ESP32 上的蓝牙开发

杨子凡

Dec 28, 2025

ESP32 作为一款高度集成的微控制器，在蓝牙开发领域脱颖而出，主要得益于其强大的硬件规格。ESP32 搭载双核 Xtensa LX6 处理器，主频可达 240 MHz，同时支持低功耗模式，这使得它特别适合资源受限的嵌入式应用。此外，ESP32 集成了 Wi-Fi 和 Bluetooth 功能，其中 Bluetooth Low Energy（BLE）支持高达 5.0 版本，提供长距离传输和 mesh 网络能力。这些优势让 ESP32 在智能家居设备如智能灯泡和门锁、穿戴设备如健身手环、物联网传感器网络以及无线遥控器等领域广泛应用。相比传统蓝牙模块，ESP32 无需额外芯片，降低了成本和功耗，并简化了电路设计。

蓝牙技术主要分为经典蓝牙（BR/EDR）和低功耗蓝牙（BLE）两种。经典蓝牙适用于高带宽场景，如音频传输，数据速率可达 3 Mbps，但功耗较高。BLE 则针对物联网优化，采用低功耗设计，广播间隔可低至几毫秒，适合电池供电设备。ESP32 支持两种协议栈：Bluedroid 是 Espressif 官方的全功能栈，支持经典蓝牙和 BLE，API 丰富但内存占用较大；NimBLE 是轻量级纯 BLE 栈，内存需求仅为 Bluedroid 的一半，更适合内存紧张的设备。本文将重点讲解 BLE 开发，同时覆盖经典蓝牙基础。

本文针对 Arduino 和 ESP-IDF 初学者到中级开发者，提供从环境搭建到实战项目的完整指南。读者需具备基本的 C/C++ 编程知识，以及 Arduino IDE 或 ESP-IDF 开发环境的搭建经验。通过阅读，你将掌握 BLE Peripheral 和 Central 模式开发、协议栈选择、低功耗优化等多项技能。文章结构从基础知识逐步深入高级主题，最后以完整项目收尾，帮助你快速上手 ESP32 蓝牙开发。

1　开发环境搭建

ESP32 蓝牙开发的首要步骤是准备硬件。推荐使用 ESP32-DevKitC 或 NodeMCU-32S 等开发板，这些板载 CP210x 或 CH340 USB 转串口芯片，便于调试。必需配件包括数据线和手机或电脑作为 BLE 测试设备。如果使用裸芯片开发，还需外接天线和电源管理模块。

软件环境安装从 Arduino IDE 开始，这是初学者友好选择。下载 Arduino IDE 2.x 版本后，在文件偏好设置中添加板卡管理器 URL：https://espressif.github.io/arduino-esp32/。然后在板卡管理器搜索“esp32”并安装最新包。ESP-IDF 适合专业开发，推荐 v5.1 或更高版本，使用 VS Code 配合官方 ESP-IDF 插件，一键安装工具链，包括编译器和调试器。PlatformIO 是另一高效选项，在 VS Code 中安装后，它自动管理依赖和库，支持 Arduino 和 ESP-IDF 框架切换。无论选择哪种，都需安装 USB 驱动：Windows 用户下载 CP210x 或 CH340 驱动，macOS 和 Linux 通常自动识别，但需检查权限。

验证环境的关键是运行“Hello World”示例。在 Arduino IDE 中，选择 ESP32 Dev Module 板卡，上传简单 Blink 代码后打开串口监视器，波特率设为 115200。若看到日志输出，即环境正常。针对蓝牙模块，上传 BLE 扫描示例，检查日志中是否出现“Bluetooth initialized”信息。常见问题包括板卡未正确选择导致上传失败、波特率不匹配引起乱码，或 Linux 下串口权限不足，可用 sudo 命令或添加用户到 dialout 组解决。通过这些

步骤，确保开发链路顺畅，为后续蓝牙编程奠基。

2 蓝牙基础知识

BLE 协议栈架构从物理层向上分层，包括 L2CAP（逻辑链路控制适配协议）提供数据分段，ATT（属性协议）定义读写操作，GATT（通用属性配置文件）封装服务和特征值，GAP（通用访问配置文件）管理设备发现和连接。核心角色有 Advertiser（广播设备，不断发送广告包）、Scanner（扫描设备监听广播）、Central（中心设备主动连接）和 Peripheral（外设设备被动等待）。典型场景中，ESP32 作为 Peripheral 广播服务，手机作为 Central 扫描并连接。

ESP32 提供两种蓝牙协议栈：Bluedroid 功能全面，支持经典蓝牙和 BLE，稳定性高但静态 RAM 占用约 100 KB，适合复杂项目；NimBLE 仅支持 BLE，内存占用仅 20 KB，低功耗优化出色，适用于电池设备。选择取决于项目需求，轻量项目优先 NimBLE。

GATT 是 BLE 数据交换核心，使用服务（Service）和特征值（Characteristic）组织数据。服务由 16 位或 128 位 UUID 标识，如标准心率服务 UUID 为 0x180D。特征值支持属性如 Read（只读）、Write（只写）、Notify（通知客户端数据变化）和 Indicate（带确认的通知）。开发者自定义 UUID 时，使用 128 位格式如 “12345678-1234-5678-1234-56789abcdef0”避免冲突。这些概念是后续开发的基石。

3 BLE 周边设备（Peripheral）开发

创建 BLE Peripheral 的基础是广播设备。首先初始化控制器并设置设备名，然后启动广告。ESP-IDF 示例代码如下：

```
esp_bt_controller_config_t bt_cfg = BT_CONTROLLER_INIT_CONFIG_DEFAULT();
esp_bt_controller_init(&bt_cfg);
esp_bt_controller_enable(ESP_BT_MODE_BLE);
esp_bluedroid_init();
esp_bluedroid_enable();
esp_ble_gap_set_device_name("ESP32_BLE");
esp_ble_adv_data_t adv_data = {
    .set_scan_rsp = false,
    .include_name = true,
    .manufacture_len = 0,
};
esp_ble_gap_config_adv_data(&adv_data);
esp_ble_gap_start_advertising(&adv_params);
```

这段代码逐行解析：esp_bt_controller_init 使用默认配置初始化硬件控制器，支持 BLE 模式；esp_bt_controller_enable 启用 BLE 并分配内存；esp_bluedroid_init 和 esp_bluedroid_enable 初始化 Bluedroid 栈，提供 GAP 和 GATT API；esp_ble_gap_set_device_name 设置可见设备名为 “ESP32_BLE”，手机扫描时显示；esp_ble_adv_data_t 结构体配置广告数据，include_name 确保名称包含在内；esp_ble_gap_config_adv_data 应用配置；esp_ble_gap_start_advertising 以默认

参数（间隔 100ms-1000ms）开始广播。此过程约需 100ms，日志显示“GAP_EVT_ADV_START”确认成功。Arduino 版本用 `BLEDevice::init(ESP32_BLE); BLEAdvertising *pAdvertising = BLEDevice::getAdvertising(); pAdvertising->start();`，更简洁封装。

实现 GATT 服务器需定义服务和特征值。以温度传感器为例，创建标准环境感知服务（UUID 0x181A），添加温度特征值（UUID 0x2A6E，支持 Notify）：

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

BLEServer *pServer = NULL;
BLECharacteristic *pTemperatureCharacteristic = NULL;
bool deviceConnected = false;

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) { deviceConnected = true; };
    void onDisconnect(BLEServer* pServer) { deviceConnected = false; };
};

void setup() {
    BLEDevice::init("ESP32_TempSensor");
    pServer = BLEDevice::createServer();
    pServer->setCallbacks(new MyServerCallbacks());
    BLEService *pService = pServer->createService("181A");
    pTemperatureCharacteristic = pService->createCharacteristic(
        "2A6E", BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY);
    pTemperatureCharacteristic->addDescriptor(new BLE2902());
    pService->start();
    BLEAdvertising *pAdvertising = pServer->getAdvertising();
    pAdvertising->start();
}

void loop() {
    if (deviceConnected) {
        float temp = 25.5; // 模拟温度
        uint8_t data[4];
        memcpy(data, &temp, 4);
        pTemperatureCharacteristic->setValue(data, 4);
        pTemperatureCharacteristic->notify();
```

```
        delay(1000);
    }
}
```

解读：BLEServerCallbacks 类重载 onConnect 和 onDisconnect，跟踪连接状态，避免无效通知。setup 中 createService(181A) 创建服务，createCharacteristic 定义特征值，属性组合支持读和 Notify；BLE2902 是标准描述符，启用客户端配置。loop 模拟温度数据，用 memcpy 打包为 4 字节浮点，转发 Notify。手机用 nRF Connect App 连接后订阅特征值，即实时接收温度更新。

安全配对使用 Just Works 模式（无输入设备自动配对）或 Passkey（6 位数字验证）。流程：Central 发送配对请求，Peripheral 响应加密密钥交换，确保数据机密。ESP-IDF 中 esp_ble_gap_set_security_param 配置模式。

调试技巧包括启用日志 esp_log_level_set(*, ESP_LOG_VERBOSE) 查看详细事件，手机 App 如 nRF Connect 显示 RSSI 和包内容，或用 Wireshark 抓包分析 MTU 和 PDU。

4 BLE 中心设备（Central）开发

BLE Central 开发从扫描开始。调用 esp_ble_gap_start_scanning(5) 扫描 5 秒，回调中解析广告数据：

```
static void gap_event_handler(esp_gap_ble_cb_event_t event, esp_ble_gap_cb_param_t *
    ↪ param) {
    if (event == ESP_GAP_BLE_SCAN_RESULT_EVT) {
        esp_ble_gap_cb_param_t *scan_result = (esp_ble_gap_cb_param_t *)param;
        if (scan_result->scan_rst.search_evt == ESP_GAP_SEARCH_INQ_RES_EVT) {
            // 过滤服务 UUID
            if (esp_ble_is_service_uuid_match(scan_result->scan_rst.ble_adv,
                0x181A, NULL)) {
                esp_ble_gap_stop_scanning();
                esp_ble_gattc_open(gattc_if, &scan_result->scan_rst.bda,
                    ↪ BLE_ADDR_TYPE_PUBLIC, true);
            }
        }
    }
}
```

此回调处理扫描结果事件，esp_ble_is_service_uuid_match 检查环境感知服务 UUID，若匹配则停止扫描并连接。解析广告包的 ble_adv 字段提取设备地址（BDA）和类型。

连接后进行 GATT 客户端操作。服务发现用 esp_ble_gattc_search_service(gattc_if, conn_id, &filter)，filter 指定 UUID。发现服务后，获取特征值句柄并读写：

```
esp_ble_gattc_read_char(gattc_if, conn_id, char_handle, ESP_GATT_AUTH_REQ_NONE);
```

读操作异步返回数据回调。订阅 Notify 用 esp_ble_gattc_register_for_notifications，客户端收到 Peripheral Notify 时触发事件。

多设备管理通过连接池实现，每个连接有唯一 conn_id，维护数组跟踪状态。自动重连监听 ESP_GAP_BLE_DISCONNECT_EVT，重启扫描和连接逻辑。

5 经典蓝牙（SPP）开发

经典蓝牙传输速度高达 3 Mbps，功耗约 BLE 的 4 倍，适用于串口替代。SPP（串口协议）模拟 RS232，实现透明传输。ESP-IDF 初始化：

```
esp_bt_controller_enable(ESP_BT_MODE_CLASSIC_BT);
esp_bluedroid_enable();
esp_bt_gap_register_callback(gap_cb);
esp_spp_register_callback(spp_cb);
esp_spp_init(ESP_SPP_MODE_CB);
esp_spp_start_srv(ESP_SPP_SEC_NONE, ESP_SPP_ROLE_SLAVE, 10, "SPP_Server");
```

esp_spp_init 以回调模式初始化，esp_spp_start_srv 启动服务器，角色为从机，安全无加密，服务名为“SPP_Server”。回调 spp_cb 处理打开、关闭和数据事件：

```
void spp_cb(esp_spp_cb_event_t event, esp_spp_cb_param_t *param) {
   if (event == ESP_SPP_SRV_OPEN_EVT) {
      // 客户端连接
   } else if (event == ESP_SPP_DATA_IND_EVT) {
      uart_write_bytes(UART_NUM_0, param->data_ind.data, param->data_ind.len);
   }
}
```

数据到达时转发到 UART，实现蓝牙到串口桥接。PC 用串口助手连接“SPP_Server”，发送数据即在 ESP32 串口输出，反之亦然。

6 高级主题与优化

低功耗优化调整广播间隔至 1s，连接参数协商 MTU 至 247 字节，进入 Light Sleep 模式降低至微安级。测试用 ESP Power Monitor 测量电流曲线。

Wi-Fi 和 BLE 共存需通道避让，BLE 默认通道 37-39，Wi-Fi 动态切换。API esp_bluedroid_ble_coex_enable() 启用共存。

BLE OTA 升级分控制服务和数据服务，客户端分包下载到 OTA 分区，验证 CRC 后重启。自定义协议用 CRC16 校验命令帧：头（1 字节命令）+ 数据 + CRC（2 字节）。

性能基准显示 BLE Notify 吞吐 10 KB/s，延迟 20 ms，功耗 5 mA；经典 SPP 为 100 KB/s、50 ms、20 mA。

7 完整项目实战

BLE 智能灯控项目使用 ESP32 连接 LED 和电位器，实现 App 控制亮度和颜色同步。服务 UUID 自定义为 “12345678-1234-5678-1234-56789abcdef0”，特征值控制 PWM 占空比和 RGB 值。完整代码包括连接回调、Notify 状态上报和 PWM 输出。模拟心率监测器用标准 HR 服务（0x180D），定时生成 60-100 bpm 数据，通过 Notify 发送，模拟 MAX30102 传感器。

部署时，用 Flutter 开发跨平台 App，集成 flutter_blue_plus 库扫描和读写。批量测试脚本循环连接多设备，记录丢包率。

8 常见问题与故障排除

无法扫描设备通常因广播未启动，检查 `esp_ble_gap_config_adv_data` 是否调用且广告数据包含服务 UUID。连接断开多为 RSSI 低于 -80 dBm，调整 `esp_ble_tx_power_set(ESP_BLE_PWR_TYPE_DEFAULT, ESP_PWR_LVL_P9)` 提升功率。内存溢出切换 NimBLE，减少 MTU 大小。工具如 BLE Scanner App 显示实时 RSSI，ESP-IDF Monitor 捕获日志。

本文从环境搭建到实战，覆盖 ESP32 蓝牙全链路，掌握后你能开发生产级应用。进阶阅读 ESP-IDF 文档和 Bluetooth SIG 规范。资源包括 Espressif BLE GitHub 示例、Arduino BLE 库和 ESP32 中文社区。欢迎分享你的项目，关注后续 Wi-Fi 系列。