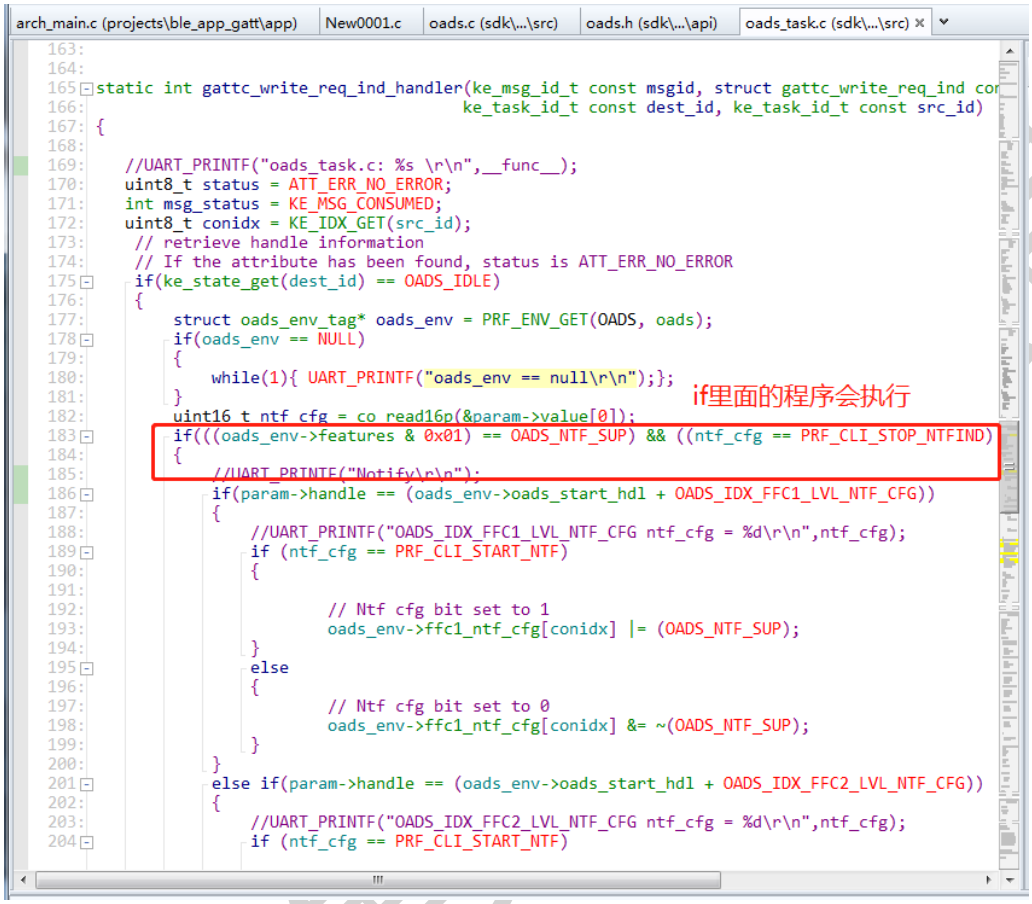
打开RC OTA，找到对应的蓝牙设备连接上后。

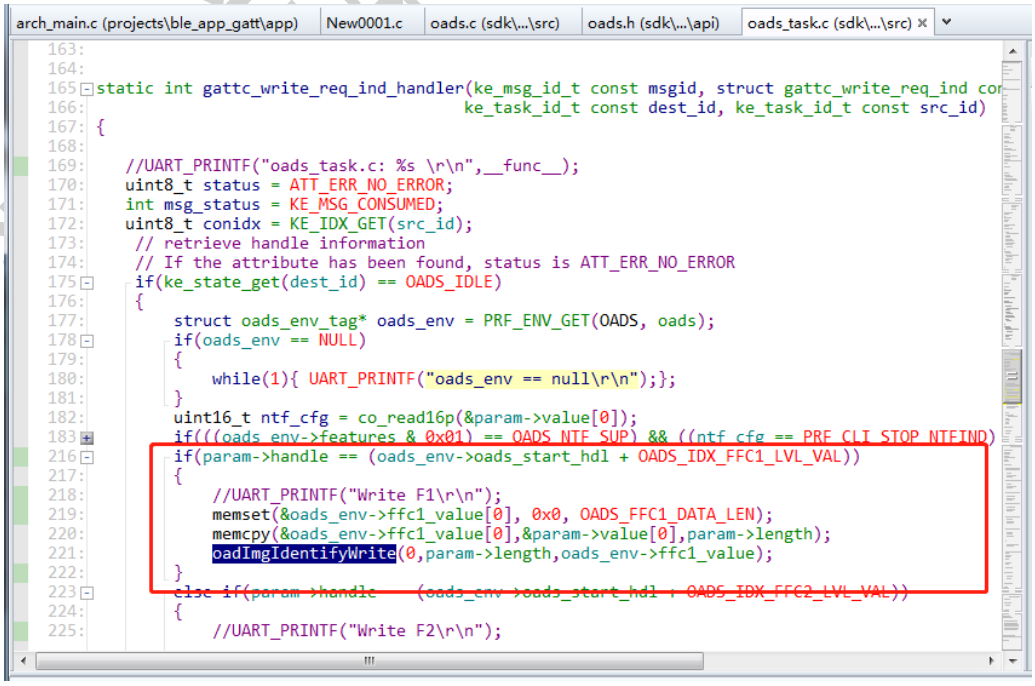
1. 点击OTA按钮，APP打开FFC1通道的notify监听，同时APP执行FFC1通道的write动作；蓝牙程序进入gattc_write_req_ind_handler函数。

（1） APP打开FFC1通道的notify监听，蓝牙端执行的对应程序：




```
163:
164:
165: static int gattc_write_req_ind_handler(ke_msg_id_t const msgid, struct gattc_write_req_ind co
166:                                        ke_task_id_t const dest_id, ke_task_id_t const src_id)
167: {
168:
169:     //UART_PRINTF("oads_task.c: %s \r\n",__func__);
170:     uint8_t status = ATT_ERR_NO_ERROR;
171:     int msg_status = KE_MSG_CONSUMED;
172:     uint8_t conidx = KE_IDX_GET(src_id);
173:      // retrieve handle information
174:      // If the attribute has been found, status is ATT_ERR_NO_ERROR
175:     if(ke_state_get(dest_id) == OADS_IDLE)
176:     {
177:         struct oads_env_tag* oads_env = PRF_ENV_GET(OADS, oads);
178:         if(oads_env == NULL)
179:         {
180:             while(1){ UART_PRINTF("oads_env == null\r\n");};
181:         }
182:         uint16_t ntf_cfg = co_read16p(&param->value[0]);
183:         if(((oads_env->features & 0x01) == OADS_NTF_SUP) && ((ntf_cfg == PRF_CLI_STOP_NTFIND)
184:         {
185:             //UART_PRINTF("Notify\r\n");
186:             if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC1_LVL_NTF_CFG))
187:             {
188:                 //UART_PRINTF("OADS_IDX_FFC1_LVL_NTF_CFG ntf_cfg = %d\r\n",ntf_cfg);
189:                 if (ntf_cfg == PRF_CLI_START_NTF)
190:                 {
191:
192:                         // Ntf cfg bit set to 1
193:                         oads_env->ffc1_ntf_cfg[conidx] |= (OADS_NTF_SUP);
194:                 }
195:                 else
196:                 {
197:                         // Ntf cfg bit set to 0
198:                         oads_env->ffc1_ntf_cfg[conidx] &= ~(OADS_NTF_SUP);
199:                 }
200:             }
201:             else if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC2_LVL_NTF_CFG))
202:             {
203:                 //UART_PRINTF("OADS_IDX_FFC2_LVL_NTF_CFG ntf_cfg = %d\r\n",ntf_cfg);
204:                 if (ntf_cfg == PRF_CLI_START_NTF)
```

（2） APP执行FFC1通道的write动作，蓝牙端执行的对应程序：



```
163:
164:
165: static int gattc_write_req_ind_handler(ke_msg_id_t const msgid, struct gattc_write_req_ind co
166:                                        ke_task_id_t const dest_id, ke_task_id_t const src_id)
167: {
168:
169:     //UART_PRINTF("oads_task.c: %s \r\n",__func__);
170:     uint8_t status = ATT_ERR_NO_ERROR;
171:     int msg_status = KE_MSG_CONSUMED;
172:     uint8_t conidx = KE_IDX_GET(src_id);
173:      // retrieve handle information
174:      // If the attribute has been found, status is ATT_ERR_NO_ERROR
175:     if(ke_state_get(dest_id) == OADS_IDLE)
176:     {
177:         struct oads_env_tag* oads_env = PRF_ENV_GET(OADS, oads);
178:         if(oads_env == NULL)
179:         {
180:             while(1){ UART_PRINTF("oads_env == null\r\n");};
181:         }
182:         uint16_t ntf_cfg = co_read16p(&param->value[0]);
183:         if(((oads_env->features & 0x01) == OADS_NTF_SUP) && ((ntf_cfg == PRF_CLI_STOP_NTFIND)
216:         if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC1_LVL_VAL))
217:         {
218:             //UART_PRINTF("Write F1\r\n");
219:             memset(&oads_env->ffc1_value[0], 0x0, OADS_FFC1_DATA_LEN);
220:             memcpy(&oads_env->ffc1_value[0],&param->value[0],param->length);
221:             oadImgIdentifyWrite(0,param->length,oads_env->ffc1_value);
222:         }
223:         else if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC2_LVL_VAL))
224:         {
225:             //UART_PRINTF("Write F2\r\n");
```

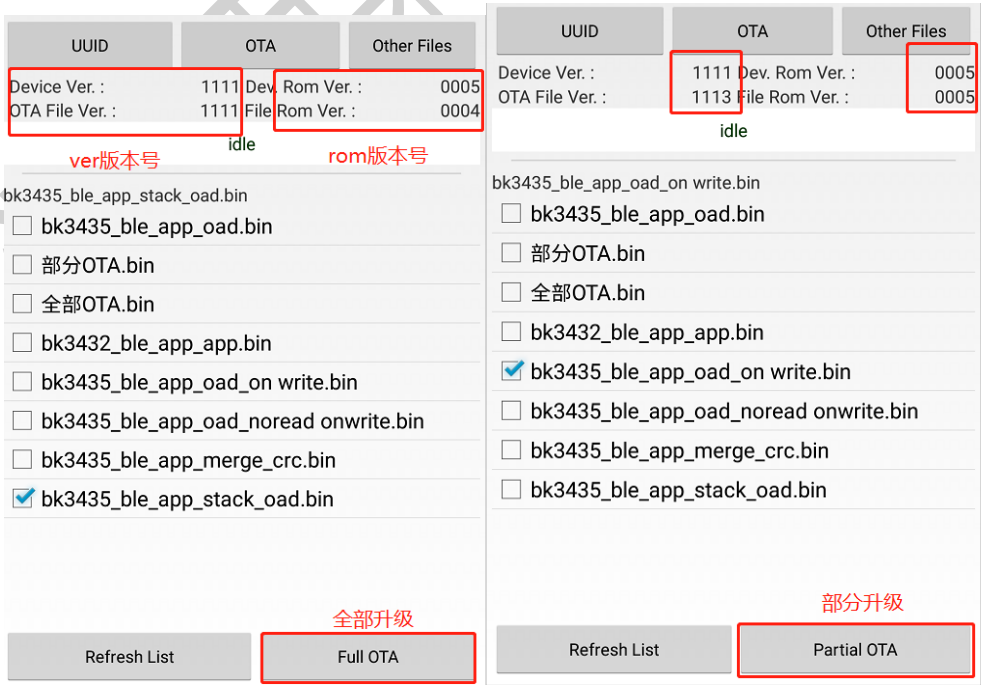
（3） 蓝牙端读出蓝牙设备程序的ver版本号和rom版本号（在0x20010地址上），蓝牙notify回传给APP。APP获取版本号后和升级的bin文件的版本号对比（如果ver相等，rom不相等，判断为部分升级；如果ver不相等，rom相等，判断为全部升级）

```
arch_main.c (projects\ble_app_gatt\app)  New0001.c  oads.c (sdk\...\src)  oads.h (sdk\...\api)  oads_task.c (sdk\...\src)
210: }
211:
212: void appm_update_param(struct gapc_conn_param *conn_param);
213: uint8_t oadImgIdentifyWrite( uint16_t connHandle, uint16_t length,uint8_t *pValue
214: {
215:     UART_PRINTF("oads.c:%s line:%d\r\n",__func__,__LINE__);
216:     img_hdr_t rxHdr;
217:     img_hdr_t ImgHdr;
218:     rxHdr.ver = co_read16p(&(pValue[4]));
219:     UART_PRINTF("rxHdr.ver.      = %x \r\n", rxHdr.ver);
220:     rxHdr.len = co_read16p(&(pValue[6]));
221:     UART_PRINTF("rxHdr.len.      = %x \r\n", rxHdr.len);
222:     rxHdr.uid = co_read32p(&(pValue[8]));
223:     UART_PRINTF("rxHdr.uid.      = %x \r\n", rxHdr.uid);
224:     rxHdr.rom_ver = co_read16p(&(pValue[14]));
225:     UART_PRINTF("rxHdr.rom_ver. = %x \r\n", rxHdr.rom_ver);
226:     if(rxHdr.uid == OAD_APP_PART_UID) ...
242:     else if(rxHdr.uid == OAD_APP_STACK_UID)//全部OTA ： bin文件的版本信息 ...
259:     else
260:     {
261:         oad_firmware_type = 0;
262:         flash_read(0,SEC_IMAGE_OAD_HEADER_APP_FADDR, sizeof(img_hdr_t),(uint8_t *)&ImgHdr, NUL
263:         UART_PRINTF("ImgHdr.rom_ver.= %x \r\n", ImgHdr.rom_ver);
264:         UART_PRINTF("ImgHdr.uid.     = %x \r\n", ImgHdr.uid);
265:         UART_PRINTF("ImgHdr.ver.     = %x \r\n", ImgHdr.ver);
266:     }
267:     if(oad_uid_check_status == 1) ...
281:     else
282:     {
283:         oadImgIdentifyReq(connHandle, &ImgHdr);
284:     }
285:     return ( 0x00 );//SUCCESS
286: }
287:
288: /*********************************************************************
289:  * @fn      oadImgBlockWrite
290:  *
291:  * @brief   Process the Image Block Write.
292:  *
293:  * @param   connHandle - connection message was received on
294:  * @param   pValue - pointer to data to be written
```

notify回传

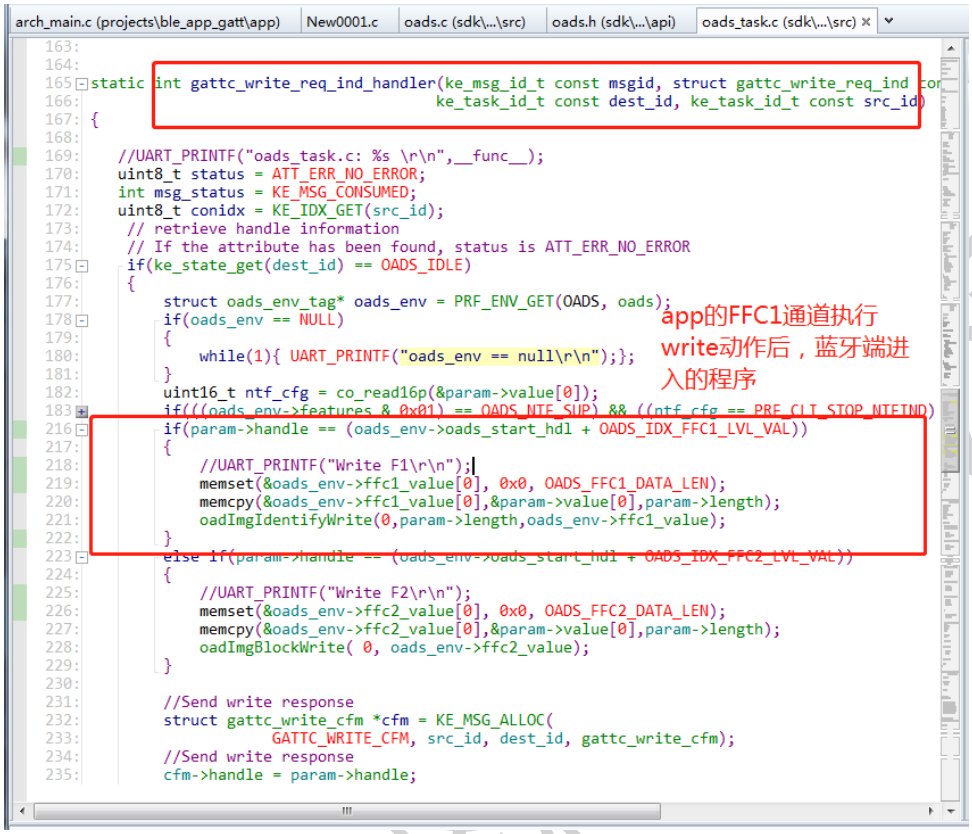
```
SEC_IMAGE_OAD_HEADER_APP_FADDR Constant in oads.h (sdk\...\api) at line 85

#define SEC_IMAGE_OAD_HEADER_APP_FADDR      (0x20010)
#define SEC_IMAGE_OAD_HEADER_STACK_FADDR    (0x20e0)

#define SEC_BACKUP_OAD_HEADER_FADDR         (0x44000) //272kb * 1024
//#define SEC_BACKUP_ALLOC_START_FADDR      (0x44000) //(272KB)
#define SEC_BACKUP_ALLOC_END_FADDR          (0x80000) //(512KB)
/*********************************************************************
```

（4） 不同升级模式的APP显示信息：



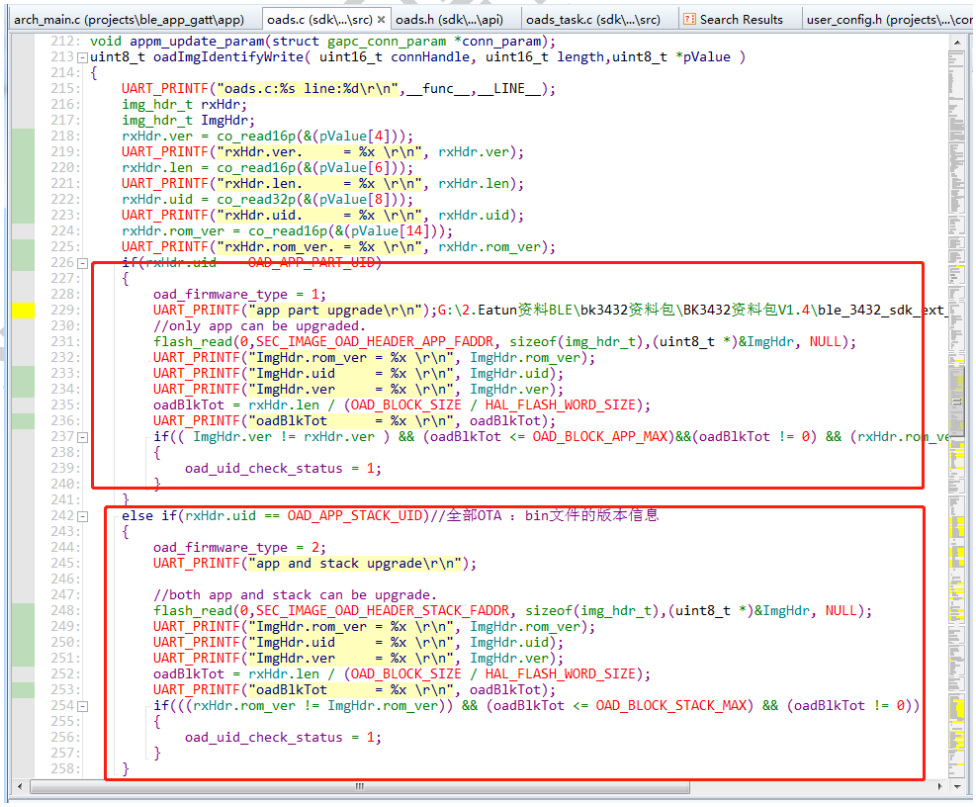
2. 点击Full OTA或者Partial OTA按钮，APP通过FFC1的write通道将ver版本号和rom版本号下发给蓝牙，蓝牙对比后判断是全部升级还是部分升级。

（1） APP端在FFC1通道上执行write动作后，蓝牙端执行的程序。




```
static int gattc_write_req_ind_handler(ke_msg_id_t const msgid, struct gattc_write_req_ind const *param,
                                       ke_task_id_t const dest_id, ke_task_id_t const src_id)
{
    //UART_PRINTF("oads_task.c: %s \r\n",__func__);
    uint8_t status = ATT_ERR_NO_ERROR;
    int msg_status = KE_MSG_CONSUMED;
    uint8_t conidx = KE_IDX_GET(src_id);
     // retrieve handle information
     // If the attribute has been found, status is ATT_ERR_NO_ERROR
     if(ke_state_get(dest_id) == OADS_IDLE)
     {
         struct oads_env_tag* oads_env = PRF_ENV_GET(OADS, oads);
         if(oads_env == NULL)
         {
             while(1){ UART_PRINTF("oads_env == null\r\n");};
         }
         uint16_t ntf_cfg = co_read16p(&param->value[0]);
         if(((oads_env->features & 0x01) == OADS_NTF_SUP) && (ntf_cfg == PRF_CLI_STOP_NTFIND))
         if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC1_LVL_VAL))
         {
             //UART_PRINTF("Write F1\r\n");
             memset(&oads_env->ffc1_value[0], 0x0, OADS_FFC1_DATA_LEN);
             memcpy(&oads_env->ffc1_value[0],&param->value[0],param->length);
             oadImgIdentifyWrite(0,param->length,oads_env->ffc1_value);
         }
         else if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC2_LVL_VAL))
         {
             //UART_PRINTF("Write F2\r\n");
             memset(&oads_env->ffc2_value[0], 0x0, OADS_FFC2_DATA_LEN);
             memcpy(&oads_env->ffc2_value[0],&param->value[0],param->length);
             oadImgBlockWrite( 0, oads_env->ffc2_value);
         }

         //Send write response
         struct gattc_write_cfm *cfm = KE_MSG_ALLOC(
                 GATTC_WRITE_CFM, src_id, dest_id, gattc_write_cfm);
         //Send write response
         cfm->handle = param->handle;
```

（2） 选择部分升级还是全部升级：



```
void appm_update_param(struct gapc_conn_param *conn_param);
uint8_t oadImgIdentifyWrite( uint16_t connHandle, uint16_t length,uint8_t *pValue )
{
    UART_PRINTF("oads.c:%s line:%d\r\n",__func__,__LINE__);
    img_hdr_t rxHdr;
    img_hdr_t ImgHdr;
    rxHdr.ver = co_read16p(&(pValue[4]));
    UART_PRINTF("rxHdr.ver.      = %x \r\n", rxHdr.ver);
    rxHdr.len = co_read16p(&(pValue[6]));
    UART_PRINTF("rxHdr.len.      = %x \r\n", rxHdr.len);
    rxHdr.uid = co_read32p(&(pValue[8]));
    UART_PRINTF("rxHdr.uid.      = %x \r\n", rxHdr.uid);
    rxHdr.rom_ver = co_read16p(&(pValue[14]));
    UART_PRINTF("rxHdr.rom_ver. = %x \r\n", rxHdr.rom_ver);
    if(rxHdr.uid    OAD_APP_PART_UID)
    {
        oad_firmware_type = 1;
        UART_PRINTF("app part upgrade\r\n");G:\2.Eatun资料BLE\bk3432资料包\BK3432资料包V1.4\ble_3432_sdk_ext_
        //only app can be upgraded.
        flash_read(0,SEC_IMAGE_OAD_HEADER_APP_FADDR, sizeof(img_hdr_t),(uint8_t *)&ImgHdr, NULL);
        UART_PRINTF("ImgHdr.rom_ver = %x \r\n", ImgHdr.rom_ver);
        UART_PRINTF("ImgHdr.uid     = %x \r\n", ImgHdr.uid);
        UART_PRINTF("ImgHdr.ver     = %x \r\n", ImgHdr.ver);
        oadBlkTot = rxHdr.len / (OAD_BLOCK_SIZE / HAL_FLASH_WORD_SIZE);
        UART_PRINTF("oadBlkTot      = %x \r\n", oadBlkTot);
        if(( ImgHdr.ver != rxHdr.ver ) && (oadBlkTot <= OAD_BLOCK_APP_MAX)&&(oadBlkTot != 0) && (rxHdr.rom_ve
        {
            oad_uid_check_status = 1;
        }
    }
    else if(rxHdr.uid == OAD_APP_STACK_UID)//全部OTA : bin文件的版本信息
    {
        oad_firmware_type = 2;
        UART_PRINTF("app and stack upgrade\r\n");

        //both app and stack can be upgrade.
        flash_read(0,SEC_IMAGE_OAD_HEADER_STACK_FADDR, sizeof(img_hdr_t),(uint8_t *)&ImgHdr, NULL);
        UART_PRINTF("ImgHdr.rom_ver = %x \r\n", ImgHdr.rom_ver);
        UART_PRINTF("ImgHdr.uid     = %x \r\n", ImgHdr.uid);
        UART_PRINTF("ImgHdr.ver     = %x \r\n", ImgHdr.ver);
        oadBlkTot = rxHdr.len / (OAD_BLOCK_SIZE / HAL_FLASH_WORD_SIZE);
        UART_PRINTF("oadBlkTot      = %x \r\n", oadBlkTot);
        if(((rxHdr.rom_ver != ImgHdr.rom_ver)) && (oadBlkTot <= OAD_BLOCK_STACK_MAX) && (oadBlkTot != 0))
        {
            oad_uid_check_status = 1;
        }
    }
```

（3） 将升级信息通过FFC2通道的notify回传给APP：

main.c (projects\ble_app_gatt\app) | oads.c (sdk\...\src) * × | oads.h (sdk\...\api) | oads_task.c (sdk\...\src) | Search Results

```
212: void appm_update_param(struct gapc_conn_param *conn_param);
213:uint8_t oadImgIdentifyWrite( uint16_t connHandle, uint16_t length,uint8_t *pValue )
214: {
215:     UART_PRINTF("oads.c:%s line:%d\r\n",__func__,__LINE__);
216:     img_hdr_t rxHdr;
217:     img_hdr_t ImgHdr;
218:     rxHdr.ver = co_read16p(&(pValue[4]));
219:     UART_PRINTF("rxHdr.ver.      = %x \r\n", rxHdr.ver);
220:     rxHdr.len = co_read16p(&(pValue[6]));
221:     UART_PRINTF("rxHdr.len.      = %x \r\n", rxHdr.len);
222:     rxHdr.uid = co_read32p(&(pValue[8]));
223:     UART_PRINTF("rxHdr.uid.      = %x \r\n", rxHdr.uid);
224:     rxHdr.rom_ver = co_read16p(&(pValue[14]));
225:     UART_PRINTF("rxHdr.rom_ver. = %x \r\n", rxHdr.rom_ver);
226:     if(rxHdr.uid == OAD_APP_PART_UID) ...
242:     else if(rxHdr.uid == OAD_APP_STACK_UID)//全部OTA ： bin文件的版本信息 ...
259:     else ...
260:     if(oad_uid_check_status == 1)
263:     {
269:         oadBlkNum = 0;
270:         latency_disable_state = 1;//失能
271:         oad_uid_check_status = 0;
272:         //update oad connect parameter.
273:         struct gapc_conn_param param;
274:         param.intv_min = 12;
275:         param.intv_max = 14;
276:         param.latency = 0;
277:         param.time_out = 300;
278:         appm_update_param(&param);
279:         oadImgBlockReq(connHandle, 0);
280:     }
281:     else
282:     {
283:         oadImgIdentifyReq(connHandle, &ImgHdr);
284:     }
285:     return ( 0x00 );//SUCCESS
286: }
287:
```

（4） APP开始将bin文件写入0x44000地址，bin写入完成后，蓝牙重启，由boot将bin文件搬到指定位置，这部分用户不需要理会：

arch_main.c (projects\ble_app_gatt\app) | oads.c (sdk\...\src) | oads.h (sdk\...\api) | oads_task.c (sdk\...\src) × | Search Results

```
163:
164:
165:static int gattc_write_req_ind_handler(ke_msg_id_t const msgid, struct gattc_write_req_ind const
166:                                       ke_task_id_t const dest_id, ke_task_id_t const src_id)
167: {
168:
169:     //UART_PRINTF("oads_task.c: %s \r\n",__func__);
170:     uint8_t status = ATT_ERR_NO_ERROR;
171:     int msg_status = KE_MSG_CONSUMED;
172:     uint8_t conidx = KE_IDX_GET(src_id);
173:      // retrieve handle information
174:      // If the attribute has been found, status is ATT_ERR_NO_ERROR
175:     if(ke_state_get(dest_id) == OADS_IDLE)
176:     {
177:         struct oads_env_tag* oads_env = PRF_ENV_GET(OADS, oads);
178:         if(oads_env == NULL)
179:         {
180:             while(1){ UART_PRINTF("oads_env == null\r\n");};
181:         }
182:         uint16_t ntf_cfg = co_read16p(&param->value[0]);
183:         if(((oads_env->features & 0x01) == OADS_NTF_SUP) && ((ntf_cfg == PRF_CLI_STOP_NTFIND) |
216:         if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC1_LVL_VAL))
217:         {
218:             //UART_PRINTF("Write F1\r\n");
219:             memset(&oads_env->ffc1_value[0], 0x0, OADS_FFC1_DATA_LEN);
220:             memcpy(&oads_env->ffc1_value[0],&param->value[0],param->length);
221:             oadImgIdentifyWrite(0,param->length,oads_env->ffc1_value);
222:         }
223:         else if(param->handle == (oads_env->oads_start_hdl + OADS_IDX_FFC2_LVL_VAL))
224:         {
225:             //UART_PRINTF("Write F2\r\n");
226:             memset(&oads_env->ffc2_value[0], 0x0, OADS_FFC2_DATA_LEN);
227:             memcpy(&oads_env->ffc2_value[0],&param->value[0],param->length);
228:             oadImgBlockWrite( 0, oads_env->ffc2_value);
229:         }
230:
231:         //Send write response
232:         struct gattc_write_cfm *cfm = KE_MSG_ALLOC(
233:                 GATTC_WRITE_CFM, src_id, dest_id, gattc_write_cfm);
234:         //Send write response
235:         cfm->handle = param->handle;
236:         cfm->status = status;
237:         ke_msg_send(cfm);
238:     }
239:     else if(ke_state_get(dest_id) == OADS_BUSY)
240:     {
241:         UART_PRINTF("OADS_BUSY\r\n");
```