谷雨物联网
iotxx.com

BLE技术揭秘

谷雨文档中心
http://doc.iotxx.com

2024-01-30

BLE技术揭秘

BLE是低功耗蓝牙的英文缩写（Bluetooth Low Energy），是蓝牙4.0版本起开始支持的新的、低功耗版本的蓝牙技术规范。

蓝牙技术联盟（Bluetooth SIG）在2010年发布了跨时代的蓝牙4.0，它并不是蓝牙3.0的简单升级版本，而是全新的技术架构，蓝牙4.0版本分两种模式：单模蓝牙和双模蓝牙。

常见的蓝牙音箱，是典型的双模蓝牙，它需要传输大量的音频数据。而小米手环，蓝牙温度计则属于单模蓝牙。行业里一般不讲单模蓝牙，而是统一称为低功耗蓝牙。

如今，蓝牙5.0已经发布和应用，4倍通信速度、2倍的通信距离以及Mesh组网特性，将使蓝牙成为物联网领域的重要的技术之一。

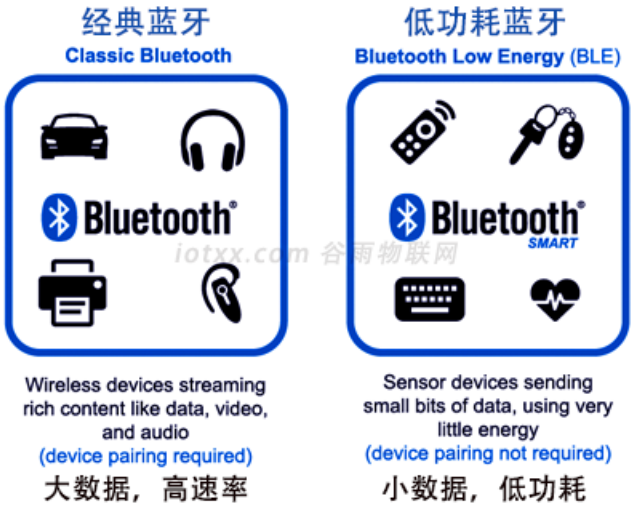
本文我们将由表及里，由浅入深，全方位的揭秘低功耗蓝牙技术。

1 蓝牙简介

蓝牙是一种近距离无线通信技术，运行在2.4GHz免费频段，目前已大量应用于各种移动终端，物联网，健康医疗，智能家居等行业。蓝牙4.0以后的版本分为两种模式，单模蓝牙和双模蓝牙。

- 单模蓝牙，即低功耗蓝牙模式，是蓝牙4.0中的重点技术，低功耗，快连接，长距离。
- 双模蓝牙，支持低功耗蓝牙的同时还兼容经典蓝牙，经典蓝牙的特点是大数据高速率，例如音频、视频等数据传输。

如下图所示，双模蓝牙具有下图所有的特点，而单模蓝牙仅如图右侧所示。



经典蓝牙支持音频（HFP/HSP，A2DP）和数据（SPP，HID等）两大类协议，在音箱，耳机，汽车电子及传统数传行业，由于苹果对经典蓝牙数据传输接口有限制（需要过MFI认证），加上功耗偏大，因此在目前移动互联应用中慢慢地被边缘化。因此低功耗蓝牙顺势而出，由于可支持苹果4S以上及安卓4.3系统以上的数据传输，且功耗极低，目前正在被越来越多的移动互联设备所采用，但低功耗蓝牙不支持音频协议，并且受数据传输速度限制，其应用也被限制在小数据传输行业。而蓝牙双模则是综合了两者的优缺点，既可以支持音频传输，同样可支持数据传输，并且兼容性也是两者之和，在对功耗要求不苛刻的情况下，是比较理想的选择。

2 BLE特点

低功耗蓝牙瞄准多个市场，特别是移动智能终端，智能家居，互联设备等领域，主要特点包括：

- 低功耗，使用纽扣电池就可以运行数月至数年。
- 快连接，毫秒级的连接速度，传统蓝牙甚至长达数分钟。
- 远距离，长达数百米的通信距离，而传统蓝牙通常10米左右。

蓝牙联盟沿用经典蓝牙的规范内容，为低功耗蓝牙定义了一些标准`Profile`，Profile理解为数据规范，只要遵守该规范，任意厂家的蓝牙设备，均可以相互连接与通信，例如无线蓝牙键盘鼠标，无论是安卓或是iOS还是Windows，均是即插即用，这便是“标准”的力量。低功耗蓝牙支持的标准Profile有：

- HID，用于无线鼠标，键盘或其他遥控设备。
- BatteryServices，电池状态服务，用于告知电池电量状态。
- HRP，心率计Profile，用于心率采集。等等。

另外，低功耗蓝牙还可以自定义Profile，伴随着智能手机的发展和普及，低功耗蓝牙的这个特性得到了发扬光大，同时也拓宽了低功耗蓝牙的应用领域。例如，可以自定义一个开关量的Profile，数据01表示开灯，数据00表示关灯，然后手机发送数据01和00就可以控制灯的亮和灭。类似的应用案例有很多，下面总结应用特点

- 支持自定义Profile，可以收发任意格式的数据，如01和00
- 支持自定义设备，支持任意设备的连接和通信，例如智能蓝牙插座等。

提示：低功耗蓝牙的Profile均基于GATT（通用属性规范，后面会详解）之上，如HID over GATT。也就是说，经典蓝牙中的HID规范与低功耗蓝牙中的HID规范用的是两个不同的通道。

3 BLE工作流程

本节我们介绍低功耗蓝牙的基本行为状态和主从机交互过程，为后面的低功耗蓝牙协议的学习准备基础。

3.1 角色

BLE设备角色主要分为两种角色，主机（Master或Central）和从机（Peripheral），当主机和从机建立连接之后才能相互收发数据

- 主机，主机可以发起对从机的扫描连接。例如手机，通常作为BLE的主机设备
- 从机，从机只能广播并等待主机的连接。例如智能手环，是作为BLE的从机设备

另外还有观察者（Observer）和广播者（Broadcaster），这两种角色不常使用，但也十分有用，例如iBeacon，就可以使用广播者角色来做，只需要广播特定内容即可。

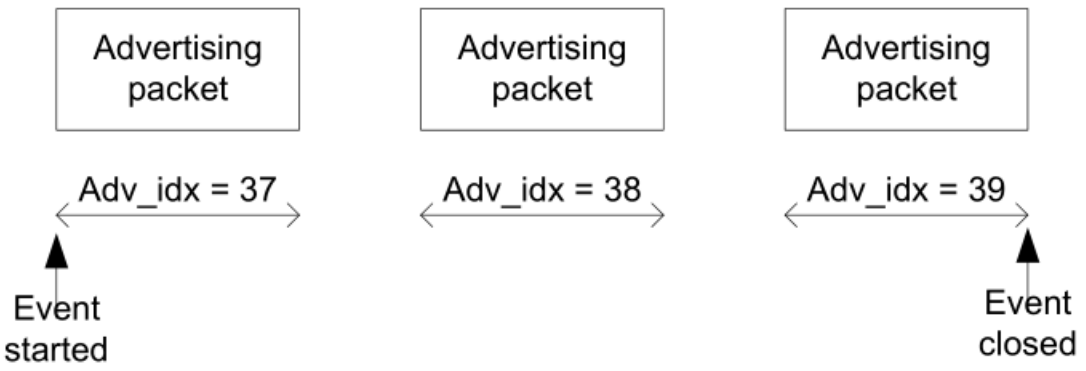
- 观察者，观察者角色监听空中的广播事件，和主机唯一的区别是不能发起连接，只能持续扫描从机。
- 广播者，广播者可以持续广播信息，和从机的唯一区别是不能被主机连接，只能广播数据

蓝牙协议栈没有限制设备的角色范围，同一个BLE设备，可以作为主机，也可以作为从机，我们称之为主从一体，主从一体的好处是，每个BLE设备都是对等的，可以发起连接，也可以被别人连接，更加实用。

3.2 广播

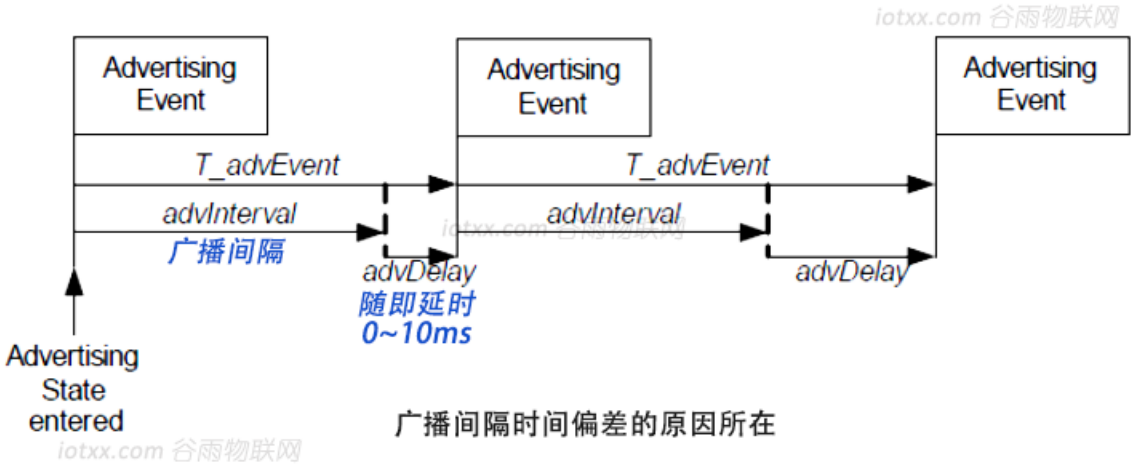
广播是指从机每经过一个时间间隔发送一次广播数据包，这个时间间隔称为广播间隔，这个广播动作叫做广播事件，只有当从机处于广播状态时，主机才能发现该从机。

在每个广播事件中，广播包会分别在37,38和39三个信道上依次广播，如下图所示。



广播时间间隔的范围是从20ms到10.24s，广播间隔影响建立连接的时间。广播间隔越大，连接的时间越长。

另外BLE链路层会在两个广播事件之间添加一个0~10ms的随机延时，保证多个设备广播时，不会一直碰撞广播。也就是说，设置100ms的广播间隔，实际上两次广播事件的时间间隔可能是100~110ms之间的任意时间。



广播间隔时间偏差的原因所在

广播数据包最多能携带31个字节的数据，一般包含可读的设备名称，设备是否可连接等信息。

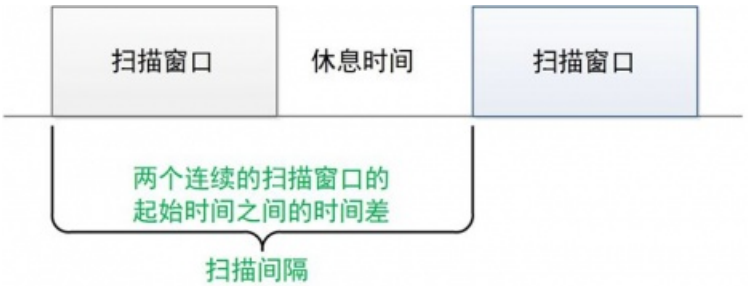
当主机收到从机广播的数据包后，它可以再发送获取更多数据包的请求，这个时候从机将广播扫描回应数据包，扫描回应数据包和广播包一样，可以携带31个字节的数据。

> 提示：蓝牙4.x，广播有效载荷最多是31个字节。而在蓝牙5.0中，通过添加额外的广播信道和新的广播PDU，将有效载荷增加到了255个字节

3.3 扫描

扫描是主机监听从机广播数据包和发送扫描请求的过程，主机通过扫描，可以获取到从机的广播包以及扫描回应数据包，主机可以对已扫描到的从机设备发起连接请求，从而连接从机设备并通信。

扫描动作有两个比较重要的时间参数：扫描窗口和扫描间隔，如果扫描窗口等于扫描间隔，那么主机将一直处于扫描状态之中，持续监听从机广播包。



- 扫描窗口和扫描间隔设置的时间不能大于10.24S。
- 扫描窗口设置的值不能大于扫描间隔的值。
- 如果扫描窗口 = 扫描间隔的话，说明主机一直在进行扫描。

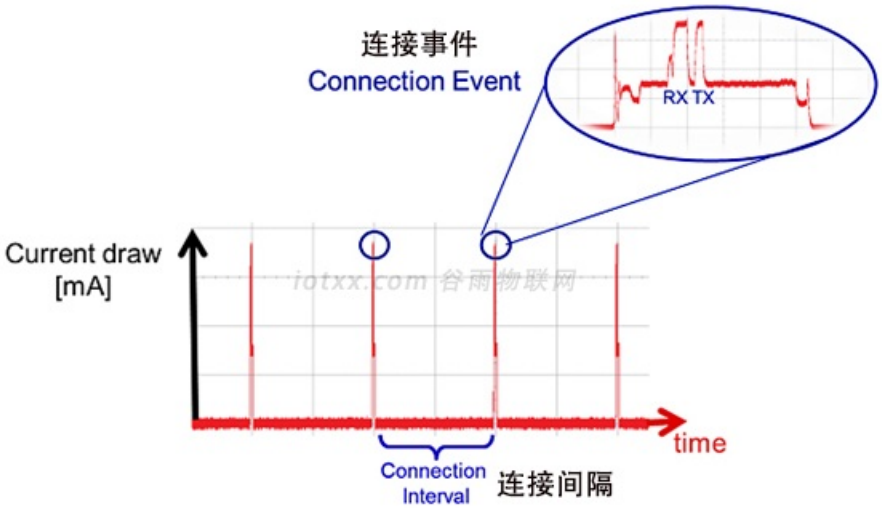
- 被动扫描，主机监听广播信道的数据，当接收到广播包时，协议栈将向上层（也就是应用层，用户可编程）传递广播包。

- 主动扫描，主动扫描除了完成被动扫描的动作外，还会向从机发送一个扫描请求，从机收到该请求时，会再次发送一个称作扫描回应的广播包。

所以，主动扫描比被动扫描，可以多收到扫描回应数据包。

3.4 连接

在BLE连接中，使用跳频方案，两个设备在特定时间、特定频道上彼此发送和接收数据。这些设备稍后在新的通道（协议栈的链路层处理通道切换）上通过这个约定的时间相遇。这次用于收发数据的相遇称为连接事件。如果没有要发送或接收的应用数据，则交换链路层数据来维护连接。两个连接事件之间的时间跨度称为连接间隔，是以1.25 ms为单位，范围从最小值7.5 ms到最大值4.0 s

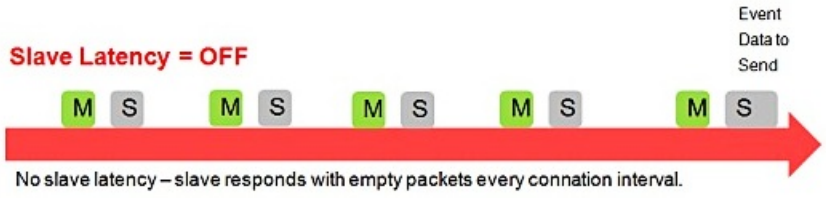


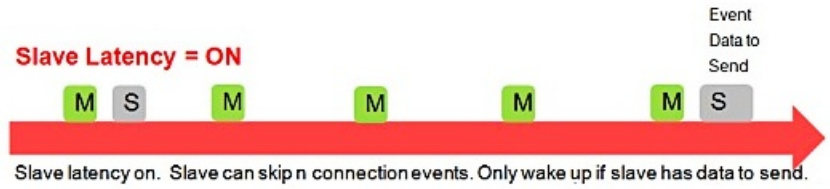
3.4.1 连接参数

Connection Interval连接间隔，两次连接事件之间的时间间隔称为连接间隔。1.25 ms为单位，范围从最小值7.5 ms到最大值4.0 s

Slave Latency从机延迟，如果从机没有要发送的数据，则可以跳过连接事件，继续保持睡眠节省电量。

Supervision Time-out监控超时，是两次成功连接事件之间的最长时间。如果在此时间内没有成功的连接事件，设备将终止连接并返回到未连接状态。该参数值以10 ms为单位，监控超时值可以从最小值10（100 ms）到3200（32.0 s）。超时必须大于有效的连接间隔。





3.4.2 连接参数更新请求

连接参数由主机发起连接的时候提供，如果从机对连接参数有自己的要求，例如要求更低的功耗，或者更高的通信速率等，从机可以向主机发送连接参数更新请求。

从机可以在连接后的任何时候发起连接参数更新请求，但最好不要在主从建立连接后立刻发起，建议延迟5s左右再发送请求。

连接参数更新请求可以修改：Connection Interval连接间隔，Slave Latency从机延迟，Supervision Time-out监控超时。

3.4.3 有效连接间隔

Effective Connection Interval有效连接间隔等于两个连接事件之间的时间跨度，假设从机跳过最大数量的连接事件，且允许从机延迟（如果从机延迟设置为0，则有效连接间隔等于实际连接间隔，）。

从机延迟表示可以跳过的最大事件数。该数字的范围可以从最小值0（意味着不能跳过连接事件）到最大值499。最大值不能使有效连接间隔（见下列公式）大于16秒。间隔可以使用以下公式计算：

Effective Connection Interval = (Connection Interval) × (1 + [Slave Latency])

Consider the following example:

- Connection Interval: 80 (100 ms)
- Slave Latency: 4
- Effective Connection Interval: (100 ms) × (1 + 4) = 500 ms

当没有数据从从机发送到主机时，从机每500ms一个连接事件交互一次。

3.4.4 iOS对连接参数的要求

不同的平台对有连接间隔有着不同的要求，例如iOS系统对ble的连接间隔有着如下的要

求。

- Interval Max * (Slave Latency + 1) <=2s
- Interval Min >=20ms
- Interval Min + 20 ms <= Interval Max
- Slave Latency <= 4
- SupervisionTimeout <= 6 s
- Interval Max * (Slave Latency + 1) * 3 < SupervisionTimeout

3.4.5 连接参数的优化考量

在许多应用中，从机跳过最大连接事件数。选择正确的连接参数组在低功耗蓝牙设备的功率优化中起重要作用。以下列表给出了连接参数设置中权衡的总体概述。

减少连接间隔如下：

- 增加两个设备的功耗
- 增加双向吞吐量
- 减少任一方向发送数据的时间

增加连接间隔如下：

- 降低两个设备的功耗
- 降低双向吞吐量
- 增加任一方向发送数据的时间

减少从机延迟（或将其设置为零）如下：

- 增加外围设备的功耗
- 减少外围设备接收从中央设备发送的数据的时间

增加从机延迟如下：

- 在周边没有数据发送期间，可以降低外设的功耗到主机设备
- 增加外设设备接收从主机设备发送的数据的时间

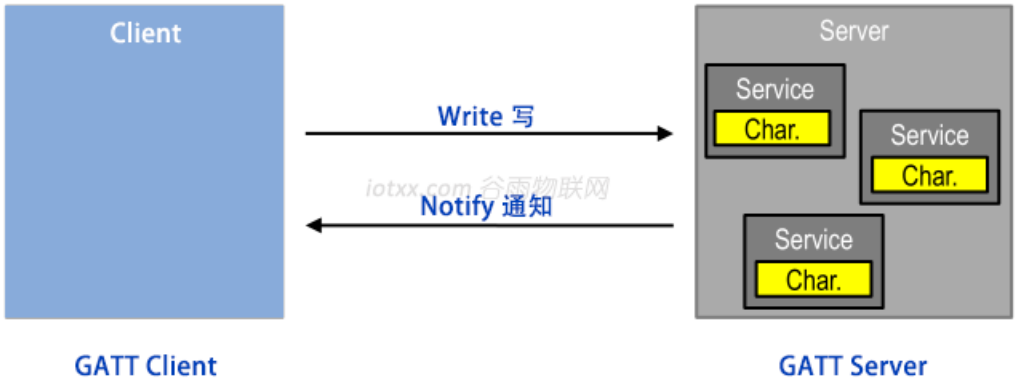
3.5 通信

通俗的说，我们将从机具有的数据或者属性特征，称之为Profile，Profile可翻译为：配置文件。

从机中添加Profile配置文件（定义和存储Profile），作为GATT的Server端，主机作为GATT的Client端。

Profile包含一个或者多个Service，每个Service又包含一个或者多个Characteristic。主机可以发现和获取从机的Service和Characteristic，然后与之通信。Characteristic是主从通信的最小单元。

- 主机可主动向从机Write写入或Read读取数据。
- 从机可主动向主机Notify通知数据。

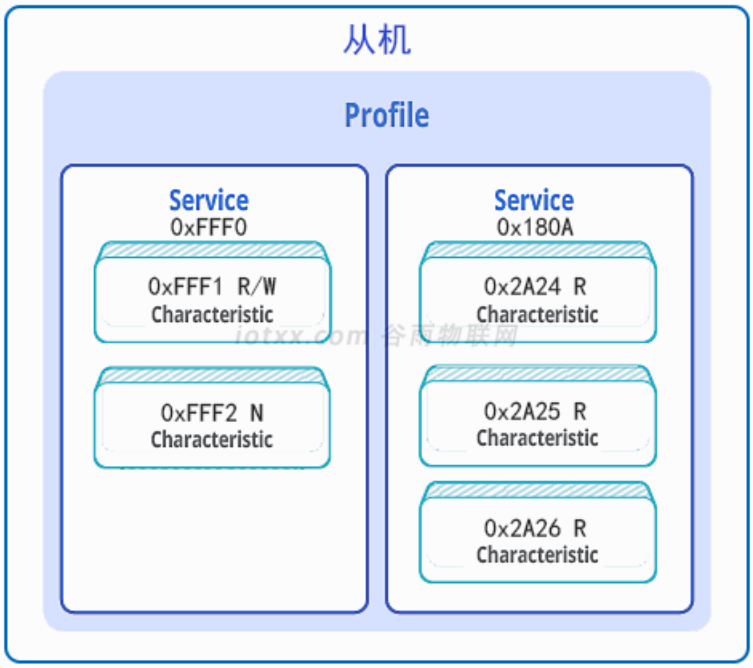


注意，这里引用了`服务 Service` 和 `特征值 Characteristic` 的概念。每个服务和特征值都有自己的唯一标识 `UUID`，标准UUID为128位，蓝牙协议栈中一般采用16位，也就是两个字节的UUID格式。

一个从机设备包括一个或者多个服务；一个服务中又可以包括一条或者多条特征值，每个特征值都有自己的`属性 Property`，属性的取值有：`可读 Read`，`可写 Write` 以及 `通知 Notify`。

- 可读可写的字面意思容易理解，表示该特征值可以被主机读取和写入数据，
- 而通知则表示从机可以主动向主机发送通知数据。这便是主从机之间两个典型的通信方式。

下图是一个典型的从机设备，该从机包含有一个Profile，两个个Service和五个Characteristic。我们先来介绍这些特征值的作用，然后介绍如何通过特征值通信。



服务0x180A

180A是蓝牙协议里标准的服务UUID，用来描述`设备信息 Device Information`，可以通过该服务，来提供从机设备的相关说明，例如硬件版本，软件版本，序列号等信息。这样，主机就可以获取从机的设备信息。上图中我们添加了三个提供具体设备信息的特征值，他们分别是：

- 特征值0x2A24，描述产品型号 `Model Number String`，例如某智能锁的产品型号为：“DSL-C07”。
- 特征值0x2A25，描述产品序列号 `Serial Number String`，例如某智能锁的产品序列号为：“lkjl00161905025002 69”
- 特征值0x2A26，描述产品固件版本号 Firmaware Revision String，例如某智能锁的固件号为：“2.7.2.0”

上述特征值仅有Read属性，因此主机只能读，不能执行写操作。

服务0xFFF0

FFF0是我们自定义的服务UUID，它包含两个特征值，用来发送和接收数据。

- 特征值0xFFF1，自定义的数据发送通道，具有Read和Write属性，主机可以通过该特征值，向从机发送数据，至于发送的数据最大长度，可以在Profile中配置。
- 特征值0xFFF2，自定义的数据接收通道，具有Notify属性，从机可以通过该特征值，主动向主机发送数据。

假设主机写特征值的协议栈函数原型为 `int GATT_WriteCharValue(uuid_t UUID, uint8 *pValue, uint8 len)`

假设从机发送通知的协议栈函数原型为 `int GATT_Notification(uuid_t UUID, uint8 *pValue, uint8 len)`

那么主机向从机发送Hello，可以这样调用协议栈的函数：`GATT_WriteCharValue(0xFFF1,"Hello",5)`

那么从机向主机发送1234，可以这样调用协议栈的函数：`GATT_Notification(0xFFF2,"1234",4)`

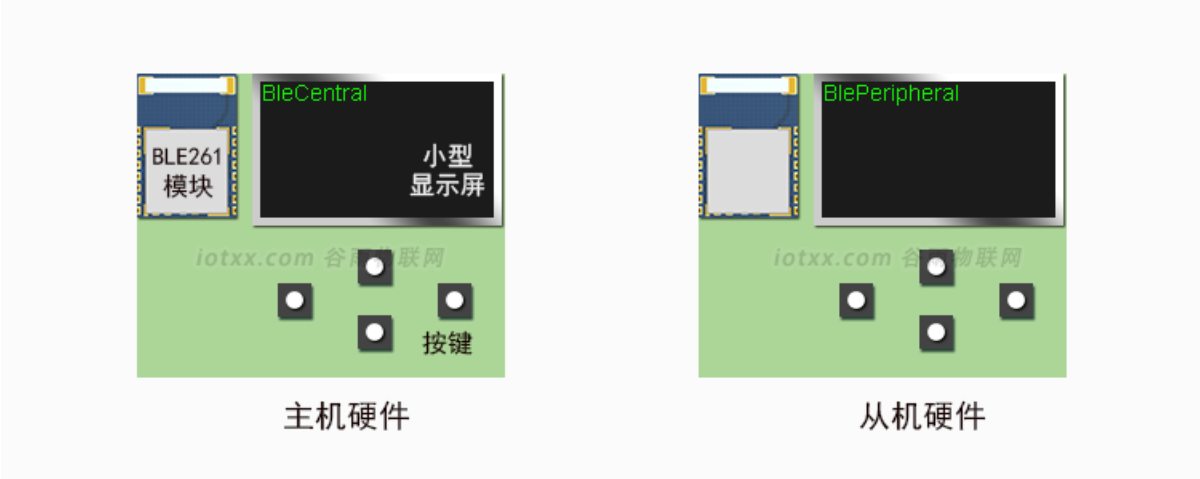
3.6 断开

主机或从机都可以发起断开连接请求，对方会收到该请求，然后断开连接恢复连接前的状态。

3.7 过程演示

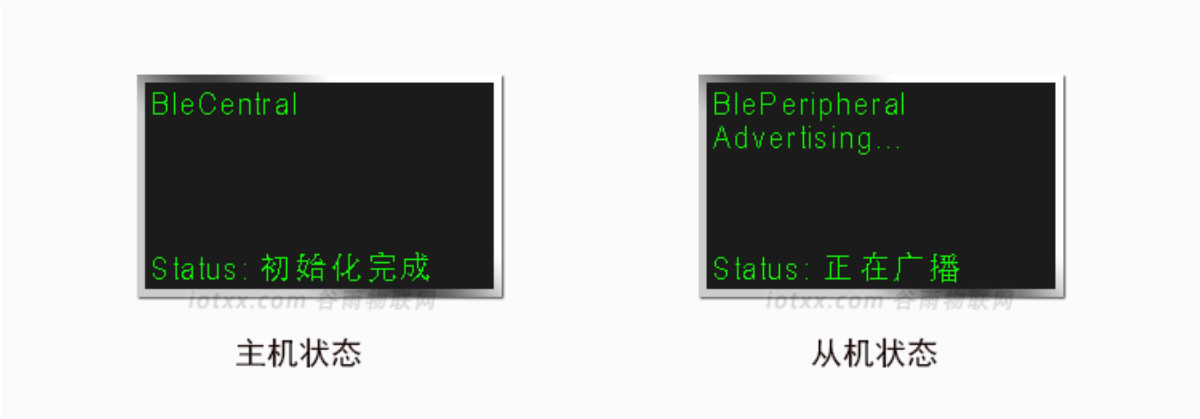
现在我们总结一下BLE的工作流程，使用两个虚拟的BLE硬件来模拟主从机的交互过程。

假设有两个BLE设备，使用的是BLE261低功耗蓝牙模块（假设已经下载了用于交互演示的功能固件），一个是主机，名称为：BleCentral，另一个是从机，名称为：BlePeripheral，如下图所示。



3.7.1 步骤1：上电初始化

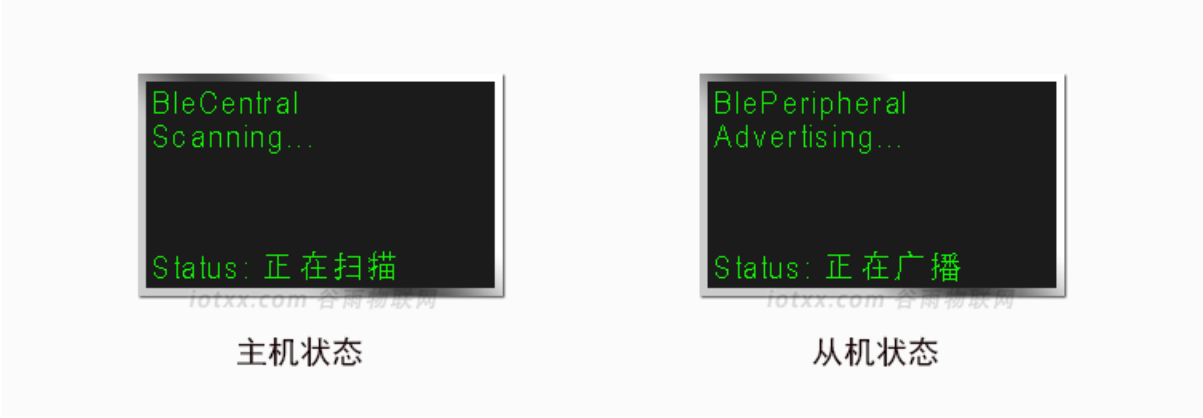
主机、从机上电后（不分先后顺序），首先进行协议栈初始化和相关功能调用，如下图所示。



- 主机设备，主机初始化时，需要设置设备类型，设置用于扫描的相关参数，初始化GATT等协议相关的参数。（下一章节详细介绍何为GATT）
- 从机设备，从机初始化时，需要设置设备名称，广播相关参数，从机Profile等。从机一般会立即开启广播，也可以等待一个事件来触发广播，例如按键触发。

3.7.2 步骤2：主机扫描从机

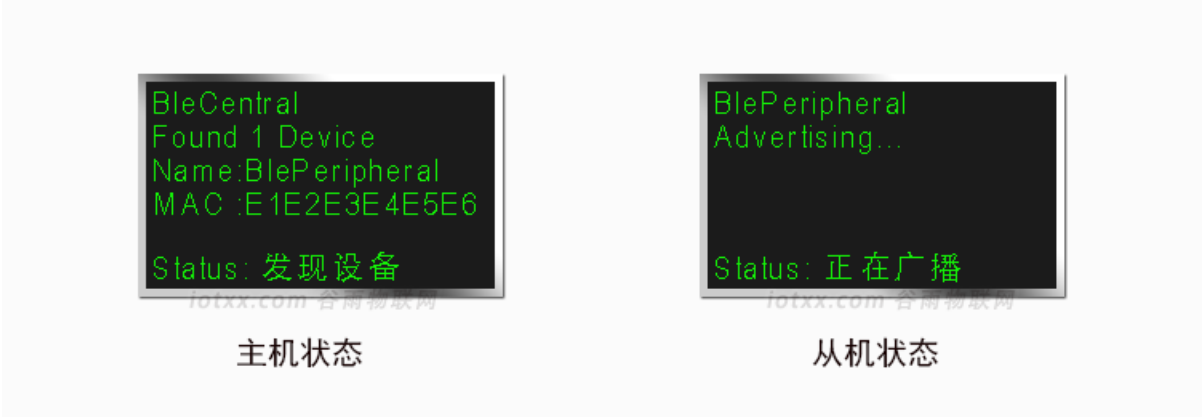
按键按下，触发主机扫描从机，此时，主机显示屏打印Scanning正在扫描。此刻的从机仍然处于广播状态。



3.7.3 步骤3：发现从机设备

当主机扫描到从机时，可以返回已扫描到的从机相关信息，例如可以提取到下图中的从机设备名称，从机MAC地址，从机的RSSI信号值等数据。

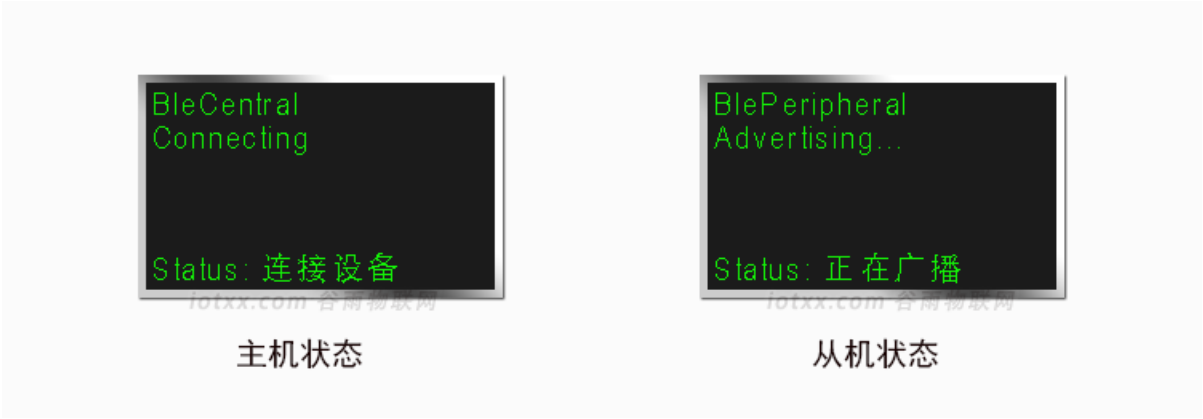
因此，有些应用在从机的广播包或者扫描回应包中添加自定义字段，这样就可以被主机通过扫描的方式拿到数据。



3.7.4 步骤4：发送连接请求

当主机扫描到从机后，通过MAC地址向从机发送连接请求。低功耗蓝牙的连接速度非常快，100ms左右即可成功连接上。如果从机的广播比较大，则会影响连接的速度。

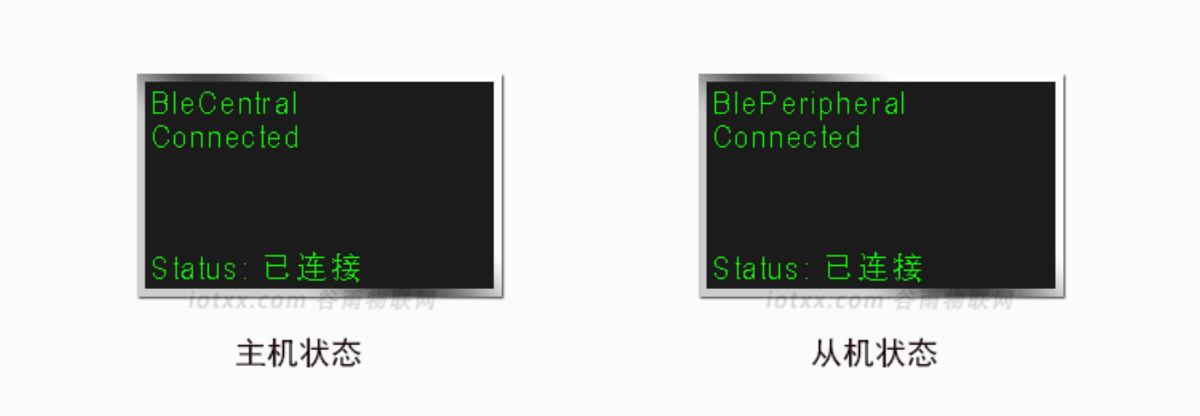
从机在未收到连接请求之前仍然处于自由的广播状态。



3.7.5 步骤5：成功连接从机

当从机收到连接请求后，双方成功建立连接，此时双方的状态均变为已连接状态。

然后主机可以调用协议栈提供的接口函数来获取从机的服务。



3.7.6 步骤6：获取从机服务

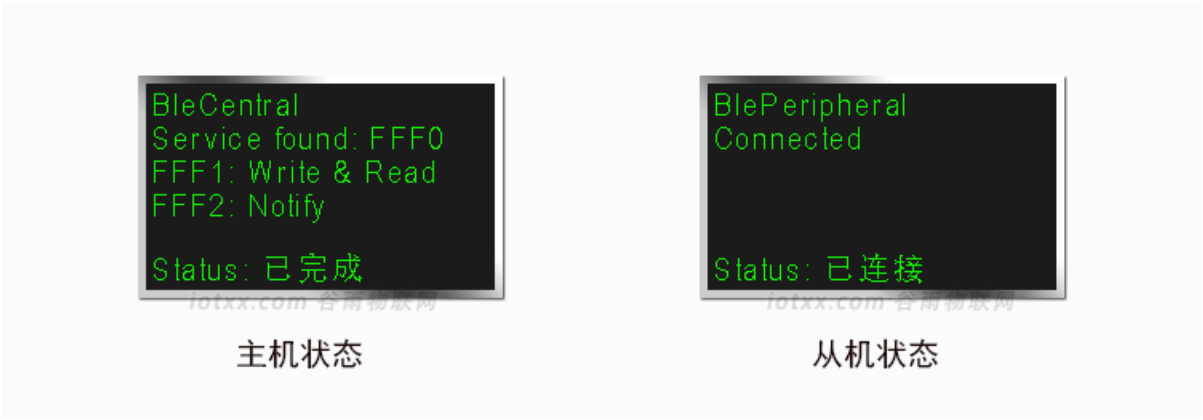
获取从机服务通常是在连接成功后就立即执行的，因为只有获取从机的服务后，才能与其通信。下图是主机向从机发送获取服务的请求。

此刻，从机处于已连接状态。响应服务获取请求是在底层自动完成，上层无需理会。

3.7.7 步骤7：成功获取服务

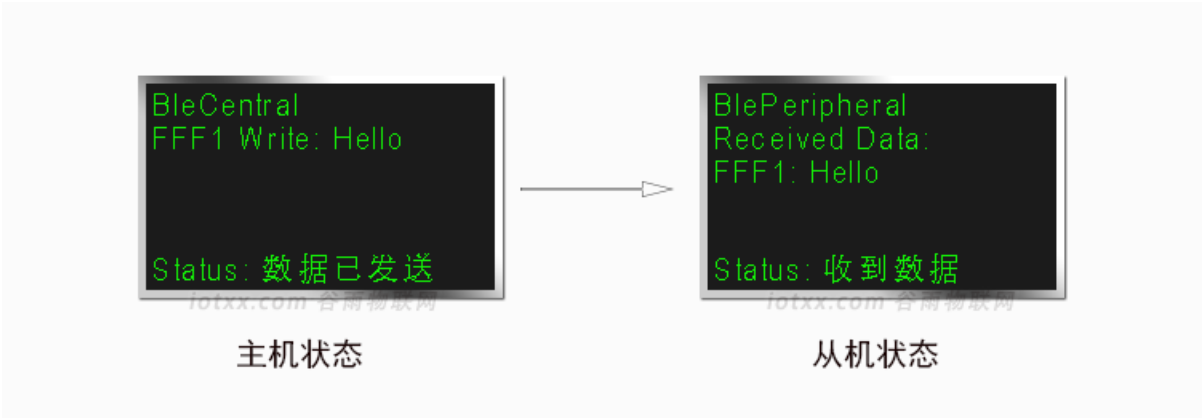
如下图所示，主机成功获取到从机的服务，例如获取到UUID为0xFFF0的Services，该Service有两个特征值，分别是具有读写属性的0xFFF1，以及具有通知属性的0xFFF2。

读写属性是指主机可以读写该特征值的内容。而通知属性是指从机可以通过该特征值向主机发送数据。



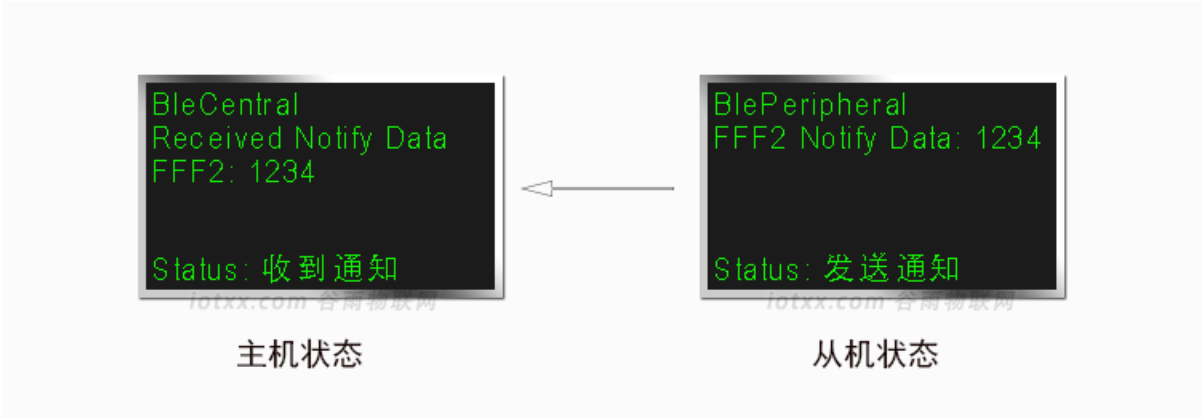
3.7.8 步骤8：主机向从机发送数据

主机通过特征值0xFFF1，主动向从机发送自定义数据Hello，当数据成功发送后，主机状态变为：数据已发送。从机将收到主机发来的数据，从机状态变为收到数据。



3.7.9 步骤9：从机向主机发送数据

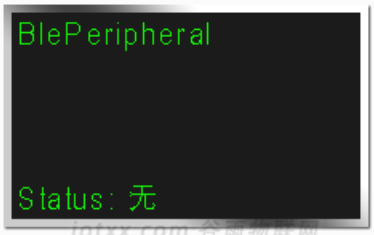
从机可以通过Norify的方式主动向主机发送数据，例如下图，从机通过特征值0xFFF2发送了一条Notify通知，数据内容为：1234



3.7.10 步骤10：发送断开请求

主机和从机任何一方均可以发起断开连接的请求，对方收到后，状态将变为已断开。

主机状态



从机状态

3.7.11 步骤11：成功断开连接

从机收到主机发来的断开请求，此刻状态变为已断开。

主机状态

从机状态

4 BLE协议栈

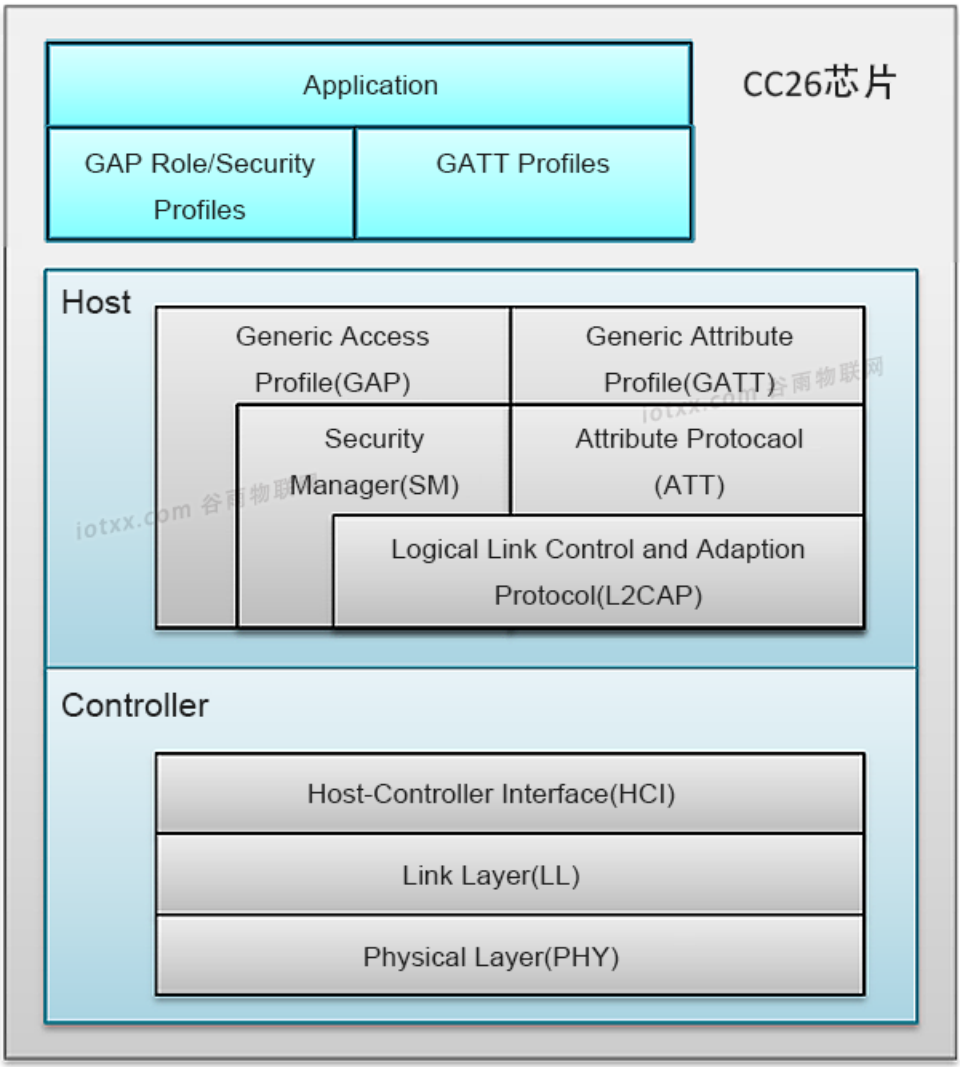
BLE协议栈一般是指芯片厂家，依据 Bluetooth SIG 发布的 `Bluetooth Core Specification` 核心协议的实现的代码固件，并提供函数接口，由芯片内部程序调用，可实现上节BLE工作流程等相关功能。

常见的协议栈有德州仪器 TI 的 `ble-stack` 和 Nordic 的 `SoftDevice`。

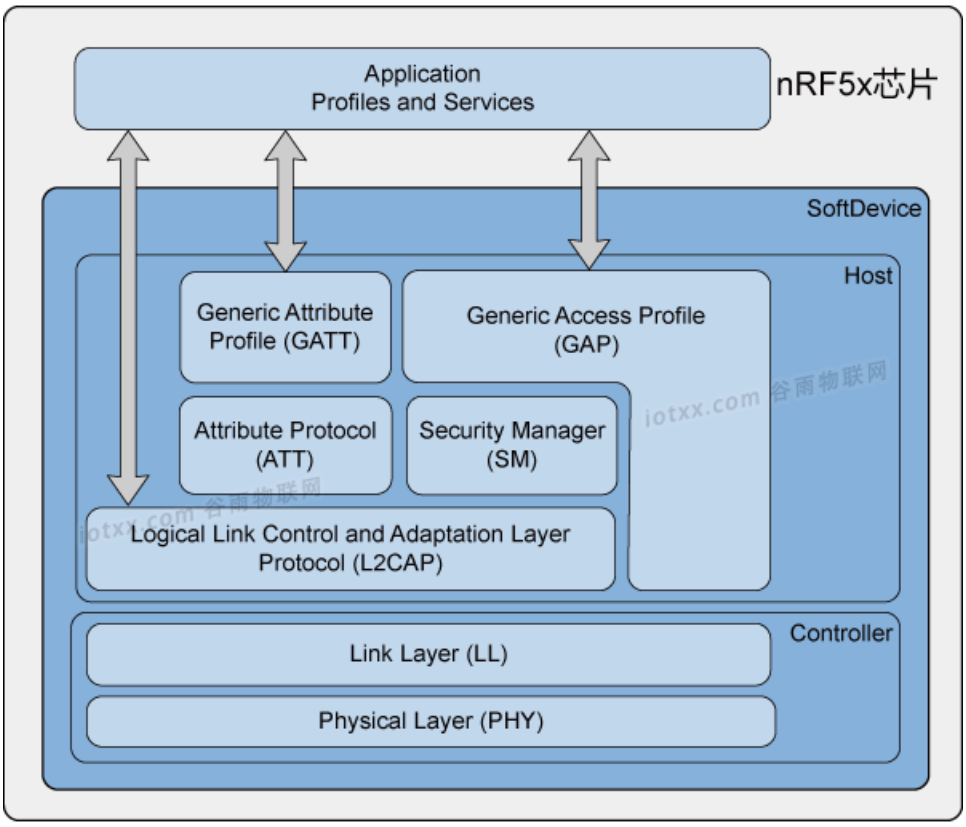
4.1 功能框图

在本节中，我们列举两家典型的蓝牙芯片厂家：TI和Noridc，来深入了解低功耗蓝牙协议栈。

下图是TI的CC26系列芯片协议栈结构图，



下图是Nordic的nRF52系列芯片的协议栈结构图。



4.2 协议栈结构

从上节的两张协议栈功能框图中可以看出，无论是哪个芯片厂商实现的BLE协议栈，其结构都非常的相似，均三个部分：

1. 底层：Controller
2. 中层：Host
3. 顶层：Application

然后每一层又分成若干个子模块。我们现在由下而上，逐层介绍。

> 提示：我们将位于顶层的应用层Application也归到协议栈中描述，其实，应用层Application不属于协议栈，它是用来调用协议栈提供的接口，然后实现蓝牙的功能。

4.2.1 控制器Controller

- `Physical Layer`，简称：`PHY`，物理层。PHY层用来指定BLE所用的无线频段，调制解调方式和方法等。PHY层做得好不好，直接决定整个BLE芯片的功耗，灵敏度以及selectivity等射频指标。
- `Link Layer`，简称：`LL`，链路层。LL层是整个BLE协议栈的核心，也是BLE协议栈的难点和重点。像Nordic的BLE协议栈能同时支持20个link（连接），就是LL层的功劳。LL层要做的事情非常多，比如具体选择哪个射频通道进行通信，怎么识别空中数据包，具体在哪个时间点把数据包发送出去，怎么保证数据的完整性，ACK如何接收，如何进行重传，以及如何对链路进行管理和控制等等。LL层只负责把数据发出去或者收回来，对数据进行怎样的解析则交给上面的GAP或者ATT
- `Host Controller Interface`，简称：`HCI`。协议栈应用开发中，我们会经常看到HCI的身影，它对上Host提供Controller的功能接口，所以称作Host Controller Interface。

4.2.2 主控Host

- `Logical Link Control Adaptation Protocol`，简称：`L2CAP`。L2CAP对LL进行了一次简单封装，LL只关心传输的数据本身，L2CAP就要区分是加密通道还是普通通道，同时还要对连接间隔进行管理。
- `Attribute Protocol`，简称：`ATT`。ATT层用来定义用户命令及命令操作的数据，比如读取某个数据或者写某个数据。BLE协议栈中，开发者接触最多的就是ATT。BLE引入了attribute概念，用来描述一条一条的数据。Attribute除了定义数据，同时定义该数据可以使用的ATT命令，因此这一层被称为ATT层。
- `Security Manager`，简称：`SM`。SMP用来管理BLE连接的加密和安全的，如何保证连接的安全性，同时不影响用户的体验，这些都是SMP要考虑的工作。
- `Generic Access Profile`，简称：`GAP`。GAP是对LL层payload（有效数据包）如何进行解析的两种方式中的一种，而且是最简单的那一种。GAP简单的对LL payload进行一些规范和定义，因此GAP能实现的功能极其有限。GAP目前主要用来进行广播，扫描和发起连接等。
- `Generic Attribute Profile`，简称：`GATT`。GATT用来规范attribute中的数据内容，并运用group（分组）的概念对attribute进行分类管理。没有GATT，BLE协议栈也能跑，但互联互通就会出问题，也正是因为有了GATT和各种各样的应用profile，BLE摆脱了ZigBee等无线协议的兼容性困境，成了出货量最大的2.4G无线通信产品。

4.2.3 应用Application

应用层是用户开发实际蓝牙应用的地方，包含必要的协议栈参数设置，以及各种功能函数的调用。我们分别从蓝牙从机和蓝牙主机两种设备来分析。

- 蓝牙从机
    1. 相关硬件和基础服务初始化
    2. 设置广播参数：广播数据，广播间隔，扫描回应等参数或者数据。
    3. 设置Profile：添加从机服务、特征是，还有设置回调函数用于接收主机数据等。
    4. 设置绑定管理参数（可选）
    5. 启动广播，开始运行。
    6. 等待相关事件，及事件处理，例如收到主机发来的数据，被链接等等。
- 蓝牙主机
    1. 相关硬件和基础服务初始化
    2. 设置扫描参数。
    3. 设置连接参数。
    4. 设置绑定管理参数（可选）
    5. 启动协议栈，开始运行。
    6. 等待相关事件，及事件处理，例如扫描事件，从机的Notify事件等等。

5 GAP和GATT

蓝牙协议栈分为两类结构：控制器（Controller）和主机（Host）。每个类别都有子类别，这些子类别执行特定的角色。我们将要研究的两个子类别是 通用访问配置文件 （GAP）和 通用属性配置文件 （GATT）。

- GAP是Generic Access Profile的缩写，中文含义是：通用访问配置文件。
- GATT是Generic Attribute Profile的缩写，中文含义是：通用属性配置文件。

5.1 GAP和GATT区别

区分GAP和GATT很重要。

- GAP 定义了 BLE网络堆栈的一般拓扑。

- GATT 详细描述了一旦设备建立连接后如何传输属性（数据）。

GATT特别关注如何根据其描述的规则格式化打包和发送数据。在BLE网络堆栈中，属性协议（ATT）与GATT紧密对齐，GATT直接位于ATT的顶部。GATT实际上使用ATT来描述如何从两个连接的设备交换数据。

5.2 通用访问配置文件（GAP）

BLE设备可以使用两种机制与外界通信：广播或连接。这些机制受通用访问配置文件（GAP）准则的约束。GAP定义了启用BLE的设备如何使其自身可用，以及两个设备如何直接相互通信。

5.2.1 建立联系（Connecting）

设备可以通过采用GAP中指定的以下角色来加入BLE网络：

A、广播（**Broadcasting**）：这些角色不必显式地相互连接即可传输数据。

- 广播者（**Broadcaster**）：广播公共数据包的设备，例如可以广播按下按钮的时间。
- 观察者（**Observer**）：侦听广播者发送的广告包中数据的设备。广播者和观察者之间没有任何连接。/2、/2

B、连接（**Connecting**）：这些角色必须显式连接和握手才能传输数据。这些角色比广播角色更常用。

- 从机设备（**Peripheral**）： 通过广播，告知其他设备自己的存在，以便主机设备可以建立连接。连接后，从机设备不再向其他主机设备广播数据，而是保持与主机设备的连接。
  - 从机设备功耗低，因为它们只需要定期发送信标即可。主机设备负责开始与从机设备的通信。
  - 手环是BLE外设的一个示例。
- 主机设备（**Central**）：一种通过侦听广播包来启动与从机设备的连接的设备。主机设备可以连接到许多其他从机设备。
  - 当主机设备要连接时，它将请求连接数据包发送到从机设备。如果从机设备接受来自主机设备的请求，则建立连接。
  - 当您的手机连接到手环时，就是BLE Central设备的一个示例。

5.2.2 连接后（Connected）

主机设备可以更新连接参数： 主机设备通常在设备与其自身之间建立连接参数。只有主机设备能修改连接参数。但是，从机设备可以要求主机设备更改连接参数，及从机发送更新参数请求。

从机设备或主机设备可以终止连接：连接可能由于多种原因而终止，例如设备的电池可能耗尽或网络干扰可能导致连接失败。设备还可以主动与对等设备断开连接。

5.3 通用属性配置文件（GATT）

5.3.1 模型角色

GATT分为两种类型，注意与从机或主机无关。

客户端（Client）：客户端可以发送请求给GATT服务端，客户端可以读（Read）/写（Write）服务端的属性（Attributes ），通过属性可以通信数据。

服务端（Server）：服务端是用来存储属性（Attributes ）的，每当客户端发送请求时，服务端会相应这些请求。

5.3.2 客户端与服务端的关系

一个示例如下：手环采集了心跳信息，希望计算机读取该信息。手环充当服务端并提供信息。手机充当客户端，读取该信息。

GAP和GATT模型角色基本上彼此独立从机设备或主机设备都可以充当服务端或客户端，这取决于数据的流动方式。

在一般的主从机通信时，主机可以通过读写从机的属性，实现接收和发送数据给从机，从机可以通过发送通知的方式实现与主机的通信。因此，一般从机是作为GATT的服务端，主机作为GATT的客户端。

6 协议栈分层协作

下面以如何发送一个无线数据包的例子来简单阐述协议栈中各分层的作用和必要性。实际上，协议栈的实现可能更加负责，它需要考虑方方面面的因素。

6.1 发送数据包

假设有设备A和设备B，设备A要把自己的电量状态83%（十六进制表示为0x53）发给设备B，该怎么做呢？作为一个开发者，他希望越简单越好，对他而言，他希望调用一个简单的API就能完成这件事，比如`send(0x53)`，实际上我们的BLE协议栈就是这样设计的，开发者只需调用`send(0x53)`就可以把数据发送出去了，其余的事情BLE协议栈帮你搞定。很多人会想，BLE协议栈是不是直接在物理层就把0x53发出去，就如下图所示：

这种方式初看起来挺美的，但由于很多细节没有考虑到，实际是不可行的。首先，它没有考虑用哪一个射频信道来进行传输，在不更改API的情况下，我们只能对协议栈进行分层，为此引入LL层，开发者还是调用`send(0x53)`，`send(0x53)`再调用`send_LL(0x53,2402M)`（注：2402M为信道频率）。这里还有一个问题，设备B怎么知道这个数据包是发给自己的还是其他人的，为此BLE引入**access address概念，用来指明接收者身份**，其中，`0x8E89BED6`这个access address比较特殊，它表示要发给周边所有设备，即广播。如果你要一对一的进行通信（BLE协议将其称为**连接**），即设备A的数据包只能设备B接收，同样设备B的数据包只能设备A接收，那么就必须生成一个独特的**随机access address**以标识设备A和设备B两者之间的连接。

6.2 广播方式

我们先来看一下简单的广播情况，这种情况下，我们把设备A叫**advertiser**（广播者），设备B叫**scanner**或者**observer**（扫描者）。广播状态下设备A的LL层API将变成`send_LL(0x53,2402M, 0x8E89BED6)`。由于设备B可以同时接收到很多设备的广播，因此数据包还必须包含设备A的device address（0xE1022AAB753B）以确认该广播包来自设备A，为此send_LL参数需要变成`send_LL(0x53,2402M, 0x8E89BED6, 0xE1022AAB753B)`。LL层还要检查数据的完整性，即数据在传输过程中有没有发生窜改，为此引入CRC24对数据包进行检验（假设为0xB2C78E）。同时为了调制解调电路工作更高效，每一个数据包的最前面会加上1个字节的preamble（前导帧），preamble一般为0x55或者0xAA。这样，整个空中包就变成（注：**空中包用小端模式表示！**）：

上面这个数据包还有如下问题：

1. 没有对数据包进行分类组织，设备B无法找到自己想要的数据0x53。为此我们需要在access address之后加入两个字段：LL header和长度字节。LL header用来表示数据包的LL类型，长度字节用来指明payload的长度
2. 设备B什么时候开启射频窗口以接收空中数据包？如上图case1所示，当设备A的数据包在空中传输的时候，设备B把接收窗口关闭，此时通信将失败；同样对case2来说，当设备A没有在空中发送数据包时，设备B把接收窗口打开，此时通信也将失败。只有case3的情况，通信才能成功，即设备A的数据包在空中传输时，设备B正好打开射频接收窗口，此时通信才能成功，换句话说，**LL层还必须定义通信时序**。
3. 当设备B拿到数据0x53后，该如何解析这个数据呢？它到底表示湿度还是电量，还是别的意思？这个就是GAP层要做的工作，GAP层引入了LTV（Length-Type-Value）结构来定义数据，比如020105，02-长度，01-类型（强制字段，表示广播flag，广播包必须包含该字段），05-值。由于广播包最大只能为31个字节，它能定义的数据类型极其有限，像这里说的电量，GAP就没有定义，因此要通过广播方式把电量数据发出去，只能使用供应商自定义数据类型0xFF，即04FF590053，其中04表示长度，FF表示数据类型（自定义数据），0x0059是供应商ID（自定义数据中的强制字段），0x53就是我们的数据(设备双方约定0x53就是表示电量，而不是其他意思)。

最终空中传输的数据包将变成：

`AAD6BE898E600E3B75AB2A02E102010504FF5900538EC7B2`

- `AA` - 前导帧(preamble)
- `D6BE898E` - 访问地址(access address)
- `60` - LL帧头字段(LL header)
- `0E` - 有效数据包长度(payload length)
- `3B75AB2A02E1` - 广播者设备地址(advertiser address)
- `02010504FF590053` - **广播数据**
- `8EC7B2` - CRC24值

AAD6BE898E600E3B75AB2A02E102010504FF5900538EC7B2

有了PHY，LL和GAP，就可以发送广播包了，但广播包携带的信息极其有限，而且还有如下几大限制：

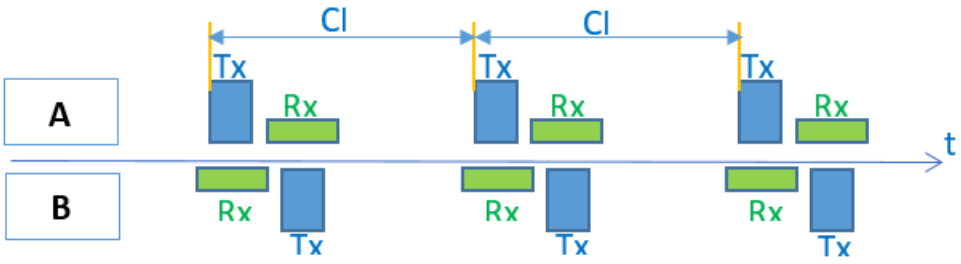
1. 无法进行一对一双向通信 （广播是一对多通信，而且是单方向的通信）
2. 由于不支持组包和拆包，因此无法传输大数据
3. 通信不可靠及效率低下。广播信道不能太多，否则将导致扫描端效率低下。为此，BLE只使用37(2402MHz) /38(2426MHz) /39(2480MHz)三个信道进行广播和扫描，因此广播不支持跳频。由于广播是一对多的，所以广播也无法支持ACK。这些都使广播通信变得不可靠。
4. 扫描端功耗高。由于扫描端不知道设备端何时广播，也不知道设备端选用哪个频道进行广播，扫描端只能拉长扫描窗口时间，并同时对37/38/39三个通道进行扫描，这样功耗就会比较高。

而连接则可以很好解决上述问题，下面我们就来看看连接是如何将0x53发送出去的。

6.3 连接方式

到底什么叫连接(connection)？像有线UART，很容易理解，就是用线（Rx和Tx等）把设备A和设备B相连，即为连接。用“线”把两个设备相连，实际是让2个设备有共同的通信媒介，并让两者时钟同步起来。蓝牙连接有何尝不是这个道理，**所谓设备A和设备B建立蓝牙连接，就是指设备A和设备B两者一对一“同步”成功**，其具体包含以下几方面：

- 设备A和设备B对接下来要使用的物理信道达成一致
- 设备A和设备B双方建立一个共同的时间锚点，也就是说，把双方的时间原点变成同一个点
- 设备A和设备B两者时钟同步成功，即双方都知道对方什么时候发送数据包什么时候接收数据包
- 连接成功后，设备A和设备B通信流程如下所示：



如上图所示，一旦设备A和设备B连接成功（此种情况下，我们把设备A称为**Master**或者**Central**，把设备B称为**Slave**或者**Peripheral**），设备A将周期性以CI（connection interval）为间隔向设备B发送数据包，而设备B也周期性地以CI为间隔打开射频接收窗口以接收设备A的数据包。同时按照蓝牙spec要求，设备B收到设备A数据包**150us后**，设备B切换到发送状态，把自己的数据发给设备A；设备A则切换到接收状态，接收设备B发过来的数据。由此可见，连接状态下，设备A和设备B的射频发送和接收窗口都是周期性地有计划地开和关，而且开的时间非常短，从而大大降低系统功耗并大大提高系统效率。

现在我们看看连接状态下是如何把数据0x53发送出去的，从中大家可以体会到蓝牙协议栈分层的妙处。

- 对开发者来说，很简单，他只需要调用send(0x53)
- GATT层定义数据的类型和分组，方便起见，我们用0x0013表示电量这种数据类型，这样GATT层把数据打包成130053（**小端模式！**）
- ATT层用来选择具体的通信命令，比如读/写/notify/indicate等，这里选择notify命令0x1B，这样数据包变成了：1B130053
- L2CAP用来指定connection interval（连接间隔），比如每10ms同步一次（CI不体现在数据包

中），同时指定逻辑通道编号0004（表示ATT命令），最后把ATT数据长度0x0004加在包头，这样数据就变为：040004001B130053

- LL层要做的工作很多，首先LL层需要指定用哪个物理信道进行传输（物理信道不体现在数据包中），然后再给此连接分配一个Access address（0x50655DAB）以标识此连接只为设备A和设备B直连服务，然后加上LL header和payload length字段，LL header标识此packet为数据packet，而不是control packet等，payload length为整个L2CAP字段的长度，最后加上CRC24字段，以保证整个packet的数据完整性，所以数据包最后变成：
    - AAAB5D65501E08040004001B130053D550F6
        - AA – 前导帧(preamble)
        - 0x50655DAB – 访问地址(access address)
        - 1E – LL帧头字段(LL header)
        - 08 – 有效数据包长度(payload length)
        - 04000400 – ATT数据长度，以及L2CAP通道编号
        - 1B – notify command
        - 0x0013 – 电量数据handle
        - 0x53 – 真正要发送的电量数据
        - 0xF650D5 – CRC24值

AAAB5D65501E08040004001B130053D550F6

虽然开发者只调用了 send(0x53)，但由于低功耗蓝牙协议栈层层打包，最后空中实际传输的数据将变成下图所示的模样，这就既满足了低功耗蓝牙通信的需求，又让用户API变得简单，可谓一箭双雕！

希望通过这个例子，让大家对协议栈的各层作用有个初步的印象。

- 本页面最后编辑于2020年1月10日 (星期五) 14:51。