Ble\_Initialize

--->.transportInterface = Ble\_HciRecv

--->Hcit\_Init(&hcitConfigStruct)

--->Ble\_HostTaskInit()

--->Ble\_HostInitialize(gapGenericCallback, Hcit\_SendPacket);

===Hcit\_SendPacket===

--->Serial\_AsyncWrite(gHcitSerMgrIf, (uint8\_t\*)(&packetType), 1, NULL, NULL );

-->Serial\_SyncWrite(gHcitSerMgrIf, (uint8\_t\*)pPacket, packetSize);

===App\_GenericCallback===

---->MSG\_Queue(&mHostAppInputQueue, pMsgIn);

---->OSA\_EventSet(mAppEvent, gAppEvtMsgFromHostStack\_c);

fsl\_os\_abstraction\_free\_rtos.c

Main

---->hardware\_init();

---->OSA\_TaskCreate(OSA\_TASK(startup\_task), NULL)

---->vTaskStartScheduler();

startup\_task

--->main\_task(argument);

App\_HandleHostMessageInput

gAppGapGenericMsg\_c

--->BleApp\_GenericCallback(&pMsg->msgData.genericMsg);

===几个回调函数===

gAppGapAdvertisementMsg\_c

-->pfAdvCallback = BleApp\_AdvertisingCallback

gAppGapScanMsg\_c

-->pfScanCallback = BleApp\_ScanningCallback

gAppGapConnectionMsg\_c

-->pfConnCallback = BleApp\_ConnectionCallback

gAppGattServerMsg\_c

--->pfGattServerCallback = BleApp\_GattServerCallback

gAppGattClientProcedureMsg\_c

--->pfGattClientProcCallback= BleApp\_GattClientCallback

--->pfGattClientNotifCallback

--->pfGattClientIndCallback

--->pfL2caLeCbDataCallback

--->pfL2caLeCbControlCallback

AppIdle\_TaskInit

---->gAppIdleTaskId = OSA\_TaskCreate(OSA\_TASK(App\_Idle\_Task), NULL);

====App\_Idle\_Task

----->App\_Idle();

---->PWR\_CheckIfDeviceCanGoToSleep()

进入低功耗

---->wakeupReason = PWR\_EnterLowPower();

--->FSCI\_SendWakeUpIndication();

按键唤醒

---->KBD\_SwitchPressedOnWakeUp();

---->PWR\_DisallowDeviceToSleep();

/\* Enter MCU Sleep \*/

--->PWR\_EnterSleep();

main\_task

----->hardware\_init();

/\* Framework init \*/

----->MEM\_Init();

----->TMR\_Init();

---->LED\_Init();

---->SecLib\_Init();

---->RNG\_Init();

---->RNG\_GetRandomNo((uint32\_t\*)(&(pseudoRNGSeed[0])));

----->KBD\_Init(App\_KeyboardCallBack);

---->NvModuleInit();

---->AppIdle\_TaskInit();

/\* Initialize peripheral drivers

---->BleApp\_Init();

--->mAppEvent = OSA\_EventCreate(TRUE);

--->MSG\_InitQueue(&mHostAppInputQueue);

--->MSG\_InitQueue(&mAppCbInputQueue);

--->Ble\_Initialize(App\_GenericCallback)

--->App\_Thread( param );

BleApp\_Init//不能的功能实现不一样

---> BOARD\_InitAdc();

--->SerialManager\_Init();

-->Serial\_InitInterface(&gAppSerMgrIf, APP\_SERIAL\_INTERFACE\_TYPE, APP\_SERIAL\_INTERFACE\_INSTANCE);

---->Serial\_SetBaudRate(gAppSerMgrIf, gUARTBaudRate115200\_c);

--->Serial\_SetRxCallBack(gAppSerMgrIf, Uart\_RxCallBack, NULL);

===Uart\_RxCallBack===

--->App\_PostCallbackMessage(BleApp\_FlushUartStream, NULL);

--->MSG\_Queue(&mAppCbInputQueue, pMsgIn);

--->OSA\_EventSet(mAppEvent, gAppEvtAppCallback\_c);

App\_Thread

--->OSA\_EventWait(mAppEvent, osaEventFlagsAll\_c, FALSE, osaWaitForever\_c , &event);

==gAppEvtMsgFromHostStack\_c===

--->pMsgIn = MSG\_DeQueue(&mHostAppInputQueue);

--->App\_HandleHostMessageInput(pMsgIn);

==gAppEvtAppCallback\_c==

--->pMsgIn = MSG\_DeQueue(&mAppCbInputQueue);

---->pMsgIn->handler (pMsgIn->param);

BleApp\_GenericCallback

--->BleConnManager\_GenericEvent(pGenericEvent);

--->BleApp\_Config();

--->BleConnManager\_GapDualRoleConfig();

--->App\_RegisterGattServerCallback(BleApp\_GattServerCallback);

--->App\_RegisterGattClientProcedureCallback(BleApp\_GattClientCallback);

--->GattServer\_RegisterHandlesForWriteNotifications(NumberOfElements

--->BleServDisc\_RegisterCallback(BleApp\_ServiceDiscoveryCallback);

--->App\_StartAdvertising(BleApp\_AdvertisingCallback, BleApp\_ConnectionCallback);

客制化profile流程

Adding a New Service to the GATT Database

frdm-kw40z\_demo/common/gatt\_uuid128.h:30

//定义Potentiometer Service

UUID128(uuid\_service\_potentiometer, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 0x04, 0x56, 0xFF, 0x02)

//定义Potentiometer Characteristic

UUID128(uuid\_characteristic\_potentiometer\_relative\_value, 0xE0, 0x1C, 0x4B, 0x5E, 0x1E, 0xEB, 0xA1, 0x5C, 0xEE, 0xF4, 0x5E, 0xBA, 0x04, 0x57, 0xFF, 0x02)

=========

服务添加到added to the GATT database

frdm-kw40z\_demo/common/gatt\_db.h

PRIMARY\_SERVICE\_UUID128(service\_potentiometer, uuid\_service\_potentiometer)

CHARACTERISTIC\_UUID128(char\_potentiometer\_relative\_value, uuid\_characteristic\_potentiometer\_relative\_value, (gGattCharPropRead\_c | gGattCharPropNotify\_c))

VALUE\_UUID128(value\_potentiometer\_relative\_value, uuid\_characteristic\_potentiometer\_relative\_value, (gPermissionFlagReadable\_c ), 1, 0x00)

CCCD(cccd\_potentiometer)

DESCRIPTOR(cpfd\_potentiometer, gBleSig\_CharPresFormatDescriptor\_d, (gPermissionFlagReadable\_c), 7, gCpfdUnsigned8BitInteger, 0x00,

0xAD/\*Unit precentage UUID in Little Endian (Lower byte)\*/,

0x27/\*Unit precentage UUID in Little Endian (Higher byte)\*/,

0x01, 0x00, 0x00)

Implementing Drivers for New Service

在gatt\_db.h中添加的新的服务，需添加俩个文件

bluetooth/profiles/frdm-kw40z\_demo/potentiometer\_interface.h

/\*! Potentiometer Service - Configuration \*/服务配置

typedef struct psConfig\_tag

{

uint16\_t serviceHandle; /\*!<Service handle \*/

uint8\_t potentiometerValue; /\*!<Input report field \*/

} psConfig\_t;

bleResult\_t Ps\_Start(psConfig\_t \*pServiceConfig);

bleResult\_t Ps\_Stop(psConfig\_t \*pServiceConfig);

bleResult\_t Ps\_Subscribe(deviceId\_t clientDeviceId);//订阅

bleResult\_t Ps\_Unsubscribe();//取消订阅

bleResult\_t Ps\_RecordPotentiometerMeasurement (uint16\_t serviceHandle, uint8\_t newPotentiometerValue);//获取value

bluetooth/profiles/frdm-kw40z\_demo/potentiometer\_service.c

/\*! Potentiometer Service - Subscribed Client\*/

static deviceId\_t mPs\_SubscribedClientId;

service\_potentiometer

Gap\_RegisterDeviceSecurityRequirements(&deviceSecurityRequirements);

-->serviceSecurity

-->.serviceHandle = service\_potentiometer

-->PRIMARY\_SERVICE\_UUID128(service\_potentiometer, uuid\_service\_potentiometer)

-->UUID128(uuid\_service\_potentiometer, 0xE0, 0x1C,xxx,xxx)

source/frdm-kw40z\_demo/common/app.c

#include "potentiometer\_driver.h"

BleApp\_GenericCallback(&pMsg->msgData.genericMsg);

(uint16\_t\*)&notifiableHandleArray[0]);

==pGenericEvent->eventType==

==gInitializationComplete\_c==//蓝牙协议栈初始化完成触发

--->BleApp\_Config();

注册相应的回调

App\_RegisterGattServerCallback(BleApp\_GattServerCallback);

GattServer\_RegisterHandlesForWriteNotifications(notifiableHandleCount,

---->Ps\_Start(&psServiceConfig);

==psServiceConfig = {service\_potentiometer, 0};

--->Ps\_RecordPotentiometerMeasurement (pServiceConfig->serviceHandle,

pServiceConfig->potentiometerValue);

main\_task

--->Ble\_Initialize

--->Ble\_HostInitialize(gapGenericCallback,

(hciHostToControllerInterface\_t) Hcit\_SendPacket);

gAppGapGenericMsg\_c

App\_GenericCallback//初始化完成触发

pMsgIn->msgType = gAppGapGenericMsg\_c

FLib\_MemCpy(&pMsgIn->msgData.genericMsg, pGenericEvent, sizeof(gapGenericEvent\_t));

App\_Thread 消息循环处理

---->App\_HandleHostMessageInput(pMsgIn);

App\_RegisterGattServerCallback

==pfGattServerCallback=BleApp\_GattServerCallback

-->GattServer\_RegisterCallback(App\_GattServerCallback);

====================================================

Client端

从server端获取data

BleApp\_GattNotificationCallback

gScanStateChanged\_c

-->Gap\_Connect(&gConnReqParams, BleApp\_ConnectionCallback);

==gConnEvtConnected\_c==

---->shell\_write("\r\nConnected!\r\n");

-->BleApp\_StateMachineHandler(mPeerInformation.deviceId, mAppEvt\_PeerConnected\_c);

BleApp\_ScanningCallback

==gDeviceScanned\_c==

-->CheckScanEvent(&pScanningEvent->eventData.scannedDevice);

gConnReqParams.peerAddress

-->foundMatch = MatchDataInAdvElementList(&adElement, &uuid\_service\_humidity, 16);

-->Gap\_StopScanning();//停止扫描

====Found device:

KeyScan

--->mpfKeyFunction(keyBase + pressedKey);

--->App\_KeyboardCallBack

--->BleApp\_HandleKeys(events);

--->BleApp\_Start

--->App\_StartScanning(&scanParams, BleApp\_ScanningCallback);

--->pfScanCallback = BleApp\_ScanningCallback;

//设置参数和扫描

-->Gap\_StartScanning(pScanningParameters, App\_ScanningCallback);

App\_HandleHostMessageInput

-->gAppGapScanMsg\_c

pfScanCallback(&pMsg->msgData.scanMsg);

App\_ScanningCallback//扫描成功的回调gatt

--->pMsgIn->msgType = gAppGapScanMsg\_c;

--->pScanningEvent->eventType=pScanningEvent->eventType

BleApp\_GattNotificationCallback

--->BleApp\_PrintHumidity(\*(uint16\_t\*)aValue);

-->shell\_write("Humidity: ");

App\_RegisterGattClientNotificationCallback(BleApp\_GattNotificationCallback);

===pfGattClientNotifCallback = callback;

-->GattClient\_RegisterNotificationCallback(App\_GattClientNotificationCallback);

App\_HandleHostMessageInput

case gAppGattClientNotificationMsg\_c:

--->pfGattClientNotifCallback

App\_GattClientNotificationCallback//接受回调

pMsgIn->msgType = gAppGattClientNotificationMsg\_c;

FLib\_MemCpy(pMsgIn->msgData.gattClientNotifIndMsg.aValue, aValue, valueLength);

=======================================

Server

BleApp\_GattServerCallback

--->value\_accelerometer\_scale

---->gEvtAttributeWritten\_c

----->//Get written value

---->//Call Scale update function

---->//Send response to client

//需返回给client

-->BleApp\_SendAttWriteResponse(&deviceId, pServerEvent, &result);

无cccd

value\_accelerometer\_scale, uuid\_characteristic\_accelerometer\_scale

nocccd

notifiableHandleArray[] ={

value\_led\_control,

value\_buzzer,

value\_accelerometer\_scale,

value\_controller\_command,

value\_controller\_configuration

}

--->GattServer\_RegisterHandlesForWriteNotifications(notifiableHandleCount, (uint16\_t\*)&notifiableHandleArray[0]);

处理的类型：

# **Handling Notifications and Write Requests**

<https://community.nxp.com/docs/DOC-332703>

处理类型：

## 1：**Handling Write Requests**

写请求函数必须要系统初始化时先注册

写的值并不会自动的存到服务器

必须要调用：GattServer\_RegisterHandlesForWriteNotifications