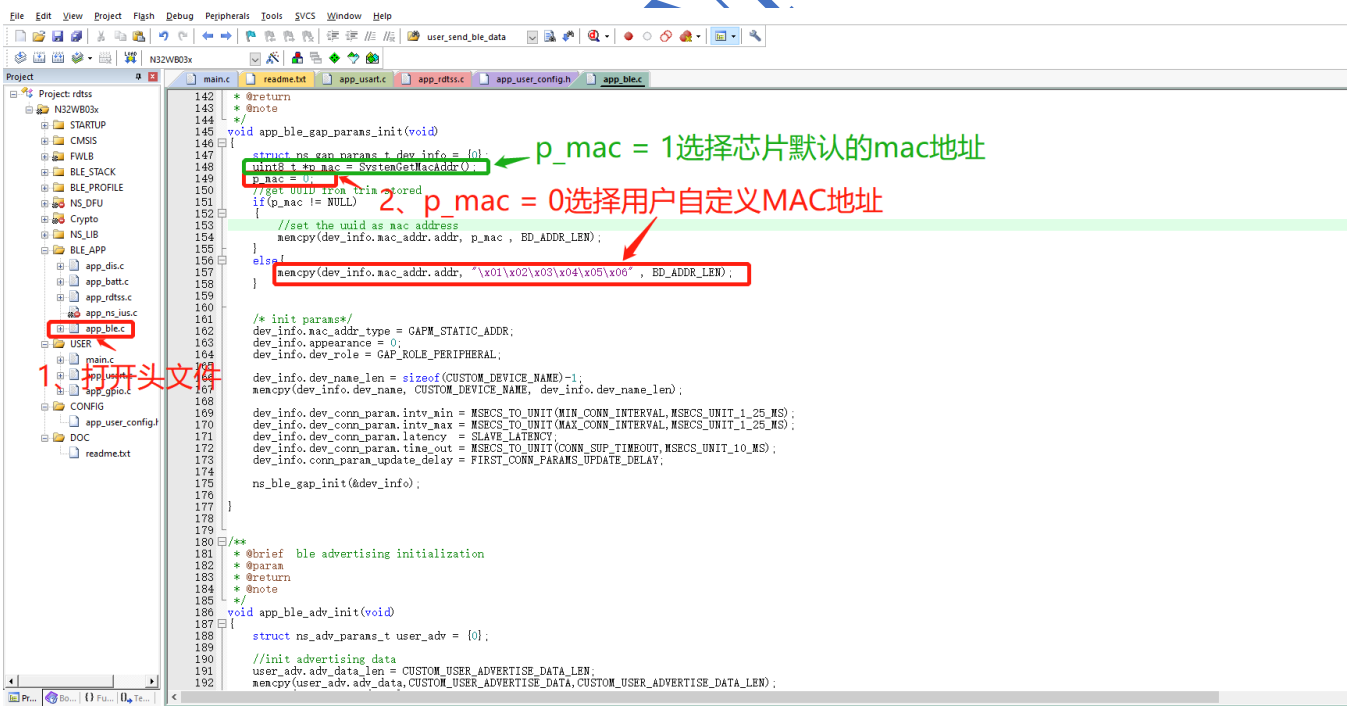


3.2 通过设备 MAC 地址进行连接

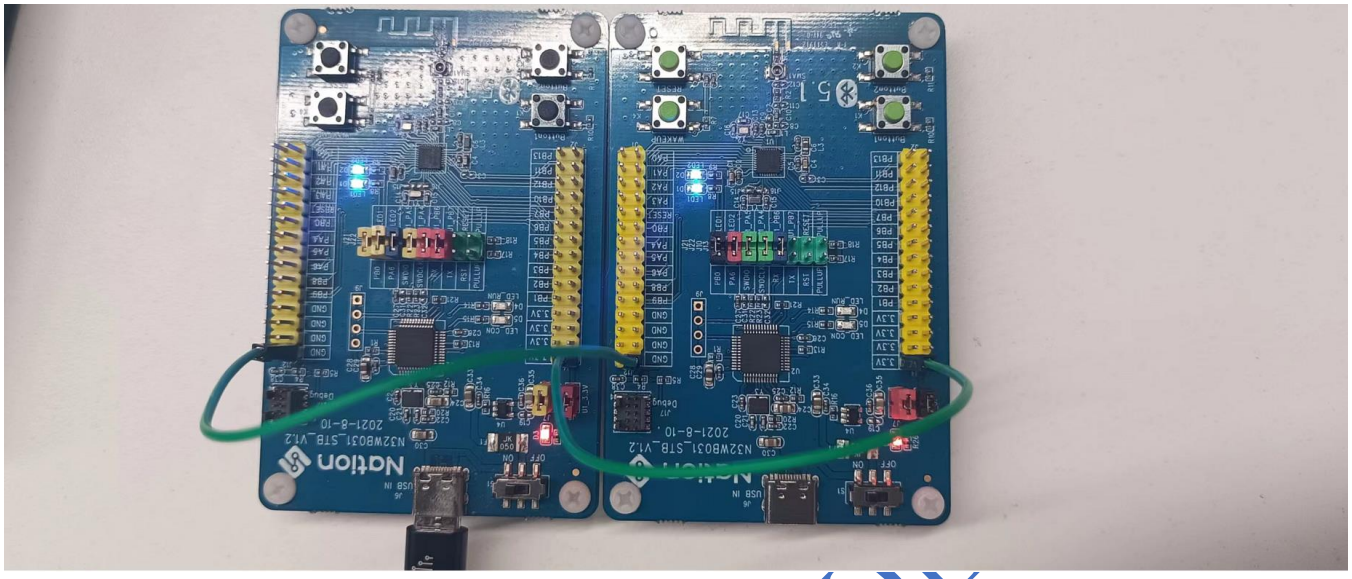
主机参数配置：app_ble_scan_init(void)



从机 MAC 地址配置：app_ble_gap_params_init(void)

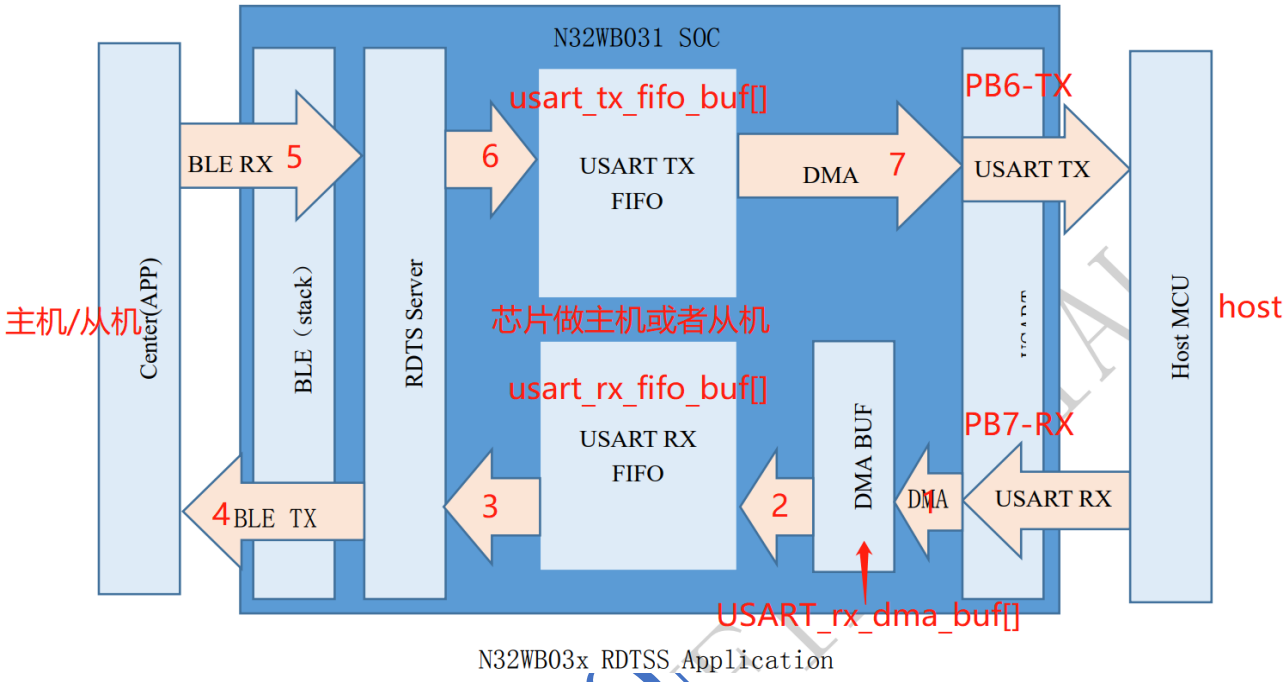


实际连接效果如下：芯片上电之后两颗蓝色的 LED 灯都亮，表示主从机蓝牙已经建立了连接



4. 主从机通信过程解析

数据传输示意图：



N32WB03x RDTSS Application

芯片做主机时，数据传输示意图中各序号调用的函数如下：

1：DmaCh1234_irq_handler() 、usart_irq_handler()

2：usart_rx_check_in_irq()、app_usart_rx_data_fifo_enter()

3：rdtsc_data_tx_cfm_handler()、usart_forward_to_ble_loop()

4：user_send_ble_data()

5：rdtsc_data_rx_ind_handler()

6：app_usart_tx_fifo_enter()

7：app_user_msg_handler()、app_usart_tx_process()

芯片做从机时，数据传输示意图中各序号调用的函数如下：

1：DmaCh1234_irq_handler() 和 usart_irq_handler()

2：usart_rx_check_in_irq()、app_usart_rx_data_fifo_enter()

3：rdtss_val_ntf_cfm_handler()、usart_forward_to_ble_loop()

4：rdtss_send_notify()

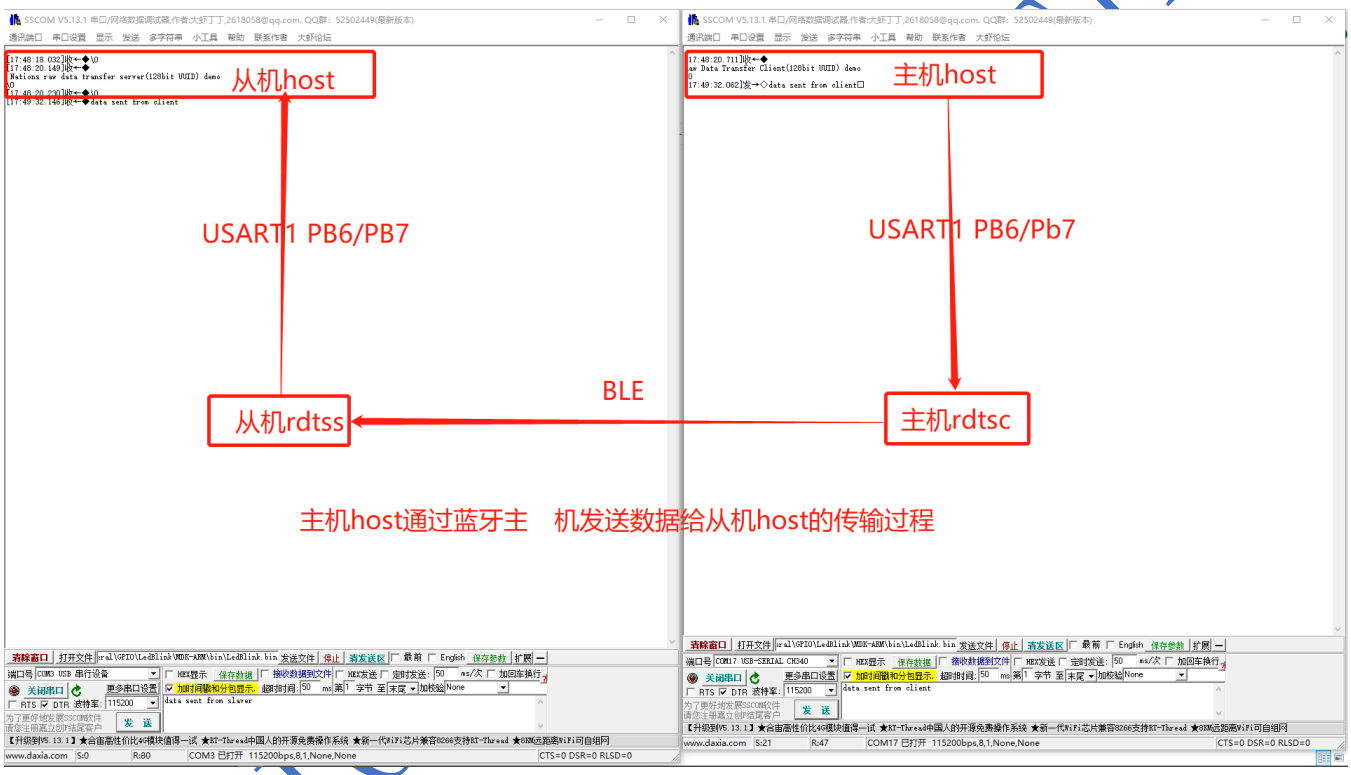
5：rdtss_val_write_ind_handler()

6：app_usart_tx_fifo_enter()

7：app_user_msg_handler()、app_usart_tx_process()

5. 主机 host 通过蓝牙发送数据给从机 host

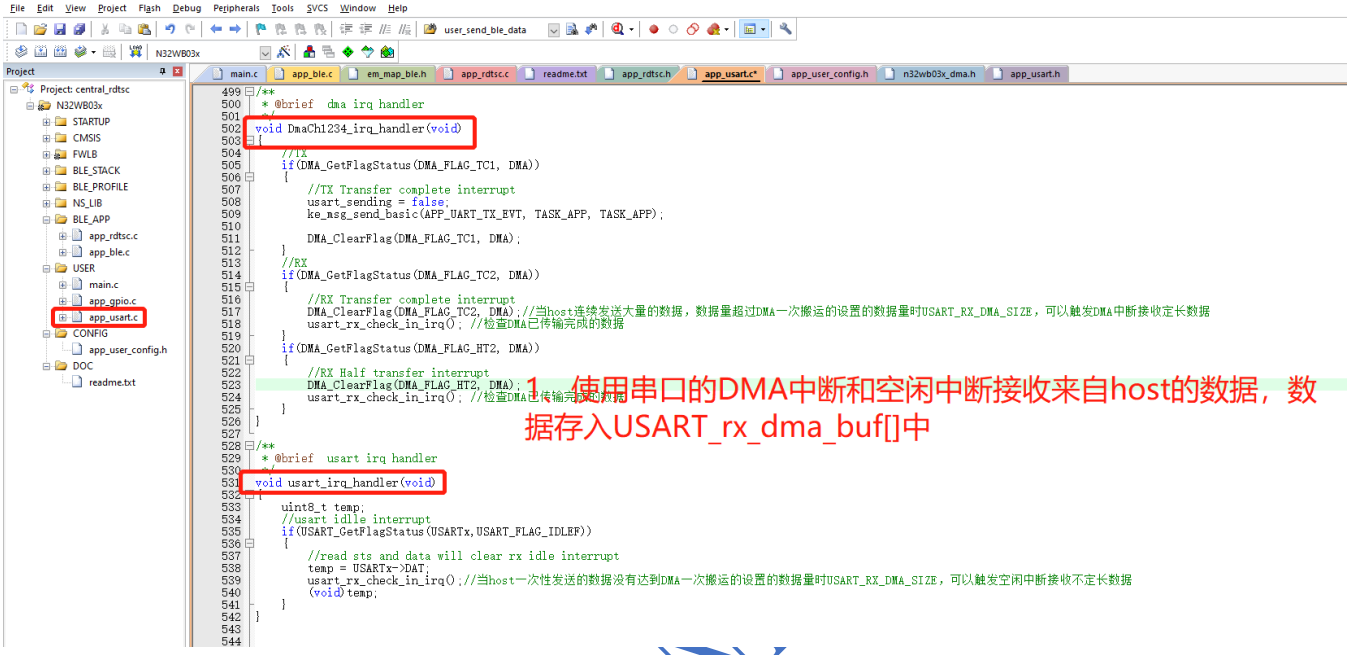
数据传输过程示意图：



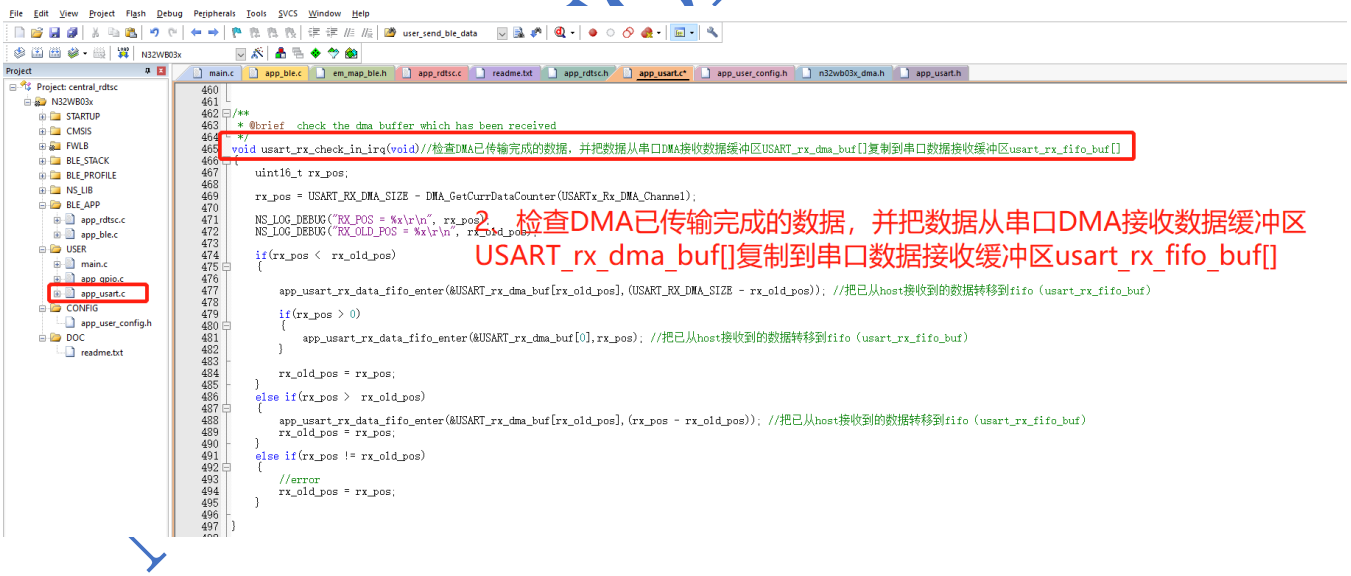
5.1 主机接收来自 host 的数据

主机的主控 MCU 通过串口 1 的 PB6-TX 和 PB7-RX 把数据发送给主机。

主机调用 DmaCh1234_irq_handler() 和 usart_irq_handler()中断函数进行数据接收

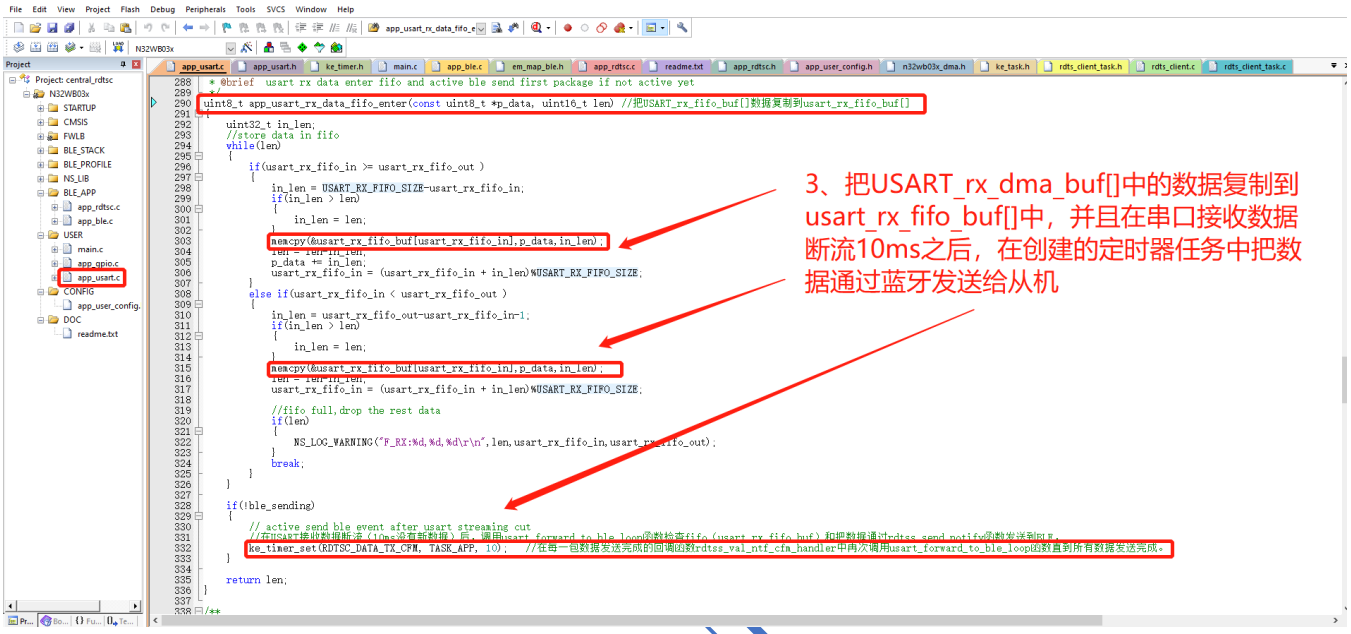


检查处理接收到的数据：usart_rx_check_in_irq()



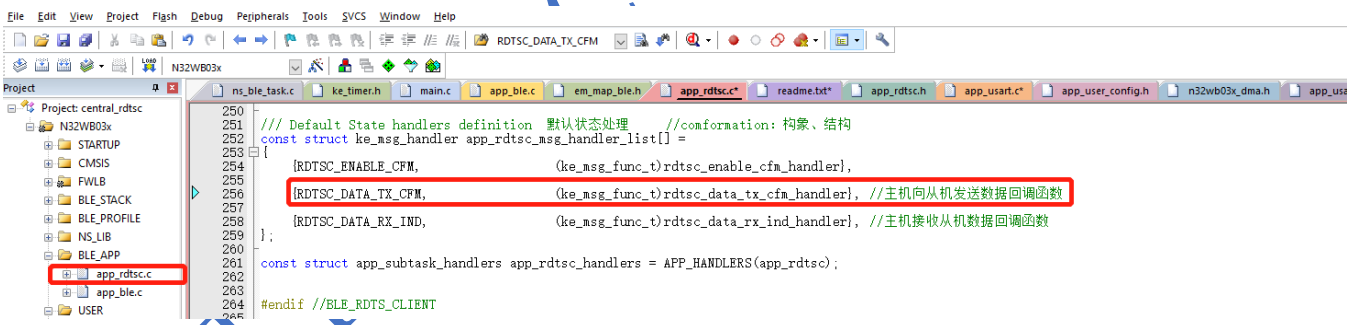
主机在串口接收到数据后，会创建一个定时器任务 ke_timer_set(RDTSC_DATA_TX_CFM, TASK_APP, 10)，在串口接收数据断流 10ms 后，调用定时器任务对应的回调函数把数据通过蓝牙发送到从机。

复制数据到 usart_rx_data_fifo []：app_usart_rx_data_fifo_enter()。



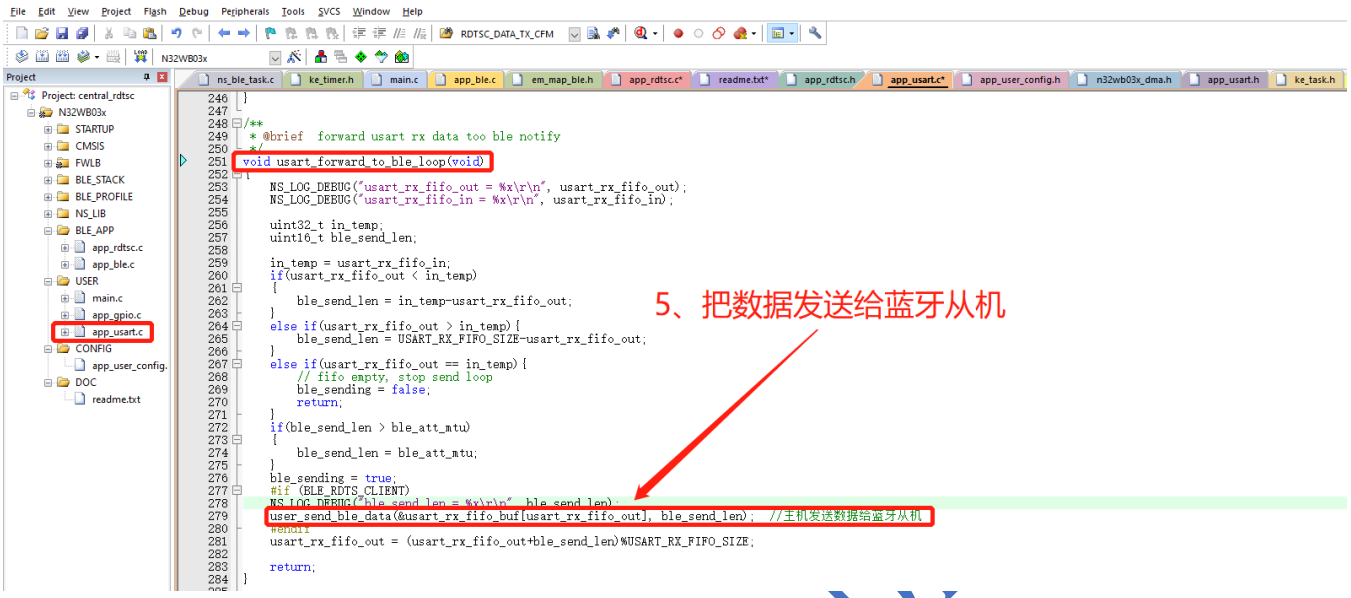
5.2 主机下发数据给从机

在用户信息处理函数中添加数据发送函数的标志：RDTSC_DATA_TX_CFM



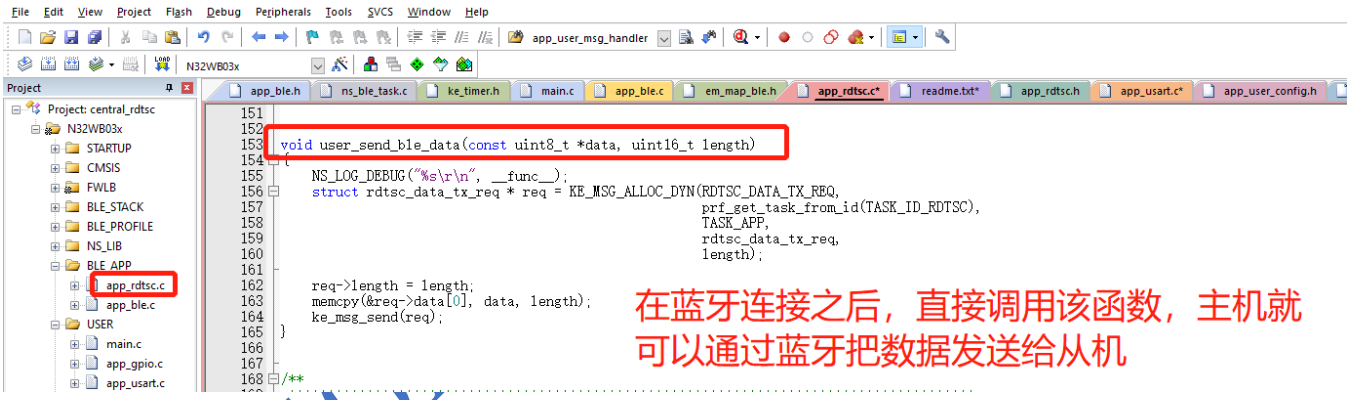
主机向从机发送数据回调函数：rdtsc_data_tx_cfm_handler()

循环调用该函数直到把所有的数据发送给从机：usart_forward_to_ble_loop()




```
246 }
247
248 /**
249  * @brief  forward usart rx data too ble notify
250  */
251 void usart_forward_to_ble_loop(void)
252 {
253     NS_LOG_DEBUG("usart_rx_fifo_out = %x\r\n", usart_rx_fifo_out);
254     NS_LOG_DEBUG("usart_rx_fifo_in = %x\r\n", usart_rx_fifo_in);
255
256     uint32_t in_temp;
257     uint16_t ble_send_len;
258
259     in_temp = usart_rx_fifo_in;
260     if(usart_rx_fifo_out < in_temp)
261     {
262         ble_send_len = in_temp-usart_rx_fifo_out;
263     }
264     else if(usart_rx_fifo_out > in_temp){
265         ble_send_len = USART_RX_FIFO_SIZE-usart_rx_fifo_out;
266     }
267     else if(usart_rx_fifo_out == in_temp){
268         // fifo empty, stop send loop
269         ble_sending = false;
270         return;
271     }
272     if(ble_send_len > ble_att_mtu)
273     {
274         ble_send_len = ble_att_mtu;
275     }
276     ble_sending = true;
277     #if (BLE_RDTS_CLIENT)
278     NS_LOG_DEBUG("ble_send_len = %x\r\n", ble_send_len);
279     user_send_ble_data(&usart_rx_fifo_buf[usart_rx_fifo_out], ble_send_len);  //主机发送数据给蓝牙从机
280     #endif
281     usart_rx_fifo_out = (usart_rx_fifo_out+ble_send_len)%USART_RX_FIFO_SIZE;
282
283     return;
284 }
```

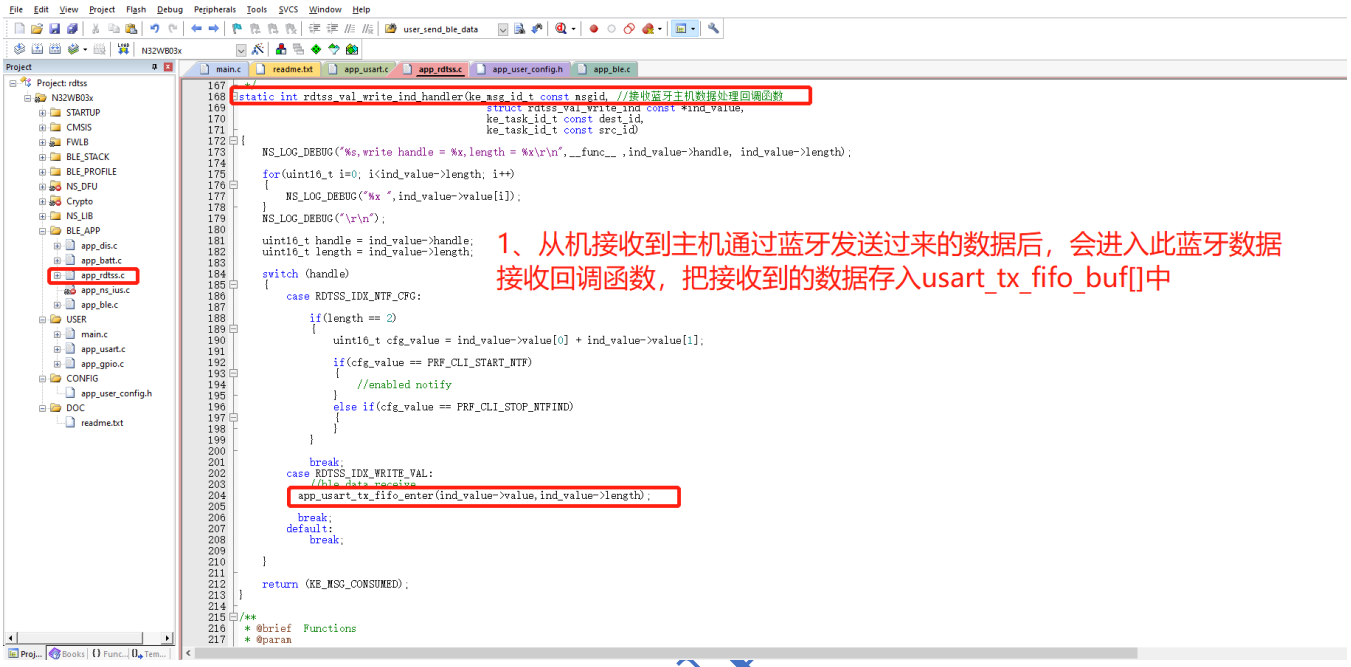
发送数据给从机：user_send_ble_data()




```
151
152
153 void user_send_ble_data(const uint8_t *data, uint16_t length)
154 {
155     NS_LOG_DEBUG("%s\r\n", __func__);
156     struct rdtsc_data_tx_req * req = KE_MSG_ALLOC_DYN(RDTSC_DATA_TX_REQ,
157                                                       prf_get_task_from_id(TASK_ID_RDTSC),
158                                                       TASK_APP,
159                                                       rdtsc_data_tx_req,
160                                                       length);
161
162     req->length = length;
163     memcpy(&req->data[0], data, length);
164     ke_msg_send(req);
165 }
166
167
168 /**
```

5.3 从机接收主机下发的数据

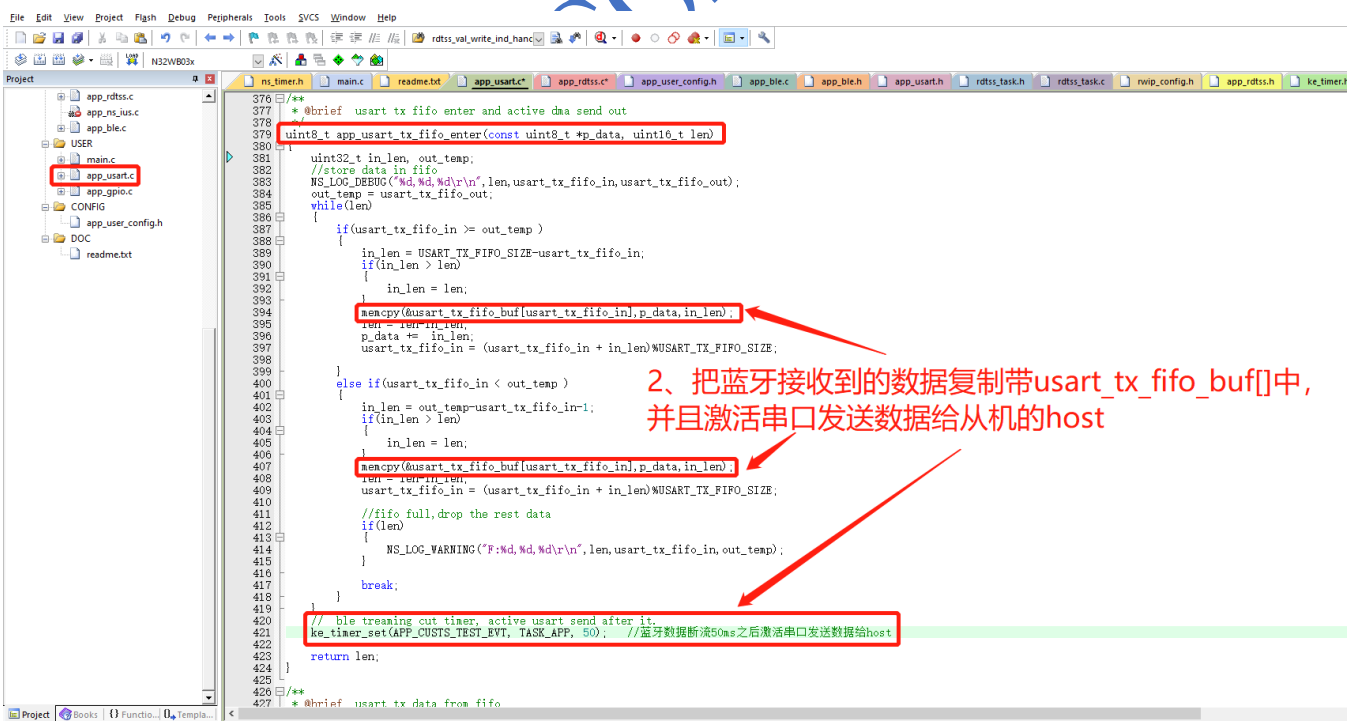
从机接收蓝牙数据回调函数：rdtss_val_write_ind_handler()



```
/*
static int rdtss_val_write_ind_handler(ke_msg_id_t const msgid, //接收蓝牙主机数据处理回调函数
                                       struct rdtss_val_write_ind const *ind_value,
                                       ke_task_id_t const dest_id,
                                       ke_task_id_t const src_id)
{
    NS_LOG_DEBUG("%s,write handle = %x,length = %x\r\n",__func__ ,ind_value->handle, ind_value->length);

    for(uint16_t i=0; i<ind_value->length; i++)
    {
        NS_LOG_DEBUG("%x ",ind_value->value[i]);
    }
    NS_LOG_DEBUG("\r\n");

    uint16_t handle = ind_value->handle;
    uint16_t length = ind_value->length;

    switch (handle)
    {
        case RDTSS_IDX_NTF_CFG:

            if(length == 2)
            {
                uint16_t cfg_value = ind_value->value[0] + ind_value->value[1];

                if(cfg_value == PRF_CLI_START_NTF)
                {
                    //enabled notify
                }
                else if(cfg_value == PRF_CLI_STOP_NTFIND)
                {
                }
            }

            break;
        case RDTSS_IDX_WRITE_VAL:
            //ble data receive
            app_usart_tx_fifo_enter(ind_value->value,ind_value->length);

          break;
        default:
            break;

    }

    return (KE_MSG_CONSUMED);
}

/**
 * @brief  Functions
 * @paran
```

1、从机接收到主机通过蓝牙发送过来的数据后，会进入此蓝牙数据接收回调函数，把接收到的数据存入usart_tx_fifo_buf[]中

把蓝牙接收到的数据复制到 usart_tx_fifo_buf[]：app_usart_tx_fifo_enter()，并创建发送数据定时器任务发送数据。



```
/**
 * @brief  usart tx fifo enter and active dma send out
 */
uint8_t app_usart_tx_fifo_enter(const uint8_t *p_data, uint16_t len)
{
    uint32_t in_len, out_temp;
    //store data in fifo
    NS_LOG_DEBUG("%d,%d,%d\r\n",len,usart_tx_fifo_in,usart_tx_fifo_out);
    out_temp = usart_tx_fifo_out;
    while(len)
    {
        if(usart_tx_fifo_in >= out_temp )
        {
            in_len = USART_TX_FIFO_SIZE-usart_tx_fifo_in;
            if(in_len > len)
            {
                in_len = len;
            }
            memcpy(&usart_tx_fifo_buf[usart_tx_fifo_in],p_data,in_len);
            len = len-in_len;
            p_data +=  in_len;
            usart_tx_fifo_in = (usart_tx_fifo_in + in_len)%USART_TX_FIFO_SIZE;

        }
        else if(usart_tx_fifo_in < out_temp )
        {
            in_len = out_temp-usart_tx_fifo_in-1;
            if(in_len > len)
            {
                in_len = len;
            }
            memcpy(&usart_tx_fifo_buf[usart_tx_fifo_in],p_data,in_len);
            len = len-in_len;
            usart_tx_fifo_in = (usart_tx_fifo_in + in_len)%USART_TX_FIFO_SIZE;

            //fifo full,drop the rest data
            if(len)
            {
                NS_LOG_WARNING("F:%d,%d,%d\r\n",len,usart_tx_fifo_in,out_temp);
            }

            break;
        }
    }
    //  ble treaming cut timer, active usart send after it.
    ke_timer_set(APP_CUSTS_TEST_EVT, TASK_APP, 50);   //蓝牙数据断流50ms之后激活串口发送数据给host

    return len;
}

/**
 * @brief  usart tx data from fifo
```

2、把蓝牙接收到的数据复制带usart_tx_fifo_buf[]中，并且激活串口发送数据给从机的host

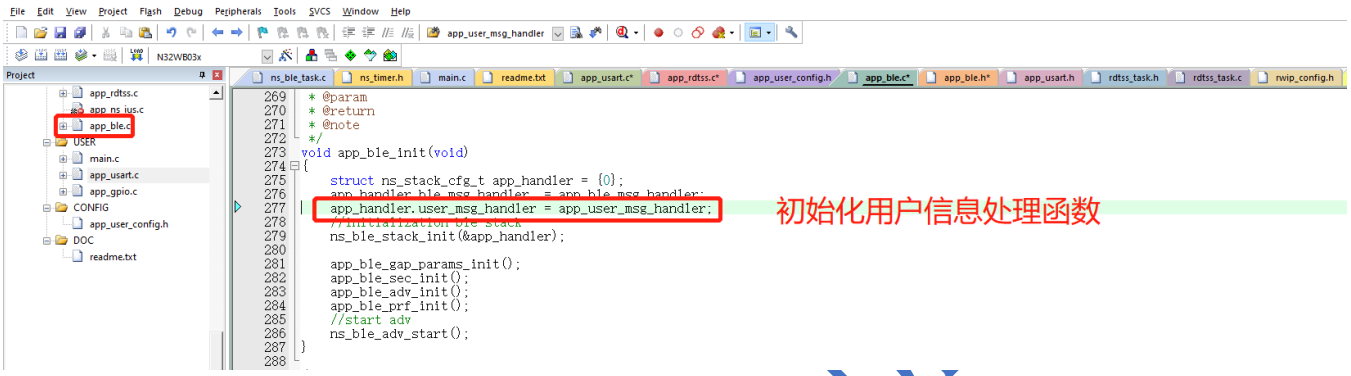
5.4 从机把数据发送给从机 host

从机通过串口 1 即 PB6-TX 和 PB7-RX 把数据发送给从机的 host

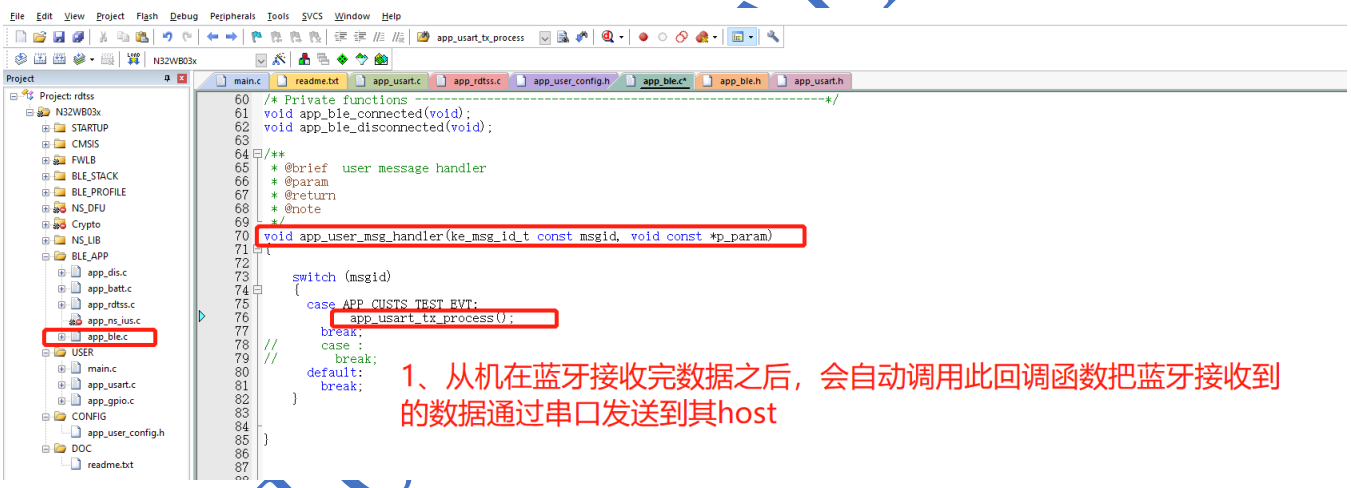
创建一个用户自定义的定时器任务：ke_timer_set(APP_CUSTS_TEST_EVT, TASK_APP, 50)

用户事件标志：APP_CUSTS_TEST_EVT

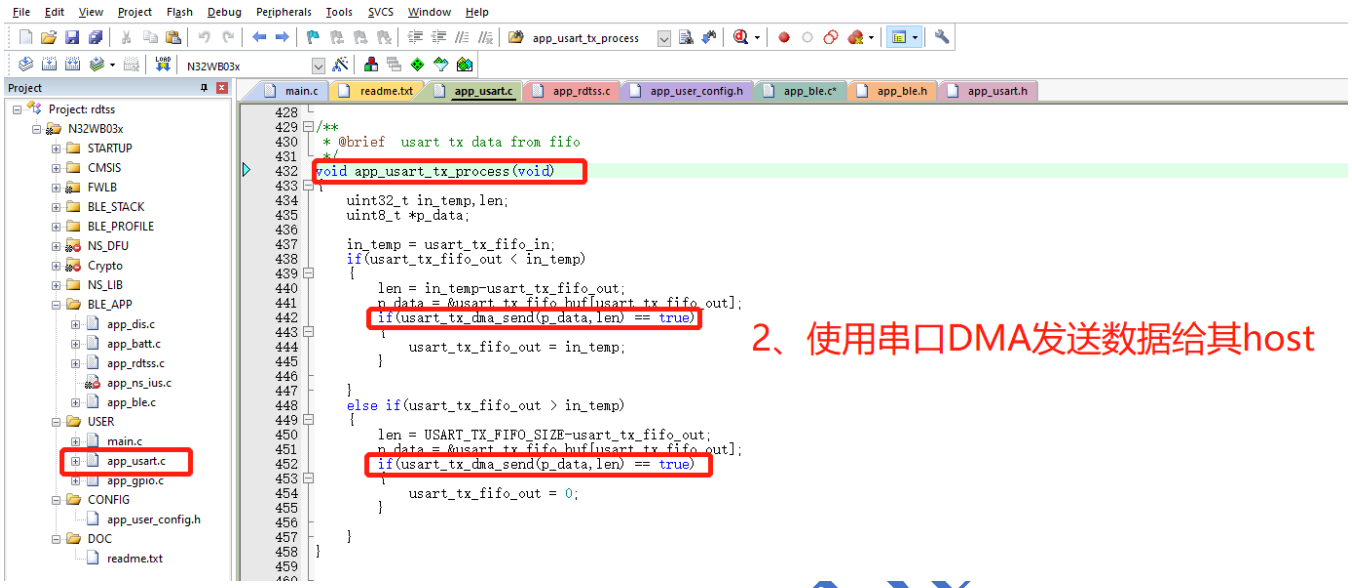
用户信息处理函数：app_user_msg_handler()



在蓝牙接收数据断流 50ms 之后，调用用户信息处理回调函数：app_user_msg_handler()



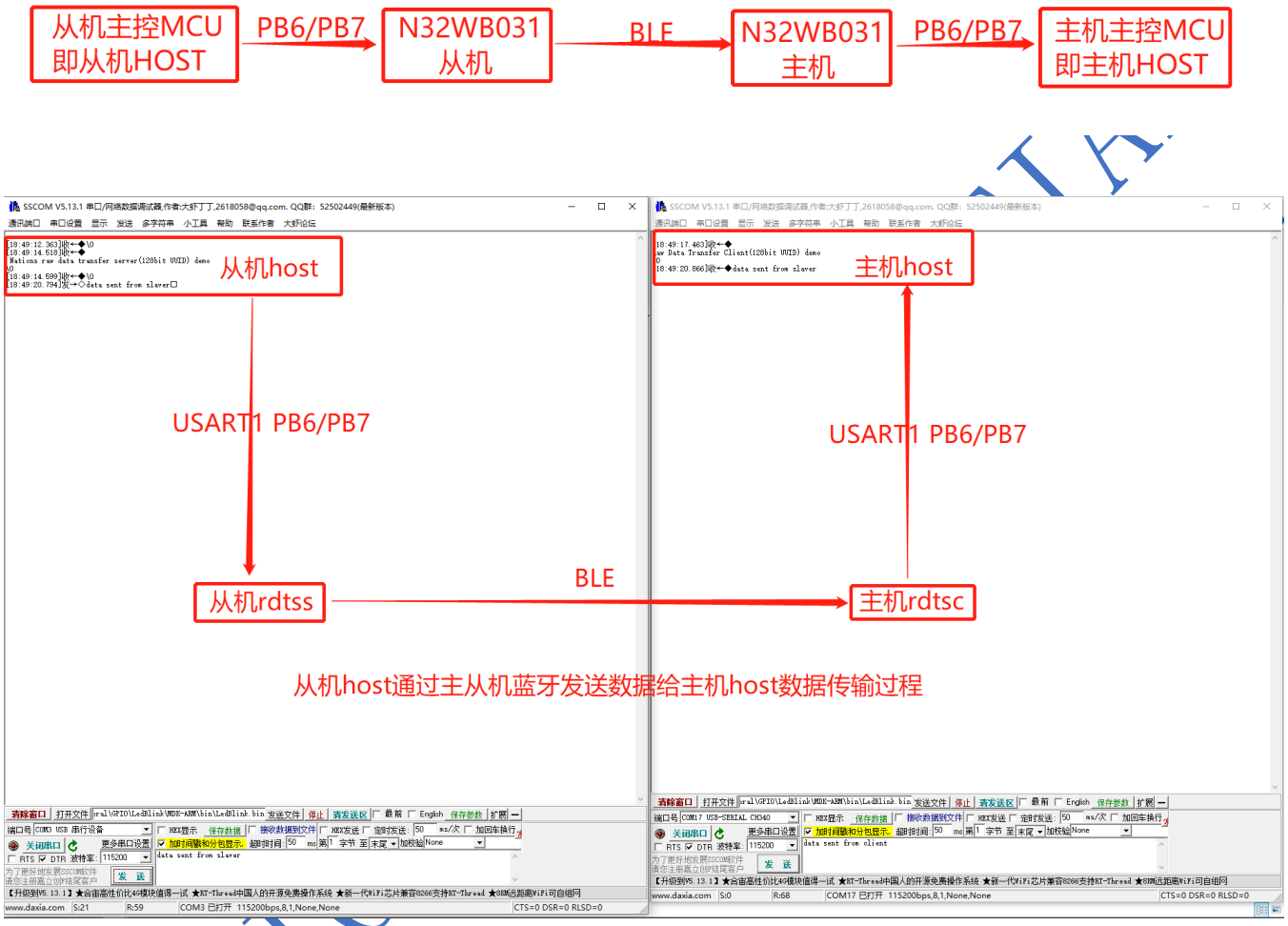
把数据通过串口发送给其 host：app_usart_tx_process()



2、使用串口DMA发送数据给其host

6. 从机 host 通过蓝牙发送数据给主机 host

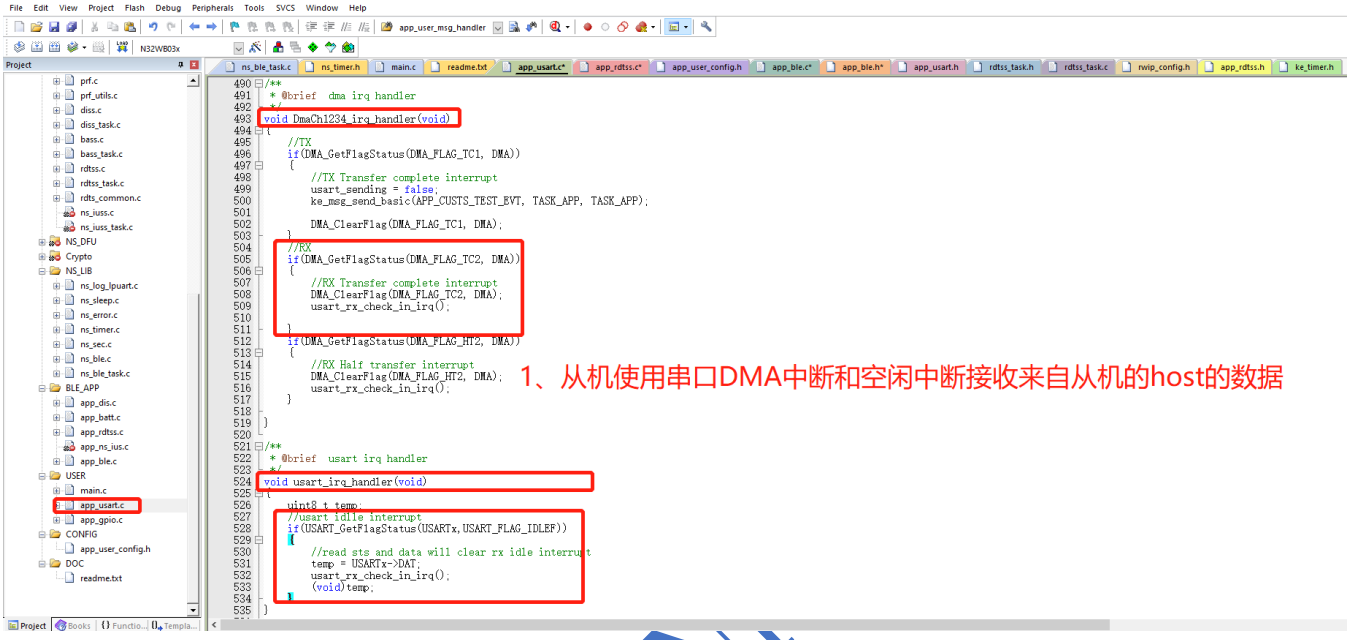
数据传输过程示意图：



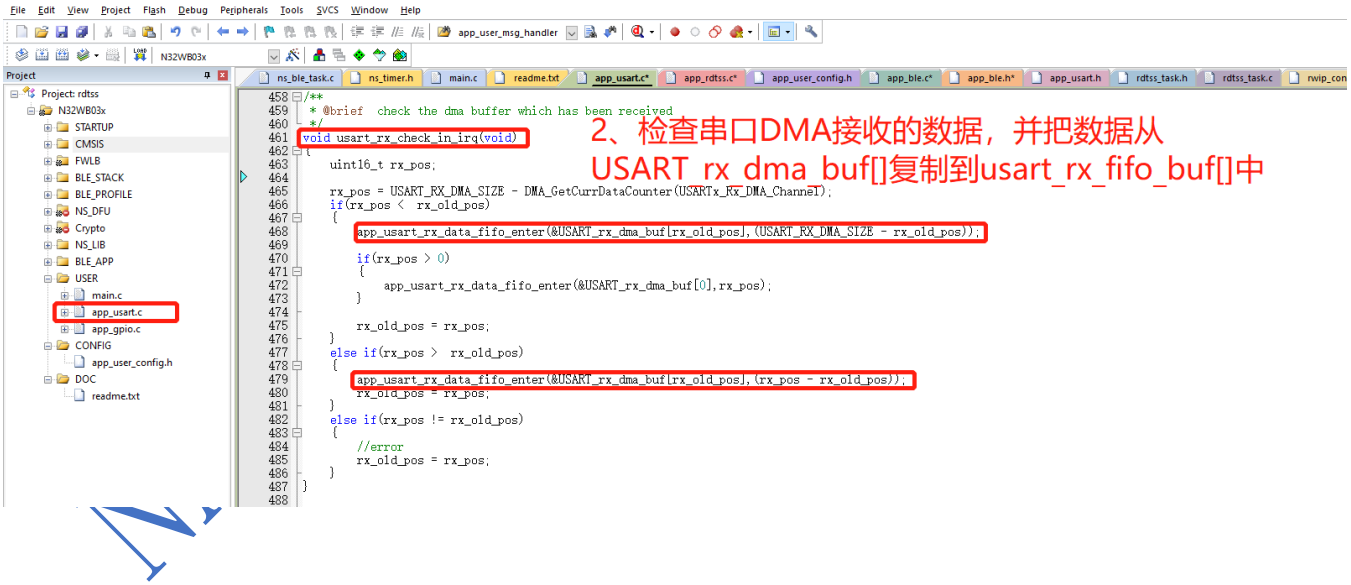
6.1 从机接收来自 host 的数据

从机主控 MCU 即从机 host 通过串口 1 的 PB6-TX 和 PB7-RX 把数据发送给从机。

从机通过 DmaCh1234_irq_handler() 和 usart_irq_handler()中断接收数据。

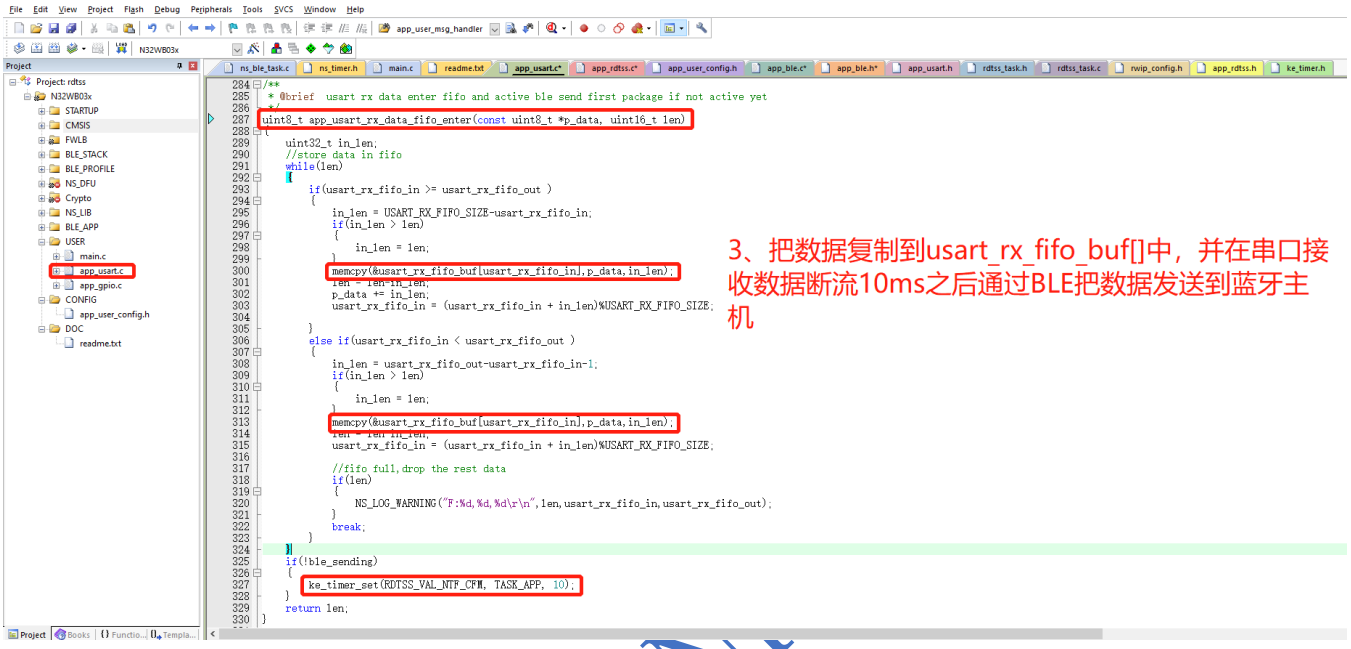


检查处理接收到的数据：usart_rx_check_in_irq()



6.2 从机上发数据给主机

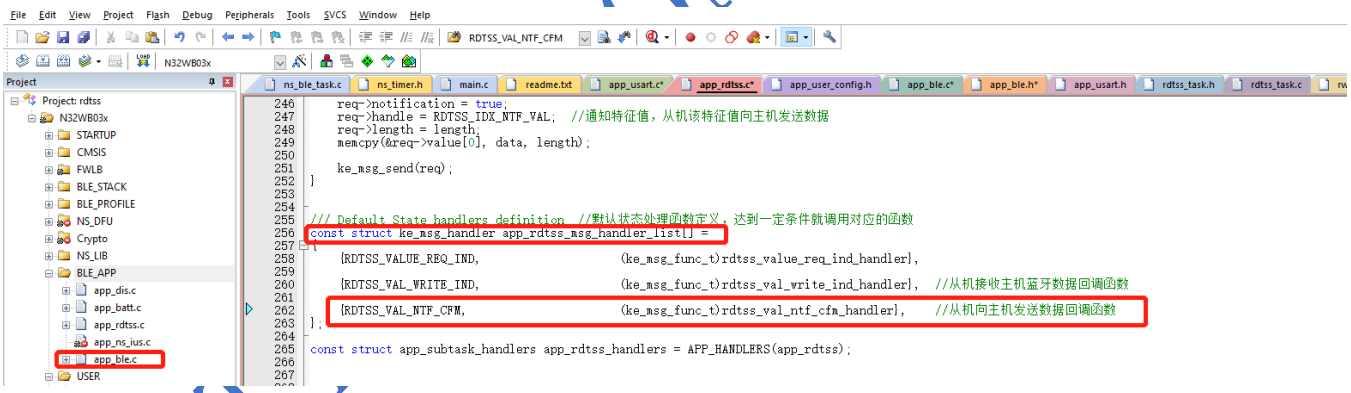
把 DMA 接收到的数据复制到 usart_rx_data_fifo 缓冲区：app_usart_rx_data_fifo_enter()，并创建发送数据定时器任务发送数据。



```
/**
 * @brief  usart rx data enter fifo and active ble send first package if not active yet
 */
uint8_t app_usart_rx_data_fifo_enter(const uint8_t *p_data, uint16_t len)
{
    uint32_t in_len;
    //store data in fifo
    while(len)
    {
        if(usart_rx_fifo_in >= usart_rx_fifo_out )
        {
            in_len = USART_RX_FIFO_SIZE-usart_rx_fifo_in;
            if(in_len > len)
            {
                in_len = len;
            }
            memcpy(&usart_rx_fifo_buf[usart_rx_fifo_in],p_data,in_len);
            len = len-in_len;
            p_data += in_len;
            usart_rx_fifo_in = (usart_rx_fifo_in + in_len)%USART_RX_FIFO_SIZE;
        }
        else if(usart_rx_fifo_in < usart_rx_fifo_out )
        {
            in_len = usart_rx_fifo_out-usart_rx_fifo_in-1;
            if(in_len > len)
            {
                in_len = len;
            }
            memcpy(&usart_rx_fifo_buf[usart_rx_fifo_in],p_data,in_len);
            len = len-in_len;
            usart_rx_fifo_in = (usart_rx_fifo_in + in_len)%USART_RX_FIFO_SIZE;

            //fifo full,drop the rest data
            if(len)
            {
                NS_LOG_WARNING("F:%d,%d,%d\r\n",len,usart_rx_fifo_in,usart_rx_fifo_out);
            }
            break;
        }
    }
    if(!ble_sending)
    {
        ke_timer_set(RDTSS_VAL_NTF_CFM, TASK_APP, 10);
    }
    return len;
}
```

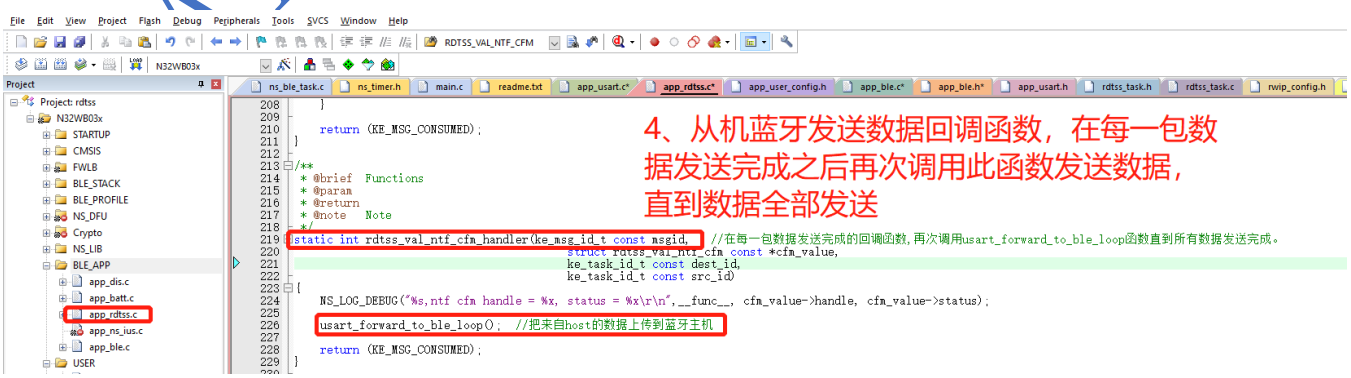
3、把数据复制到usart_rx_fifo_buf[]中，并在串口接收数据断流10ms之后通过BLE把数据发送到蓝牙主机

从机发送数据事件标志：RDTSS_VAL_NTF_CFM



```
    req->notification = true;
    req->handle = RDTSS_IDX_NTF_VAL;  //通知特征值，从机该特征值向主机发送数据
    req->length = length;
    memcpy(&req->value[0], data, length);

    ke_msg_send(req);
}

/// Default State handlers definition  //默认状态处理函数定义，达到一定条件就调用对应的函数
const struct ke_msg_handler app_rdtss_msg_handler_list[] =
{
    {RDTSS_VALUE_REQ_IND,           (ke_msg_func_t)rdtss_value_req_ind_handler},

    {RDTSS_VAL_WRITE_IND,           (ke_msg_func_t)rdtss_val_write_ind_handler},  //从机接收主机蓝牙数据回调函数

    {RDTSS_VAL_NTF_CFM,             (ke_msg_func_t)rdtss_val_ntf_cfm_handler},    //从机向主机发送数据回调函数
};

const struct app_subtask_handlers app_rdtss_handlers = APP_HANDLERS(app_rdtss);
```

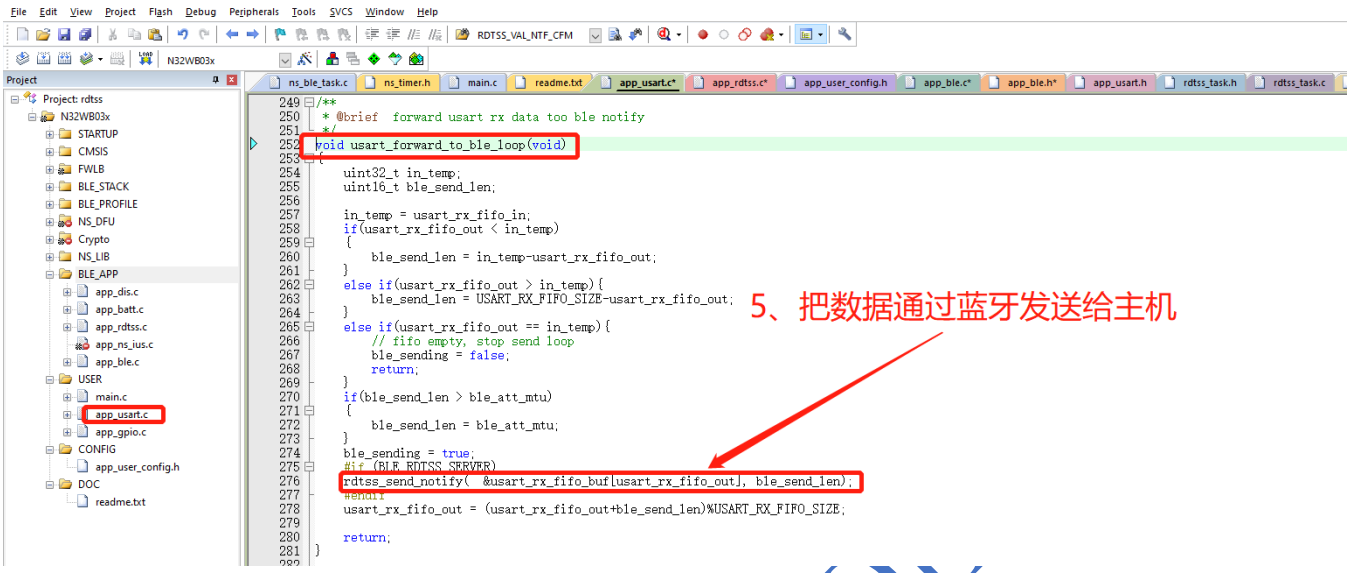
从机发送数据回调函数：rdtss_val_ntf_cfm_handler()



```
    }

    return (KE_MSG_CONSUMED);
}

/**
 * @brief  Functions
 * @param
 * @return
 * @note   Note
 */
static int rdtss_val_ntf_cfm_handler(ke_msg_id_t const msgid,  //在每一包数据发送完成的回调函数,再次调用usart_forward_to_ble_loop函数直到所有数据发送完成。
                                     struct rdtss_val_ntf_cfm const *cfm_value,
                                     ke_task_id_t const dest_id,
                                     ke_task_id_t const src_id)
{
    NS_LOG_DEBUG("%s,ntf cfm handle = %x,  status = %x\r\n",__func__,  cfm_value->handle,  cfm_value->status);

    usart_forward_to_ble_loop();  //把来自host的数据上传到蓝牙主机

    return (KE_MSG_CONSUMED);
}
```

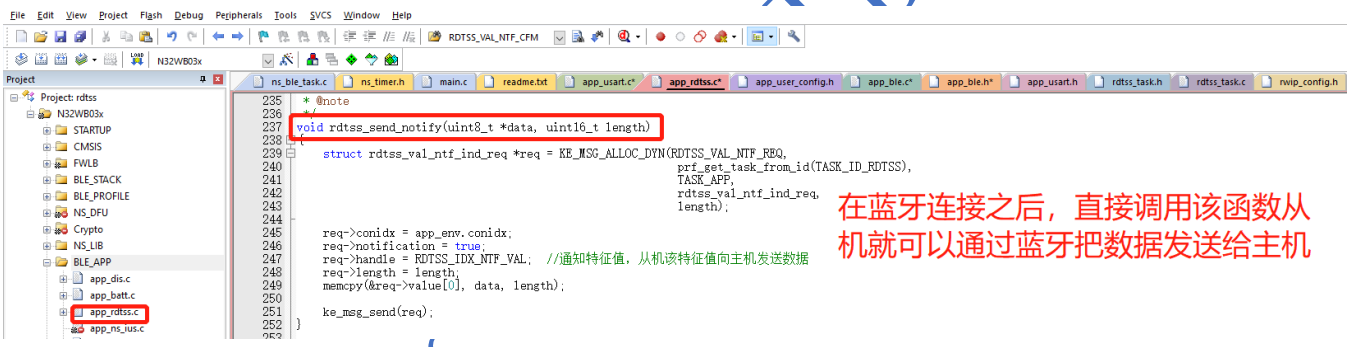
4、从机蓝牙发送数据回调函数，在每一包数据发送完成之后再次调用此函数发送数据，直到数据全部发送

循环调用该函数直到把所有的数据发送给主机：usart_forward_to_ble_loop()




```
/**
 * @brief  forward usart rx data too ble notify
 */
void usart_forward_to_ble_loop(void)
{
    uint32_t in_temp;
    uint16_t ble_send_len;

    in_temp = usart_rx_fifo_in;
    if(usart_rx_fifo_out < in_temp)
    {
        ble_send_len = in_temp-usart_rx_fifo_out;
    }
    else if(usart_rx_fifo_out > in_temp){
        ble_send_len = USART_RX_FIFO_SIZE-usart_rx_fifo_out;
    }
    else if(usart_rx_fifo_out == in_temp){
        // fifo empty, stop send loop
        ble_sending = false;
        return;
    }
    if(ble_send_len > ble_att_mtu)
    {
        ble_send_len = ble_att_mtu;
    }
    ble_sending = true;
    #if (BLE_RDTSS_SERVER)
    rdtss_send_notify(  &usart_rx_fifo_buf[usart_rx_fifo_out], ble_send_len);
    #endif
    usart_rx_fifo_out = (usart_rx_fifo_out+ble_send_len)%USART_RX_FIFO_SIZE;

    return;
}
```

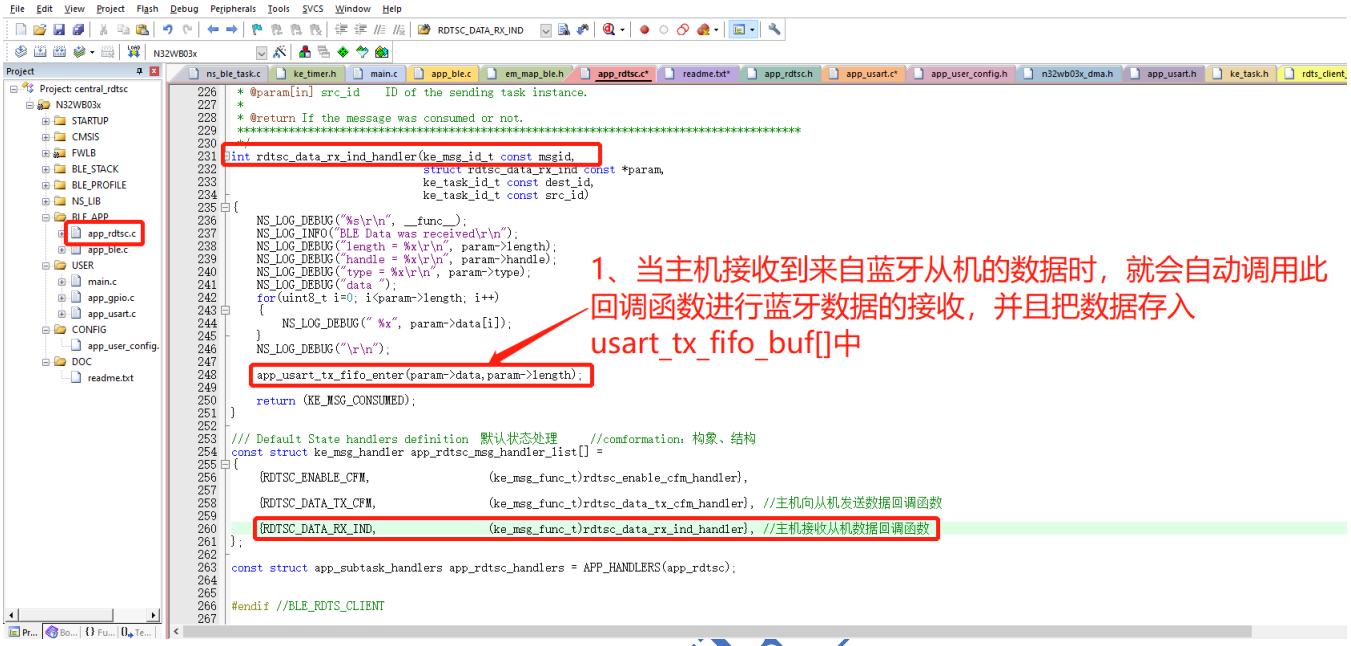
把数据发送给主机：rdtss_send_notify()



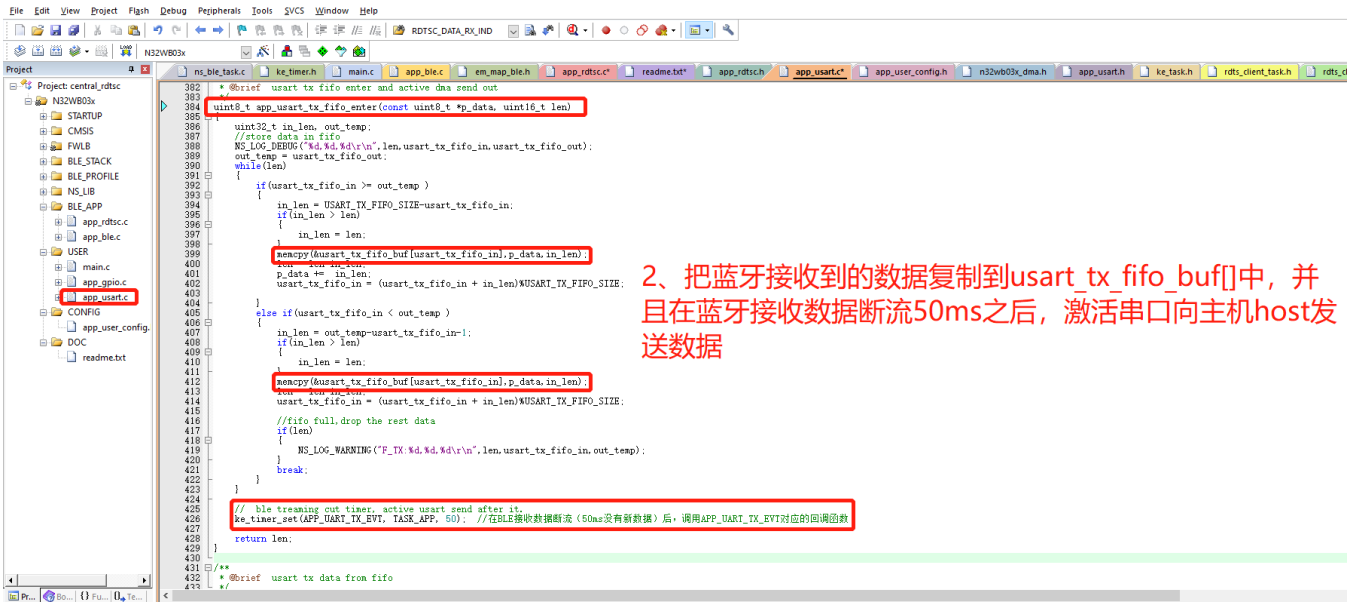

```
 * @note
 */
void rdtss_send_notify(uint8_t *data, uint16_t length)
{
    struct rdtss_val_ntf_ind_req *req = KE_MSG_ALLOC_DYN(RDTSS_VAL_NTF_REQ,
                                                         prf_get_task_from_id(TASK_ID_RDTSS),
                                                         TASK_APP,
                                                         rdtss_val_ntf_ind_req,
                                                         length);

    req->conidx = app_env.conidx;
    req->notification = true;
    req->handle = RDTSS_IDX_NTF_VAL;  //通知特征值，从机该特征值向主机发送数据
    req->length = length;
    memcpy(&req->value[0], data, length);

    ke_msg_send(req);
}
```

6.3 主机接收从机上发的数据

主机接收蓝牙数据回调函数：rdtsc_data_rx_ind_handler()



把接收到的数据复制到 usart_tx_fuf[]中：app_usart_tx_fifo_enter()，并创建定时器任务发送数据。

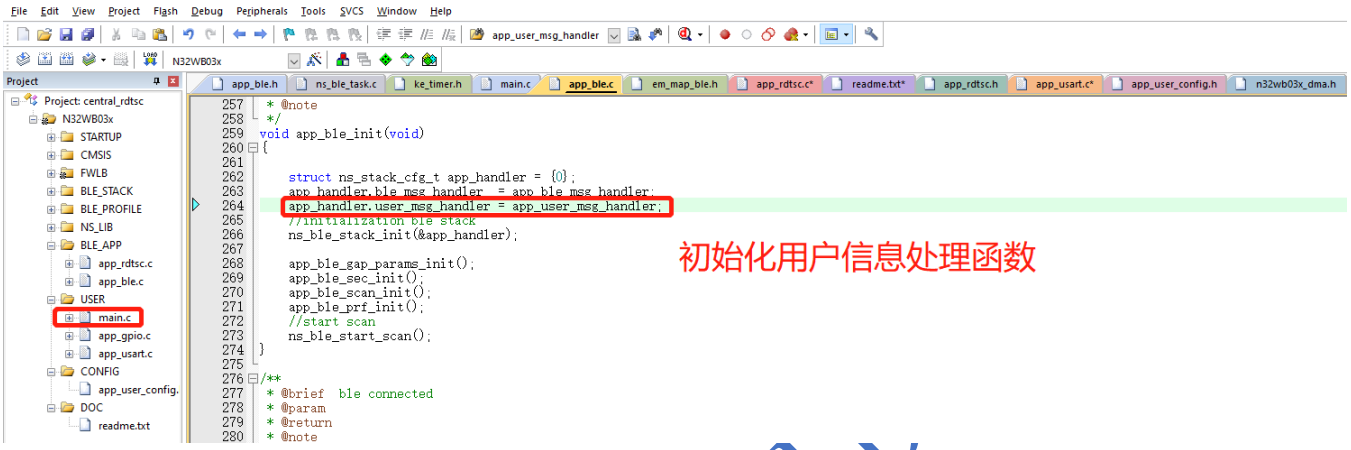


6.4 主机把数据发送给主机 host

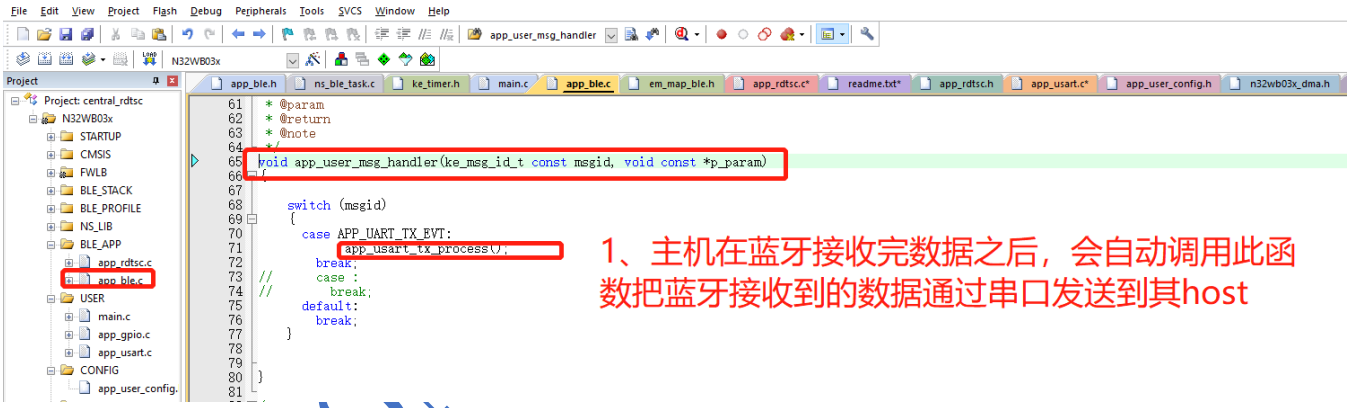
创建一个事件处理定时器，定时处理指定的事件：ke_timer_set(APP_CUSTS_TEST_EVT, TASK_APP, 50)

用户事件标志：APP_CUSTS_TEST_EVT

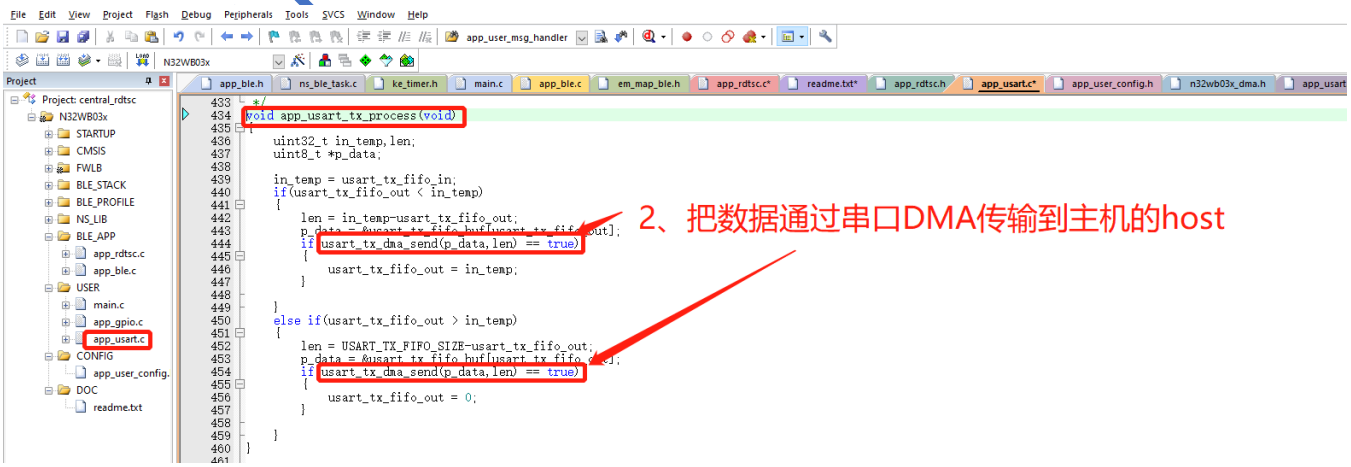
用户信息处理函数：app_user_msg_handler()



在蓝牙接收数据断流 50ms 之后，调用用户信息处理回调函数：app_user_msg_handler()



把数据通过串口发送给其主机：app_usart_tx_process()



注意事项

主从机收发数据接口：

主机 rdtsc 通过蓝牙下发数据给从机 rdtss 的接口：user_send_ble_data()，主机调用该函数给从机发数据；

主机 rdtsc 接收从机 rdtss 上发的蓝牙数据接口：rdtsc_data_rx_ind_handler()，主机在此函数中接收从机数据；

从机 rdtss 通过蓝牙上发数据给主机 rdtsc 的接口：rdtss_send_notify()，从机调用该函数给主机发数据；

从机 rdtss 接收主机 rdtsc 下发的蓝牙数据接口：rdtss_val_write_ind_handler()，从机在此函数中接收主机数据。

历史版本

版本	日期	备注
V1.0	2022.10.17	新建文档

声 明

国民技术股份有限公司（以下简称国民技术）保有在不事先通知而修改这份文档的权利。国民技术认为提供的信息是准确可信的。尽管这样，国民技术对文档中可能出现的错误不承担任何责任。在购买前请联系国民技术获取该器件说明的最新版本。对于使用该器件引起的专利纠纷及第三方侵权国民技术不承担任何责任。另外，国民技术的产品不建议应用于生命相关的设备和系统，在使用该器件中因为设备或系统运转失灵而导致的损失国民技术不承担任何责任。国民技术对本手册拥有版权等知识产权，受法律保护。未经国民技术许可，任何单位及个人不得以任何方式或理由对本手册进行使用、复制、修改、抄录、传播等。