1 在 gatt_uuid128.h 自定义 UUID，使用宏定义 UUID128（服务名/特征名，128 位 uuid）
UUID128(uuid_service_adcTst,0x01,0x03,0x07,0x09,0x0b,0x0d,0x0f,0x02,0x03,0x01,0x04,0x43,0x23,0x32,0x33,0x78)

UUID128(uuid_characteristic_adcTst,0x01,0x03,0x07,0x0a,0x0b,0x0d,0x0f,0x02,0x04,0x01,0x04,0x93,0x23,0x32,0x33,0x79)
名字+16 字节 uuid

2 gatt_db.h 定义服务

PRIMARY_SERVICE_UUID128(service_device_adc, uuid_service_adcTst)
CHARACTERISTIC_UUID128(char_adc_value,uuid_characteristic_adcTst,(gGattCharPropRead_c | gGattCharPropNotify_c | gGattCharPropWrite_c))
    VALUE_UUID128(value_adc_value, uuid_characteristic_adcTst, gPermissionFlagReadable_c | gPermissionFlagWritable_c ,1, 0x00)
    CCCD(cccd_adc)

3 在 profiles 下创建一个新文件夹，放自定义服务，在文件夹下放两个文件，xxx_interface.h 和 xxx_services.c

4 在 interface.h 自定义数据结构，类似
```
typedef struct psConfig_tag
{
    uint16_t     serviceHandle;             /*!<Service handle */
    uint8_t      potentiometerValue;        /*!<Input report field */
} psConfig_t;
```
声明两个开启服务的函数，类似
```
bleResult_t Ps_Start(psConfig_t *pServiceConfig);
bleResult_t Ps_Stop(psConfig_t *pServiceConfig);
```
两个注册客户端注册服务，类似
```
bleResult_t Ps_Subscribe(deviceId_t clientDeviceId);
bleResult_t Ps_Unsubscribe();
```
用于读写更新特征的函数，类似
```
bleResult_t Ps_RecordPotentiometerMeasurement (uint16_t
serviceHandle, uint8_t newPotentiometerValue);
```

5 service.c 要实现
一个静态变量用于存储 ID
```
static deviceId_t mPs_SubscribedClientId;
```

实现以上函数
```
bleResult_t Ps_Start (psConfig_t *pServiceConfig)
{
    /* Clear subscibed clien ID (if any) */
    mPs_SubscribedClientId = gInvalidDeviceId_c;

    /* Set the initial value defined in pServiceConfig to the characteristic values */
    return Ps_RecordPotentiometerMeasurement
(pServiceConfig->serviceHandle,
                                                    pServiceConfig-
>potentiometerValue);
}

bleResult_t Ps_Stop (psConfig_t *pServiceConfig)
{
  /* Unsubscribe current client */
    return Ps_Unsubscribe();
}

bleResult_t Ps_Subscribe(deviceId_t deviceId)
{
   /* Subscribe a new client to this service */
    mPs_SubscribedClientId = deviceId;

    return gBleSuccess_c;
}

bleResult_t Ps_Unsubscribe()
{
   /* Clear current subscribed client ID */
    mPs_SubscribedClientId = gInvalidDeviceId_c;
    return gBleSuccess_c;
}
```

这函数的 特征 uuid 变量是定义在 uuid128 里的，但是我实验时候发现，直接填进函数里会报未定义错误，后来使用了 extern 解决了问题

```
bleResult_t Ps_RecordPotentiometerMeasurement (uint16_t
serviceHandle, uint8_t newPotentiometerValue)
{
    uint16_t  handle;
    bleResult_t result;

    /* Get handle of Potentiometer characteristic */
    result = GattDb_FindCharValueHandleInService(serviceHandle,
        gBleUuidType128_c,
(bleUuid_t*)&potentiometerCharacteristicUuid128, &handle);

```

```
    if (result != gBleSuccess_c)
        return result;

    /* Update characteristic value */
    result = GattDb_WriteAttribute(handle, sizeof(uint8_t), 
(uint8_t*)&newPotentiometerValue);

    if (result != gBleSuccess_c)
        return result;

    Ps_SendPotentiometerMeasurementNotification(handle);

    return gBleSuccess_c;
}
```

实现发送提醒的函数

```
static void Ps_SendPotentiometerMeasurementNotification
(
  uint16_t handle
)
{
    uint16_t  hCccd;
    bool_t isNotificationActive;

    /* Get handle of CCCD */
    if (GattDb_FindCccdHandleForCharValueHandle(handle, &hCccd) != 
gBleSuccess_c)
        return;

    if (gBleSuccess_c == Gap_CheckNotificationStatus
        (mPs_SubscribedClientId, hCccd, &isNotificationActive) 
&&
        TRUE == isNotificationActive)
    {
        GattServer_SendNotification(mPs_SubscribedClientId, handle);
    }
}
```

6 在文件里，放置服务数据的地方，定义自定义的数据，类似

Static psConfig_t psServiceConfig = {service_potentiometer,0};

这个 service_potentiometer 就是 UUID 定义的主服务

7 在 BleApp_Config() 里开启服务

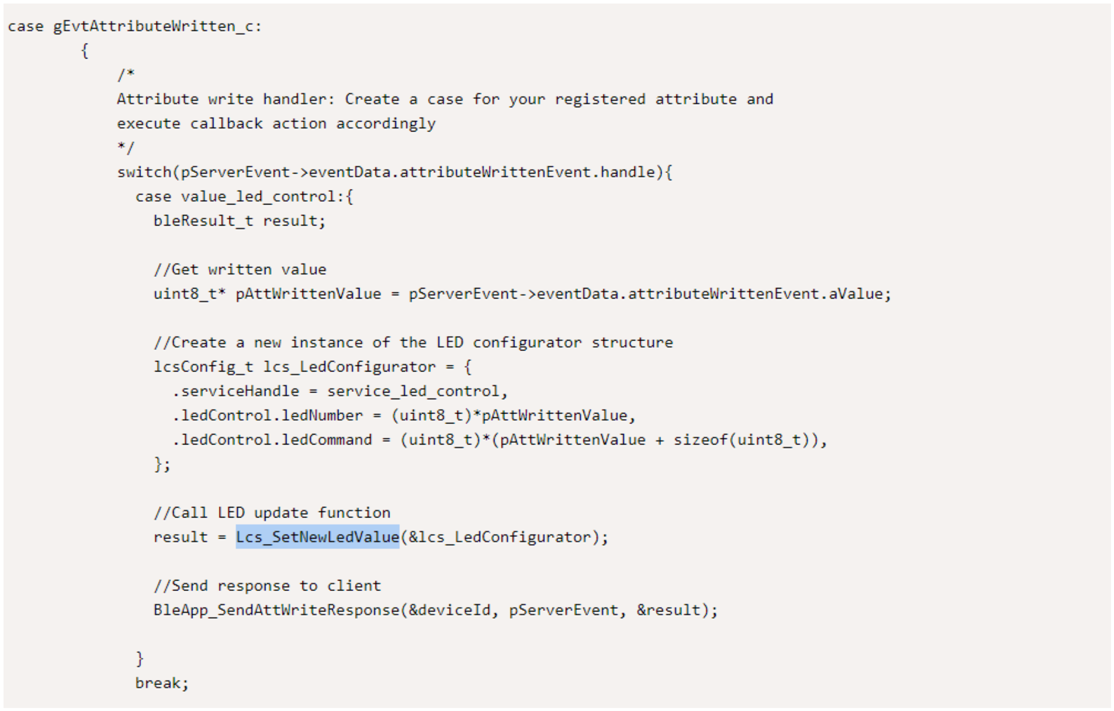
```
Ps_Start(&psServiceConfig);
```

8 在 BleApp_ConnectionCallback，在 gConnEvtConnected_c 这个状态下，调用 Ps_Subscribe(peerDeviceId); 注册 ID，开启对应服务定时器，实现定时器回调函数

9 如果需要处理，Notifications **和写请求**
在 BleApp_GattServerCallback 里的 ccd 写情况下，开启定时器，定时发送数据

```
写请求则要先用 GattServer_RegisterHandlesForWriteNotifications 来注册函数
uint16_t notifiableHandleArray[] = {value_led_control,
value_buzzer, value_accelerometer_scale,
value_controller_command, value_controller_configuration};
    uint8_t notifiableHandleCount =
sizeof(notifiableHandleArray)/2;
    bleResult_t initializationResult =
GattServer_RegisterHandlesForWriteNotifications(notifiableHandleCount,
(uint16_t*)&notifiableHandleArray[0]);
```

在 BleApp_GattServerCallback 里的属性写情况下，
同时要用 GattServer_RegisterHandlesForWriteNotifications 来注册句柄，cpHandles 放了 VALUE 值的句柄，需要在里面添加 VALUE 对应的第一个名字


```
case gEvtAttributeWritten_c:
        {
            /*
            Attribute write handler: Create a case for your registered attribute and
            execute callback action accordingly
            */
            switch(pServerEvent->eventData.attributeWrittenEvent.handle){
              case value_led_control:{
                bleResult_t result;

                //Get written value
                uint8_t* pAttWrittenValue = pServerEvent->eventData.attributeWrittenEvent.aValue;

                //Create a new instance of the LED configurator structure
                lcsConfig_t lcs_LedConfigurator = {
                  .serviceHandle = service_led_control,
                  .ledControl.ledNumber = (uint8_t)*pAttWrittenValue,
                  .ledControl.ledCommand = (uint8_t)*(pAttWrittenValue + sizeof(uint8_t)),
                };

                //Call LED update function
                result = Lcs_SetNewLedValue(&lcs_LedConfigurator);

                //Send response to client
                BleApp_SendAttWriteResponse(&deviceId, pServerEvent, &result);

              }
              break;
```

实现 sendAttWrite…

```
static void BleApp_SendAttWriteResponse (deviceId_t* pDeviceId,
gattServerEvent_t* pGattServerEvent, bleResult_t* pResult){
  attErrorCode_t attErrorCode;
  // Determine response to send (OK or Error)
  if(*pResult == gBleSuccess_c)
    attErrorCode = gAttErrCodeNoError_c;
  else{
    attErrorCode = (attErrorCode_t)(*pResult & 0x00FF);
  }
  // Send response to client
  GattServer_SendAttributeWrittenStatus(*pDeviceId,
pGattServerEvent->eventData.attributeWrittenEvent.handle,
attErrorCode);
}
```

按下 notify 按钮则会触发 cccd 写事件 gEvtCharacteristicCccdWritten_c

启用配对验证在 app_preinclude.h 定义为 1

```
/*! Enable/disable use of bonding capability */
#define gAppUseBonding_d      1

/*! Enable/disable use of pairing procedure */
#define gAppUsePairing_d      1
```

密钥定义

```
#define gPasskeyValue_c                          999999
```