STM32WB BLE 协议栈编程指南

引言

本文档的主要目的是为开发人员提供有关如何使用 STM32WB BLE 协议栈 API 和相关事件回调开发低功耗蓝牙（BLE）应用的一些参考编程指南。

本文档介绍了允许访问 STM32WB 片上系统所提供的低功耗蓝牙功能的 STM32WB 低功耗蓝牙协议栈库框架、API 接口和事件回调。

本编程手册还提供一些与低功耗蓝牙（BLE）技术有关的基本概念，以便将 STM32WB BLE 协议栈 API、参数及相关事件回调与 BLE 协议栈特性联系起来。用户必须具备有关 BLE 技术及其主要功能的基本知识。

有关 STM32WB 系列和低功耗蓝牙规范的更多信息，请参考本文档结尾处的第 6 节“参考文档”。

STM32WB 是一种功率极低的低功耗蓝牙（BLE）单模网络处理器，符合蓝牙规范 v5.3 并支持主设备或从设备角色。

手册结构如下：

- 低功耗蓝牙（BLE）技术的基本原理
- STM32WB BLE 协议栈库 API 和事件回调概述
- 如何利用 STM32WB 库 API 和事件回调设计应用（使用“switch case”事件处理程序提供一些示例，而不使用事件回调框架）

1 概述

本文档适用于 STM32WB 系列基于双核 Arm®的微控制器。

注意： *Arm 是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

2 低功耗蓝牙技术

低功耗蓝牙（BLE）无线技术由蓝牙技术联盟（SIG）开发，目的是使设备能够以极低功耗标准使用纽扣电池工作数年。

传统蓝牙作为一种无线技术标准，可以取代连接便携式和/或固定式电子设备的线缆，但是由于采取了快速跳频、以连接为导向的行为方式和相对复杂的连接流程，无法采用电池供电的方式。

低功耗蓝牙设备的功耗仅为标准蓝牙产品的一小部分，让使用纽扣电池的设备能够无线连接到启用了标准蓝牙的设备。

图 1. 支持低功耗蓝牙技术的以钮扣电池供电的设备

低功耗蓝牙技术广泛应用于传输少量数据的传感器应用中：

- 汽车
- 运动与健身
- 医疗
- 娱乐
- 家庭自动化
- 安全和接近感测

2.1 BLE 协议栈架构

低功耗蓝牙技术已被蓝牙核心规范 4.0 正式采纳（参见第 6 节 参考文件）。

低功耗蓝牙技术工作在工业、科学和医疗（ISM）频段 2.4~2.485GHz，可以在全球许多国家使用而无需官方授权。它使用扩频、跳频、全双工信号。低功耗蓝牙技术的关键特性：

- 稳健性
- 性能
- 可靠性
- 互操作性
- 低速率
- 低功耗

另外，低功耗蓝牙技术的目的是为了实现在传输极小数据包的同时，其功耗显著低于基础速率（BR）、增强速率（EBR）以及高速设备（HS）等经典蓝牙设备。

蓝牙低能耗技术旨在解决两种替代方案：

- 智能设备（Smart device）
- 智能就绪设备（Smart ready device）

智能设备仅支持 BLE 标准。它适用于低功耗并使用纽扣电池供电的应用（例如传感器）。

智能就绪设备支持 BR/EDR/HS 和 BLE 标准（通常为移动设备或笔记本电脑）。

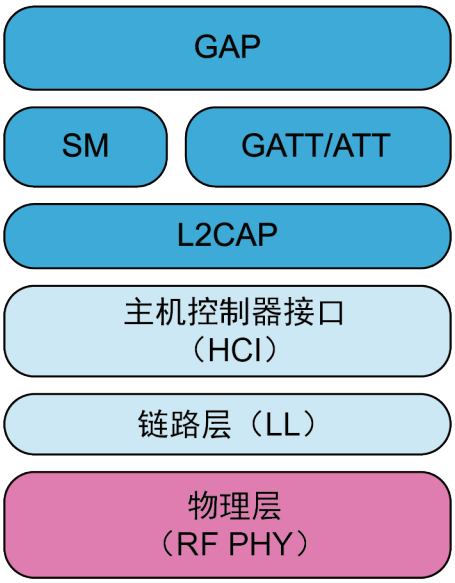
低功耗蓝牙协议栈有两个组成部分：

- 控制器（Controller）
- 主机（Host）

控制器包含物理层和链路层。

主机包括逻辑链路控制和适配协议（L2CAP）、安全管理器（SM）、属性协议（ATT）、通用属性配置文件（GATT）和通用访问配置文件（GAP）。两个组成部分之间的接口被称为主机控制器接口（HCI）。

图 2. 低功耗蓝牙栈架构



目前已发布支持新功能的蓝牙规范 v4.1、v4.2、v5.0、v5.1 和 v5.2：

- **STM32WB 支持的蓝牙规范 V4.1 中的功能：**
  - 同时支持多个角色
  - 支持同时广播并扫描
  - 支持同时作为最多两个主设备的从设备
  - Privacy V1.1
  - 低占空比定向广播
  - 连接参数请求流程
  - 32 位 UUID
  - L2CAP 以连接为导向的行为
- **STM32WB 支持的蓝牙规范 V4.2 中的功能：**
  - LE 数据长度扩展
  - 地址解析
  - LE Privacy 1.2
  - LE 安全连接
- **STM32WB 支持的蓝牙规范 V5.0 中的功能：**
  - LE 2M PHY

2.2 物理层

物理层是 1 Mbps 自适应跳频高斯频移键控（GFSK）无线电或 2Mbit/s 2 级高斯频移键控（GFSK）无线电。它在无需执照的 2.4 GHz ISM 波段中以 2400-2483.5 MHz 的频率工作。很多其他标准也使用此波段，如：IEEE 802.11，IEEE 802.15。

BLE 系统使用 40 射频通道（0-39），2 MHz 间隔。这些射频通道的频率中心为：

$$240 + k * 2MHz，其中\ k = 0.39 \quad (1)$$

有两种类型的通道：

1. 广播通道，使用三个固定的射频通道（37、38 和 39）用于：
   a. 传输广播通道数据包
   b. 用于发现/连接的数据包
   c. 广播/扫描
2. 数据物理通道使用其他 37 个射频通道进行已连接设备的双向通信。

表 1. BLE RF 通道类型和频率

通道索引	射频中心频率	通道类型
37	2402 MHz	广播通道
0	2404 MHz	数据通道
1	2406 MHz	数据通道
….	….	数据通道
10	2424 MHz	数据通道
38	2426 MHz	广播通道
11	2428 MHz	数据通道
12	2430 MHz	数据通道
….	….	数据通道
36	2478 MHz	数据通道
39	2480 MHz	广播通道

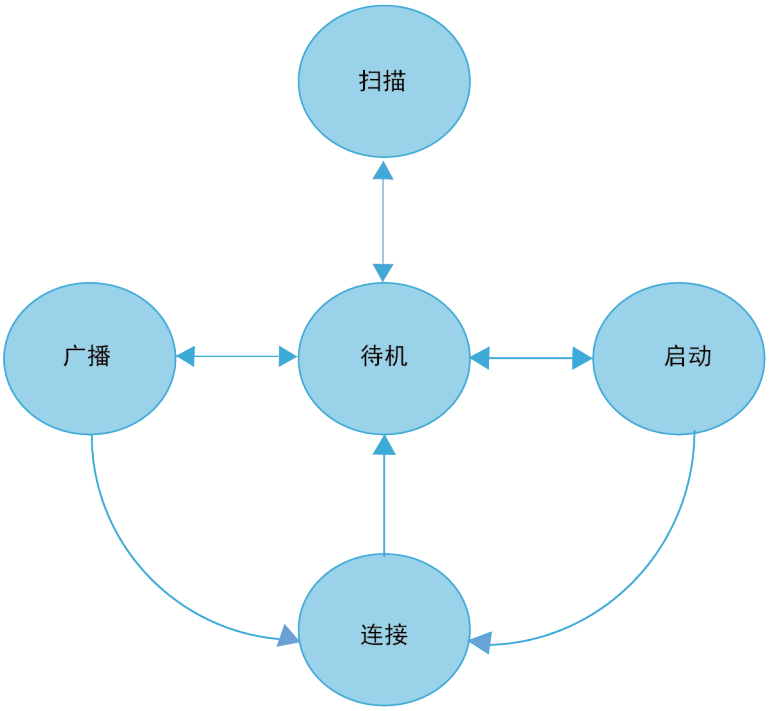
BLE 是一种自适应跳频（AFH）技术，只能使用所有可用频率的一个子集，以避开其他非自适应技术使用的所有频率。通过使用特定的跳频算法确定下一个要使用的良好通道，可以从不良通道转移到已知良好通道。

2.3 链路层（LL）

链路层（LL）定义了两个设备使用射频信号在彼此之间传递信息的方式。

链路层定义了具有五种状态的状态机：

图 3. **链路层状态机**

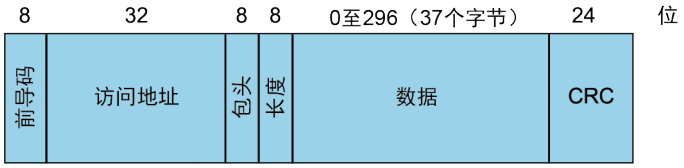


- 待机：设备不发送或接收数据包
- 广播：设备在广播通道中广播广播（称为广播设备）
- 扫描：设备寻找广播设备（称为扫描设备）
- 发起：设备发起与广播设备的连接
- 连接：发起设备为主设备，与从设备进行通讯并定义通讯的方式和时序
- 广播设备是从设备，只能和单个主设备进行通讯

2.3.1 BLE 数据包

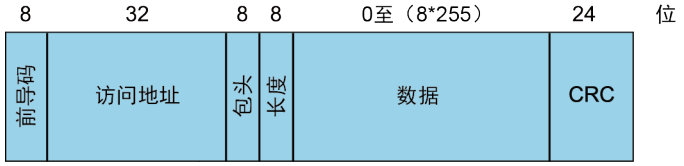
数据包是带标签的数据，由一个设备发送并由一个或多个其他设备接收。BLE 数据包的结构如下。

图 4. 数据包结构



蓝牙低功耗 BLE 规范 v4.2 定义了 LE 数据包长度扩展功能，它将 LE 链路层 PDU 的数据有效负载从 27 字节扩展到 251 字节。

图 5. LE 数据包长度扩展功能的数据包结构



长度字段的范围为 0 到 255 字节。使用加密时，数据包末尾的消息完整性检查字段（MIC）为 4 字节，从而导致实际最大可用有效数据长度为 251 字节。

- 前导码：RF 同步序列
- 访问地址：32 位，广播或数据访问地址（用于识别物理层通道上的通信数据包）
- 包头：其内容取决于数据包类型（广播或数据包）
- 广播数据包包头：

表 2. 广播数据头内容

广播数据包类型	保留	发送地址类型	接收地址类型
（4 位）	（2 位）	（1 位）	（1 位）

- 广播数据包类型定义如下：

表 3. 广播数据包类型

数据包类型	说明	注释
ADV_IND	可连接非定向广播	由广播方在需要另一台设备与之连接时使用。设备可被扫描设备扫描到，或者在收到连接请求时作为从设备建立连接。
ADV_DIRECT_IND	可连接定向广播	由广播方在需要特定设备与之连接时使用。 ADV_DIRECT_IND 数据包仅包含广播方地址和发起方地址。
ADV_NONCONN_IND	不可连接非定向广播	由广播方在需要向所有设备提供某些信息，但是不希望其他设备向其请求更多信息或与之连接时使用。 设备只在相关通道上发送广播数据包，但不希望任何其他设备与之连接或扫描。
ADV_SCAN_IND	可扫描非定向广播	由希望允许扫描设备请求更多信息的广播方使用。设备不能连接，但是对于广播数据和扫描响应数据而言可发现。
SCAN_REQ	扫描请求	由处于扫描状态的设备用于向广播方请求更多信息。
SCAN_RSP	扫描响应	由广播设备用于向扫描设备提供更多信息。
CONNECT_REQ	连接请求	由发起设备发送给处于可连接/可发现模式的设备。

广播事件类型决定了允许的响应：

表 4. 广播事件类型和允许的响应

广播事件类型	是否允许响应	
	SCAN_REQ	CONNECT_REQ
ADV_IND	是	是
ADV_DIRECT_IND	否	是
ADV_NONCONN_IND	否	否
ADV_SCAN_IND	是	否

- 数据包包头：

表 5. 数据包包头内容

链路层标识符	下一个序号	序号	更多数据	保留
（2 位）	（1 位）	（1 位）	（1 位）	（3 位）

下一个序号（NESN）位用于实现数据包响应功能。它将发送设备预期将发送的下一个序号通知接收设备。NESN 和序号（SN）值相同的数据包为重传数据包。

更多数据位用于向设备发送信号，表明在当前连接事件期间发送设备有更多数据准备发送。

关于广播和数据包头内容及类型的详细描述，请参考第 6 节 参考文件中的蓝牙规范[第 2 卷]。

- 长度：数据字段中的字节数

表 6. 数据包长度字段和有效值

	长度字段位数
广播数据包	6 位，有效值为 0 至 37 字节
数据包	5 位，有效值为 0 至 31 字节 8 位，有效值为 0 到 255 字节，支持 LE 数据包长度扩展

- 数据或有效负载：实际发送的数据（广播数据、扫描响应数据、连接建立数据或连接期间发送的应用数据）
- CRC（24 位）：用于保护数据，避免发生位错误。它通过包头、长度和数据字段计算得出

2.3.2 广播状态

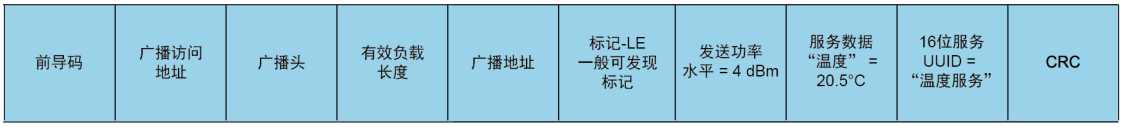
广播状态允许链路层发送广播数据包，并对正在执行扫描的设备发出的扫描请求作出扫描响应。

可通过停止广播使广播设备进入待机状态。

设备每次进行广播时，通过三个广播通道发送相同数据包。此三个数据包序列称为一个“广播事件”。两个广播事件之间的时间被称为广播间隔，长度从 20 毫秒到 10.24 秒不等。

广播数据包的示例列出了设备实现的服务 UUID（一般可发现标记，发送功率 = 4dbm，服务数据 = 温度服务和 16 位服务 UUID）。

图 6. 具有 AD 类型标记的广播数据包



AD 类型标记字节（Flags）包含以下标记位：

- 有限可发现模式（第 0 位）
- 一般可发现模式（第 1 位）
- 不支持 BR/EDR（第 2 位，在 BLE 上为 1）
- 可同时以 LE 和 BR/EDR 连接到同一设备（控制器）（第 3 位）
- 可同时以 LE 和 BR/EDR 连接到同一设备（主机）（第 4 位）

如果任意位为非零，则在广播数据中包含 AD 类型标记（不包括在扫描响应中）。

在启用广播之前，可设置下列广播参数：

- 广播间隔
- 广播地址类型
- 广播设备地址
- 广播通道映射：应使用三个广播通道中的哪一个

- 广播过滤策略：
  - 处理来自白名单中的设备的扫描/连接请求
  - 处理所有扫描/连接请求（默认的广播方过滤策略）
  - 处理来自所有设备的连接请求，但仅处理来自白名单中的设备的扫描请求
  - 处理来自所有设备的扫描请求，但仅处理来自白名单中的设备的连接请求

白名单是设备控制器用于过滤设备的已保存设备地址列表。当白名单正在使用时，不能修改其内容。如果设备处于广播状态且正在使用白名单过滤设备（扫描请求或连接请求），必须禁用广播模式才能修改其白名单。

2.3.3 扫描状态

有两种类型的扫描：

- 被动扫描：允许接收来自广播设备的广播数据
- 主动扫描：在接收到广播数据包时，设备可以发回扫描请求数据包，以便得到广播方的扫描响应。这使扫描设备能够获取来自广播设备的额外信息。

可以设置下列扫描参数：

- 扫描类型（被动或主动）
- 扫描间隔：控制器的扫描频率
- 扫描窗口：对于每个扫描间隔而言，它定义了设备扫描的持续时间
- 扫描过滤策略：它可以接受所有广播数据包（默认策略）或仅接受来自白名单设备的广播数据包。

在设置扫描参数后，可以启用设备扫描。扫描设备的控制器向上层发送任何在广播报告事件中接收的广播数据包。该事件包含该广播数据包的广播方地址、广播方数据和接收信号强度指示（RSSI）。可将 RSSI 与广播数据包中包含的发送功率水平信息一起使用，以确定信号的路径损耗和设备距离：

路径损耗 = 发送功率 – RSSI

2.3.4 连接状态

当待发送数据的复杂度超过广播数据允许的水平，或者需要在两个设备之间建立双向可靠通信时，建立连接。

当发起设备接收到它要连接的广播设备发出的广播数据包时，它可以向广播设备发送连接请求数据包。该数据包包含建立和处理两个设备之间的连接所需的所有必要信息：

- 用于识别所建立连接的物理链路层通讯的访问地址
- CRC 初始值
- 发送窗口大小（第一个数据包的时序窗口）
- 发送窗口偏移（发送窗口起点）
- 连接间隔（两个连接事件之间的时间）
- 从设备延迟（从设备在被强制监听前可以忽略的连接事件次数）
- 监控超时（在链路被视为丢失之前两次正确接收到数据包之间的最长时间）
- 通道映射表：37 位（1= 良好；0 = 不佳）
- 跳频值（5 至 16 之间的随机数）
- 睡眠时钟精度范围（用于确定连接事件中从设备的不确定性窗口）

关于连接请求数据包的详细描述，请参考蓝牙规范[第 6 卷]。

表 7. 连接请求时序间隔中总结了允许的时序范围：

表 7. 连接请求时序间隔

参数	最小值	最大值	注释
发送窗口长度	1.25 毫秒	10 毫秒	-
发送窗口偏移	0	连接间隔	1.25 毫秒的倍数
连接间隔	7.5 毫秒	4 秒	1.25 毫秒的倍数
监控超时	100 毫秒	32 秒	10 毫秒的倍数

发送窗口在连接请求数据包末尾加上发送窗口偏移加上 1.25 ms 的强制延迟后启动。发送窗口启动时，从设备进入接收器模式并等待来自主设备的数据包。如果这段时间内没有接收到数据包，从设备将退出接收器模式，并且在下一个连接间隔重新尝试接收来自主设备的数据。在连接建立后，主设备必须就每个连接事件向从设备发送数据包，以允许从设备向主设备发送数据包。或者，从设备可以跳过给定数量的连接事件（从设备延迟）。

连接事件是上一个连接事件的起点与下一个连接事件的起点之间的时间。

一个 BLE 从设备只能连接一个 BLE 主设备，但是一个 BLE 主设备可以连接多个 BLE 从设备。蓝牙 SIG 对一个主设备可以连接的从设备数量并无限制（只受限于使用的特定 BLE 技术或协议栈）。

2.4 主机控制器接口（HCI）

主机控制器接口（HCI）层使主机和控制器之间可以通过软件 API 或者通过 SPI、UART 或 USB 等硬件接口：实现通信。它来自标准蓝牙规范，包含用于低功耗特定功能的附加新指令。

2.5 逻辑链路控制和适配层协议（L2CAP）

逻辑链路控制和适配层协议（L2CAP）支持更高层协议复用、数据包分割和重组操作以及服务信息质量的通知。

2.6 属性配置文件（ATT）

属性配置文件（ATT）允许设备向另一设备公开某些数据，即属性。公开属性的设备被称为服务器，而使用它们的对端设备被称为客户端。

属性是一种包含下列组成部分的数据：

- 属性句柄：16 位值，用于标识服务器端相应属性数据。客户端在读写请求中引用该句柄
- 属性类型：通过通用唯一标识符（UUID）定义，决定了对应属性值的含义。蓝牙 SIG 定义了标准 16 位属性 UUID
- 属性值：长度为 0~512 字节的数据。
- 属性权限：由使用该属性的更高层协议定义。它们指定读和/或写访问以及通知和/或指示需要的安全级别。使用属性协议不能获取这些权限。几种不同的许可类型如下：
  - 访问权限：决定了可以对属性执行的请求类型（可读、可写、可读且可写）
  - 验证权限：决定了属性是否需要验证。如果发生了验证错误，客户端可以尝试使用安全管理器进行验证并发回请求
  - 授权权限（无授权、授权）：这是服务器的特性，决定了是否授权客户端访问一组属性（客户端不能解决授权错误）

表 8. 属性示例

句柄属性	属性类型	属性值	属性权限
0x0008	“温度 UUID”	“温度值”	“只读，无授权，无验证”

- “温度 UUID”由“温度特征”规范定义，为有符号 16 位整数。

属性的集合被称为数据库，始终包含在属性服务器中。

属性协议定义了一组方法协议，用于在对端设备上发现、读取和写入属性。它在如下属性服务器和属性客户端之间实现端到端的客户端-服务器协议：

- 服务器角色
  - 包含所有属性（属性数据库）
  - 接收请求、执行、响应指令
  - 在数据变化时可以指示、通知属性值
- 客户端角色
  - 与服务器通信
  - 发送请求，等待响应（可以访问（读取）和更新（写入）数据）
  - 确认指示

服务器公开的属性可以被客户端发现、读取和写入，并且可通过服务器指示和通知，如表 9.属性协议消息所示：

表 9. 属性协议消息

协议数据单元（PDU 消息）	发送者	说明
请求	客户端	客户端询问服务器（总会导致响应）
响应	服务器	服务器发送对客户端请求的响应
指令	客户端	客户端命令服务器做某事（无响应）
通知	服务器	服务器将新值通知客户端（无确认）
指示	服务器	服务器向客户端指示新值（总是导致确认）
确认	客户端	对指示的确认

2.7 安全管理器（SM）

低功耗蓝牙链路层支持使用 CBC-MAC（加密块链-消息验证码）算法的计数器模式和 128 位 AES 分组密码（AES-CCM）进行加密和验证。当在连接中使用加密和认证时，为数据通道 PDU 的有效负载添加 4 字节的消息完整性检查字段（MIC）。

加密应用于 PDU 有效负载和 MIC 字段。

当两个设备要在连接期间进行通信加密时，安全管理器使用配对流程。此流程允许通过交换身份信息来验证两个设备，交换信息是为了创建可作为受信任关系或（单个）安全连接基础的安全密钥。有一些方法用于执行配对过程。其中一些方法提供了

- 中间人（MITM）攻击防护：设备能够监控和修改两个设备之间的通信通道或向其添加新消息。典型场景为设备能够连接每个设备，并通过与每个设备通信来充当其他设备
- 被动窃听攻击：通过嗅探设备监听其他设备的通信

低功耗蓝牙规范 v4.0 或 v4.1 的配对也被称为 LE 传统配对，该配对支持基于设备 IO 功能的以下方法：直接工作法、输入密钥法和带外（OOB）法。

在低功耗蓝牙规范 v4.2 中，已定义 LE 安全连接配对模型。新安全模型的主要特性为：

1. 密钥交换过程使用了椭圆曲线（ECDH）算法：该算法允许通过不安全的通道交换密钥，并可以防止被动窃听攻击（通过嗅探设备秘密监听其他设备的通信）
2. 在 3 种可用的 LE 传统配对方法的基础上，增加了一种名为“数字比较”的新方法

根据设备 IO 功能选择配对程序。

有三种输入功能：

- 无输入
- 能够选择是/否
- 能够使用键盘输入数字

有两种输出功能：

- 无输出
- 数字输出：能够显示六位数字

下表显示了可能的 IO 功能组合

表 10. BLE 设备上输入/输出功能的组合

	无输出	显示
无输入	无输入，无输出	仅显示
是/否	无输入，无输出	显示是/否
键盘	仅键盘	键盘，显示

LE 传统配对

LE 传统配对算法使用并生成 2 个密钥：

- 临时密钥（TK）：用于生成短期密钥（STK）的 128 位临时密钥
- 短期密钥（STK）：用于在配对后加密链接的 128 位临时密钥

配对流程分为三个阶段。

第 1 阶段：配对特征交换

两个已建立连接的设备通过配对请求消息交换其输入/输出能力。该消息还包括用于说明带外数据是否可以用的指示位以及身份验证需求等信息。在第 1 阶段交换的信息将用于选择第 2 阶段使用的 STK 生成配对方法。

第 2 阶段：短期密钥（STK）生成

配对设备首先使用下列密钥生成方法之一定义临时密钥（TK）

1. 带外（OOB）法将带外通信（如 NFC）用于 TK 协议。它提供了身份认证（MITM 保护）。仅在两个设备上都设置了带外位时才选择此方法，否则必须使用设备的 IO 功能来确定可以使用其他方法（输入密钥法或直接工作法）
2. 输入密钥法：用户输入六位数字作为设备之间的 TK。它提供了身份认证（MITM 保护）
3. 直接工作法：该方法不提供认证，也不提供中间人（MITM）攻击防护

根据下表所定义的 IO 功能在输入密钥法和直接工作法之间进行选择。

表 11. 计算临时密钥（TK）的方法

	仅显示	显示是/否	仅键盘	无输入，无输出	键盘，显示
仅显示	直接工作	直接工作	输入密钥	直接工作	输入密钥
显示是/否	直接工作	直接工作	输入密钥	直接工作	输入密钥
仅键盘	输入密钥	输入密钥	输入密钥	直接工作	输入密钥
无输入，无输出	直接工作	直接工作	直接工作	直接工作	直接工作
键盘，显示	输入密钥	输入密钥	输入密钥	直接工作	输入密钥

第 3 阶段：用于计算临时密钥（TK）的传输特定密钥分配方法

一旦第 2 阶段完成，可通过使用 STK 密钥加密的消息分配最多三个 128 位密钥：

1. 长期密钥（LTK）：用于生成链路层加密和验证使用的 128 位密钥
2. 连接签名解析密钥（CSRK）：用于在 ATT 层上执行数据签名和验证的 128 位密钥
3. 身份解析密钥（IRK）：用于生成和解析随机地址的 128 位密钥

LE 安全连接

LE 安全连接配对方法使用并生成一个密钥：

- 长期密钥（LTK）：用于在配对和后续连接后加密连接的 128 位密钥

配对流程分为三个阶段：

第 1 阶段：配对特征交换

两个已建立连接的设备通过配对请求消息交换其输入/输出能力。该消息还包括用于说明带外数据是否可以用的指示位以及身份验证需求等信息。在第 1 阶段交换的信息将用于选择第 2 阶段使用的配对方法。

第 2 阶段：长期密钥（LTK）生成

配对流程由发起设备启动，该设备将公钥发送至接收设备。接收设备将回复其公钥。对于所有配对方法（除了 OOB 以外），公钥交换阶段已完成。每个设备均产生自己的椭圆 Diffie-Hellman (ECDH)公钥 - 私钥对。每个密钥对均包含一个私钥（密钥）和一个公钥。密钥对应仅在每个设备上生成一次，并可以在执行配对之前计算。

支持以下配对密钥生成方法：

1. 使用带外通信来建立公钥的带外（OOB）法。如果带外位在配对请求/响应中至少由一个设备设置，则选择该方法，否则必须使用设备的 IO 功能来确定可以使用其他方法（输入密钥法、直接工作法或数字比较法）
2. Just Works：该方法无需验证，并且不提供针对中间人（MITM）攻击的任何保护
3. 输入密钥法：该方法需要认证。用户输入六位数字。该六位数字是设备认证的基础
4. 数字比较：该方法需要认证。两种设备都将 IO 能力设为是/否或键盘显示。两个设备都计算在两个设备上向用户显示的六位数字确定值：用户需要输入是或否来确认是否匹配。如果在两个设备上都选择是，则配对执行成功。当多个设备具有相同名字的情况下，此方法可以确保用户设备与正确的设备连接

根据下表在可能的方法中选择。

表 12. 将 IO 功能映射到可能的密钥生成方法

发起方/响应方	仅显示	显示是/否	仅键盘	无输入，无输出	键盘，显示
仅显示	直接工作	直接工作	输入密钥	直接工作	输入密钥
显示是/否	直接工作	直接工作 （LE 传统） 数字比较（LE 安全连接）	输入密钥	直接工作	输入密钥（LE 传统） 数字比较（LE 安全连接）
仅键盘	输入密钥	输入密钥	输入密钥	直接工作	输入密钥
无输入，无输出	直接工作	直接工作	直接工作	直接工作	直接工作
键盘，显示	输入密钥	输入密钥 （LE 传统） 数字比较（LE 安全连接）	输入密钥	直接工作	输入密钥（LE 传统） 数字比较（LE 安全连接）

注意： *如果可能的密钥生成方法未提供与安全属性匹配的密钥（经过身份验证 - MITM 保护或未经过身份验证 - 无 MITM 保护），则设备将发送配对失败指令，并显示“验证要求”的错误代码。*

第 3 阶段：传输特定密钥分配

主设备和从设备之间交换以下密钥：

- 用于验证未加密数据的连接签名解析密钥（CSRK）
- 用于设备身份和隐私的身份解析密钥（IRK）

当设备保存确定的加密密钥以用于未来验证时，设备被绑定。

数据签名

还可以使用 CSRK 密钥，通过未加密的链路层连接传输已验证数据：12 字节签名位于数据有效负载之后。签名算法还使用可防止重放攻击的计数器（一种外部设备，可以只获取某些数据包然后发送，无需对数据包内容有任何理解：接收设备只检查数据包计数器并丢弃，因为它的帧计数器少于最近接收的良好数据包）。

2.8 隐私

总是用相同地址信息（公共地址或静态随机地址）的广播设备，很容易被其他扫描设备追踪。为了防止这种情况发生，可以在广播设备上启用私有功能。在启用了私有的设备上，将使用私有地址。有两种私有地址：

- 不可解析私有地址
- 可解析私有地址

不可解析私有地址是完全随机的（两个最高有效位除外），不能进行解析。因此，使用不可解析私有地址的设备不能被没有预先配对的设备识别。可解析私有地址由 24 位随机部分和哈希部分组成。哈希部分来源于随机数和 IRK（身份解析密钥）。因此，只有知晓该 IRK 的设备能够解析地址并识别该设备。在配对过程中分发 IRK。

两种类型的地址都经常改变，从而增强了设备身份机密性。在 GAP 发现模式和过程期间不使用私有功能，仅在 GAP 连接模式和过程期间使用私有功能。

在 v4.1 及以下的低功耗蓝牙协议栈中，通过主机解析和产生私有地址。在蓝牙 4.2 中，私有功能已从版本 1.1 更新到版本 1.2。在低功耗蓝牙协议栈 v4.2 中，控制器可以使用主机提供的设备身份信息来解析和生成私有地址。

外设

启用了私有功能的处于不可连接模式的外设将使用不可解析或可解析私有地址。

如需连接中央设备，外设设备中如启用主机私有，则必须且只能使用非定向可连接模式。如启用控制器私有，则外设设备还可使用定向可连接模式。在可连接模式中，外设设备使用可解析私有地址。

无论使用的是不可解析还是可解析私有地址，均以 15 分钟的间隔时间自动重新生成。设备不在要发送的广播数据中包含设备名称。

中央设备

启用了私有功能的执行主动扫描的中央设备只使用不可解析或可解析私有地址。为了连接外设，如果启用了主机私有，应使用通用连接建立流程。对于基于控制器的私有，可使用任何连接流程。中央设备将使用可解析私有地址作为发起方的设备地址。每间隔 15 分钟后生成可解析或不可解析私有地址。

广播设备

启用了私有的广播设备使用不可解析或可解析私有地址。每间隔 15 分钟后将自动生成新地址。广播设备不应在要发送的广播数据中包含设备名称或唯一性数据。

监听设备

启用了私有的监听设备使用不可解析或可解析私有地址。每间隔 15 分钟后将自动生成新地址。

2.8.1 设备过滤

蓝牙 LE 可通过减少设备的响应数量来降低功耗，因为这样可以减少控制器与上层之间的传输和交互。设备过滤通过白名单来实现。启用白名单后，链路层将忽略不在白名单中的设备。

在蓝牙 4.2 之前，当远程设备使用私有功能后，无法实现设备过滤。由于引入了链路层私有，可以在检查远程设备身份地址是否在白名单中之前解析远程设备身份地址。

通过将“Filter_Duplicates”模式设为 1，用户可以激活 LL 层级的广播过滤。具体工作如下所述。

LL 保留两组缓存区（每组 4 个缓存区）：一组用于 4 个广播指示地址，另一组用于 4 个扫描响应地址。

当收到广播指示数据包时，将比较其地址（6 字节）与存储的 4 个地址。如果与其中一个地址匹配，则丢弃此数据包。如果不匹配，则将指示报告至上层并将其地址存储到缓冲区，同时删除缓冲区中最早的地址。

相同的过程分别适用于扫描响应。

2.9 通用属性配置文件（GATT）

通用属性配置文件（GATT）定义了使用 ATT 协议的框架，它被用于服务、特征、描述符发现、特征读取、写入、指示和通知。

就 GATT 而言，当两个设备已经连接时，有两种设备角色：

- GATT 客户端：通过读取、写入、通知或指示操作访问远程 GATT 服务器上的数据的设备
- GATT 服务器：在本地保存数据并向远程 GATT 客户端提供数据访问方法的设备

一个设备可以既是 GATT 服务器又是 GATT 客户端。

设备的 GATT 角色在逻辑上独立于主、从角色。主、从角色定义了 BLE 无线连接的管理方式，而 GATT 客户端/服务器角色由数据存储和数据流动来决定。

因此，不要求从（外围）设备必须是 GATT 服务器以及主（中央）设备必须是 GATT 客户端。

ATT 传输的属性封装在下列基础类型中：

1. 特征（具有相关描述符）
2. 服务（主要、次要和包含）

2.9.1 特征属性类型

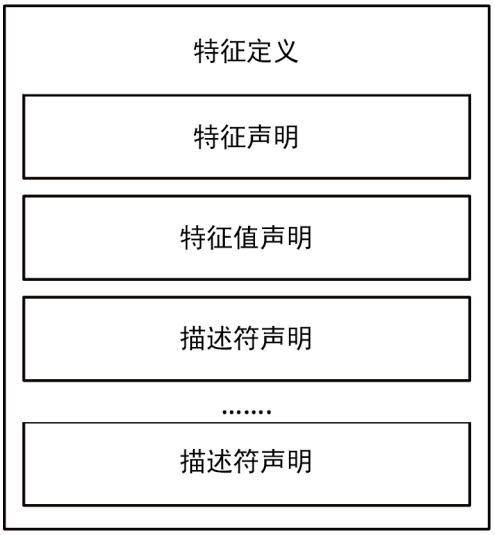
特征是一种包含单个值和任意数量描述符的属性类型，描述符描述的特征值使用户能够理解该特征。

特征揭示了值代表的数据类型、值是否能够读取或写入以及如何配置要指示或通知的值，它还描述了值的含义。

特征具有以下组成部分：

1. 特征声明
2. 特征值
3. 特征描述符

图 7. **特征定义示例**



特征声明是一种属性，其定义如下：

表 13. **特征声明**

属性句柄	属性类型	属性值	属性权限
0xNNNN	0x2803 （特征属性类型的 UUID）	特征值属性（读取、广播、写入、写入但不响应、通知、指示等）。确定如何能够使用特征值或如何能够访问特征描述符	“只读，无验证，无授权”
		特征值属性句柄	
		特征值 UUID（16 或 128 位）	

特征声明包含特征值。该值是特征声明后的第一个属性：

表 14. 特征值

属性句柄	属性类型	属性值	属性权限
0xNNNN	0xuuuu – 16 位或 128 位（特征 UUID）	特征值	取决于更高层配置文件或应用

2.9.2 特征描述符类型

特征描述符用于描述特征值，以便为特征添加特定“含义”，使客户能够理解特征。有以下特征描述符可供使用：

1. 特征扩展属性：允许为特征添加扩展属性
2. 特征用户描述：使设备能够将文本字符串关联到特征
3. 客户端特征配置：如果特征可以通知或指示，为强制要求。客户端应用必须写入该特征描述符以使特征通知或指示成为可能（前提是特征属性允许通知或指示）
4. 服务器特征配置：可选描述符
5. 特征表达格式：它允许通过一些字段（例如，格式、指数、单位命名空间、描述）定义特征值表达格式，以便正确显示相关值（例如，oC 格式的温度测量值）
6. 特征聚合格式：可以聚合多种特征表达格式。

关于特征描述符的详细描述，请参考蓝牙规范。

2.9.3 服务属性类型

服务是特征的集合，共同为一个可实现的应用配置文件提供通用服务类型。例如，健康体温计服务包含温度测量值特征和每次测量的间隔时间特征。服务或主要服务可以引用被称为次要服务的其他服务。

服务的定义如下：

表 15. 服务声明

属性句柄	属性类型	属性值	属性权限
0xNNNN	0x2800 – “主要服务”的 UUID，或 0x2801 – “次要服务”的 UUID	0xuuuu – 16 位或 128 位（服务 UUID）	“只读，无验证，无授权”

服务包含服务声明，其他服务包含定义（可选）和特征定义。服务包含声明位于服务声明之后。

表 16. 包含声明

属性句柄	属性类型	属性值			属性权限
0xNNNN	0x2802（包含属性类型的 UUID）	包含服务属性句柄	结束组句柄	服务 UUID	“只读，无验证，无授权”

“包含服务属性句柄”是所包含次要服务的属性句柄，“结束组句柄”是所包含次要服务中最后一个属性的句柄。

2.9.4 GATT 流程

通用属性配置文件（GATT）定义了一组发现服务、特征和相关描述符的标准流程以及它们的使用方法。有以下流程可供使用：

- 发现流程（表 17. 发现流程和相关响应事件）
- 客户端发起的流程（表 18. 客户端发起的流程和相关响应事件）
- 服务器发起的流程（表 19. 服务器发起的流程和相关响应事件）

表 17. 发现流程和相关响应事件

步骤	响应事件
发现所有主要服务	按组读取响应
通过服务 UUID 发现主要服务	按类型值响应查找
查找包含的服务	按类型响应事件读取
发现服务的所有特征	按类型响应读取
通过 UUID 发现特征	按类型响应读取
发现所有特征描述符	查找信息响应

表 18. 客户端发起的流程和相关响应事件

步骤	响应事件
读取特征值	读取响应事件
通过 UUID 读取特征值	读取响应事件
读取长特征值	读取 BLOB 响应事件
读取多个特征值	读取响应事件
写入特征值，无响应	不生成事件
签名写入，无响应	不生成事件
写入特征值	写入响应事件
写入长特征值	准备写入响应 执行写入响应
可靠写入	准备写入响应 执行写入响应

表 19. 服务器发起的流程和相关响应事件

流程	响应事件
通知	不生成事件
指示	确认事件

关于 GATT 流程和相关响应事件的详细描述，请参考第 6 节 参考文件中的蓝牙规范。

2.10 通用访问配置文件（GAP）

蓝牙系统为所有的蓝牙设备定义了一个基础配置文件，叫做通用访问配置文件（GAP）。此配置文件定义了一个蓝牙设备所需具备的基本要求。

下表中描述了四种 GAP 配置文件的角色：

表 20. GAP 角色

角色[1]	描述	发送器	接收器	典型示例
广播设备	发送广播事件	M	O	发送温度值的温度传感器
监听设备	接收广播事件	O	M	只接收和显示温度值的温度显示装置
外设	始终为从设备。 处于可连接广播模式。 支持所有 LL 控制流程；可选择加密或不加密	M	M	观看
中央设备	始终为主设备。 从不广播。 支持主动或被动扫描。支持所有 LL 控制流程；可选择加密或不加密	M	M	移动电话

1. *1.M = 强制；O = 可选*

在 GAP 中，定义了两个基本概念：

- GAP 模式：配置设备，使之以特定方式长时间工作。GAP 模式有四种类型：广播、可发现、可连接和可绑定类型
- GAP 流程：配置设备，使之在特定的时间段或有限时间内执行某一动作。有四种类型的 GAP 流程：监听、发现、连接和绑定流程

不同类型的可发现和可连接模式可以同时使用。定义的 GAP 模式如下：

表 21. GAP 广播器模式

模式	说明	注释	GAP 角色
广播模式	设备仅使用链路层广播通道和数据包广播数据（不在 AD 类型标记上设置任何位）	设备可使用监听流程检测广播数据	广播设备

表 22. GAP 可发现模式

模式	说明	注释	GAP 角色
非可发现模式	不能在 AD 类型标记上设置有限和一般可发现位	不能通过执行通用和有限发现流程的设备对其进行发现	外设
有限可发现模式	在 AD 类型标记上设置有限可发现位	允许约 30 s。由用户最近交互的设备使用。例如用户按下设备上的按钮	外设
一般可发现模式	在 AD 类型标记上设置一般可发现位	在设备希望成为可发现设备时使用。对可发现时间无限制	外设

表 23. GAP 可连接模式

模式	说明	注释	GAP 角色
不可连接模式	只能使用 ADV_NONCONN_IND 或 ADV_SCAN_IND 广播数据包	在广播时不能使用可连接广播数据包	外设
直接可连接模式	使用 ADV_DIRECT 广播数据包	由要快速连接中央设备的外设使用。只能使用 1.28 秒，并且需要外设和中央设备地址	外设
非定向可连接模式	使用 ADV_IND 广播数据包	在设备希望成为可连接设备时使用。由于 ADV_IND 广播数据包可以包含 AD 类型标记，因此设备可同时处于可发现和非定向可连接模式。 当设备进入连接模式或不可连接模式时，可连接模式终止	外设

表 24. GAP 可绑定模式

模式	说明	注释	GAP 角色
不可绑定模式	不允许与对端设备建立绑定	设备不保存密钥	外设
可绑定模式	设备接受来自中央设备的绑定请求。	-	外设

表 25. GAP 监听流程中定义了以下 GAP 流程：

表 25. GAP 监听流程

流程	说明	注释	角色
监听流程	它允许设备搜索广播设备数据	-	监听设备

表 26. GAP 发现流程

流程	说明	注释	角色
有限可发现流程	用于在有限发现模式下发现外设	根据 AD 类型标记信息应用设备过滤	中央设备
通用可发现流程	用于在通用和有限发现模式下发现外设	根据 AD 类型标记信息应用设备过滤	中央设备
名称发现流程	用于从可连接设备检索“蓝牙设备名称”的流程	-	中央设备

为了实现名称发现 GAP 流程，用户可以执行以下操作：

- 调用 ACI_GAP_CREATE_CONNECTION 指令
- 等待 HCI_LE_CONNECTION_COMPLETE_EVENT
- 使用以下参数调用 ACI_GATT_READ_USING_CHAR_UUID 指令：
  - Start_Handle = 0x0001
  - End_Handle = 0xFFFF
  - UUID_Type = 1
  - UUID = DEVICE_NAME_UUID

注意： *此指令可以替代不再支持的 ACI_GAP_START_NAME_DISCOVERY_PROC 指令。*

表 27. GAP 连接流程

流程	说明	注释	角色
自动连接建立流程	允许连接一个或多个处于定向可连接模式或非定向可连接模式的设备	使用白名单	中央设备
通用连接建立流程	允许连接一组处于定向可连接模式或非定向可连接模式的已知对端设备	当在被动扫描期间检测到具有私有地址的设备时，它使用直接连接建立流程支持私有地址	中央设备
选择性连接建立流程	使用主机的选定连接配置参数与白名单中的一组设备建立连接	使用白名单，并按照该白名单进行扫描	中央设备
直接连接建立流程	使用一组连接间隔参数与特定设备建立连接	由通用和选择性流程使用	中央设备
连接参数更新流程	在连接期间更新使用的连接参数	-	中央设备
终止流程	终止 GAP 流程	-	中央设备

表 28. GAP 绑定流程

流程	说明	注释	角色
绑定流程	在配对请求上设置了绑定位的情况下，启动配对过程	-	中央设备

关于 GAP 流程的详细描述，请参考蓝牙规范。

2.11 BLE 配置文件和应用

服务集合一组特征并公开这些特征的行为（设备的操作内容，而不是设备如何使用它们）。服务不定义特征用例。用例决定了需要的服务（如何在设备上使用服务）。这是通过配置文件实现的，它定义了特定用例需要的服务：

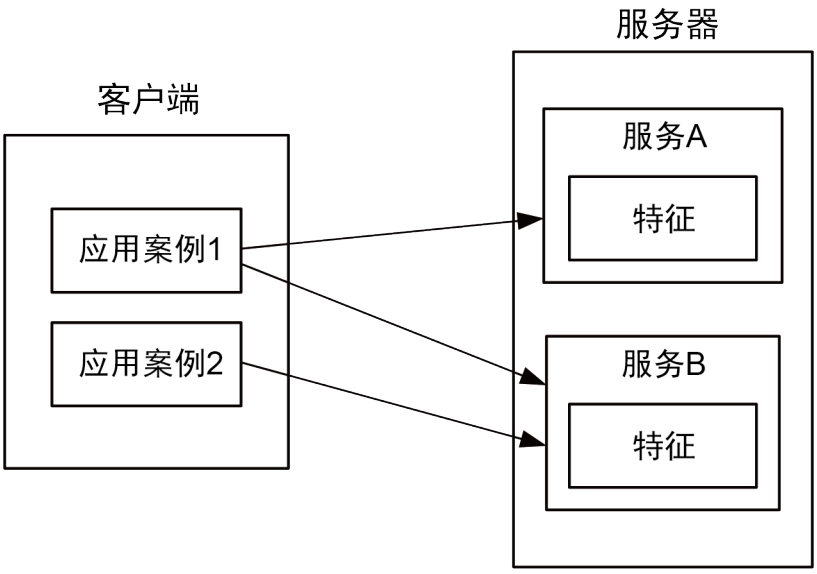
- 配置文件客户端实现用例
- 配置文件服务器实现服务

可以使用标准配置文件或专有配置文件。在使用非标准配置文件时，需要 128 位 UUID，并且必须是随机生成的。

目前，任何标准蓝牙 SIG 配置文件（服务和特征）均使用 16 位 UUID。可从以下 SIG 网页下载服务和特征规范及 UUID 分配：

- https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx
- https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx

图 8. 客户端和服务器配置文件



- 应用案例1使用服务A和B
- 应用案例2使用服务B

2.11.1 接近感测示例

本节将简单介绍接近感测的目标、工作原理和需要的服务：

目标

- 当一个设备由近及远，到一定距离时：
  - 导致报警

工作原理

- 如果设备断开，则会导致报警
- 链路丢失报警：«链路丢失» 服务
  - 如果设备距离太远
  - 导致路径丢失报警：«即时报警»和«发送功率»服务
- «链路丢失» 服务
  - «报警级别» 特征
  - 行为：链路丢失时，因被枚举导致报警
- «即时报警» 服务
  - «报警级别» 特征
  - 行为：写入时，因被枚举导致报警
- «发送功率» 服务
  - «发送功率» 特征
  - 行为：读取时，报告连接的当前发送功率

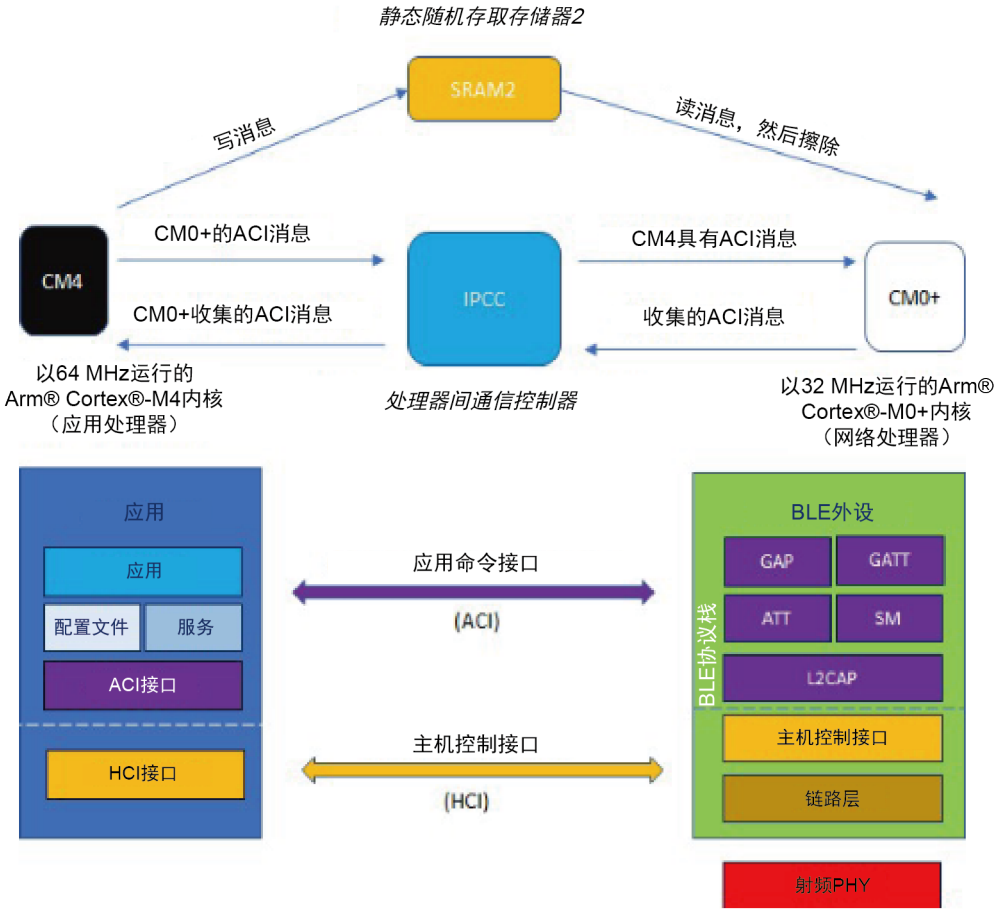
3 STM32WB 低功耗蓝牙协议栈

STM32WB 设备是网络协处理器，提供高层接口以控制其低功耗蓝牙功能。该接口被称为 ACI（应用指令接口）。STM32WB 系列设备内置了 Arm Cotex-M0 核，各自安全地集成了蓝牙智能协议栈。因此，外部微控制器 Arm Cortex-M4 不需要 BLE 库。Cortex-M4 微控制器通过进程间通信控制器（IPCC）接口通信协议即可向 Cortex-M0 微控制器协处理器发送和接收 ACI 命令。目前的安全低功耗蓝牙（BLE）协议栈均基于二进制格式的标准 C 库。发送任何 BLE 指令前，Cortex-M4 首先将系统指令 SHCI_C2_BLE_Init()发送到 Cortex-M0，以启动 BLE 协议栈。有关系统指令和 BLE 启动流程的更多说明，请参考 AN5289。

BLE 二进制库提供以下功能：

- 用于 BLE 协议栈初始化、BLE 协议栈应用指令接口（HCI 指令前缀为 hci_，供应商特定指令前缀为 aci_）、睡眠定时器访问和 BLE 协议栈状态机处理的协议栈 API
- 协议栈事件回调通知用户应用有关 BLE 协议栈事件和睡眠定时器事件
- 为无线电 IP 提供中断处理程序

图 9. 安全 Arm Cortex-M0 与 Arm Cortex-M4 之间的 STM32WB 协议栈架构和接口



3.1 BLE 协议栈库框架

BLE 协议栈库框架允许将指令发送至 STM32WB SoC BLE 协议栈，并且还提供了 BLE 事件回调函数定义。BLE 协议栈 API 利用并扩展了在蓝牙规范中定义的标准 HCI 数据格式。

提供的一组 API 支持以下命令：

- 蓝牙规范定义的控制器标准 HCI 指令
- 供应商特定的（VS）控制器 HCI 指令
- 供应商特定的（VS）主机 HCI 指令（L2CAP, ATT, SM, GATT, GAP）

STM32WB 套件软件包中提供了参考 ACI 接口框架（请参考第 6 节“参考文档”）。ACI 接口框架包含用于在两个 STM32WB 处理器：Arm® Cortex®-M0（网络处理器）和以 64 MHz 运行的 Arm® Cortex®-M4 内核（应用处理器）之间发送 ACI 指令的代码。它还提供设备事件的定义。以下头文件定义了 ACI 框架接口：

表 29. BLE 应用协议栈库框架接口

文件	说明
ble_hci_le.h	HCI 库函数原型和错误代码定义。
ble_events.h	包含 STM32WB FW 协议栈的指令和事件的头文件
ble_gatt_aci.h	GATT 服务器定义的头文件
ble_l2cap_aci.h	包含 STM32WB FW 协议栈的 L2CAP 指令的头文件
ble_gap_aci.h	STM32WB GAP 层的头文件
ble_hal_aci.h	包含 STM32WB FW 协议栈的 HCI 指令的头文件
ble_types.h	包含 STM32WB FW 协议栈的 ACI 定义的头文件

4 使用 STM32WB BLE 协议栈设计应用

本节提供关于如何使用 BLE 协议栈 v2.x 二进制库在 STM32WB 设备上设计和实现低功耗蓝牙应用的信息和代码示例。

在 STM32WB 设备上实现 BLE 应用的用户必须完成一些基本的常见步骤：

1. 初始化阶段和应用程序主循环
2. STM32WB 事件和事件回调设置
3. 服务和特征配置（在 GATT 服务器上）
4. 创建连接：可发现、可连接模式与流程
5. 安全（配对和绑定）
6. 服务和特征发现
7. 特征通知/指示、写入、读取
8. 基本/典型错误条件描述

注意： *在后面的章节中，将使用一些用户应用“定义”，以便轻松识别设备低功耗蓝牙的角色（中央设备、外设、客户端和服务器）。*

欲了解 BLE 协议栈初始化和 BLE 协议栈 NVM 使用的更多详情，请参考 AN5289。

表 30. BLE 设备角色的用户应用定义

定义	说明
GATT_CLIENT	GATT 客户端角色
GATT_SERVER	GATT 服务器角色

4.1 初始化阶段和应用程序主循环

正确配置 STM32WB 设备需执行以下主要步骤。

1. 初始化 HAL 库：
   a. 配置 Flash 预取、指令缓存和数据缓存。
   b. 通过配置 SysTick 来产生 1 毫秒的中断，该中断由 MSI 计时（在该阶段，时钟尚未配置，因此系统通过内部 MSI 以 4 MHz 的频率运行）。
   c. 将 NVIC 组优先级设为 4。
   d. 调用用户文件“stm32wbxx_hal_msp.c”中定义的 HAL_MspInit()回调函数，以执行全局低级硬件初始化
2. 配置系统时钟
3. 配置外设时钟
4. 配置系统功耗模式
5. 初始化所有配置的外设
6. APPE_Init()：
   a. 配置系统功耗模式
   b. 初始化定时器服务
   c. 初始化调试
   d. 初始化所有传输层
7. 增加一个调用 UTIL_SEQ_Run(UTIL_SEQ_DEFAULT)的 while(1)循环
   a. 处理用户操作/事件（广播、连接、服务和特征发现、通知和相关事件）的定序器。

以下伪代码示例描述了需要的初始化步骤：

```
int main(void)
{/* 复位所有外设，初始化 Flash 接口和 Systick。*/ HAL_Init();
/* USER CODE BEGIN Init */ Reset_Device(); Config_HSE();
/* USER CODE END Init */
/* 配置系统时钟 */ SystemClock_Config();
/* USER CODE BEGIN SysInit */ PeriphClock_Config();
Init_Exti(); /**< 配置系统功耗模式 */
/* USER CODE END SysInit */
/* 初始化所有配置的外设 */ MX_GPIO_Init(); MX_DMA_Init(); MX_RF_Init();
MX_RTC_Init(); APPE_Init();
/* 无限循环 */ while(1){
UTIL_SEQ_Run( UTIL_SEQ_DEFAULT ); }
}/* 结束 main() */
```

注意：

1. *当执行 GATT_Init() & GAP_Init() API 时，STM32WB 协议栈始终会添加两个标准服务：包含服务更改特征的属性配置文件服务（0x1801）以及包含设备名称和外观特征的 GAP 服务（0x1800）。*
2. *当未在 aci_gap_init() API 上启用私有或基于主机的私有时，为标准 GAP 服务保留的最后一个属性句柄为 0x000B，若在 aci_gap_init() API 上启用基于控制器的私有，则为 0x000D。*

表 31. GATT、GAP 默认服务

默认服务	开始句柄	结束句柄	服务 UUID
属性配置文件服务	0x0001	0x0004	0x1801
通用访问配置文件（GAP）服务	0x0005	0x000B	0x1800

表 32. GATT、GAP 默认特征

默认服务	特征	句柄属性	特征属性	特征值句柄	特征 UUID	特征值长度（字节）
属性配置文件服务						
	服务更改	0x0002	指示	0x0003	0x2A05	4
通用访问配置文件（GAP）服务	-	-	-	-	-	-
-	设备名称	0x0006	读取|写入，无响应 | 写入 | 验证签名写入	0x0007	0x2A00	8
-	外观	0x0008	读取|写入，无响应 | 写入 | 验证签名写入	0x0009	0x2A01	2
-	外设首选的连接参数	0x000A	读/写	0x000B	0x2A04	8
-	中央设备地址解析(1)	0x000C	无需身份验证或授权即可读取。 不可写	0x000D	0x2AA6	1

1. *仅在 aci_gap_init() API 上启用基于控制器的隐私（0x02）时添加。*

aci_gap_init()角色参数值如下：

表 33. aci_gap_init()角色参数值

参数	角色参数值	注释
角色	0x01：外设设备 0x02：广播设备 0x04：中央设备 0x08：监听设备	角色参数可以是支持的任意值的按位或（同时支持多个角色）
enable_Privacy	0x00 用于禁用隐私； 0x01 用于启用隐私； 0x02 用于启用基于控制器的主机隐私	-
device_name_char_len	-	它允许指定设备名称特性的长度。

关于该 API 和相关参数的完整描述，请参考第 6 节 参考文件中的蓝牙 LE 协议栈 API 和事件文档。

4.1.1 BLE 地址

STM32WB 设备支持以下设备地址：

- 公共地址
- 随机地址
- 私有地址

公共 MAC 地址（6 字节- 48 位地址）唯一标识了 BLE 设备，它们由电气和电子工程师协会（IEEE）定义。

公共地址的前 3 个字节标识了发布标识符的公司，称为组织唯一标识符（OUI）。组织唯一标识符（OUI）是一种从 IEEE 购买的 24 位数字。该标识符唯一标识了某个公司，并可以为公司独家使用特定 OUI 预留一组可能的公共地址（最多 2^24 个，来自公共地址的其余 3 个字节）。

当之前分配的地址组的至少 95%已被使用时，任何组织/公司均可以请求一组新的 6 字节地址（最多 2^24 个，通过特定 OUI 提供可能的地址）。

公共地址保持静态和唯一，用户仅可读取。

如果用户想要编写自己的自定义 MAC 地址，则可以通过应用利用在 OTP 中定义的有效预分配 MAC 地址设置特定的公共地址。

设置 MAC 地址的 ACI 指令是 ACI_HAL_WRITE_CONFIG_DATA（操作码：0xFC0C），其指令参数如下：

- 偏移：0x00（0x00 标识 BTLE 公共地址，例如 MAC 地址）
- 长度：0x06（MAC 地址的长度）
- 值：0xaabbccddeeff（MAC 地址的 48 位数组）

启动 BLE 操作前（每次通电或复位后）应发送 ACI_HAL_WRITE_CONFIG_DATA 指令。

以下伪代码示例描述了如何设置公共地址：

```
uint8_t bdaddr[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
ret=aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,CONFIG_DATA_PUBAD DR_LEN, bdaddr);
if(ret)PRINTF("Setting address failed.\n")}
```

STM32WB 设备没有有效的预分配 MAC 地址，但有唯一的序列号（对用户为只读）。唯一序列号是存储在地址 0x100007F4 中的六字节值：它作为两个字（8 字节）存储在地址 0x100007F4 和 0x100007F8 中，并用 0xAA55 填充唯一序列号。

静态随机地址是在设备第一次启动时生成并被写入 Flash 的指定位置。Flash 上的值是设备使用的实际值：用户每次复位设备时，协议栈都会检查专用 Flash 区域是否存在有效数据并使用该数据（FLASH 上的特殊有效标记用于标识是否存在有效数据）。如果用户执行了批量擦除，则会删除存储值（包括标记），下一次设备重启时又会生成新的 MAC 地址，并存储到 Flash 的指定位置上。

当启用了私有时，按照低功耗蓝牙规范使用私有地址。欲了解私有地址的更多信息，请参考第 2.7 节“安全管理器（SM）”。

4.1.2 设置发送功率

在初始化阶段，用户还可以使用以下 API 选择发送功率：

aci_hal_set_tx_power_level（高功率水平）

下面是将无线发送功率设置为高功率和--2 dBm 输出功率的伪代码示例：

ret= aci_hal_set_tx_power_level (1,4);

关于该 API 和相关参数的完整描述，请参考第 6 节 参考文件中的蓝牙 LE 协议栈 API 和事件文档。

4.2 服务和特征配置

一个服务必须配置专用句柄、属性类型、UUID 和权限。

如前文第 2.9 节“通用属性配置文件（GATT）”所述，一个服务可以具有很多不同的特征，为此，也可以有很多不同的句柄。

特征应始终依附于/取决于服务。

共有 4 种不同的特征“类型”，每种类型都具有各自的句柄，客户可以逐一选择：

- 特征扩展属性
- 特征声明属性
- 特征值属性
- 客户端特征配置描述符（CCCD）。

任意 CCCD 的 UUID 始终是标准的 16 位 UUIDCCCD（0x2902）

为了添加服务和相关特征，用户应用必须定义要寻址的特定配置文件：

1. 标准配置文件由蓝牙 SIG 组织定义。用户必须遵循配置文件规范和服务、特征规范文档，以便使用定义的相关配置文件、服务和特征 16 位 UUID 对其进行实现（请参考蓝牙 SIG 网页：www.bluetooth.org/en-%20us/specification/adopted-specifications）。
2. 专有的非标准配置文件。用户必须定义其自己的服务和特征。在本例中，需要 128 位 UUID，并且由用户自己生成（请参考 UUID 生成器网页：www.famkruithof.net/uuid/uuidgen）。
3. 默认有两种服务，必须包括这些服务以及专用特征，详情如下：
   - 通用访问服务：
     - UUID 0x1800 服务及其三个强制特征：
       - 特征：设备名称。UUID 0x2A00.
       - 特征：外观。UUID 0x2A01.
       - 特征：外设首选的连接参数。UUID 0x2A04.
   - 通用属性服务。
     - UUID 0x1801 以及一个可选特征：
       - 特征：服务更改。UUID 0x2A05.

可使用以下指令添加服务：

```
aci_gatt_add_service(uint8_t Service_UUID_Type,
                     Service_UUID_t *Service_UUID,
                     uint8_t Service_Type,
                     uint8_t Max_Attribute_Records,
                     uint16_t *Service_Handle);
```

该指令返回服务句柄的指针（Service_Handle），用于标识用户应用中的该服务。可使用以下指令为该服务添加特征：

```
aci_gatt_add_char(uint16_t Service_Handle,
                  uint8_t Char_UUID_Type,
                  Char_UUID_t *Char_UUID,
                  uint8_t Char_Value_Length,
                  uint8_t Char_Properties,
                  uint8_t Security_Permissions,
                  uint8_t GATT_Evt_Mask,
                  uint8_t Enc_Key_Size,
                  uint8_t Is_Variable,
                  uint16_t *Char_Handle);
```

该指令返回特征句柄的指针（Char_Handle），用于标识用户应用中的该特征。

以下伪代码示例描述了在专有非标准配置文件中添加服务和两个关联的特征所要执行的步骤。

```
/* 服务和特征 UUID 变量。*/
Service_UUID_t service_uuid;
Char_UUID_t char_uuid;


tBleStatus Add_Server_Services_Characteristics(void)
{
    tBleStatus ret = BLE_STATUS_SUCCESS;
    /*
    以下 128 位 UUID 由随机 UUID 生成器生成：
    D973F2E0-B19E-11E2-9E96-0800200C9A66:服务 128 位 UUID
    D973F2E1-B19E-11E2-9E96-0800200C9A66:Characteristic_1 128bits UUID
    D973F2E2-B19E-11E2-9E96-0800200C9A66:Characteristic_2 128bits UUID
    */
    /*服务 128 位 UUID */
    const uint8_t uuid[16] =
    {0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe0,0xf2,0x73,0xd9};
    /*Characteristic_1 128bits UUID */
    const uint8_t charUuid_1[16] =
    {0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe1,0xf2,0x73,0xd9};
    /*Characteristic_2 128bits UUID */
    const uint8_t charUuid_2[16] =
    {0x66,0x9a,0x0c,0x20,0x00,0x08,0x96,0x9e,0xe2,0x11,0x9e,0xb1,0xe2,0xf2,0x73,0xd9};
    Osal_MemCpy(&service_uuid.Service_UUID_128, uuid, 16);
    /* 将具有 service_uuid 128 位 UUID 的服务添加到 GATT 服务器数据库。返回服务句柄 Service_Handle。
    */
    ret = aci_gatt_add_service(UUID_TYPE_128, &service_uuid, PRIMARY_SERVICE, 6, &Service_Handle);
    if(ret != BLE_STATUS_SUCCESS) return(ret);
    Osal_MemCpy(&char_uuid.Char_UUID_128, charUuid_1, 16);

    /*  将具有 charUuid_1 128 位 UUID 的特征添加到服务 Service_Handle。该特征具有最大长度为 20 字节的特征值、
        通知属性（CHAR_PROP_NOTIFY）、无安全许可（ATTR_PERMISSION_NONE）、无 GATT 事件掩码(0)、
        加密密钥长度 16 以及长度可变的特征(1)。
        返回特征句柄（CharHandle_1）。
        */
    ret = aci_gatt_add_char(Service_Handle, UUID_TYPE_128, &char_uuid, 20,
                            CHAR_PROP_NOTIFY, ATTR_PERMISSION_NONE, 0,16, 1,
                            &CharHandle_1);
    if (ret != BLE_STATUS_SUCCESS) return(ret);
    Osal_MemCpy(&char_uuid.Char_UUID_128, charUuid_2, 16);

    /*  将具有 charUuid_2 128 位 UUID 的特征添加到服务 Service_Handle。该特征具有最大长度为 20 字节的特征值、
        读取/写入/写入且无响应属性、无安全许可（ATTR_PERMISSION_NONE）、在将 GATT 事件掩码作为属性写入
        时通知应用（GATT_NOTIFY_ATTRIBUTE_WRITE）、加密密钥长度 16 以及长度可变的特征（1）。返回特征
        句柄（CharHandle_2）。
    */
    ret = aci_gatt_add_char(Service_Handle, UUID_TYPE_128, &char_uuid, 20,
                            CHAR_PROP_WRITE|CHAR_PROP_WRITE_WITHOUT_RESP,
                            ATTR_PERMISSION_NONE, GATT_NOTIFY_ATTRIBUTE_WRITE,
                            16, 1, &&CharHandle_2);
    if (ret != BLE_STATUS_SUCCESS)return(ret) ;
}/*结束 Add_Server_Services_Characteristics() */
```

4.3 创建连接：可发现和可连接 API

为了在 BLE GAP 中央设备（主）和 BLE GAP 外围（从）设备之间建立连接，可以使用表 34. GAP 模式 API、表 35. GAP 发现流程 API 以及表 36. 连接流程 API 中所述的 GAP 可连接/可发现模式和流程，并使用提供的相关 BLE 协议栈 API。

GAP 外设可发现和可连接模式 API

可按照以下 API 的描述使用不同类型的可发现和可连接模式：

表 34. GAP 模式 API

API	支持的广播事件类型	说明
aci_gap_set_discoverable()	0x00：可连接非定向广播（默认）	将设备设置为一般可发现模式。 设备处于可发现模式，直至设备调用 aci_gap_set_non_discoverable() API。
	0x02：可扫描非定向广播	
	0x03：不可连接非定向广播	
aci_gap_set_limited_discoverable()	0x00：可连接非定向广播（默认）	将设备设置为有限可发现模式。设备处于可发现模式的最长时间为 TGAP (lim_adv_timeout) = 180 秒。随时可通过调用 aci_gap_set_non_discoverable() API 禁用广播。
	0x02：可扫描非定向广播	
	0x03：不可连接非定向广播	
aci_gap_set_non_discoverable()	NA	将设备设置为不可发现模式。该指令禁用 LL 广播并将设备设置为待机状态。
aci_gap_set_direct_connectable()	NA	将设备设置为定向可连接模式。设备的定向可连接模式仅持续 1.28 秒。如果在此期间没有建立连接，设备将进入不可发现模式，并且必须明确地重新启用广播功能。
aci_gap_set_non_connectable()	0x02：可扫描非定向广播	使设备进入不可连接模式。
	0x03：不可连接非定向广播	
aci_gap_set_undirect_connectable ()	NA	使设备进入非定向可连接模式。

表 35. GAP 发现流程 API

API	说明
aci_gap_start_limited_discovery_proc()	启动有限发现流程。命令控制器开始主动扫描。当该流程启动时，仅处于有限可发现模式的设备返回上层。
aci_gap_start_general_discovery_proc()	启动一般发现流程。命令控制器开始主动扫描。

表 36. 连接流程 API

API	说明
aci_gap_start_auto_connection_establish_proc()	启动自动连接建立流程。主机将指定设备添加到控制器的白名单，在发起方过滤策略被设置为“使用白名单确定要连接的广播设备”的情况下，GAP 从控制器调用创建连接。
aci_gap_create_connection()	启动直接连接建立流程。在发起方过滤策略被设置为“忽略白名单，只处理特定设备的可连接广播数据包”的情况下，GAP 从控制器调用创建连接。
aci_gap_start_general_connection_establish_proc()	启动通用连接建立流程。在扫描过滤策略设置为“接受所有广播数据包”的情况下，主机在控制器中启用扫描，并使用事件 callback hci_le_advertising_report_event()将扫描结果中的所有设备发送至上层。
aci_gap_start_selective_connection_establish_proc()	启动选择性连接建立流程。GAP 将指定设备地址添加到白名单中，并在扫描过滤策略设置为“仅接受来自白名单中的设备的数据包”的控制器中启用扫描。通过事件回调 hci_le_advertising_report_event()将找到的所有设备发送至上层。
aci_gap_terminate_gap_proc()	终止指定的 GAP 流程。

4.3.1 设置可发现模式并使用直接连接建立流程

下列伪代码示例仅描述了使 GAP 外设进入一般可发现模式，以及通过直接连接建立流程将 GAP 中央设备直接连接到该 GAP 外设要执行的特定步骤。

注意：假设在初始化阶段设置了设备公共地址，如下所示：

```
uint8_t bdaddr[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
ret=aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,CONFIG_DATA_PUBAD DR_LEN, bdaddr);
if(ret != BLE_STATUS_SUCCESS)PRINTF(“Failure.\n”);

/* GAP 外设：一般可发现模式（不发送扫描响应）
*/

void GAP_Peripheral_Make_Discoverable(void )
{

   tBleStatus ret;
const charlocal_name[]=
{AD_TYPE_COMPLETE_LOCAL_NAME,’S’,’T’,’M’,’3’,’2’,’W’,’B’,’x’,’5’,’T’,’e’,’s’,’t’};/* 禁用
扫描响应：被动扫描 */ hci_le_set_scan_response_data (0,NULL);

   /*   禁用扫描响应：被动扫描 */
   hci_le_set_scan_response_data (0,NULL);

   /*   使 GAP 外设进入一般可发现模式：
        Advertising_Type:ADV_IND(undirected scannable and connectable);
        Advertising_Interval_Min:100;
        Advertising_Interval_Max:100;
        Own_Address_Type:PUBLIC_ADDR (public address:0x00);
        Adv_Filter_Policy:NO_WHITE_LIST_USE (no whit list is used);
        Local_Name_Lenght:14
        Local_Name:STM32WBx5Test;
        Service_Uuid_Length:0 (no service to be advertised); Service_Uuid_List:NULL;
        Slave_Conn_Interval_Min:0 (Slave connection internal minimum value);
        Slave_Conn_Interval_Max:0 (Slave connection internal maximum value).
        */

   ret = aci_gap_set_discoverable(ADV_IND, 100, 100, PUBLIC_ADDR,
                                  NO_WHITE_LIST_USE,
                                  sizeof(local_name), local_name,
                                  0, NULL, 0, 0);
   if (ret != BLE_STATUS_SUCCESS) PRINTF(“Failure.\n”);
} /* 结束 GAP_Peripheral_Make_Discoverable() */

/* GAP 中央设备：启动直接连接建立流程以连接处于可发现模式的 GAP 外设
*/

void GAP_Central_Make_Connection(void)

{
 /* 启动直接连接建立流程以连接处于一般可发现模式的 GAP 外设，使用以下连接参数：
    LE_Scan_Interval:0x4000;
    LE_Scan_Window:0x4000;
    Peer_Address_Type:PUBLIC_ADDR (GAP peripheral address type: public address);
    Peer_Address:{0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};
    Own_Address_Type:
    PUBLIC_ADDR (device address type);
    Conn_Interval_Min:40 (Minimum value for the connection event interval);
    Conn_Interval_Max:40 (Maximum value for the connection event interval);
    Conn_Latency:0 (Slave latency for the connection in a number of connection events);
    Supervision_Timeout:60 (Supervision timeout for the LE Link);
    Minimum_CE_Length:2000 (Minimum length of connection needed for the LE connection);
    Maximum_CE_Length:2000 (Maximum length of connection needed for the LE connection).

   */

    tBDAddr GAP_Peripheral_address = {0xaa, 0x00, 0x00, 0xE1, 0x80, 0x02};
    ret= aci_gap_create_connection(0x4000, 0x4000, PUBLIC_ADDR,
                                   GAP_Peripheral_address,PUBLIC_ADDR, 40,
                                   40,
                                   0, 60, 2000 , 2000);
    if(ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

}/* GAP_Central_Make_Connection(void )*/
```

注意：

1. *如果在 GAP 流程终止时返回了 ret = BLE_STATUS_SUCCESS，则调用 hci_le_connection_complete_event() 事件回调，以指示已与 GAP_Peripheral_address 建立连接（在 GAP 外设上返回相同事件）。*
2. *可通过发出 API aci_gap_terminate_gap_proc() 明确终止连接流程。*
3. *aci_gap_create_connection() 的最后两个参数 Minimum_CE_Length 和 Maximum_CE_Length 是 BLE 连接所需连接事件的长度。这些参数允许用户指定主设备必须为单个从设备分配的时间量，因此必须精心选择。具体来说，当主设备连接更多从设备时，每个从设备的连接间隔必须等于其他连接间隔或是它们的倍数，并且用户不得为每个从设备设置过长的连接事件长度。有关时序分配政策的详细信息，请参考第 5 节“BLE 多连接时序策略”。*

4.3.2 设置可发现模式并使用一般发现流程（主动扫描）

下列伪代码示例仅描述了使 GAP 外设进入一般可发现模式，以及 GAP 中央设备启动一般发现流程（以便发现无线传输范围内的设备）要执行的特定步骤。

注意：假设在初始化阶段设置了设备公共地址，如下所示：

```
      uint8_t bdaddr[] = {0x12, 0x34, 0x00, 0xE1, 0x80, 0x02};
      ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
                                      CONFIG_DATA_PUBADDR_LEN,
                                      bdaddr);
      if (ret != BLE_STATUS_SUCCESS)PRINTF("Failure.\n");

/* GAP 外设：一般可发现模式（发送扫描响应）：
*/
void GAP_Peripheral_Make_Discoverable(void)
{
      tBleStatus ret;
      const char local_name[] =
{AD_TYPE_COMPLETE_LOCAL_NAME,'S','T','M','3','2','W','B','x','5',}; /* 作为扫描响应数据，使用了一个专有 128 位服
务 UUID。
        此 128 位数据无法插入广播数据包（ADV_IND），理由是于长度受限（31 字节）。
        AD 类型描述：
        0x11：长度
        0x06：128 位服务 UUID 类型
        0x8a,0x97,0xf7,0xc0,0x85,0x06,0x11,0xe3,0xba,0xa7,0x08,0x00,0x20,0x0c,
        0x9a,0x66：128 位服务 UUID
      */
      uint8_t ServiceUUID_Scan[18]=
{0x11,0x06,0x8a,0x97,0xf7,0xc0,0x85,0x06,0x11,0xe3,0xba,0xa7,0x08,0x00,0x2,0x0c,0x9a,0x66};
/*    启用在 GAP 外设接收到
      来自 GAP 中央设备的执行一般
      发现流程（主动扫描）的扫描请求时发送扫描响应的功能 */

hci_le_set_scan_response_data(18,ServiceUUID_Scan);
/*    使 GAP 外设进入一般可发现模式：
      Advertising_Type:ADV_IND (undirected scannable and connectable);
Advertising_Interval_Min:100;
      Advertising_Interval_Max:100;
      Own_Address_Type:PUBLIC_ADDR (public address:0x00); Advertising_Filter_Policy:
NO_WHITE_LIST_USE (no whit list is used);
      Local_Name_Length:8
      Local_Name:STM32WB;
      Service_Uuid_Length:0 (no service to be advertised); Service_Uuid_List:NULL;
      Slave_Conn_Interval_Min:0 (Slave connection internal minimum value);
Slave_Conn_Interval_Max:0 (Slave connection internal maximum value).
      */
      ret = aci_gap_set_discoverable(ADV_IND, 100, 100, PUBLIC_ADDR,
                                      NO_WHITE_LIST_USE,sizeof(local_name),
                                      local_name, 0, NULL, 0, 0);
   if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

} /* 结束 GAP_Peripheral_Make_Discoverable() */

/* GAP 中央设备：启动一般发现流程以发现处于可发现模式的 GAP 外设 */
void GAP_Central_General_Discovery_Procedure(void)
{
tBleStatus ret;

/*    使用以下参数启动一般发现流程（主动扫描）：
     LE_Scan_Interval:0x4000;
     LE_Scan_Window:0x4000;
     Own_address_type:0x00 (public device address);
     Filter_Duplicates:0x00 (duplicate filtering disabled);
*/
ret =aci_gap_start_general_discovery_proc(0x4000,0x4000,0x00,0x00);
if (ret != BLE_STATUS_SUCCESS)PRINTF("Failure.\n");
}
```

通过 hci_le_advertising_report_event()事件回调提供流程响应。通过 aci_gap_proc_complete_event()事件回调（其 Procedure_Code 参数等于 GAP_GENERAL_DISCOVERY_PROC (0x2)）来指示流程结束。

```
/* 收到广播报告时调用此回调 */
void hci_le_advertising_report_event(uint8_t Num_Reports,
                                     Advertising_Report_t
                                     Advertising_Report[])
{
    /*  Advertising_Report 包含所有预期参数。
        User application should add code for decoding the received
        Advertising_Report event databased on the specific evt_type
        (ADV_IND, SCAN_RSP, ..)
    */

    /* 示例：存储收到的 Advertising_Report 字段 */
    uint8_t bdaddr[6];

    /* 对端地址的类型 (PUBLIC_ADDR,RANDOM_ADDR) */
    uint8_t bdaddr_type = Advertising_Report[0].Address_Type;

    /* 事件类型（广播数据包类型） */
    uint8_t evt_type = Advertising_Report[0].Event_Type ;

    /* RSSI 值 */
    uint8_t RSSI = Advertising_Report[0].RSSI;

    /* 在扫描期间发现的对端设备地址 */
    Osal_MemCpy(bdaddr, Advertising_Report[0].Address,6);

    /* 广播或扫描响应数据的长度 */
    uint8_t data_length = Advertising_Report[0].Length_Data;

    /*  格式化的广播或扫描响应数据的 data_length 字节位于 Advertising_Report[0]。数据字段：将根据特定用户应用场
        景存储/过滤*/

} /* hci_le_advertising_report_event() */
```

具体来说，在这一特定背景下，由于 GAP 外设处于可发现模式并启用了扫描响应，GAP 中央设备上的 hci_le_advertising_report_event ()回调将发起下列事件：

1. 具有广播数据包类型(evt_type =ADV_IND)的广播报告事件
2. 具有扫描响应数据包类型(evt_type =SCAN_RSP)的广播报告事件

表 37. ADV_IND 事件类型

事件类型	地址类型	地址	广播数据	RSSI
0x00 (ADV_IND)	0x00 (公共地址)	0x0280E1003412	0x02, 0x01, 0x06, 0x08, 0x0A, 0x53, 0x54, 0x4D, 0x33, 0x32, 0x57, 0x42, 0x78, 0x35	0xCE

广播数据可作如下解释（请参考第 6 节 参考文件中的蓝牙规范版本）：

表 38. ADV_IND 广播数据

AD 类型标记字段	本地名称字段	发送功率水平
0x02：字段长度 0x01：AD 类型标记 0x06：0b110（比特 2：BR/EDR 不支持；比特 1：一般可发现模式）	0x09：字段的长度 0x0A：完整的本地名称类型 0x53, 0x54, 0x4D, 0x33, 0x32, 0x57, 0x48, 0x78, 0x35:STM32WB	0x02：字段的长度 0x0A：发送功率类型 0x08：功率值

表 39. SCAN_RSP 事件类型

事件类型	地址类型	地址	扫描响应数据	RSSI
0x04 (SCAN_RS P)	0x01（随机地址）	0x0280E1003412	0x12,0x66,0x9A,0x0C, 0x20,0x00,0x08,0xA7,0 xBA,0xE3,0x11,0x06,0x 85,0xC0,0xF7,0x97,0x8 A,0x06,0x11	0xDA

扫描响应数据可作如下解释（请参考蓝牙规范）：

表 40. 扫描响应数据

扫描响应数据
0x12：数据长度 0x11：服务 UUID 广播数据的长度；0x06：0x06：128 位服务 UUID 类型； 0x66,0x9A,0x0C,0x20,0x00,0x08,0xA7,0xBA,0xE3,0x11,0x06,0x85,0xC0,0xF7,0x97,0x8A: 128 位服务 UUID

4.4 BLE 协议栈事件和事件回调

为了处理其应用中的 ACI 事件，用户可以在两种不同方法之间选择：

- 使用嵌套的“switch case”事件处理程序
- 使用事件回调框架

根据具体的应用场景，用户必须标识要检测和处理的必要设备事件，以及发生这些事件后要执行的应用特定的操作。

在实现 BLE 应用时，最常见且使用最广泛的设备事件是与发现、连接、终止流程、服务和特征发现流程、GATT 服务器上的属性修改事件和 GATT 客户端上的通知/指示事件相关的事件。

表 41. BLE 协议栈：主要事件回调

事件回调	说明	处于...
hci_disconnection_complete_event()	连接终止	GAP 中央设备/外设
hci_le_connection_complete_event()	向要建立连接的两个设备指示新的连接已建立	GAP 中央设备/外设
aci_gatt_attribute_modified_event()	如果事件启用，GATT 客户端修改了服务器端的任何属性时在服务器端生成。	GATT 服务器
aci_gatt_notification_event()	GATT 服务器端发送通知给客户端后，在客户端生成。	GATT 客户端

事件回调	说明	处于...
aci_gatt_indication_event()	GATT 服务器端发送指示给客户端后，在客户端生成。	GATT 客户端
aci_gap_pass_key_req_event()	在配对过程中需要提交密钥时，安全管理器向应用层提交该时间，应用层接收到该事件后必须调用 API “aci_gap_pass_key_resp()”进行反馈。	GAP 中央设备/外设
aci_gap_pairing_complete_event()	在配对过程成功完成或发生配对过程超时或配对失败时生成	GAP 中央设备/外设
aci_gap_bond_lost_event()	在配对请求发出时生成的事件，响应来自主设备（之前已与从设备绑定）的从设备安全请求。在接收到该事件时，上层必须发出指令 aci_gap_allow_rebond()，以允许从设备继续完成与主设备的配对过程	GAP 外设
aci_att_read_by_group_type_resp_event()	发送按组读取型响应，以应答接收到的按组读取型请求，包含已读取的属性的句柄和值	GATT 客户端
aci_att_read_by_type_resp_event()	发送按类型读取的响应，以应答接收到的按类型读取的请求，包含已读取属性的句柄和值	GATT 客户端
aci_gatt_proc_complete_event()	GATT 流程已完成	GATT 客户端
hci_le_advertising_report_event	在上层启动 GAP 流程之一后，在扫描期间发现设备时 GAP 层向上层上报的事件	GAP 中央设备

有关 BLE 事件和相关格式的详细说明，请参考第 6 节“参考文档”中的 STM32WB 蓝牙 LE 协议栈 API 和事件文档。

以下伪代码提供了处理一些描述的 BLE 协议栈事件（断开完成事件、连接完成事件、GATT 属性修改事件、GATT 通知事件）的事件回调示例：

```
/*  该事件回调表示与对端设备的连接断开。
    它在 BLE 无线电中断上下文中调用。
*/
void hci_disconnection_complete_event(uint8_t Status,
                                      uint16_t Connection_Handle,
                                      uint8_t Reason)
{
    /*  基于应用场景添加用于处理 BLE 连接断开完成事件的用户代码。
    */
}/* 结束 hci_disconnection_complete_event() */

/* 该事件回调表示连接流程结束。
*/
void hci_le_connection_complete_event(uint8_t Status,
                                      uint16_t Connection_Handle,
                                      uint8_t Role,
                                      uint8_t Peer_Address_Type,
                                      uint8_t Peer_Address[6],
                                      uint16_t Conn_Interval,
                                      uint16_t Conn_Latency,
                                      uint16_t Supervision_Timeout,
                                      uint8_t Master_Clock_Accuracy)
{
    /*  基于应用场景添加用于处理 BLE 连接完成事件的用户代码。
    */

/* 存储连接句柄 */
connection_handle = Connection_Handle;
...
}/* 结束 hci_le_connection_complete_event() */
```

```
#if GATT_SERVER

/*  该事件回调表示已修改对端设备的属性。
*/
void aci_gatt_attribute_modified_event(uint16_t Connection_Handle,
                                       uint16_t Attr_Handle,
                                       uint16_t Offset,
                                       uint8_t Attr_Data_Length,
                                       uint8_t Attr_Data[])
{
    /*  基于应用场景添加用于处理属性修改事件的用户代码。
    */
    ...
} /* 结束 aci_gatt_attribute_modified_event() */

#endif /* GATT_SERVER */

#if GATT_CLIENT
/* 该事件回调表示已收到来自对端设备的属性通知。
*/
void aci_gatt_notification_event(uint16_t Connection_Handle,
                                 uint16_t Attribute_Handle,
                                 uint8_t Attribute_Value_Length,
                                 uint8_t Attribute_Value[])
{
/*  基于应用场景添加用于处理属性修改事件的用户代码。
   */
...
} /* 结束 aci_gatt_notification_event() */
#endif /* GATT_CLIENT */
```

4.5 安全（配对和绑定）

本节介绍在两个设备之间建立配对要使用的主要功能（验证设备身份，加密链路，以及分发下一次重新连接时要使用的密钥）。

为了与设备成功配对，必须根据所选设备的可用 IO 能力正确配置 IO 能力。

aci_gap_set_io_capability(io_capability)应与下列 io_capability 值之一一起使用：

```
0x00：'IO_CAP_DISPLAY_ONLY'
0x01：'IO_CAP_DISPLAY_YES_NO',
0x02：'KEYBOARD_ONLY'
0x03：'IO_CAP_NO_INPUT_NO_OUTPUT'
0x04：'IO_CAP_KEYBOARD_DISPLAY'
```

2 个 STM32WB 设备（Device_1,Device_2）采用输入密钥方式进行配对的示例：

下列伪代码示例仅描述了使用输入密钥法将两个设备配对时要执行的特定步骤。

如表 11. 计算临时密钥（TK）的方法所述，Device_1 和 Device_2 必须设置 IO 能力，以选择输入密钥法作为安全方法。

在本例中，对 Device_1 选择“仅显示”，对 Device_2 选择“仅键盘”，如下所示：

```
/*Device_1:
*/ tBleStatus ret;\
ret= aci_gap_set_io_capability(IO_CAP_DISPLAY_ONLY);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");

/*Device_2:
*/ tBleStatus ret;
ret= aci_gap_set_io_capability(IO_CAP_KEYBOARD_ONLY);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

在定义了 IO 能力后，应使用 aci_gap_set_authentication_requirement()设置设备需要的所有安全验证要求（MITM 模式（链路是否经过验证），是否有 OOB 数据，是否使用固定配对码，以及是否启用绑定）。

下列伪代码示例仅描述了使用输入密钥法为具有以下配置的设备设置授权要求应遵循的特定步骤：“MITM 保护，无 OOB 数据，请勿使用固定引脚”：此配置用于在配对过程中授权链路并使用非固定引脚。

```
ret=aci_gap_set_authentication_requirement(BONDING,/*绑定启用 */
                                           MITM_PROTECTION_REQUIRED,
                                           SC_IS_SUPPORTED,/*支持安全连接
                                                           但为可选 */
                                           KEYPRESS_IS_NOT_SUPPORTED,
                                           7, /* 加密密钥的最小长度 */
                                           16, /* 最大加密
                                                  密钥大小 */
                                           0x01, /* 未使用固定配对码*/
                                           0x123456, /*固定配对码*/
                                           0x00 /* 公共身份地址类型 */);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

在定义了安全 IO 能力和验证要求后，应用可通过以下方式启动配对流程：

1. 在 GAP 外围（从）设备上使用 aci_gap_slave_security_req()（它向主设备发送从设备安全请求）：

```
tBleStatus ret;
ret= aci_gap_slave_security_req(conn_handle,
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

- 或在 GAP 中央（主）设备上使用 aci_gap_send_pairing_req()。

由于未设置固定配对码，一旦两个设备之一启动了配对流程，BLE 设备调用 aci_gap_pass_key_req_event()事件回调（携带相关连接句柄），以要求用户应用提供用于建立加密密钥的密码。BLE 应用必须通过使用 aci_gap_pass_key_resp(conn_handle, passkey)API 提供正确密码。

当在 Device_1 上调用 aci_gap_pass_key_req_event()回调时，它应该生成一个随机配对码，并通过 aci_gap_pass_key_resp() API 对其进行设置，如下所示：

```
void aci_gap_pass_key_req_event(uint16_t Connection_Handle)
{
   tBleStatus ret;
   uint32_t 引脚;
   /* 使用特定用户函数生成随机引脚 */
   pin = generate_random_pin();
   ret= aci_gap_pass_key_resp(Connection_Handle,pin);
   if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
}
```

由于 Device_1 的 I/O 能力被设置为“仅显示”，它应在设备显示装置上显示生成的配对码。由于 Device_2 的 I/O 能力被设置为“仅键盘”，用户可以使用键盘，并通过相同的 aci_gap_pass_key_resp() API 为 Device_2 提供 Device_1 上显示的配对码。

或者，如果用户想要设置具有固定引脚 0x123456 的验证要求（无需输入密钥事件），可以使用下列伪代码：

```
tBleStatus ret;

ret= aci_gap_set_auth_requirement(BONDING, /* 绑定启用 */
                                  MITM_PROTECTION_REQUIRED,
                                  SC_IS_SUPPORTED, /* 支持
                                  安全连接
                                  但为可选 */
                                  KEYPRESS_IS_NOT_SUPPORTED,
                                  7, /* 最小加密
                                  密钥大小 */
                                  16, /* 最大加密
                                  密钥大小 */
                                  0x00, /* 使用固定配对码*/
                                  0x123456, /*固定配对码*/
                                  0x00 /* 公共身份地址类型 */);
if (ret != BLE_STATUS_SUCCESS) PRINTF("Failure.\n");
```

注意：

1. *如果通过调用所述 API (aci_gap_slave_security_req() 或 aci_gap_send_pairing_req() 启动配对流程并在流程结束时返回值 ret = BLE_STATUS_SUCCESS，将在事件回调时返回 aci_gap_pairing_complete_event()事件以指示配对状态：*
   - *0x00：成功*
   - *0x01：SMP 超时*
   - *0x02：配对失败*

   aci_gap_pairing_complete_event()的状态字段会提供配对状态

   如果失败，则原因参数提供配对失败原因代码（如果状态参数返回成功或超时，则为 0）。
2. *如果 2 个设备成功配对，将在首次连接时自动加密链路。如果还启用了绑定（存储了密钥以备后用），在重新连接 2 个设备时，可以直接加密链路（无需再次执行配对流程）。用户应用可以直接使用相同的 API，不执行配对流程而直接加密链路：*
   - *aci_gap_slave_security_req)（对于 GAP 外围（从）设备）或*
   - *aci_gap_send_pairing_req()（对于 GAP 中央（主）设备）。*
3. *如果从设备已与主设备绑定，可以向主设备发送从设备安全请求以加密链路。当收到从设备安全请求时，主设备可以加密链路、启动配对流程或拒绝请求。通常情况下，主设备只加密链路，不执行配对流程。相反地，如果主设备启动配对流程，这意味着主设备因某些原因丢失了绑定信息，因此必须重新启动配对流程。因此，从设备调用 aci_gap_bond_lost_event()事件回调，以通知用户应用其已不再与之前绑定的主设备绑定。然后，从设备应用可以决定允许安全管理器完成配对流程，并通过调用指令 aci_gap_allow_rebond()与主设备重新绑定，或者只是关闭连接并将安全问题通知用户。*
4. *或者，也可以调用 aci_gap_set_oob_data() API，从而选择带外法。这意味着两个设备都在使用该方法，并设置了通过带外通信（例如，NFC）定义的相同 OOB 数据。*
5. *此外，将 aci_gap_set_authentication_requirement() API 的 SC_Support 字段设为 2，即可使用“安全连接”功能。*

4.5.1 配对流程图：主序列发起的配对请求（传统）

配对流程图：主序列发起的配对请求（传统）

以下流程图描述了主设备在传统模式下创建安全链路应遵循的具体步骤

假设在初始化阶段设置了设备公共地址，如下所示：

```
初始化：
Aci_gap_set_IO_capability(keyboard/display)
Aci_gap_set_auth_requirement(MITM,fixed pin,bonding=1,SC_Support=0x00)
```

图 10. 主序列发起的配对请求（传统）1/3

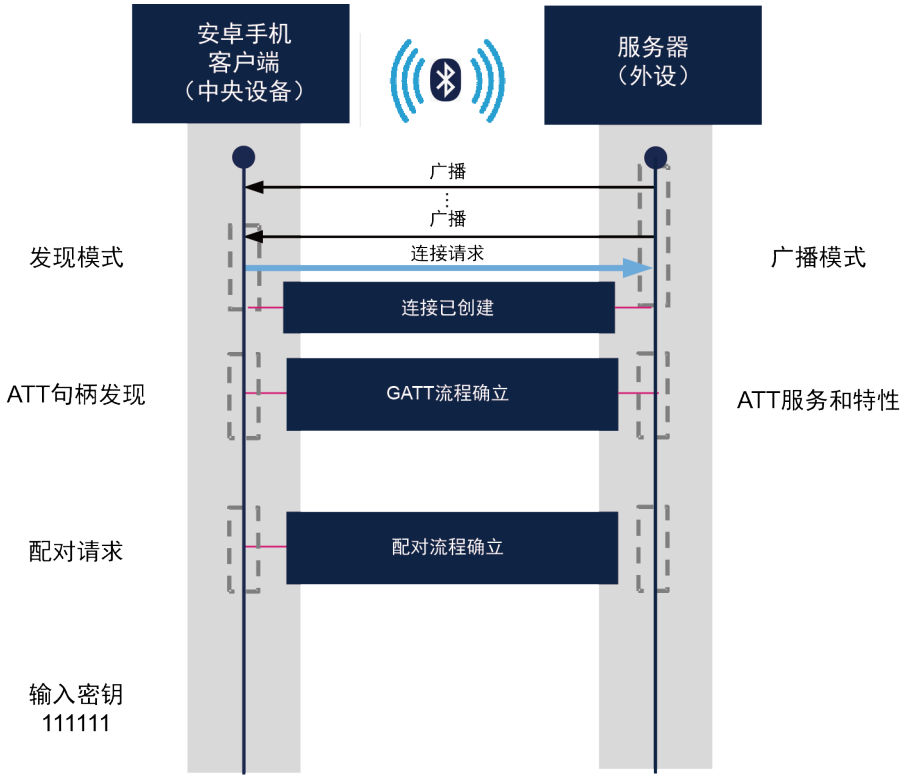


图 11. 主序列发起的配对请求（传统）2/3

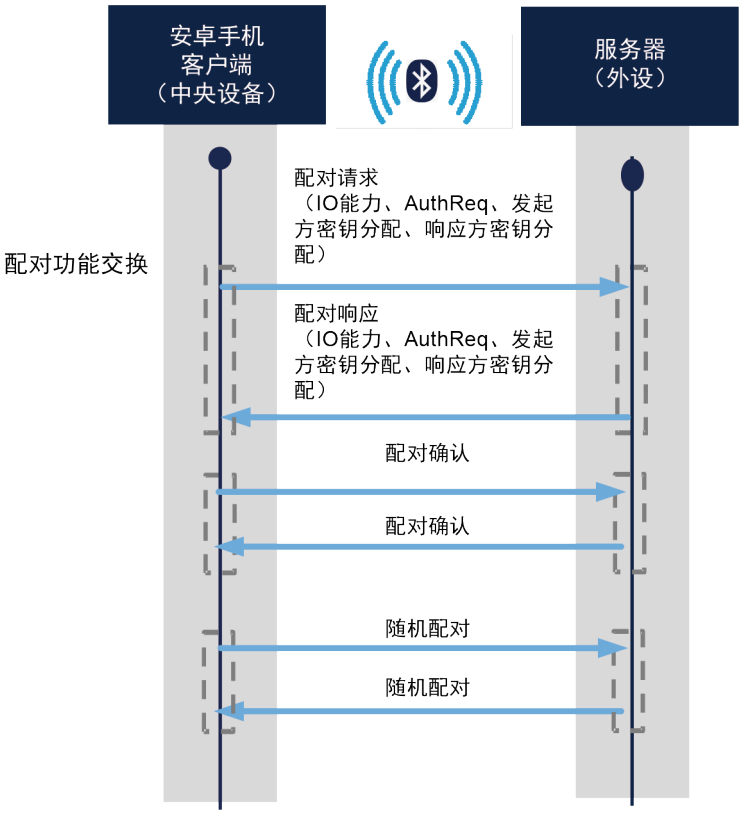
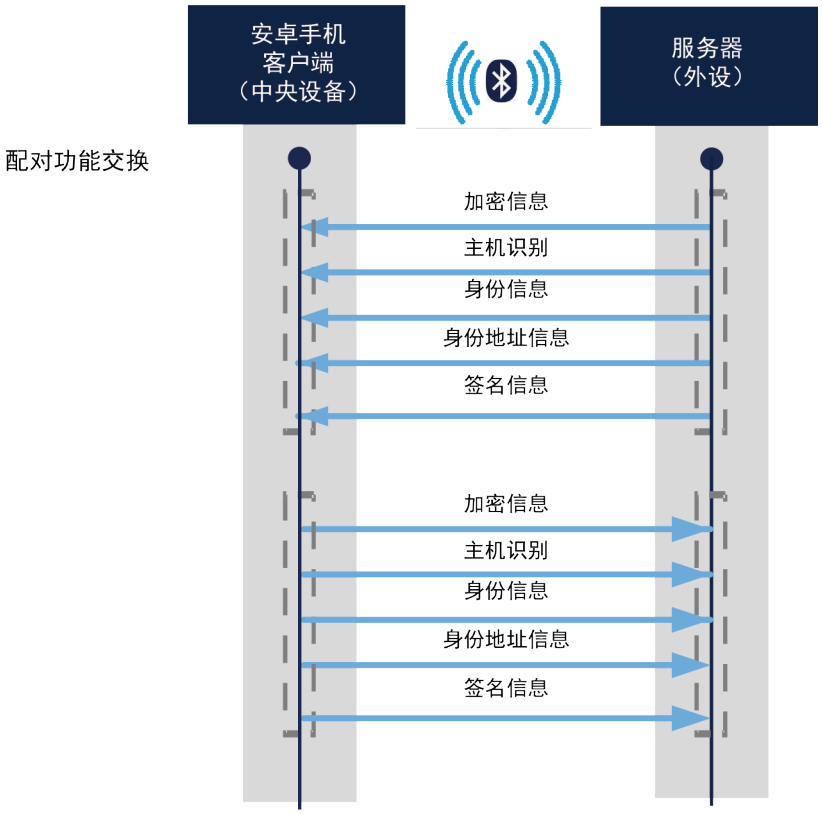


图 12. 主序列发起的配对请求（传统）3/3



4.5.2 配对流程图：主序列发起的配对请求（安全）

配对流程图：主序列发起的配对请求（安全）

以下流程图描述了主设备在安全模式下创建安全链路应遵循的具体步骤。

假设在初始化阶段设置了设备公共地址，如下所示：

```
初始化：
Aci_gap_set_IO_capability(display_yes_no)
Aci_gap_set_auth_requirement(MITM,no
fixed pin,bonding=1,SC only mode)
```

图 13. **主序列发起的配对请求（安全连接）1/3**

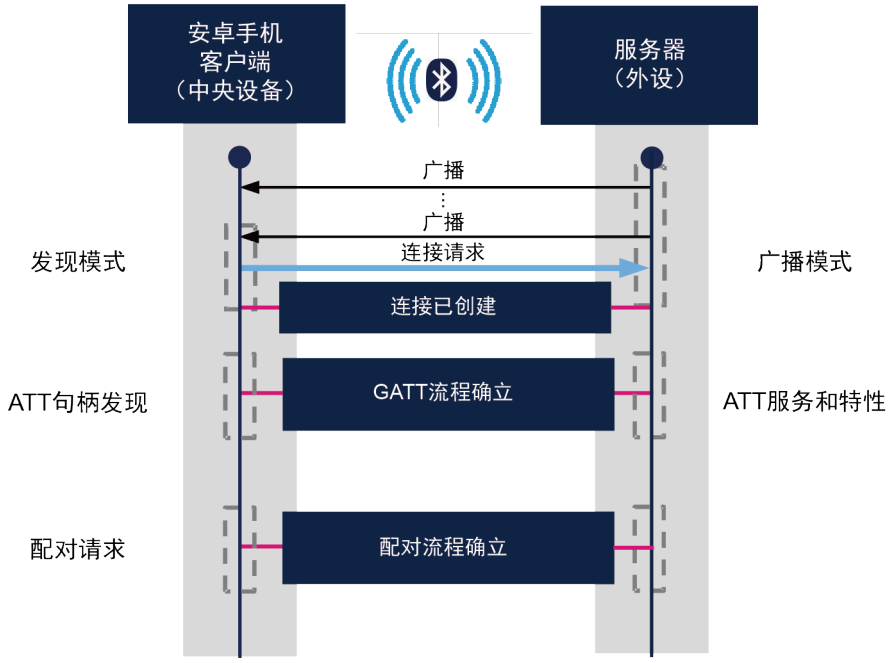


图 14. 主序列发起的配对请求（安全连接）2/3

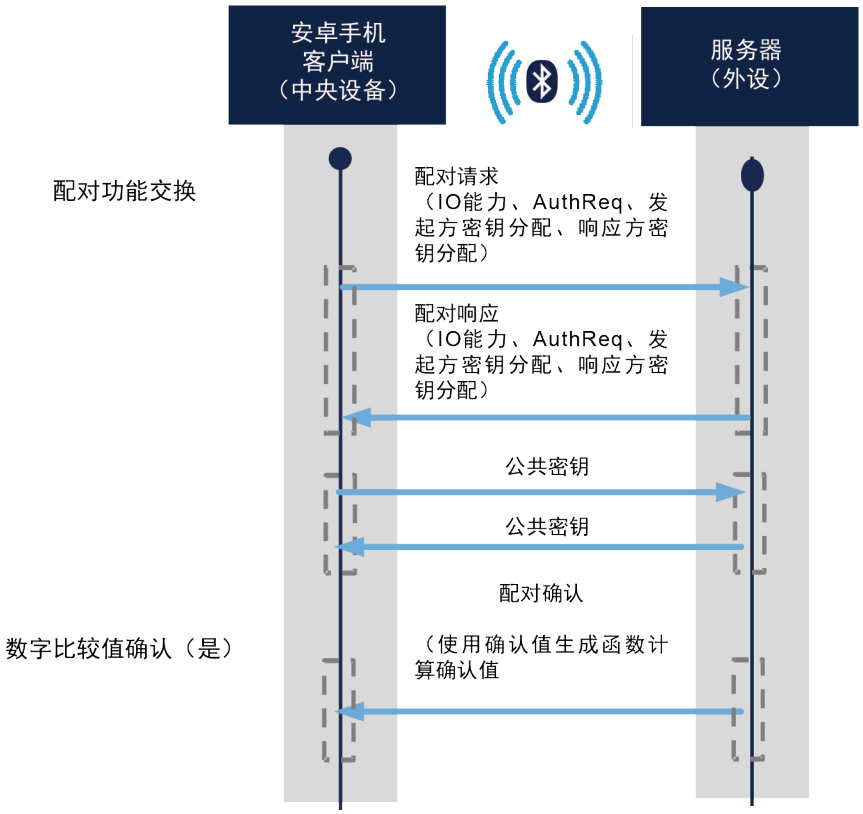
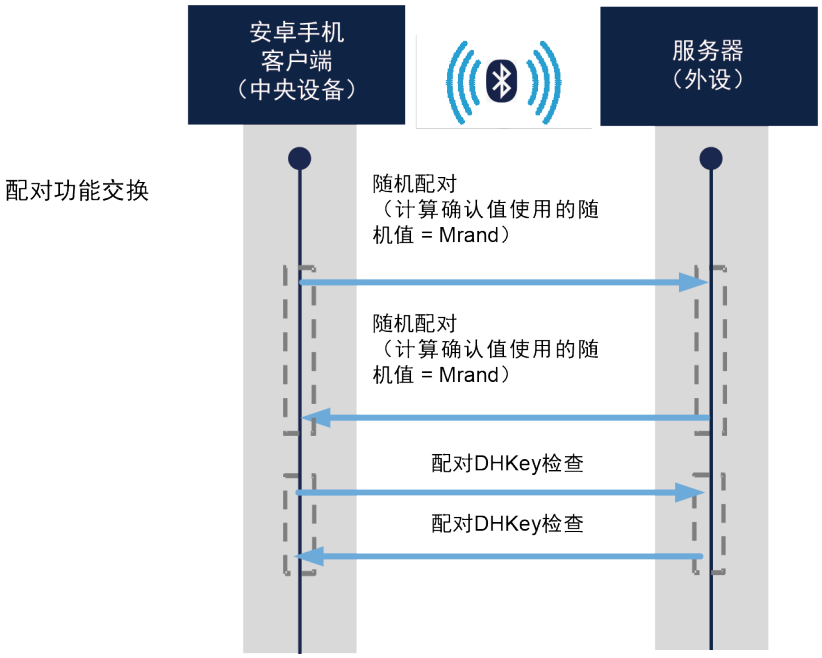


图 15. 主序列发起的配对请求（安全连接）3/3



4.5.3 配对流程图：从序列发起的配对请求（安全）

配对流程图：从序列发起的配对请求（安全）。

以下流程图描述了从设备在安全模式下创建安全链路应遵循的具体步骤

假设在初始化阶段设置了设备公共地址，如下所示：

初始化：

Aci_gap_set_IO_capability(display_yes_no)

Aci_gap_set_auth_requirement(MITM,no fixed pin,bonding=1,SC only mode)

初始化：
Aci_gap_set_IO_capability(display_yes_no)
Aci_gap_set_auth_requirement(MITM,no
fixed pin,bonding=1,SC only mode)

图 16. 从序列发起的配对请求（安全连接）1/2

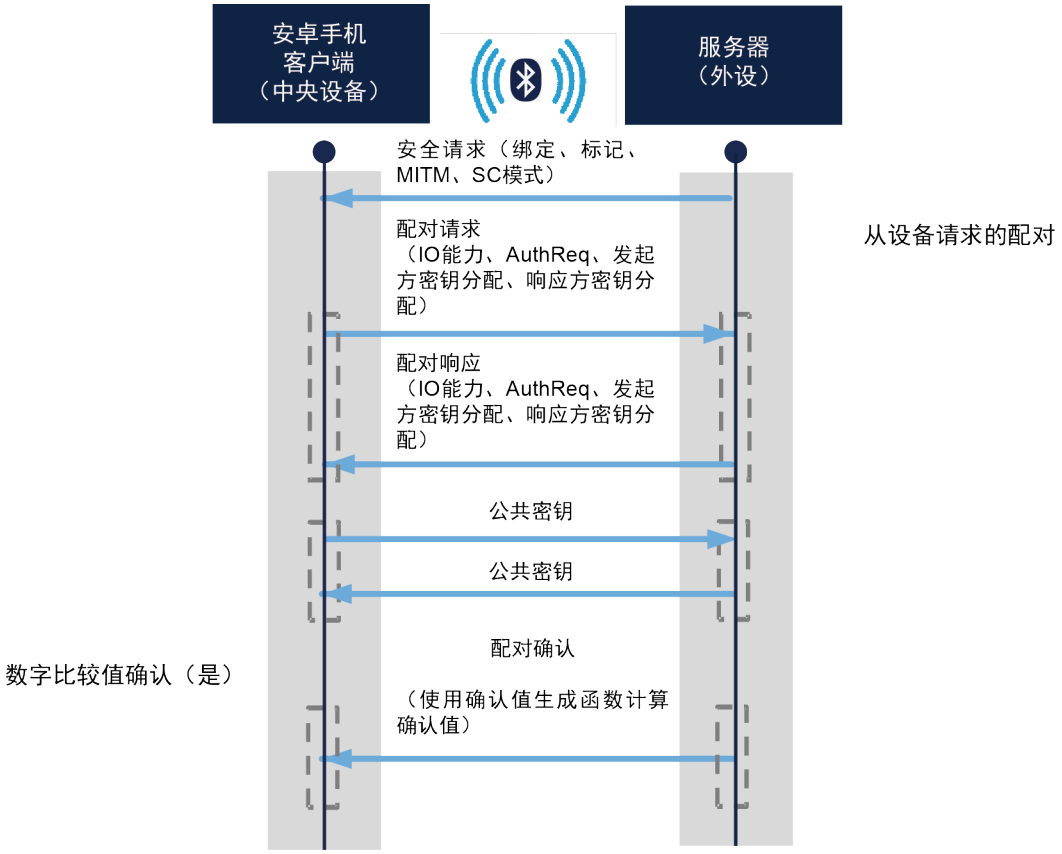
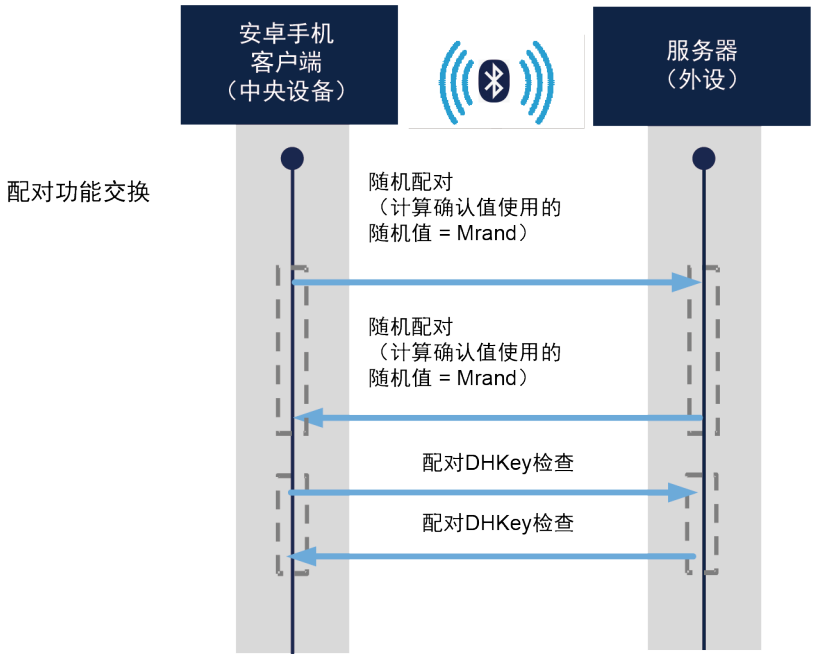


图 17. 从序列发起的配对请求（安全连接）2/2



4.6 服务和特征发现

本节介绍在两个设备建立连接后，允许 STM32WB GAP 中央设备发现 GAP 外设服务和特征的主要函数。

P2PServer 服务与特征及相关句柄用作以下伪代码示例的参考服务和特征。

此外，假设 GAP 中央设备（P2PClient 应用）连接了运行 P2PServer 应用的 GAP 外设。GAP 中央设备使用服务和发现流程寻找 GAP 外设 P2PServer 服务和特征。GAP 中央设备运行 P2PClient 应用。

表 42. BLE 传感器感测演示服务和特征句柄

服务	特征	服务/特征句柄	特征值句柄	特征客户端描述符配置句柄	特征格式句柄
点对点	NA	0x000C	NA	NA	NA
-	LED	0x000D	0x000E	NA	NA
-	按钮	0x000F	0x0010	0x0011	NA

注意：由于具有不同的属性值句柄，仅最后一个属性句柄保留用于标准 GAP 服务。在以下示例中，STM32WB GAP 外设 P2PServer 服务仅定义 LED 特征和按钮特征。有关 tP2Pserver 的详细信息，请参考第 6 节“参考文档”。

下面是包含相关描述的服务发现 API 列表：

表 43. 服务发现流程 API

发现服务 API	说明
aci_gatt_disc_all_primary_services()	此 API 启动 GATT 客户端流程以发现 GATT 服务器上的所有主要服务。在 GATT 客户端连接了设备，并想要寻找设备上提供的所有主要服务以确定其功能时使用。
aci_gatt_disc_primary_service_by_uuid()	此 API 启动 GATT 客户端流程，以通过使用其 UUID 发现 GATT 服务器上的主要服务。 在 GATT 客户端连接了设备并想要寻找特定服务而无需获取任何其他服务时使用。
aci_gatt_find_included_services()	此 API 启动寻找所有已包含服务的流程。在 GATT 客户端已发现主要服务并想要发现次要服务时使用。

以下伪代码示例描述了 aci_gatt_disc_all_primary_services() API：

```
/*GAP 中央设备启动发现所有服务的流程：
conn_handle 是 hci_le_advertising_report_event()事件回调时返回的连接句柄
*/
if (aci_gatt_disc_all_primary_services(conn_handle) !=BLE_STATUS_SUCCESS)
{
   PRINTF("Failure.\n");

}
```

通过 aci_att_read_by_group_type_resp_event() 事件回调提供流程响应。通过 aci_gatt_proc_complete_event()事件回调()调用指示流程结束。

```
/*在响应按组类型读取请求时生成该事件：参考 aci_gatt_disc_all_primary_services() */
void aci_att_read_by_group_type_resp_event(uint16_t Conn_Handle,
                                           uint8_t
Attr_Data_Length,
                                           uint8_t Data_Length,
                                           uint8_t Att_Data_List[]);
{
/*
 Conn_Handle：与响应相关的连接句柄；
 Attr_Data_Length：每个属性数据的长度；
 Data_Length：Attribute_Data_List 的字节长度；
 Att_Data_List:属性数据列表如蓝牙核心规范所定义。
  一系列属性句柄、结束组句柄、属性值元组：
 [2 个字节的属性句柄，2 字节的结束组句柄，
  （Attribute_Data_Length - 4 个字节）的属性值]。
*/
/* 添加用户代码，用于解码 Att_Data_List 字段并获取服务属性句柄、
结束组句柄和服务 uuid
*/
}/* aci_att_read_by_group_type_resp_event() */
```

就传感器感测演示而言，GAP 中央设备应用应获取三个按组读取型响应事件（通过相关的 aci_att_read_by_group_type_resp_event()事件回调），并具有以下回调参数值。

第一个按组类型读取的响应事件回调参数：

Connection_Handle:0x0801（连接句柄）；
Attr_Data_Length:0x06（每个已发现服务数据（服务
句柄、结束组句柄、服务 UUID）的长度)；
Data_Length:0x0C（Attribute_Data_List 的字节长度）
Att_Data_List:0x0C 字节如下：

表 44. 第一个按组类型读取的响应事件回调参数

句柄属性	结束组句柄	服务 UUID	注释
0x0001	0x0004	0x1801	属性配置文件服务（由 GATT_Init()添加）。 标准 16 位服务 UUID
0x0005	0x000B	0x1800	GAP 配置文件服务（由 GAP_Init()添加）。 标准 16 位服务 UUID。

第二个按组类型读取的响应事件回调参数：

Conn_Handle:0x0801（连接句柄）；
Attr_Data_Length:0x14（每个已发现服务数据：
服务句柄、结束组句柄、服务 UUID）的长度)；
Data_Length:0x14（Attribute_Data_List 的字节长度）；
Att_Data_List:0x14 字节如下：

表 45. 第二个按组类型读取的响应事件回调参数

句柄属性	结束组句柄	服务 UUID	注释
0x000C	0x0012	0x02366E80CF3A11E19A B40002A5D5C51B	128 位服务专有 UUID 的加速服务

第三个按组类型读取的响应事件回调参数：

Connection_Handle:0x0801（连接句柄）；
Attr_Data_Length:0x14（每个已发现服务数据：
服务句柄、结束组句柄和服务 uuid）；
Data_Length:0x14（Attribute_Data_List 的字节长度）；
Att_Data_List:0x14 字节如下：

表 46. 第三个按组类型读取的响应事件回调参数

句柄属性	结束组句柄	服务 UUID	注释
0x0013	0x0019	0x42821A40E47711E282D 00002A5D5C51B	128 位服务专有 UUID 的环境服务

就传感器感测演示而言，当发现所有主要服务流程完成时，在 GAP 中央设备应用上调用 aci_gatt_proc_complete_event()事件回调，并具有以下参数：

Conn_Handle:0x0801 (connection handle;
Error_Code:0x00

4.6.1 特征发现流程与相关 GATT 事件

下面是包含相关描述的特征发现 API 列表：

表 47. 特征发现流程 API

发现服务 API	说明
aci_gatt_disc_all_char_of_service ()	此 API 启动 GATT 流程以发现给定服务的所有特征
aci_gatt_disc_char_by_uuid ()	此 API 启动 GATT 流程以发现通过 UUID 指定的所有特征
aci_gatt_disc_all_char_desc ()	此 API 启动 GATT 服务器上发现所有特征描述符的流程

就 BLE 传感器感测演示而言，下面是描述 GAP 中央设备应用如何发现所有加速服务特征的简单虚拟代码（请参考表 45. 第二个按组类型读取的响应事件回调参数）：

```
uint16_t service_handle= 0x000C;
uint16_t end_group_handle = 0x0012;
```

```
/*GAP 中央设备启动发现所有服务特征的
流程：conn_handle 是 hci_le_advertising_report_event()事件回调返回的连接句柄 */
if(aci_gatt_disc_all_char_of_service(conn_handle,
                                     service_handle,/* Servicehandle */
                                     end_group_handle/* 结束组句柄
                                                      */
                                     );) != BLE_STATUS_SUCCESS)
{
    PRINTF("Failure.\n");
}
```

通过 aci_att_read_by_type_resp_event()事件回调提供流程响应。通过 aci_gatt_proc_complete_event()事件回调调用指示流程结束。

```
/* 该事件为响应 aci_att_read_by_type_req()而产生。请参见
aci_gatt_disc_all_char() API */
```

```
void aci_att_read_by_type_resp_event(uint16_t Connection_Handle ,
                                     uint8_t Handle_Value_Pair_Length,
                                     uint8_t Data_Length,
                                     uint8_t Handle_Value_Pair_Data[])
{

/*
    Connection_Handle：与响应相关的连接句柄；
    Handle_Value_Pair_Length：每个属性 handle-value 对的长度；

    Data_Length：Handle_Value_Pair_Data 的字节长度。
    Handle_Value_Pair_Data:属性数据列表如蓝牙核心规范所定义。
    handle-value 对的顺序：[2 个字节的属性句柄，
     （Handle_Value_Pair_Length - 2 个字节）的属性值]。
*/
/* 添加用户代码，用于解码 Handle_Value_Pair_Data 字段并获取特征句柄、
    属性、特征值句柄、特征 UUID*/
  */

}/* aci_att_read_by_type_resp_event() */
```

就 BLE 传感器感测演示而言，GAP 中央设备应用应获取两个读取类型响应事件（通过相关 aci_att_read_by_type_resp_event()事件回调），并具有以下回调参数值。

按类型响应事件回调参数的第一次读取：

conn_handle :0x0801（连接句柄）；
Handle_Value_Pair_Length:0x15（每个已发现
特征数据（特征句柄、属性、
特征值句柄、特征 UUID）的长度)；
Data_Length:0x16 (事件数据的长度);
Handle_Value_Pair_Data:0x15 字节如下：

表 48. 按类型响应事件回调参数的第一次读取

特征句柄	特征属性	特征值句柄	特征 UUID	注释
0x000D	0x10（通知）	0x000E	0xE23E78A0CF4A11E18FFC0002A5D5C51B	128 位特征专有 UUID 的自由落体特征

按类型响应事件回调参数的第二次读取：

conn_handle :0x0801（连接句柄）；
Handle_Value_Pair_Length:0x15（每个已发现
特征数据（特征句柄、属性、
特征值句柄、特征 UUID）的长度)；
Data_Length:0x16 (事件数据的长度);
Handle_Value_Pair_Data:0x15 字节如下：

表 49. 按类型响应事件回调参数的第二次读取

特征句柄	特征属性	特征值句柄	特征 UUID	注释
0x0010	0x12（通知并读取）	0x0011	0x340A1B80CF4B11E1AC360002A5D5C51B	128 位特征专有 UUID 的加速特征

就传感器感测演示而言，当发现所有主要服务流程完成时，在 GAP 中央设备应用上调用 aci_gatt_proc_complete_event()事件回调，并具有以下参数：

Connection_Handle:0x0801（连接句柄）；
Error_Code:0x00.

可以执行类似步骤，以便发现环境服务的所有特征（表 42. BLE 传感器感测演示服务和特征句柄）。

4.7 特征通知/指示、写入、读取

本节介绍用于访问 BLE 设备特征的主要函数。

表 50. 特征更新、读取、写入 API

发现服务 API	说明	处于...
aci_gatt_update_char_value_ext()	如果服务器端对特征启用了通知（或指示），该 API 向客户端发送通知（或指示）。	GATT 服务器
aci_gatt_read_char_value()	启动读取属性值的流程。	GATT 客户端

发现服务 API	说明	处于...
aci_gatt_write_char_value()	启动写入属性值的流程（当流程完成时，生成 GATT 流程完成事件）。	GATT 客户端
aci_gatt_write_without_resp()	启动写入特征值的流程，不等待服务器的任何响应。	GATT 客户端
aci_gatt_write_char_desc()	启动写入特征描述符的流程。	GATT 客户端
aci_gatt_confirm_indication()	确认指示。当应用接收到特征指示时，必须发送该指令。	GATT 客户端

就 P2PServer 演示而言，请遵循 GAP 中央设备应使用的部分代码，以配置按钮特征客户端描述符配置用于通知：

```
/* 启用按钮特征客户端描述符配置用于通知 */
aci_gatt_write_char_desc(aP2PClientContext[index].connHandle,
aP2PClientContext[index].P2PNotificationDescHandle,
2,
uint8_t *)&enable);
```

一旦从 GAP 中央设备启用了特征通知，GAP 外设就可以通知自由落体和加速特征的新值，如下所示：

```
void P2PS_Send_Notification(void)
{
if(P2P_Server_App_Context.ButtonControl.ButtonStatus == 0x00){
P2P_Server_App_Context.ButtonControl.ButtonStatus=0x01;
} else {
P2P_Server_App_Context.ButtonControl.ButtonStatus=0x00;
}
if(P2P_Server_App_Context.Notification_Status){
APP_DBG_MSG("-- P2P APPLICATION SERVER :INFORM CLIENT BUTTON 1 USHED \n ");
APP_DBG_MSG(" \n\r");
P2PS_STM_App_Update_Char(P2P_NOTIFY_CHAR_UUID,
(uint8_t*)&P2P_Server_App_Context.ButtonControl);
} else {
APP_DBG_MSG("-- P2P APPLICATION SERVER :CAN'T INFORM CLIENT - NOTIFICATION DISABLED\n ");
}
return;
}
tBleStatus P2PS_STM_App_Update_Char(uint16_t UUID, uint8_t *pPayload)
{
tBleStatus result = BLE_STATUS_INVALID_PARAMS;
switch(UUID)
{
case P2P_NOTIFY_CHAR_UUID:
result = aci_gatt_update_char_value(aPeerToPeerContext.PeerToPeerSvcHdle,
aPeerToPeerContext.P2PNotifyServerToClientCharHdle,
0, /* charValOffset */
2, /* charValueLen */
(uint8_t *) pPayload);
break;
default:
break;
}
return result;
}/* 结束 P2PS_STM_Init() */
```

在 GAP 中央设备上，Event_Handler（EVT_VENDOR 作为主要事件），在接收到来自 GAP 外设的特征通知（按钮）时发起 EVT_BLUE_GATT_NOTIFICATION。

```
static SVCCTL_EvtAckStatus_t Event_Handler(void *Event)
{
SVCCTL_EvtAckStatus_t return_value;
hci_event_pckt *event_pckt;
evt_blue_aci *blue_evt;
P2P_Client_App_Notification_evt_t 通知；
return_value = SVCCTL_EvtNotAck;
event_pckt = (hci_event_pckt *)(((hci_uart_pckt*)Event)->data);
switch(event_pckt->evt) {
case EVT_VENDOR:
{
blue_evt = (evt_blue_aci*)event_pckt->data;
switch(blue_evt->ecode) {
….
case EVT_BLUE_GATT_NOTIFICATION:
{
aci_gatt_notification_event_rp0 *pr = (void*)blue_evt->data;
uint8_t index;
索引 = 0；
while((index < BLE_CFG_CLT_MAX_NBR_CB) &&
(aP2PClientContext[index].connHandle != pr->Connection_Handle))
index++;
if(index < BLE_CFG_CLT_MAX_NBR_CB) {
if ( (pr->Attribute_Handle == aP2PClientContext[index].P2PNotificationCharHdle) &&
(pr->Attribute_Value_Length == (2)) )
{
Notification.P2P_Client_Evt_Opcode = P2P_NOTIFICATION_INFO_RECEIVED_EVT;
Notification.DataTransfered.Length = pr->Attribute_Value_Length;
Notification.DataTransfered.pPayload = &pr->Attribute_Value[0];
Gatt_Notification(&Notification);
/* 告知应用按钮被终端设备按下 */
}
}
}
Break;/* 结束 EVT_BLUE_GATT_NOTIFICATION */
….
void Gatt_Notification(P2P_Client_App_Notification_evt_t *pNotification) {
switch(pNotification->P2P_Client_Evt_Opcode) {
case P2P_NOTIFICATION_INFO_RECEIVED_EVT:
{
P2P_Client_App_Context.LedControl.Device_Led_Selection=pNotification-
>DataTransfered.pPayload[0];
switch(P2P_Client_App_Context.LedControl.Device_Led_Selection) {
case 0x01 :{
P2P_Client_App_Context.LedControl.Led1=pNotification->DataTransfered.pPayload[1];
if(P2P_Client_App_Context.LedControl.Led1==0x00){
BSP_LED_Off(LED_BLUE);
APP_DBG_MSG(" -- P2P APPLICATION CLIENT :NOTIFICATION RECEIVED - LED OFF \n\r");
APP_DBG_MSG(" \n\r");
} else {
APP_DBG_MSG(" -- P2P APPLICATION CLIENT :NOTIFICATION RECEIVED - LED ON\n\r");
APP_DBG_MSG(" \n\r");
BSP_LED_On(LED_BLUE);
}
break;
}
default : break;
}
….
}
```

4.7.1 访问 BLE 设备长特征。

本节介绍访问 BLE 设备长特征值的主要函数。

表 51. 长值的特征更新、读取、写入 API

特征处理 API	说明	API 调用端	将在客户端使用的事件
Aci_gatt_read_long_char_value()	读取长特征值。	GATT 客户端	ACI_GATT_READ_EXT_EVENT (mask = 0x00100000)
Aci_gatt_write_long_char_value()	写入长特征值。	GATT 客户端	ACI_ATT_EXEC_WRITE_RESP_EVENT (mask = 0x00001000) ACI_ATT_PREPARE_WRITE_RESP_EVENT (mask = 0x00000800)
Aci_gatt_update_char_value_ext()	aci_gatt_update_char_value 版本支持更新长达 512 字节的属性，并有选择地指示生成指示/通知。	GATT 服务器	ACI_GATT_NOTIFICATION_EXT_EVENT (mask = 0x00400000) or ACI_GATT_INDICATION_EXT_EVENT (mask = 0x00200000)
Aci_gatt_read_handle_value()	从本地 GATT 数据库读取指定属性句柄的值。	GATT 服务器	-

1. 当 char_length > ATT_MTU – 4 时特征为长值
2. 因事件的协议栈接口导致的限制：事件参数长度是一个 8 位的值。

读取长远程数据（客户端侧）

为了避免限制 2，添加了新事件：ACI_GATT_READ_EXT_EVENT
（将通过以下掩码启用：0x00100000 using aci_gatt_set_event_mask command)
它替代了以下三个事件：

```
ACI_ATT_READ_RESP_EVENT (1)
ACI_ATT_READ_BLOB_RESP_EVENT (2)
ACI_ATT_READ_MULTIPLE_RESP_EVENT (3)
```

旨在响应以下事件：

```
Aci_gatt_read_char_value (1)
Aci_gatt_read_long_char_value (2)
Aci_gatt_read_multiple_char_value (3)
```

条件：ATT_MTU > 多个特征总长度的总和

写入长远程数据（客户端侧）

```
Aci_gatt_write_long_char_value()
```

待写入数据的长度限于 245（ATT_MTU = 251）

读取长本地数据（服务器端）

```
Aci_gatt_read_handle_value()
```

此指令需要多次调用。

写入长本地数据（服务器端）

ACI_GATT_NOTIFICATION_EXT_EVENT

（将通过以下掩码启用：0x00400000 using aci_gatt_set_event_mask command)

响应：

Aci_gatt_update_char_value_ext

指令

如何使用 aci_gatt_update_char_value_ext：

当

ATT_MTU > (BLE_EVT_MAX_PARAM_LENGTH – 4) i.e ATT_MTU > 251

时，需要两个指令。

第一个指令：

```
Aci_gatt_update_char_value_ext (conn_handle, Service_handle, TxCharHandle,
Update_Type = 0x00,
Total_length,
Value_offset,
Param_length,
&payload)
```

第二个指令

```
Aci_gatt_update_char_value_ext (conn_handle, Service_handle, TxCharHandle,
Update_Type = 0x01,
Total_length,
Value_offset = Param_length,
param_length2,
(&payload) + param_length)
```

第二个指令后，将实时发送总长度通知，并通过 ACI_GATT_NOTIFICATION_EXT_EVENT 事件接收此通知。

根据 ACI_GATT_NOTIFICATION_EXT_EVENT 事件的偏移参数，可以重新组合数据。位 15 用作标志：当置为 1 时，表示将要传输更多数据（在属性数据较长的情况下使用分段事件）

同样适用于：ACI_GATT_INDICATION_EXT_EVENT（将通过以下掩码启用：0x00200000 using aci_gatt_set_event_mask command)

响应：Aci_gatt_update_char_value_ext() command。

在本例中，第一个指令使用 Update_Type = 0x00，第二个指令使用 Update_Type = 0x02。如果我们以长数据传输为例：

一旦从 GAP 中央设备启用了特征通知，GAP 外设就可以通知新值：

```
static void SendData( void )
{
tBleStatus status = BLE_STATUS_INVALID_PARAMS;
uint8_t crc_result;
if( (DataTransferServerContext.ButtonTransferReq != DTS_APP_TRANSFER_REQ_OFF)
&& (DataTransferServerContext.NotificationTransferReq != DTS_APP_TRANSFER_REQ_OFF)
&& (DataTransferServerContext.DtFlowStatus != DTS_APP_FLOW_OFF) )
{
/*发送到远端的数据包*/
Notification_Data_Buffer[0] += 1;
/* 计算 CRC */
crc_result = APP_BLE_ComputeCRC8((uint8_t*) Notification_Data_Buffer,
(DATA_NOTIFICATION_MAX_PACKET_SIZE - 1));
Notification_Data_Buffer[DATA_NOTIFICATION_MAX_PACKET_SIZE - 1] = crc_result;
DataTransferServerContext.TxData.pPayload = Notification_Data_Buffer;
//DataTransferServerContext.TxData.Length = DATA_NOTIFICATION_MAX_PACKET_SIZE; /*
DATA_NOTIFICATION_MAX_PACKET_SIZE */
DataTransferServerContext.TxData.Length = Att_Mtu_Exchanged-10;
status = DTS_STM_UpdateChar(DATA_TRANSFER_TX_CHAR_UUID, (uint8_t *)
&DataTransferServerContext.TxData);
if (status == BLE_STATUS_INSUFFICIENT_RESOURCES)
{
DataTransferServerContext.DtFlowStatus = DTS_APP_FLOW_OFF;
(Notification_Data_Buffer[0])-=1;
}
else
{
UTIL_SEQ_SetTask(1 << CFG_TASK_DATA_TRANSFER_UPDATE_ID, CFG_SCH_PRIO_0);
}
}
return;
}
tBleStatus DTS_STM_UpdateChar( uint16_t UUID , uint8_t *pPayload )
{
tBleStatus result = BLE_STATUS_INVALID_PARAMS;
switch (UUID)
{
case DATA_TRANSFER_TX_CHAR_UUID:
result = TX_Update_Char((DTS_STM_Payload_t*) pPayload);
break;
default:
break;
}
return result;
}/* 结束 DTS_STM_UpdateChar() */
static tBleStatus TX_Update_Char( DTS_STM_Payload_t *pDataValue )
{
tBleStatus ret;
/**
* 通知数据传输包
*/
/* 总长度与将通过通知发送的总数据长度一致
值偏移与待修改值的偏移一致
参数长度与将在之前定义偏移时修改的值长度一致 */
```

在 GAP 中央设备上，DTC_Event_Handler（EVT_VENDOR 作为主要事件），在接收到来自 GAP 外设的特征通知（按钮）时发起 EVT_BLUE_GATT_NOTIFICATION_EXT。

```
static SVCCTL_EvtAckStatus_t DTC_Event_Handler(void *Event)
{
SVCCTL_EvtAckStatus_t return_value;
hci_event_pckt *event_pckt;
evt_blue_aci *blue_evt;
P2P_Client_App_Notification_evt_t 通知；
return_value = SVCCTL_EvtNotAck;
event_pckt = (hci_event_pckt *)(((hci_uart_pckt*)Event)->data);
switch(event_pckt->evt)
{
case EVT_VENDOR:
{
blue_evt = (evt_blue_aci*)event_pckt->data;
switch(blue_evt->ecode)
{
….
case EVT_BLUE_GATT_NOTIFICATION_EXT:
{
aci_gatt_notification_event_rp0 *pr = (void*)blue_evt->data;nnnn
uint8_t index;
索引 = 0；
while((index < BLE_CFG_CLT_MAX_NBR_CB) &&
(aP2PClientContext[index].connHandle != pr->Connection_Handle))
index++;
if(index < BLE_CFG_CLT_MAX_NBR_CB)
{
if ( (pr->Attribute_Handle == aP2PClientContext[index].P2PNotificationCharHdle) &&
(pr->Attribute_Value_Length == (2)) )
{
Notification.P2P_Client_Evt_Opcode = P2P_NOTIFICATION_INFO_RECEIVED_EVT;
Notification.DataTransfered.Length = pr->Attribute_Value_Length;
Notification.DataTransfered.pPayload = &pr->Attribute_Value[0];
Gatt_Notification(&Notification);
/* 告知应用按钮被终端设备按下 */
}
}
}
Break;/* 结束 EVT_BLUE_GATT_NOTIFICATION */
```

4.8 使用 GATT 实现端到端接收流控制

当使用 GATT 接收来自对端设备的数据时，可从优化的接收流控制中受益。

通常，对端设备会多次使用 GATT 写入流程，通过数据包将数据发送到本地设备 GATT 特征。然后本地设备的用户应用通过连续的 GATT 事件（ACI_GATT_ATTRIBUTE_MODIFIED_EVENT）接收数据包。

为了获得接收流控制，用户应用需要在使用 ACI_GATT_ADD_CHAR 原语创建特征时设置 AUTHOR_WRITE 标记。接着将每个对端设备写入尝试通知用户应用，然后通过专门事件（ACI_GATT_WRITE_PERMIT_REQ_EVENT）执行。用户应用仅需使用 ACI_GATT_WRITE_RESP 原语（写入状态 = 0）回复该事件。如果用户应用需要一定时间来回复此事件（例如，仍在处理之前的数据包），当本地内部 RX ACL 数据 FIFO 已满时（此 FIFO 大小取决于 BLE 协议栈配置），这将阻碍本地 GATT，并继而阻碍对端设备。

4.9 基本/典型错误条件描述

在 STM32WB BLE 协议栈 API 框架中，会定义 tBleStatus 类型，以返回 STM32WB 协议栈错误条件。头文件“ble_status.h”中定义了错误代码。

在调用协议栈 API 时，为了追踪潜在错误状态，建议获取 API 返回状态并对其进行监控。

当 API 成功执行时，返回 BLE_STATUS_SUCCESS (0x00)。有关与每个 ACI API 有关的错误条件列表，请参考第 6 节“参考文档”中的 STM32WB 蓝牙 LE 协议栈 API 和事件文档

4.10 BLE 同时作为主、从设备的场景

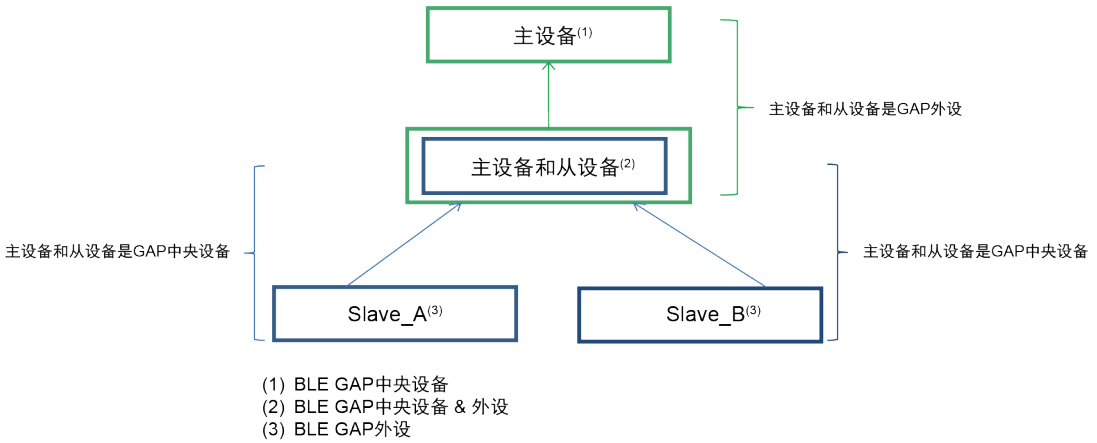
STM32WB BLE 协议栈同时支持多个角色（有关更多详情，请参见第 5 节“BLE 多连接时序策略”）。因此，同一设备既可作为一个或多个连接上的主设备（最多支持八个连接），也可作为另一个连接上的从设备。

下面的伪代码描述了如何初始化 BLE 协议栈设备以便能够同时支持中央设备和外设角色：

```
uint8_t role= GAP_PERIPHERAL_ROLE | GAP_CENTRAL_ROLE;
ret= aci_gap_init(role, 0, 0x07, &service_handle,
&dev_name_char_handle, &appearance_char_handle);
```

同时作为主、从设备的测试场景如下：

图 18. BLE 同时作为主、从设备的场景



1. 通过在 GAP_Init() API 上将角色设置为 GAP_PERIPHERAL_ROLE |GAP_CENTRAL_ROLE，将一个 BLE 设备（称为“主/从设备”）配置为中央设备及外设。假设该设备也定义了相关的服务和特征。
2. 通过在 GAP_Init() API 上将角色设置为 GAP_PERIPHERAL_ROLE，将两个 BLE 设备（称为 Slave_A、Slave_B）配置为外设。Slave_A 和 Slave_B 定义与主设备和从设备相同的服务和特征。
3. 通过在 GAP_Init() API 上将角色设置为 GAP_CENTRAL_ROLE，将一个 BLE 设备（称为主设备）配置为中央设备。
4. Slave_A 和 Slave_B 设备均进入发现模式，如下所示：

```
ret =aci_gap_set_discoverable(Advertising_Type=0x00,
                              Advertising_Interval_Min=0x20,
                              Advertising_Interval_Max=0x100,
                              Own_Address_Type= 0x0;
                              Advertising_Filter_Policy= 0x00;
                              Local_Name_Length=0x05,
                              Local_Name=[0x08,0x74,0x65,0x73,0x74],
                              Service_Uuid_length = 0;
                              Service_Uuid_length = NULL;
                              Slave_Conn_Interval_Min = 0x0006,
                              Slave_Conn_Interval_Max = 0x0008);
```

5. 主/从设备执行发现流程，以便发现外设 Slave_A 和 Slave_B：

```
ret = aci_gap_start_gen_disc_proc (LE_Scan_Interval=0x10,
                                   LE_Scan_Window=0x10,
                                   Own_Address_Type = 0x0,
                                   Filter_Duplicates = 0x0);
```

通过使用 hci_le_advertising_report_event()事件回调通知的广播报告事件发现这两个设备。

6. 在发现两个设备后，主/从设备启动两个连接流程（作为中央设备），以便分别连接到 Slave_A 和 Slave_B 设备：

```
/* 连接到 Slave_A：Slave_A 地址类型和地址
已在借助广播报告事件的发现流程中找到。
*/
ret= aci_gap_create_connection(LE_Scan_Interval=0x0010,
                               LE_Scan_Window=0x0010
                               Peer_Address_Type= ”Slave_A 地址类型”
                               Peer_Address= ”Slave_A 地址,
                               Own_Address_Type = 0x0;
                               Conn_Interval_Min=0x6c,
                               Conn_Interval_Max=0x6c,
                               Conn_Latency=0x00,
                               Supervision_Timeout=0xc80,
                               Minimum_CE_Length=0x000c,
                               Maximum_CE_Length=0x000c);
```

```
/* 连接到 Slave_B：Slave_B 地址类型和地址
已在借助广播报告事件的发现流程中找到。
*/
ret= aci_gap_create_connection(LE_Scan_Interval=0x0010,
                               LE_Scan_Window=0x0010,
                               Peer_Address_Type= ”Slave_B 地址类型”,
                               Peer_Address= ”Slave_B 地址”,
                               Own_Address_Type = 0x0;
                               Conn_Interval_Min=0x6c,
                               Conn_Interval_Max=0x6c,
                               Conn_Latency=0x00,
                               Supervision_Timeout=0xc80,
                               Minimum_CE_Length=0x000c,
                               Maximum_CE_Length=0x000c);
```

7. 一旦连接，主/从设备使用 aci_gatt_write_char_desc() API 在两个从设备上启用特征通知。Slave_A 和 Slave_B 设备使用 aci_gatt_upd_char_val() API 启动特征通知。

8. 在该阶段，主/从设备进入发现模式（作为外设）：

```
/* 使主/从设备进入可发现模式，Name = 'Test' =
[0x08,0x74,0x65,0x73,0x74*/
ret =aci_gap_set_discoverable(Advertising_Type=0x00,
                              Advertising_Interval_Min=0x20,
                              Advertising_Interval_Max=0x100,
                              Own_Address_Type= 0x0;
                              Advertising_Filter_Policy= 0x00;
                              Local_Name_Length=0x05,
                              Local_Name=[0x08,0x74,0x65,0x73,0x74],
                              Service_Uuid_length = 0;
                              Service_Uuid_List = NULL;
                              Slave_Conn_Interval_Min = 0x0006,
                              Slave_Conn_Interval_Max = 0x0008);
```

由于主/从设备也是中央设备，它分别从 Slave_A 和 Slave_B 设备接收特征值的相关通知事件。

9. 一旦主/从设备进入发现模式，它还等待来自被配置为 GAP 中央设备的其他 BLE 设备（称为主设备）的连接请求。主设备启动发现流程，以便发现主/从设备：

```
ret = aci_gap_start_gen_disc_proc(LE_Scan_Interval=0x10,
                                  LE_Scan_Window=0x10,
                                  Own_Address_Type = 0x0,
                                  Filter_Duplicates = 0x0);
```

10. 一旦发现主/从设备，主设备启动连接流程以便与之连接：

```
/*   主设备连接主/从设备：主/从设备
     地址类型和地址已在借助广播报告事件的发现流程中找到 */
     ret= aci_gap_create_connection(LE_Scan_Interval=0x0010,
                                    LE_Scan_Window=0x0010,
                                    Peer_Address_Type=“主/从设备地址类型”,
                                    Peer_Address=“主/从设备地址”,
                                    Own_Address_Type = 0x0;
                                    Conn_Interval_Min=0x6c,
                                    Conn_Interval_Max=0x6c,
                                    Conn_Latency=0x00,
                                    Supervision_Timeout=0xc80,
                                    Minimum _CE_Lenght=0x000c
                                    Maximum_CE_Length=0x000c);
```

通过使用 hci_le_advertising_report_event()事件回调函数可得知主/从设备被发现。

11. 一旦连接，主设备使用 aci_gatt_write_char_desc()API 在主/从设备上启用特征通知。

12. 在该阶段，由于是 GAP 中央设备，主/从设备接收来自从设备 A、从设备 B 的特征通知，而作为 GAP 外设，它还能将这些特征值通知主设备。

4.11 低能耗蓝牙私有 1.2

BLE 协议栈 v2.x 支持低功耗蓝牙私有 1.2。

私有功能通过频繁修改相关 BLE 地址来降低跟踪特定 BLE 的能力。频繁修改的地址被称为私有地址，可通过可信设备进行解析。

为使用该功能，通信中涉及的设备需要先经过配对：使用在先前的配对/绑定过程期间交换的设备 IRK 创建私有地址。

私有功能有两种变体：

1. 基于主机的隐私私有地址通过主机解析和生成
2. 基于控制器的隐私私有地址通过控制器解析和生成，在主机提供控制器设备身份信息后，主机不再介于。

当支持控制器隐私时，可进行设备过滤，因为在控制器中执行地址解析（可在检查对端设备是否处于白名单中之前解析对端设备的身份地址）。

4.11.1 基于控制器的私有与设备过滤方案

在 STM32WB 上，aci_gap_init() API 支持 privacy_enabled 参数的以下选项：

- 0x00：私有禁用
- 0x01：主机私有启用
- 0x02：控制器私有启用

当从设备想要对可解析的私有地址进行解析，并希望能够过滤私有地址以重新连接已绑定和可信设备时，必须执行以下步骤：

1. 在 aci_gap_init()上启用私有控制器：将 0x02 用作 privacy_enabled 参数。
2. 使用允许的安全方法之一连接、配对和绑定候选可信设备：使用设备 IRK 创建私有地址。
3. 调用 aci_gap_configure_whitelist() API，以将绑定设备的地址添加到 BLE 设备控制器的白名单中。
4. 使用 aci_gap_get_bonded_devices() API 获取绑定的设备身份地址和类型。
5. 使用 aci_gap_add_devices_to_resolving_list() API 将绑定设备的身份地址和类型添加到控制器中的可解析私有地址转换列表中。
6. 通过使用参数 Own_Address_Type = 0x02（可解析私有地址）和 Adv_Filter_Policy = 0x03（只允许来自白名单的扫描请求，只允许来自白名单的连接请求）来调用 aci_gap_set_undirected_connectable() API，设备进入了非定向可连接模式。
7. 当绑定的主设备执行重新连接到从设备的连接过程时，从设备能够解析并过滤主设备的地址并与之连接。

4.11.2 解析地址

在与绑定设备重新连接之后，不必通过严格解析对端设备的地址来加密链路。事实上，STM32WB 协议栈将自动寻找正确的 LTK 以加密链路。

但是，在某些情况下必须解析对端设备的地址。当设备收到可解析私有地址时，可通过主机或控制器来解析该地址（如链路层）。

基于主机的私有

如果没有启用控制器私有，则可以使用 aci_gap_resolve_private_addr()来解析可解析的私有地址。如果可以在为绑定设备存储的 IRK 列表中找到相应的 IRK，则解析该地址。可通过以下方式接收可解析的私有地址，当 STM32WB 正在扫描时，通过 hci_le_advertising_report_event()，或当连接已建立时，通过 hci_le_connection_complete_event()。

基于控制器的私有

如果在链路层启用了地址解析，则在收到可解析的私有地址时使用解析列表。要将绑定设备添加到解析列表，必须调用 aci_gap_add_devices_to_resolving_list()。此函数搜索相应的 IRK 并将其添加到解析列表中。

当启用私有时，如果已将设备添加到解析列表，则其地址通过链路层自动解析并报告给应用，而无需明确调用任何其他功能。在与设备连接后，返回 hci_le_enhanced_connection_complete_event()。如果已成功解析，则该事件报告设备的身份地址（如果 hci_le_enhanced_connection_complete_event()被屏蔽，则只返回 hci_le_connection_complete_event()）。

在扫描时，如果该设备使用可解析的私有地址并且其地址已正确解析，则 hci_le_advertising_report_event()包含广播中的设备身份地址。在这种情况下，报告的地址类型为 0x02 或 0x03。如果找不到可解析该地址的IRK，则报告可解析的私有地址。在广播设备使用了定向广播的情况下，如果已取消屏蔽，并将扫描过滤策略设为 0x02 或 0x03，则通过 hci_le_advertising_report_event()或通过 hci_le_direct_advertising_report_event()报告已解析的私有地址。

4.12 ATT_MTU 和交换 MTU API 事件

ATT_MTU 定义为在客户端和服务器之间发送的任何数据包的最大大小：

- 默认 ATT_MTU 值：23 字节

这决定了用于执行特征操作时的当前最大属性值的大小（通知/写入最大值为 ATT_MTU-3）。

客户端和服务器可以使用交换 MTU 请求和响应消息交互所能接收的最大数据包大小信息。两个设备均使用这些交互信息中的最小值来进行所有的进一步通信：

```
tBleStatus aci_gatt_exchange_config(uint16_t Connection_Handle);
```

为响应交换 MTU 请求，在两个设备上触发 aci_att_exchange_mtu_resp_event()回调：

```
void aci_att_exchange_mtu_resp_event(uint16_t Connection_Handle, uint16_t
                                     Server_RX_MTU);
```

Server_RX_MTU 指定了服务器和客户端之间所商定的 ATT_MTU 值。

4.13 LE 数据包长度扩展 API 和事件

在 BLE 规范 v4.2 中，包数据单元（PDU）的大小已从 27 字节增加到 251 字节。这允许通过降低数据包所需的开销（包头，MIC）来提高数据率。因此，由于降低了开销，可实现更快的 OTA 固件升级操作和更高的通讯效率。

STM32WB 协议栈支持 LE 数据包长度扩展功能和相关 API、事件：

- HCI LE API（API 原型）
  - hci_le_set_data_length()
  - hci_le_read_suggested_default_data_length()
  - hci_le_write_suggested_default_data_length()
  - hci_le_read_maximum_data_length()
- HCI LE 事件（事件回调原型）
  - hci_le_data_length_change_event()

hci_le_set_data_length() API 使用户应用程序能够建议用于给定连接的最大传输数据包大小(TxOctets)和最大数据包(TxTime)传输时间：

```
tBleStatus hci_le_set_data_length(uint16_t Connection_Handle,
                                  uint16_t TxOctets,
                                  uint16_t TxTime);
```

支持的 TxOctets 值在[27-251]的范围内，TxTime 按以下方式提供：(TxOctets +14)*8.

在设备连接后，一旦在 STM32WB 设备上执行 hci_le_set_data_length() API，如果连接的对端设备支持 LE 数据包长度扩展，则会在两个设备上引发以下事件：

```
hci_le_data_length_change_event(uint16_t Connection_Handle,
                                uint16_t MaxTxOctets,
                                uint16_t MaxTxTime,
                                uint16_t MaxRxOctets,
                                uint16_t MaxRxTime)
```

该事件通知主机更改最大链路层有效负载长度或链路层数据通道 PDU 在任一方向（TX 和 RX）的最长时间。所报告的值(MaxTxOctets, MaxTxTime, MaxRxOctets, MaxRxTime)是在更改后实际用于连接的最大值。

4.14 STM32WB LE 2M PHY

蓝牙核心规范 5.0 版中引入了 LE 2M PHY，它使物理层可以在高达 2Mb/s 的数据速率下运行。LE 2M PHY 的数据速率是标准 LE 1M PHY 的两倍，这降低了采用相同传输功率时的功耗。但是，由于符号速率增加，传输距离低于 LE 1M PHY。STM32WB 协议栈同时支持 LE 1M PHY 和 LE 2M PHY，由应用层决定所要使用的默认 PHY。应用可以随时根据需要发起更改 PHY 参数，在每个连接通道上（通过连接句柄）选择不同的 PHY 参数。由于 STM32WB 处理不对称连接，因此应用还可以在每个接收和发送方向上（通过连接句柄）使用不同的 PHY。PHY 协商在应用端是透明的，并依赖于远程功能。STM32WB 协议栈支持以下指令：

- HCI_LE_SET_DEFAULT_PHY：允许主机指定其首选的 TX & RX PHY 参数。
- HCI_LE_SET_PHY：允许主机为 TX & RX PHY 参数设置适合当前连接（通过连接句柄标识）的 PHY 偏好。
- HCI_LE_READ_PHY：允许主机读取当前连接（通过连接句柄标识）上的 TX & RX PHY 参数。

4.15 STM32WB LE 附加信标

此功能作为专有解决方案引入，使最终用户可以获得附加广播信标，作为基本广播特性之外的额外信标（不可连接）。

STM32WB 额外信标解决方案具有非定向的不可连接模式，且未忽略隐私特性和白名单。额外信标包括三个固定的 1 Mbit/s PHY 通道选集（通道 37、38 和 39），此通道集采用专门的发送功率水平，并且其广播数据符合可供应用使用的原始 0..31 字节长度有效负载。

地址可以是随机地址或公共地址（如果当前未被标准广播使用），将由最终用户决定写入此新 BD 地址（两个地址：标准广播和附加信标采用不同的地址）。

应用可以通过 GAP 指令初始化额外信标功能，（无相关的 HCI 指令），以使 STM32WB 支持 LE 附加信标：

- GAP_Additional_Beacon_Mode_Start，从而允许主机通过以下参数启动附加信标：
  - 广播类型：ADV_NONCONN_IND（不可连接非定向广播）以及用于未来增强的参数
    - 广播间隔最小值/最大值：20 ms - 10.24 s
    - 信标地址类型和值：地址类型（公共或随机）以及最终用户写入新的 BD 地址值。
    - AdDataLen 和 AdvData：数据及其数据值的长度
    - PA 水平：从- 40dBm 至 5 dBm 之间的传输输出水平
- GAP_Additional_Beacon_Mode_Stop：允许主机停止附加信标
- GAP_Additional_Beacon_Update_Data：允许主机通过以下参数更改附加信标数据：
  - AdvDataLen 和 AdvData：数据及其数据值的长度

注意： *广播始终预留 14.6 ms（10 ms 用于随机广播延迟，其余时间用于 3 个通道广播、扫描请求/响应和保护时间）*

选择广播间隔时将为附加广播留出空间。

第一个启用的广播会选择 HostBaseTime，而所有附加最小/最大间隔将作为模数适应此时间（选择最小/最大广播时需要计算时间裕度）。

旧广播时隙预留仍会影响新的广播最小间隔。如果最小间隔随着一个时隙长度而增加，则接受预留，否则，如果小于一个时隙长度，则不接受。

4.16 STM32WB LE 扩展广播

蓝牙核心规范 5.0 版中引入了 LE 扩展广播，它使最终用户可以广播和发现比以前的“传统广播”更多的数据。

此广播扩展功能可以：

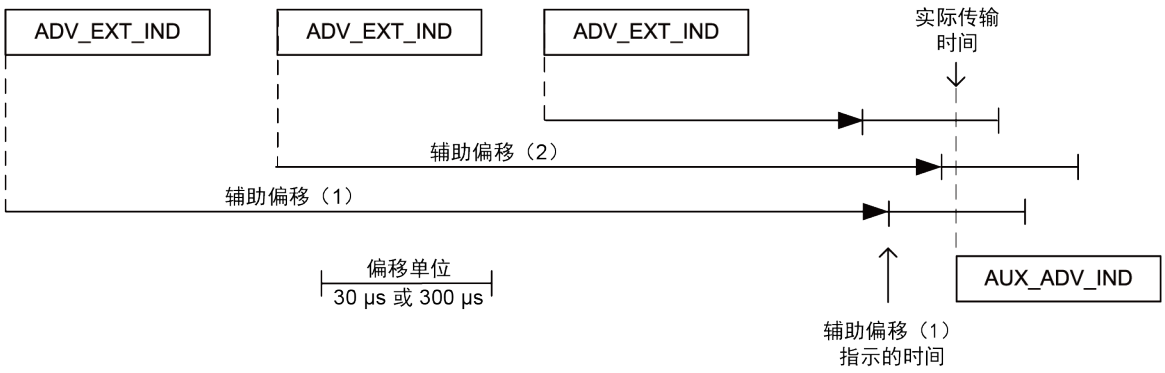
- 扩展无连接场景中的数据长度
- 发送多组广播数据
- 以确定的方式发送广播

4.16.1 扩展广播组

初始广播和传统 PDU 通过 3 个 RF 通道（37、38、39）传输，这些通道称为“主广播物理通道”。新的扩展广播包可以利用低功耗蓝牙® 4.x 连接通道（0-36）扩展广播有效负载，这些通道称为“辅助广播物理通道”。

ADV_EXT_IND 新数据包可以通过主广播 PHY 通道发送。包头字段包括一个新数据 AuxPtr，其包含通道编号（0-36）以及辅助广播 PHY 通道上的辅助数据包指针：大多数信息位于称为 AUX_ADV_IND 的辅助数据包中（参见图 19）。

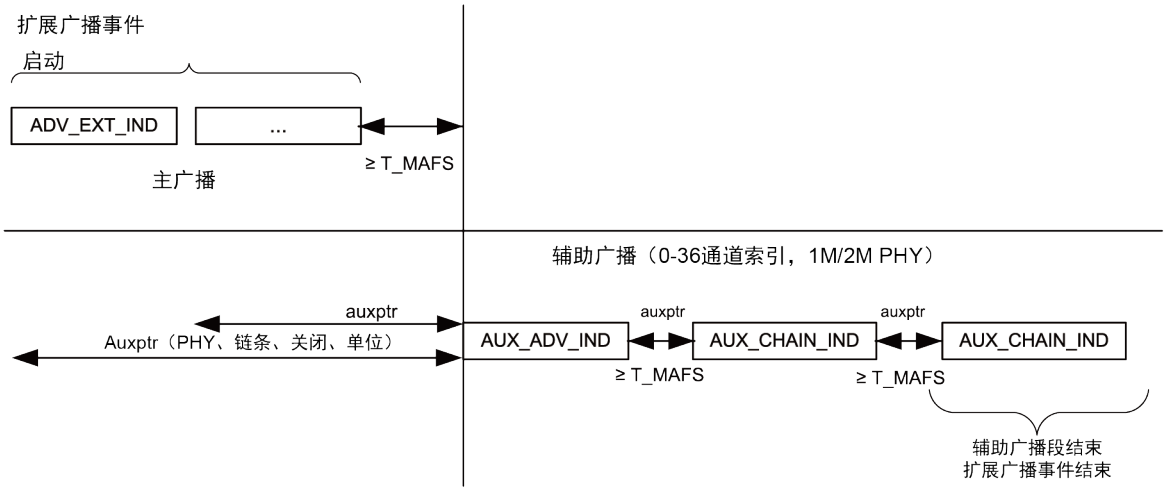
图 19. 广播组示例



此 AUX_ADV_IND 数据包（最多 207 字节）可以通过之前在 ADV_EXT_IND 数据包中定义的 1Mbit PHY 或 2Mbit PHY 发送。

另外，还可以在辅助通道上创建广播数据包链，以传输更多的广播有效负载数据：AUX_CHAIN_IND 数据包（最多 1650 字节），如图 20 所示。辅助通道上的每个广播数据包都包括其 AuxPtr，即用于链条上的下一个广播数据包的下一个辅助通道编号。因此，广播数据包链中的每个链条都可以在不同的辅助通道上发送。

图 20. 链式广播组示例



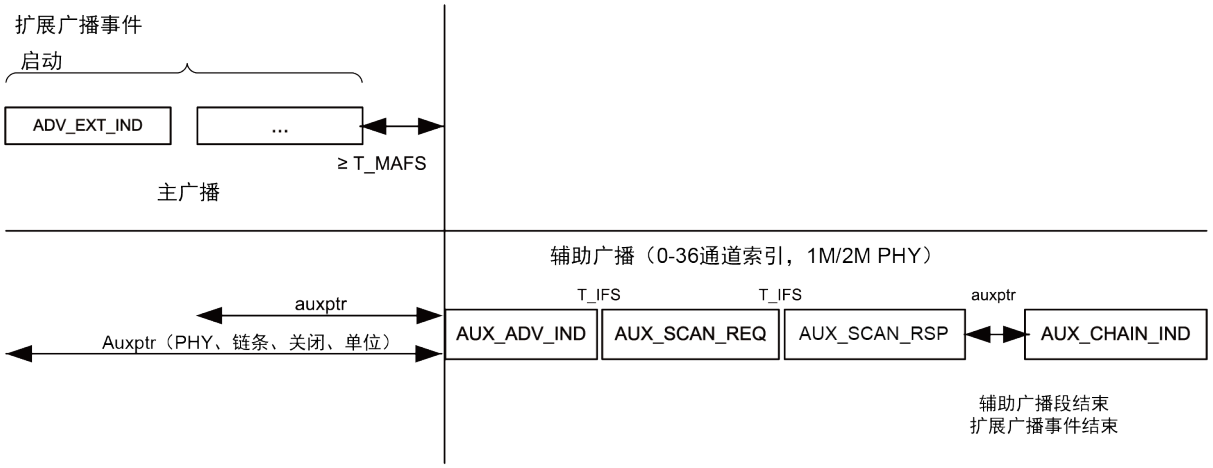
4.16.2 扩展可扫描组

扩展可扫描组允许广播方仅在收到扫描请求时发送数据，并仅对辅助广播通道索引上的数据做出响应。

如果在主通道中使用 ADV_EXT_IND 事件类型进行广播，意味着后续的广播数据将使用 AUX_ADV_IND 在辅助广播通道上进行发送。扫描方通过 AUX_SCAN_REQ 请求更多信息，则广播方在同一个辅助广播通道上通过 AUX_SCAN_RSP 做出响应（如图 21 所示）。

与广播组一样，在扩展可扫描组中也可以在辅助通道上创建广播数据包链，以传输更多的广播有效负载数据：AUX_CHAIN_IND 数据包（最多 1650 字节）。

图 21. 可扫描组示例

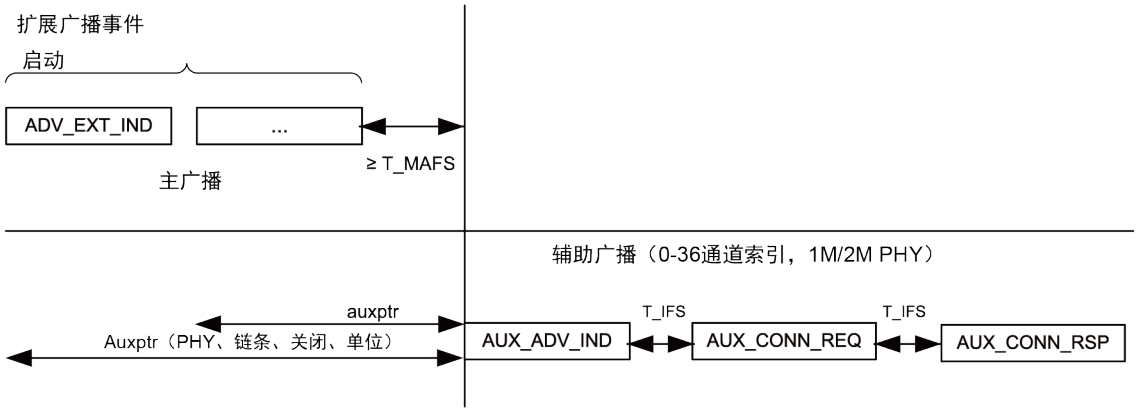


4.16.3 扩展可连接组

使用 ADV_EXT_IND 的可连接定向广播事件类型还允许发起方对辅助广播物理通道上的连接请求做出响应，以建立 ACL 连接。

与此事件相关每个 AUX_ADV_IND 之后，扫描方在同一辅助广播通道上发送 AUX_CONNECT_REQ，则广播方通过 AUX_CONNECT_RSP 做出响应。如图 22 所示。

图 22. 可连接组示例



4.16.4 扩展多组

扩展多组可由最终用户定义（如图 23 所示）。在 STM32WB 协议栈中，多组可以是扩展广播组、扩展可扫描组或扩展可连接组的组合。

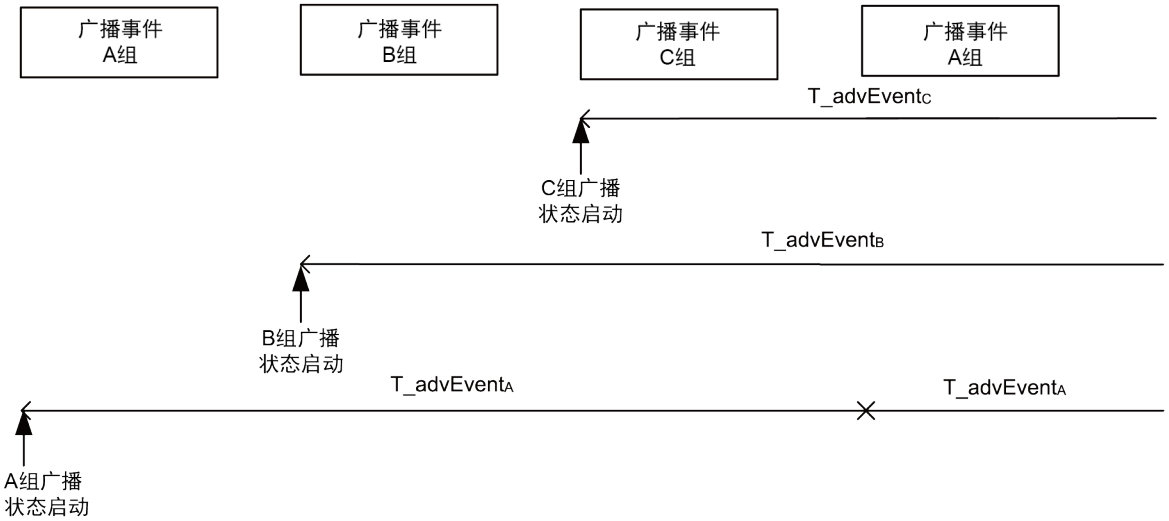
在 STM32WB 协议栈中，最终用户可以定义最多 8 个不同组（最大数据高达 207 字节）。

或者，最终用户可以定义 3 个不同组（最大数据高达 1650 字节）。

每组具有不同的广播参数，如广播 PDU 类型、广播间隔和 PHY。

当使用 ADV_EXT_IND 或 AUX_ADV_IND PDU 广播时，将通过广播 ADI 字段的 SID 子字段，标识广播组。

图 23. 扩展多组示例



4.16.5 LE 扩展扫描

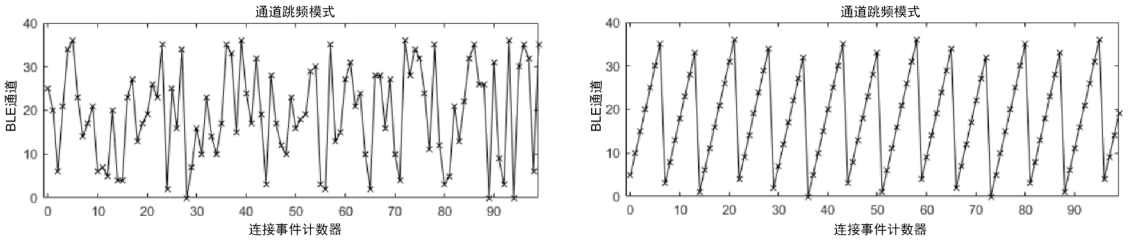
基于相同的原理，STM32WB 协议栈支持扩展扫描功能。

由于扩展广播采用了新的数据包类型和新的 PHY， 这些变化也会体现到扫描流程中去。 在 LE 1M PHY 上扫描主要通道，可获取：

- 传统事件
- 扩展广播事件，有可能会切换到其他 PHY 的辅助广播通道上。扩展扫描也支持多组扫描（最多 8 组）。然而目前在使用扩展扫描接口时尚不支持私有功能。

在扩展广播和扩展扫描中，必须使用新的通道选择算法（CSA）#2。这种新算法更加复杂，更难以跟踪获取下一连接事件的通道索引。但是，与 CSA #1 相比，它可以更加有效地避免干扰和多径衰减效应。图 24 展示了这两种不同的算法。

图 24. 两种不同的通道跳频系统



CSA#1　　CSA#2

4.16.6 STM32WB LE 以连接为导向的通道

L2CAP 连接导向通道通过减少数据包格式开销，支持高效的大数据量传输。服务数据单元（SDU）借助流控制进行可靠传输。大 SDU 的分割和重组由 L2CAP 实体自动完成。多路复用支持同时执行多个服务。

蓝牙核心规范 5.2 版中引用了一种新的 L2CAP 模式，称为基于增强信用的流控制模式。

STM32WB 协议栈上支持这种新 L2CAP 基于增强信用的流控制模式，将允许使用新的 L2CAP PDU、L2CAP_CREDIT_BASED_CONNECTION_REQ 和 L2CAP_CREDIT_BASED_CONNECTION_RSP 动态配置 MTU（最大传输单位）和最大 PDU 有效负载大小（MPS）。

当动态分配通道时，除了 MTU 大小外，还应设置 CID 和 PSM 参数。

动态分配的 CID，用以识别逻辑链路和本地端点（0x0040 至 0xFFFF）。

动态 PSM（协议/服务多路复用器）范围在 0x0080 和 0x00FF 之间。固定 PSM 由 SIG 指定，而动态 PSM 可通过 GATT 发现。

4.17 转换 RSSI 原始值的 STM32WB 公式（dBm）

本节介绍应用层如果读取 STM32WB 射频部分检测到的远程设备的 RSSI（接收信号强度指示）值。为了将接收到的信号强度转换为 dBm 表示的值，用户必须遵循以下步骤：

- READ RSSI SPI 编程：

```
globalParameters.rssiLevel[0] = 0x84; /* 3 个数据字节的读取命令 */
globalParameters.rssiLevel[1] = SPI_RSSI0_DIG_OUT_ADD; /* 3 字节读取：2 字节用于
RSSImeasurement + 1 字节用于 AGC (SPI_AGC_DIG_OUT_ADD) */
globalParameters.rssiLevel[5] = 0; /* 最后一个字节应为 0，以强制使 BLE 内核停止读取 */
```

- RSSI 读取 intoIRQ：

```
if (BLUE_CTRL->RADIO_CONFIG == ((uint32_t)(&(globalParameters.rssiLevel[5])+2)&
0xffff)) /* 检查 BLE Core 是否已正确递增了指针 */
{
globalParameters.rssiValid = globalParameters.rssiLevel[2] +
globalParameters.rssiLevel[3] <<8) + (globalParameters.rssiLevel[4] <<16);
globalParameters.current_action_packet->rssi = IPBLE_LLD_Read_RSSI_dBm() ;
}*/
```

- RSSI Code Conversion in dBm:

```
int32_t IPBLE_LLD_Read_RSSI_dBm(void)
{
      int i = 0 ; int rsi_dbm;
      while(i < 100 && (BLUE_CTRL->RADIO_CONFIG & 0x10000) != 0)
{
      rsi_dbm = i++;
}
int rssi_int16 = globalParameters.rssiValid & 0xFFFF ; /* 前 2 个字节包含在接收信号上测量的 Rssi */
      int reg_agc = (globalParameters.rssiValid >> 16) & 0xFF ; /* 第三个字节是用于 RX 的 AGC */
if(rssi_int16 == 0 || reg_agc > 0xb)
      rsi_dbm = 127 ;
}
else
{
rsi_dbm = reg_agc * 6 -127 ;
while(rssi_int16 > 30)
      {
rsi_dbm = rsi_dbm + 6 ;
rssi_int16 = rssi_int16 >> 1 ;
      }
      rsi_dbm = rsi_dbm + ((417*rssi_int16 + 18080)>>10) ;
}
return rsi_dbm ;
}
```

5 BLE 多连接时序策略

本节提供当多主多从状态下，STM32WB 协议栈的连接时序管理策略的概述。

5.1 关于低功耗蓝牙时序的基本概念

本节介绍与广播、扫描和连接操作相关的低功耗蓝牙时序管理的基本概念。

5.1.1 广播时序

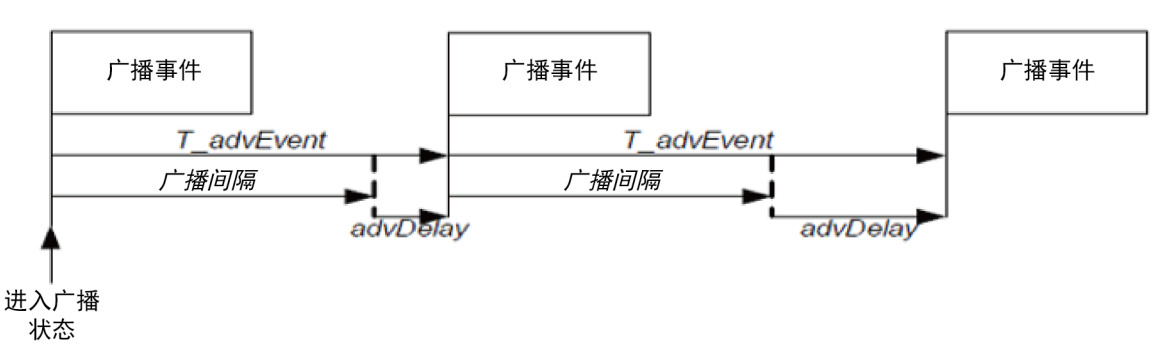
广播状态的时序由 3 个时序参数来定义，它们通过以下公式串联：

T_advEvent = advInterval + advDelay

其中：

- T_advEvent：两个连续广播事件的启动时间之间的时间间隔；如果广播事件类型为可扫描非定向事件类型或不可连接非定向类型，advInterval 应不小于 100 ms；如果广播事件类型为低占空比模式下使用的可连接非定向事件类型或可连接定向事件类型，advInterval 可大于等于 20 ms。
- advDelay：链路层为每个广播事件生成的虚拟随机值，范围为 0 ms 至 10 ms。

图 25. 广播时序



5.1.2 扫描时序

扫描状态的时序由 2 个时序参数来定义：

- scanInterval：两个连续扫描事件的启动时间之间的间隔
- scanWindow：链路层在广播通道上侦听的时间（广播通道的索引（37/38/39）顺序，在每个新的 ScanWindow 时间窗口内可变）。

scanWindow 和 scanInterval 参数小于等于 10.24 s。

scanWindow 小于等于 scanInterval。

5.1.3 连接时序

连接事件的时序取决于 2 个参数：

- 连接事件间隔（connInterval）：两个连续连接事件的启动时间之间的间隔，绝不重叠；连接事件启动的时间点被称为锚点。

主设备在锚点开始向从设备发送数据通道 PDU，而从设备监听其主设备在锚点发送的数据包。

主设备确保连接事件在下一个连接事件的锚点的至少 T_IFS=150 µs（帧间间隔时间，如相同通道索引上连续数据包之间的时间间隔）前结束。

connInterval 是 1.25 ms 的倍数，范围为 7.5 ms 至 4.0 s。

- 从设备延迟（connSlaveLatency）：允许从设备使用更少的连接事件。该参数定义了从设备无需监听主设备的连续连接事件数量。

当主机想要创建连接时，它为控制器提供连接间隔（Conn_Interval_Min、Conn_Interval_Max）和连接长度（Minimum_CE_Length、Maximum_CE_Length）的最大和最小值，从而在选择当前参数方面给予了控制器一定的灵活性，以便满足其他时序限制，例如有多个连接时。

5.2 BLE 协议栈时序和时隙分配概念

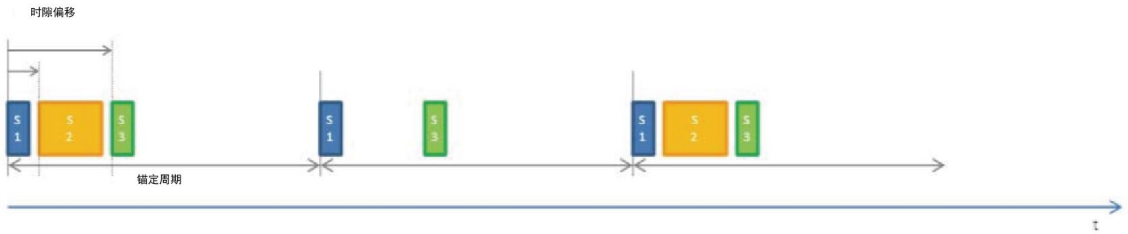
STM32WB BLE 协议栈采用了一种时间槽分配机制，以实现主/从议题设备的连接场景。其基本控制参数，如下表所述：

表 52. 时隙算法的时序参数

参数	说明
锚定周期	循环时间间隔，最多可划分出 8 个连接时隙。 在这 8 个时隙中，同一时间只能有一个时隙可用于扫描或广播时间（扫描和广播事件是互斥的）
时隙持续时间	时间间隔，在此期间发生完整事件（如广播或扫描及连接）；时隙持续时间是分配给连接时隙的持续时间，与连接事件的最长持续时间有关
时隙偏移	锚定周期的开始时间与连接时隙的开始时间之间的延时对应的时间值
时隙延迟	代表连续锚定周期内特定连接时隙的实际利用率的值。 （例如，时隙延迟等于‘1’表示在每个锚定周期内实际使用了特定连接时隙；时隙延迟等于 n 表示每 n 个锚定周期内只实际使用一次特定连接时隙）

时序分配概念允许对多个连接进行明确的时间处理，但同时对控制器可以接受的实际连接参数强加了一些限制。时基参数和连接时隙分配的示例如下图所示

图 26. 三个连接时隙的分配示例



时隙 1 就锚定周期而言的偏移为 0，时隙 2 的时隙延迟 = 2，所有时隙相隔 1.5 ms 的保护时间。

5.2.1 为第一个主设备连接设置时序

上述时基机制在创建首个主设备连接时实际启动。此类首个连接的参数决定了锚定周期的初始值，并影响与首个连接同步的任何其他主设备连接可以接受的时序设置。

尤其是：

- 选择的初始锚定周期等于主机请求的最大和最小连接周期的平均值
- 第一个连接时隙位于锚定周期开始处
- 第一个连接时隙的持续时间设置为等于请求的连接长度的最大值

很明显，此类首个连接时隙与锚定周期相比的相对持续时间，限制了为其他主设备连接分配其他连接时隙的可能性。

5.2.2 为其他主设备连接设置时序

在配置并启动了时基（如上文所述）后，时隙分配算法将尝试在一定范围内动态地重新配置时基，以分配其他主机请求。

具体来说，考虑下列三种情况：

1. 当前锚定周期处于为新连接指定的 Conn_Interval_Min 至 Conn_Interval_Max 的范围内。这种情况下，不能修改时基，将新连接的连接间隔设置为等于当前锚定周期。
2. 当前锚定周期小于新连接所要求的 Conn_Interval_Min。这种情况下，算法搜索满足下列条件的整数 m：
   Conn_Interval_Min ≤ Anchor_Period × m ≤ Conn_Interval_Max
   如果找到这样的值，则维持当前锚定周期，并将新连接的连接间隔设置为等于 Anchor_Period*m，时隙延迟等于 m。
3. 当前锚定周期大于新连接所要求的 Conn_Interval_Max。这种情况下，算法搜索满足下列条件的整数 k：

   $$Conn\_Interval\_Min \le \frac{Anchor\_Period}{k} \le Conn\_Interval\_Max$$

   如果找到这样的值，则将当前锚定周期缩短为：

   $$\frac{Anchor\_Period}{k}$$

   将新连接的连接间隔设置为：

   $$\frac{Anchor\_Period}{k}$$

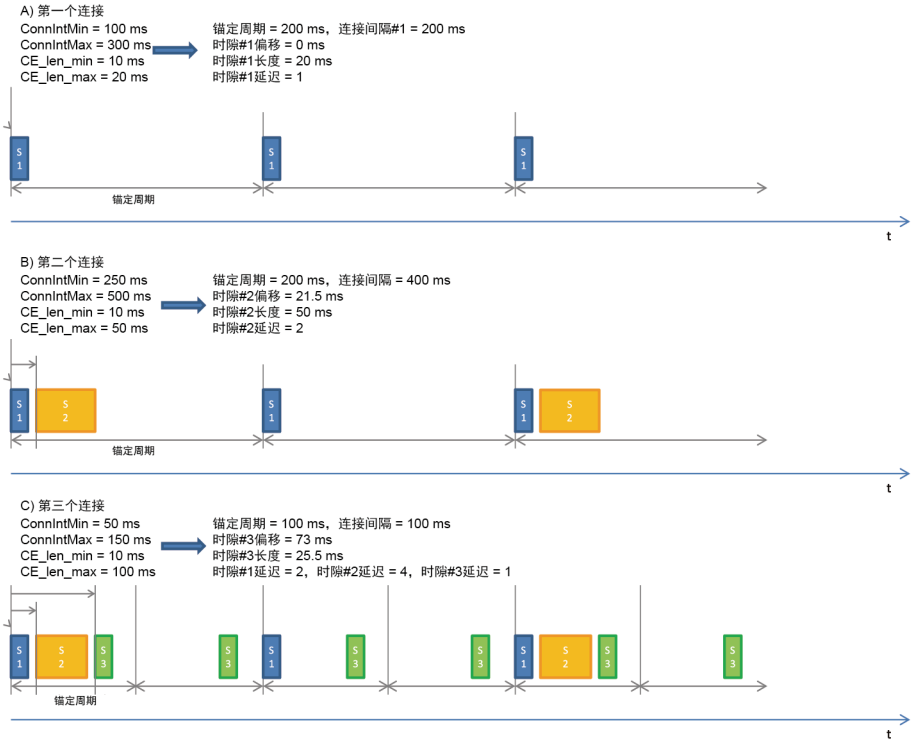
   将现有连接的时隙延迟乘以因数 k。注意，在本例中，还必须满足以下条件：
   - *Anchor_Period/k* 必须是 1.25 ms 的倍数
   - *Anchor_Period/k* 必须足够大，以包含已分配给之前的连接的所有连接时隙

一旦找到符合上述标准的合适锚定周期，在其中为实际连接时隙分配时间间隔。一般而言，如果锚定周期内有足够间隔，算法将分配最大请求连接事件长度，否则将缩短为实际可用间隔。

如果创建了多个连续连接，通常按顺序排列相对连接时隙，中间以很短的保护时间（1.5 ms）隔开；当连接关闭时，通常在两个连接时隙之间产生未使用的间隙。在之后创建新连接时，算法尝试将新连接时隙安插到现有间隙之一；如果间隙不够宽，则直接将连接时隙排列在最后一个连接时隙之后。

图 27. 三个连续连接的时序分配示例展示了当创建连续连接时如何管理时基参数的示例。

图 27. 三个连续连接的时序分配示例



5.2.3 广播事件时序

广播事件的周期（由 advInterval 控制）是基于下列参数计算的，这些参数由从设备通过主机在 HCI_LE_Set_Advertising_parameters 指令中指定：

- *Advertising_Interval_Min, Advertising_Interval_Max;*
- *Advertising_Type;*

如果将 Advertising_Type 设置为高占空比定向广播，则将广播间隔设置为 3.75ms，无论 Advertising_Interval_Min 和 Advertising_Interval_Max 的值为多少；这种情况下，超时也设置为 1.28 s，即这种情况下广播事件的最长持续时间。

在所有其他情况下，选择的广播间隔等于（Advertising_Interval_Min + 5 ms）和（Advertising_Interval_Max + 5 ms）的平均值。和前一种情况相比，这类广播事件没有最长的持续时间，只在连接建立时或主机明确请求时停止。

每个广播事件的长度由软件默认设置为等于 14.6 ms（如允许的最大广播事件长度），并且不能缩短。

在主设备时隙的相同时基内分配广播时隙（如扫描和连接时隙）。因此，将由软件在至少一个主设备时隙时接受的广播启用指令处于激活状态，广播间隔必须是实际锚定周期的整数倍。

5.2.4 扫描时序

主设备通过下列参数（由主机在 HCI_LE_Set_Scan_parameters 指令中指定）请求扫描时序：

- LE_Scan_Interval：用于计算扫描时隙的周期
- LE_Scan_Window：用于计算要分配到主设备时基中的扫描时隙的长度

分配的扫描时隙位于其他激活的主设备时隙（如连接时隙）和广播时隙（如果有一个处于激活状态）的相同时基内。

如果已存在活动时隙，则扫描间隔始终适应锚定周期。

鉴于扫描参数只是 BT 官方规范提供的建议（v.4.1 第 2 卷 E 部分第 7.8.10 节），每当 LE_Scan_Interval 大于实际锚定周期时，为了维持主机要求的相同占空比，软件自动尝试截取 LE_Scan_Interval 并缩短分配的扫描时隙长度（最多达到 LE_Scan_Window 的¼）。

5.2.5 从设备时序

从设备时序由主设备在创建连接时定义，这意味着这种情况下将以异步方式管理从设备链路的连接时隙。从设备假设主设备可以使用与连接间隔一样长的连接事件长度。调度算法基于连接事件持续时间的连续计算动态估计从设备时隙长度，如果发生冲突，则优先选择较短的连接时隙。

调度器还可以对从设备连接时隙持续时间使用动态限制，以保留主设备和从设备连接。详见下一节。

5.3 多个主设备和从设备微网拓扑结构连接指南

STM32WB 设备可用于不同的微网拓扑结构中。STM32WB BLE 协议栈主/从一体设备（在本文中称为 Master_Slave）提供同时处理多达 8 个连接的功能，如下所示：

1. 作为多个从设备的主设备：
   - Master_Slave 可连接最多 8 个从设备
2. 多个主设备的从设备：
   - Master_Slave 可连接最多 8 个主设备
3. 同时多个主设备和从设备：
   a. Master_Slave 作为主设备时，可连接最多 x 个从设备（x ≤ 8），同时，相同的 Master_Slave 设备作为从设备时，可连接最多 8-x 个主设备
   b. 设备可以在多从设备模式下扫描、广播和连接为主设备
   c. 设备可以在多主设备模式下扫描、广播和连接为从设备

为了使用 STM32WB 设备正确处理多个主设备和从设备连接，必须遵循下列指南：

1. 避免分配过长的连接事件长度：选择尽可能小的 Minimum_CE_Length 和 Maximum_CE_Length 以严格满足应用需求。这样做有助于分配算法在锚定周期内分配多个连接并缩短锚定周期，如果需要，为连接分配较小的连接间隔。
2. 对于第一个主设备连接：
   a. 如可能，创建具有最短连接间隔的连接作为第一个连接，以便为其他连接分配为初始锚定周期倍数的连接周期。
   b. 如可能，为 Conn_Interval_Min = Conn_Interval_Max 选择 10 ms 的倍数，以便为其他连接分配既是初始锚定周期的 2、4 和 8（或更多）倍的约数又是 1.25 ms 的倍数的连接间隔。
3. 对于其他主设备连接：
   a. 使 ScanInterval 等于现有主设备连接之一的连接间隔
   b. 选择 ScanWin，使已分配主设备时隙之和（包括广播（如激活））小于最短已分配连接间隔
   c. 选择 Conn_Interval_Min 和 Conn_Interval_Max，使间隔包含：
      - 最短已分配连接间隔的倍数
      - 最短已分配连接间隔的因数也是 1,25 ms 的倍数
   d. 选择 Maximum_CE_Length = Minimum_CE_Length，使已分配主设备时隙之和（包括广播（如激活））加上 Minimum_CE_Length 后小于最短已分配连接间隔

4. 每次启动其他从设备连接的广播时：
   a. 选择 Advertising_Interval_Min = Advertising_Interval_Max = 最短已分配连接间隔的整数倍。
   b. 与主设备连接后，对于其他从设备与其他主设备的连接，建议分配以下数值，作为最小主设备连接间隔：
      - 双链路从设备：18.75 ms
      - 三链路从设备：25 ms
      - 四链路从设备：31.25 ms
      - 五链路从设备：37 ms
      - 六链路从设备：50 ms
      - 七链路从设备：55 ms
      - 八链路从设备：62 ms
5. 每次开始扫描时：
   a. 使 ScanInterval 等于现有主设备连接之一的连接间隔
   b. 使 ScanInterval 等于现有主设备连接之一的连接间隔
   c. 选择 ScanWin，使已分配主设备时隙之和（包括广播（如激活））小于最短已分配连接间隔
6. 注意，创建多个连接然后关闭其中一些连接并重新创建新连接的过程，随着时间的推移，会降低时隙分配算法的总体效率。如果在分配新连接时遇到困难，可将时基复位为初始状态（将关闭所有现有连接）。

6 参考文档

表 53. 参考文档

名称	标题/描述
AN5289	使用 STM32WB 系列微控制器构建无线应用
AN5379	STM32WB 系列微控制器上的 AT 指令示例
AN5270	STM32WB 蓝牙低功耗（BLE）无线接口
AN5155	用于 STM32WB 系列的 STM32Cube MCU 封装示例
蓝牙规范	蓝牙系统规范（v4.0、v4.1、v4.2、v5.0、v5.1、5.2）
AN5378	STM32WB 系列微控制器起动流程
AN5071	STM32WB 系列微控制器超低功耗特性概述

7 缩写和缩略语列表

本节列出了文档中使用的标准缩写和缩略语。

表 54. 缩略语列表

术语	意义
ACI	应用命令接口
ATT	属性协议
BLE	低功耗蓝牙
BR	基础速率（BR）
CRC	循环冗余校验
CSRK	连接签名解析密钥
EDR	增强数据率
DK	开发套件
EXTI	外部中断
GAP	通用访问配置文件
GATT	通用属性配置文件
GFSK	高斯频移键控
HCI	主机控制器接口
IFR	信息寄存器
IRK	身份解析密钥
ISM	工业、科学和医疗
LE	低能耗
L2CAP	逻辑链路控制和适配协议
LTK	长期密钥
MCU	微控制器单元
MITM	中间人
NA	不适用
NESN	下一个序号
OOB	带外
PDU	协议数据单元
射频器件	射频
RSSI	接收的信号强度指示
SIG	技术联盟
SM	安全管理器
SN	序号
USB	通用串行总线
UUID	通用唯一标识符
WPAN	无线个人局域网

版本历史

表 55. 文档版本历史

日期	版本	变更
2020 年 7 月 2 日	1	初始版本
2020 年 12 月 11 日	2	增加了： • 第 4.5.1 节 配对流程图：主序列发起的配对请求（传统） • 第 4.5.2 节 配对流程图：主序列发起的配对请求（安全） • 第 4.5.3 节 配对流程图：从序列发起的配对请求（安全） • 第 4.8 节 使用 GATT 实现端到端接收流控制 • 第 4.17 节 转换 RSSI 原始值的 STM32WB 公式（dBm） 更新了： • 第 2.8.1 节 设备过滤
2021 年 2 月 11 日	3	更新了： • 章节介绍
2021 年 12 月 1 日	4	更新了： • 第 2.10 节 通用访问配置文件（GAP） • 第 4.1.1 节 BLE 地址 • 第 4.2 节 服务和特征配置 • 第 4.15 节 STM32WB LE 附加信标 • 第 5 节 BLE 多连接时序策略 • 第 5.1.2 节 扫描时序 • 第 5.2.2 节 为其他主设备连接设置时序 • 第 5.2.5 节 从设备时序 • 第 5.3 节 多个主设备和从设备微网拓扑结构连接指南 • 微网拓扑结构连接公式 删除了： • 图 10 BLE MAC 地址存储 • 第 5.4 节 微网拓扑结构连接公式
2022 年 7 月 6 日	5	更新了： • 引言 • 第 4 节 使用 STM32WB BLE 协议栈设计应用 增加了： • 第 4.16 节 STM32WB LE 扩展广播 • 第 4.16.1 节 扩展广播组 • 第 4.16.2 节 扩展可扫描组 • 第 4.16.3 节 扩展可连接组 • 第 4.16.4 节 扩展多组 • 第 4.16.5 节 LE 扩展扫描 • 第 4.16.6 节 STM32WB LE 以连接为导向的通道

目录

表格索引

图片目录