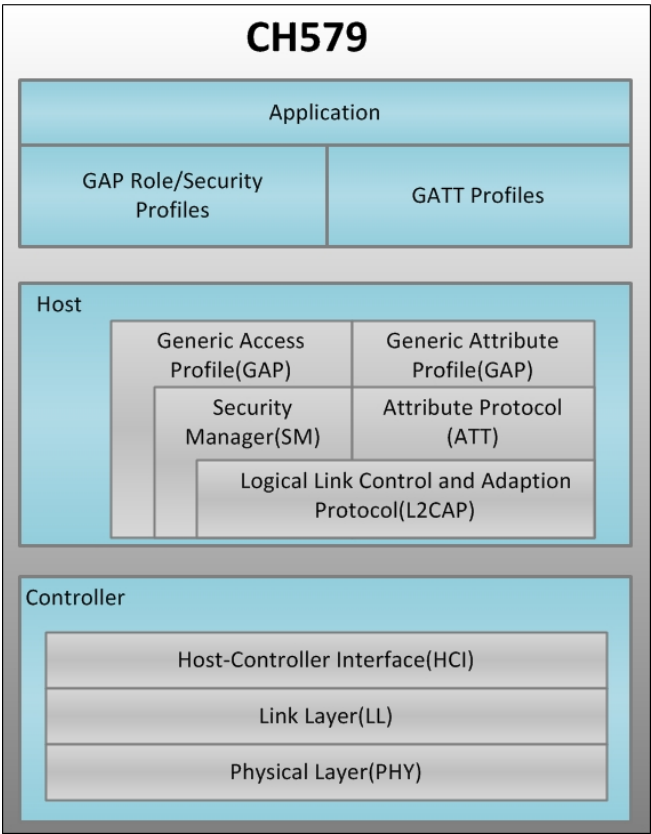


# CH57x BLE 协议栈子程序库说明

版本： V1.5  
<http://wch.cn>

## 1、概述

CH57x集成2.4GHz RF收发器和基带及链路控制，单端RF接口，无需外部电感，简化板级设计。  
CH57xBLE.LIB提供了低功耗蓝牙4.2子程序库，支持4种广播类型，连接通信，支持长包传输，以函数库的形式实现，提供应用层的API调用，快速开发低功耗蓝牙产品。  
协议栈由 Host（主机协议层）和 Controller（控制协议层）组成，结构如下：



### 链路层（Link Layer）

RF 控制器。它控制设备处于准备（standby）、广播（advertising）、监听/扫描（scanning）、初始化（initiating）、连接（connected）这五种状态中一种。围绕这几种状态，BLE 设备可以执行广播和连接等操作，链路层定义了在各种状态下的数据包格式、时序规范和接口协议。

### 通用访问规范（Generic Access Profile）

BLE 设备内部功能对外接口层。它规定了三个方面：GAP 角色、模式和规程、安全问题。主要管理蓝牙设备的广播，连接和设备绑定。

广播者——不可以连接的一直在广播的设备

观察者——可以扫描广播设备，但不能发起建立连接的设备

从机——可以被连接的广播设备，可以在单个链路层连接中作为从机

主机——可以扫描广播设备并发起连接，在单个链路层或多链路层中作为主机

### 逻辑链路控制协议（Logical Link Control and Adaptation Protocol）

主机与控制器直接的适配器，提供数据封装服务。它向上连接应用层，向下连接控制器层，使上层应用操作无需关心控制器的数据细节，。

**安全管理协议 (Security Manager)**

提供配对和密钥分发服务，实现安全连接和数据交换。

**属性传输协议 (Attribute Protocol)**

定义了属性实体的概念，包括 UUID、句柄和属性值，规定了属性的读、写、通知等操作方法和细节。

**通用属性规范 (Generic Attribute Profile)**

定义了使用 ATT 的服务框架和配置文件的结构，两个设备应用数据的通信是通过协议栈的 GATT 层实现。

GATT 服务端——为 GATT 客户端提供数据服务的设备

GATT 客户端——从 GATT 服务器读写应用数据的设备

## 2、库配置变量

### 2.1 协议栈全局变量

通过 BLE\_LibInit 函数配置。见 CH57x\_BLEInit 函数。

MEMAddr	蓝牙协议栈缓存首地址（默认值：无，必须配置）
MEMLen	蓝牙协议栈缓存的长度（默认值：无，必须配置）
SNVAddr	存储配置信息首地址，必须是 data-flash（默认值：无，不保存配对信息）
SNVBlock	存储配置信息块大小（默认值：512）
SNVNum	存储配置信息块数（默认值：1，可存储三个绑定信息）
BufMaxLen	最大支持的长度，范围 27~251，ATT_MTU=BufMaxLen-4（默认值：27）
BufNumber	控制器接收和发送缓存数据包个数（默认值：5）
TxNumEvent	一个连接事件中允许发送的最大包个数（默认值：1）
TxPower	发送功率（默认值：0x15 (0dBm)）
WakeUpTime	配置唤醒后需要的时间，以一个 RTC 时钟为单位（默认值：80）
SeIRTCClock	RTC 时钟选择，0：LSE 1：LSI-32000Hz 2：LSI-32768Hz（默认值：0）
BLEIrqOff	蓝牙中断禁止，0：开启中断 1：关闭中断（默认：0）
MacAddr	配置 MAC 地址（小端）（默认值：出厂值，如配置则优先使用配置值）
ConnectNumber	连接个数，低 2 位 peripheral 个数，紧接着 central 个数
WindowWidening	等待 RF 启动的窗口
WaiteWindow	等待连接事件启动的窗口
srandCB	注册一个随机数种子，可不配置
sleepCB	注册一个空闲回调函数，常用于低功耗模式
tsCB	注册一个获取芯片当前温度函数，根据温度变化完成校准
rcCB	注册一个 LSE 校准函数，根据温度变化协议栈自动完成校准
staCB	注册一个状态回调函数，可在调试时使用

注：灰色标记为需配置信息。

### 2.2 GAP 参数

写函数 GAP\_GetParamValue，读函数 GAP\_SetParamValue。

广播 (Broadcaster)、从机 (Peripheral)	
TGAP_GEN_DISC_ADV_MIN	通用广播模式的广播时长，单位:0.625ms（默认值：0）
TGAP_LIM_ADV_TIMEOUT	限时可发现广播模式的广播时长，单位:1s（默认值：180）
TGAP_DISC_ADV_INT_MIN	最小广播间隔，单位:0.625ms（默认值：160）
TGAP_DISC_ADV_INT_MAX	最大广播间隔，单位:0.625ms（默认值：160）
扫描 (Observer)	
TGAP_DISC_SCAN	扫描时长，单位:0.625ms（默认值：16384）

TGAP_DISC_SCAN_INT	扫描间隔, 单位:0.625ms (默认值: 16)
TGAP_DISC_SCAN_WIND	扫描窗口, 单位:0.625ms (默认值: 16)
主机 (Central)	
TGAP_CONN_EST_SCAN_INT	建立连接的扫描间隔, 单位:0.625ms (默认值: 16)
TGAP_CONN_EST_SCAN_WIND	建立连接的扫描窗口, 单位:0.625ms (默认值: 16)
TGAP_CONN_EST_INT_MIN	建立连接的最小连接间隔, 单位:1.25ms (默认值: 80)
TGAP_CONN_EST_INT_MAX	建立连接的最大连接间隔, 单位:1.25ms (默认值: 80)
TGAP_CONN_EST_SUPERV_TIMEOUT	建立连接的连接管理超时时间, 单位:10ms (默认值:2000)
TGAP_CONN_EST_LATENCY	建立连接的从机潜伏连接事件次数 (默认值: 0)

### 2.3 GAP Role 参数

读函数: GAPRole\_GetParameter, 写函数: GAPRole\_SetParameter。

GAPROLE_PROFILEROLE	当前 GAP 角色配置值, 只读
GAPROLE_BD_ADDR	当前设备地址, 只读
GAPROLE_ADVERT_ENABLED	广播使能配置, 0: 关闭广播, 1: 打开广播
GAPROLE_ADVERT_DATA	广播数据配置, 最大 31 字节
GAPROLE_SCAN_RSP_DATA	扫描应答数据配置, 最大 31 字节
GAPROLE_ADV_EVENT_TYPE	广播类型配置, 默认 GAP_ADTYPE_ADV_IND
GAPROLE_ADV_CHANNEL_MAP	广播通道配置, 默认 37/38/39 三个通道
GAPROLE_MIN_CONN_INTERVAL	连接参数更新允许的最小连接间隔, 单位: 1.25ms (默认值: 6)
GAPROLE_MAX_CONN_INTERVAL	连接参数更新允许的最大连接间隔, 单位: 1.25ms (默认值: 3200)

### 2.4 GAP 服务参数

读函数: GGS\_GetParameter, 写函数: GGS\_SetParameter。

GGS_DEVICE_NAME_ATT	最大 21 字节, 设备名称, 连接后显示在系统中
---------------------	---------------------------

### 2.5 BondMgr 参数

读写函数 GAPBondMgr\_GetParameter, GAPBondMgr\_SetParameter。

GAPBOND_PERI_BONDING_ENABLED GAPBOND_CENT_BONDING_ENABLED	连接绑定信息是否保存 0: 关闭 (默认) 1: 开启
GAPBOND_PERI_IO_CAPABILITIES GAPBOND_CENT_IO_CAPABILITIES	连接绑定过程中配对的交互能力 0x00: 只显示 (默认) 0x01: 只显示, 只可以输入是或否 0x02: 只有输入 0x03: 无显示无输入 0x04: 既有显示, 又可以输入
GAPBOND_PERI_PAIRING_MODE GAPBOND_CENT_PAIRING_MODE	连接绑定过程中的配对模式 0x00: 不支持配对 0x01: 等待配对请求 0x02: 主动发起配对请求 (默认)
GAPBOND_PERI_DEFAULT_PASSCODE GAPBOND_CENT_DEFAULT_PASSCODE	连接绑定过程中的配对 PIN 码, 范围:000000~999999 默认值: 6 个 0
GAPBOND_DISABLE_SINGLEBOND	指定绑定列表中的一个设备无效, 拒绝回连
GAPBOND_ENABLE_SINGLEBOND	解除绑定列表中的一个设备无效, 可以回连
GAPBOND_DISABLE_ALLBONDS	绑定列表中的设备全都无效, 拒绝回连
GAPBOND_ENABLE_ALLBONDS	绑定列表中的设备全都有效, 可以回连

3、库函数

3.1 库子程序

分类	函数名	简要说明
基本函数	BLE_LibInit	库初始化
	TMOS_TimerInit	系统时钟初始化
	TMOS_ProcessEventRegister	注册一个 TMOS 回调函数，并分配 ID
	LL_ConnectEventRegister	注册一个连接事件完成回调函数
	TMOS_SystemProcess	执行系统处理
GAP 函数	GAP_GetParamValue	读 GAP 参数
	GAP_SetParamValue	写 GAP 参数
	GAPRole_GetParameter	读 GAP Role 参数
	GAPRole_SetParameter	写 GAP Role 参数
	GGG_GetParameter	读 GAP 服务参数
	GGG_SetParameter	写 GAP 服务参数
	GAPBondMgr_GetParameter	读绑定相关参数
	GAPBondMgr_SetParameter	写绑定相关参数
GATT 函数	GATTServApp_InitCharCfg	初始化特性配置值
	GATTServApp_ReadCharCfg	读特性配置值
	GATTServApp_RegisterService	注册一个 GAAT 属性列表并添加该服务的读写回调函数
	pfnGATTReadAttrCB_t	读属性值回调函数
	pfnGATTWriteAttrCB_t	写属性值回调函数
	GATT_Notification	主动通知函数

3.2 BLE\_LibInit

函数原型	u32 BLE_LibInit( bleConfig_t* pCfg )
输入	pCfg: 库配置结构体指针
输出	无
返回	返回 0 表示成功，其他值错误
作用	初始化蓝牙库，并配置功能，必须在使用其他库函数之前调用

3.3 TMOS\_TimerInit

函数原型	pfnTIMHandleCB_t TMOS_TimerInit(u8 enable_rtc_irq )
输入	是否开启 RTC 触发中断
输出	无
返回	RTC 中断回调函数

作用	初始化系统时间
----	---------

3.4 TMOS\_ProcessEventRegister

函数原型	tmosTaskID TMOS_ProcessEventRegister( pTaskEventHandlerFn eventCb )
输入	TMOS 任务回调函数
输出	无
返回	分配的 ID 值，0xFF 表示无效
作用	注册一个 TMOS 任务回调函数，并分配一个 ID

3.5 LL\_ConnectEventRegister

函数原型	void LL_ConnectEventRegister( pfnConnectEventCB connectEventCB )
输入	回调处理函数
输出	无
返回	无
作用	注册一个连接事件回调函数

3.6 TMOS\_SystemProcess

函数原型	void TMOS_SystemProcess( void )
输入	无
输出	无
返回	无
作用	执行系统处理

3.7 GAP\_GetParamValue

函数原型	uint16 GAP_GetParamValue( uint16 paramID )
输入	paramID: 需要读的 GAP 参数的索引值 IDs
输出	无
返回	参数值
作用	获取当前 GAP 参数值

3.8 GAP\_SetParamValue

函数原型	bStatus_t GAP_SetParamValue( uint16 paramID, u16 paramValue )
输入	paramID: 需要写的 GAP 参数的索引值 IDs paramValue: 写入的值
输出	无

返回	0：表示成功，2：表示参数无效
作用	写 GAP 参数值

### 3.9 GAPRole\_GetParameter

函数原型	bStatus_t GAPRole_GetParameter( uint16 param, void *pValue )
输入	paramID: 需要读的 GAP 参数的索引值 IDs pValue: 数组指针
输出	获取的 GAP Role 参数
返回	0：表示成功，2：表示参数无效
作用	获取当前 GAP Role 参数值

### 3.10 GAPRole\_SetParameter

函数原型	bStatus_t GAPRole_SetParameter( uint16 param, uint8 len, void *pValue )
输入	paramID: 需要写的 GAP 参数的索引值 IDs len: 写入参数长度 pValue: 写入参数值
输出	无
返回	0：表示成功，2：表示参数无效
作用	写 GAP Role 参数值

### 3.11 GGS\_GetParameter

函数原型	bStatus_t GGS_GetParameter( uint8 param, void *value )
输入	paramID: 需要读的 GAP 参数的索引值 IDs value: 数组指针
输出	获取的 GAP 服务参数
返回	0：表示成功，2：表示参数无效
作用	获取当前 GAP 服务参数值

### 3.12 GGS\_SetParameter

函数原型	bStatus_t GGS_SetParameter( uint8 param, uint8 len, void *value )
输入	paramID: 需要写的 GAP 参数的索引值 IDs len: 写入参数长度 value: 写入参数值
输出	无
返回	0：表示成功，2：表示参数无效
作用	写 GAP 服务参数值

3.13 GAPBondMgr\_GetParameter

函数原型	bStatus_t GAPBondMgr_GetParameter( uint16 param, void *pValue )
输入	paramID: 需要读的 GAP 参数的索引值 IDs pValue: 数组指针
输出	获取的 GAP 绑定参数
返回	0: 表示成功, 2: 表示参数无效
作用	获取当前 GAP 绑定参数值

3.14 GAPBondMgr\_SetParameter

函数原型	bStatus_t GGS_SetParameter( uint8 param, uint8 len, void *value )
输入	paramID: 需要写的 GAP 参数的索引值 IDs len: 写入参数长度 value: 写入参数值
输出	无
返回	0: 表示成功, 2: 表示参数无效
作用	写 GAP 绑定参数值

3.15 GATTServApp\_InitCharCfg

函数原型	void GATTServApp_InitCharCfg( uint16 connHandle, gattCharCfg_t *charCfgTbl )
输入	connHandle: 连接句柄. charCfgTbl: 特性值列表
输出	无
返回	无
作用	初始化特性值置值

3.16 GATTServApp\_ReadCharCfg

函数原型	uint16 GATTServApp_ReadCharCfg( uint16 connHandle, gattCharCfg_t *charCfgTbl )
输入	connHandle: 连接句柄. charCfgTbl: 特性值列表
输出	无
返回	特性值
作用	读取特性值

3.17 GATTServApp\_RegisterService

函数原型	bStatus_t GATTServApp_RegisterService( gattAttribute_t *pAttrs, uint16 numAttrs,
------	--

	uint8 encKeySize, gattServiceCBs_t *pServiceCBs )
输入	pAttrs: 要注册的属性表 numAttrs: 属性数量 encKeySize: 加密 KEY 最小值 pServiceCBs: 服务回调函数指针
输出	无
返回	0: 表示成功, 其他值表示失败
作用	注册一个 GAAT 属性列表并添加该服务的读写回调函数

GATTServApp\_RegisterService 函数内部会分配一个 gattService\_t 结构体, 并且把 pAttrs, numAttrs, encKeySize 这 3 个参数赋值给 gattService\_t 结构体, CH579 协议栈会把所有 gattService\_t 结构体链成一个列表。所有服务都是有 gattAttribute\_t 数组组成, gattServiceCBs\_t 结构体中的 pfnReadAttrCB, pfnWriteAttrCB 回调函数提供对当前服务的读写操作, 如果属性权限中定义 GATT\_PERMIT\_AUTHOR\_READ 或者 GATT\_PERMIT\_AUTHOR\_WRITE, 协议栈会调用 pfnAuthorizeAttrCB 回调函数来处理, 只有 pfnAuthorizeAttrCB 返回成功后, 才会允许读写当前属性。

3. 18 pfnGATTReadAttrCB\_t

函数原型	typedef uint8 (*pfnGATTReadAttrCB_t)( uint16 connHandle, gattAttribute_t *pAttr, uint8 *pValue, uint16 *pLen, uint16 offset, uint16 maxLen, uint8 method );
输入	connHandle: 连接句柄, 表示不同的连接; pAttr: 指向被读取的属性; pValue: 指向被读取的属性值; pLen: 函数返回后, 实际被读取的数据长度; Offset: 被读取的属性值的偏移, 如果属性长度超过 ATT_MTU-1 时, 用块读取 (Read Blob) 才会用到 Offset, 其他情况都是 0; maxLen: pValue 缓冲区的最大长度; Method: 读属性时所用的方法, 可能值 (0x0A=Read Request), (0x0C=Read Blob Request) 等。
输出	读取的数据内容和长度
返回	0: 表示成功, 其他值表示失败
作用	读属性值回调函数

3. 19 pfnGATTWriteAttrCB\_t

函数原型	typedef uint8 (*pfnGATTWriteAttrCB_t)( uint16 connHandle, gattAttribute_t *pAttr, uint8 *pValue, uint16 len, uint16 offset, uint8 method );
------	--



输入	connHandle: 连接句柄, 表示不同的连接; pAttr: 指向被写入的属性; pValue: 指向被写入的属性值; Len: 被写入的属性值长度; Offset: 被写入的属性值的偏移; Method: 写属性时所用的方法, 可能值 (0x12 = Write Request), ( 0x52 = Write Command) 等。
输出	无
返回	0: 表示成功, 其他值表示失败
作用	写属性值回调函数

3.20 GATT\_ Notification

函数原型	bStatus_t GATT_Notification( uint16 connHandle, attHandleValueNoti_t *pNoti, uint8 authenticated )
输入	connHandle: 连接句柄, 表示不同的连接; pNoti: 通知信息; authenticated: whether an authenticated link is required
输出	无
返回	0: 表示成功, 其他值表示失败
作用	用于服务端主动发送数据包, 并且不需要应答

4、使用指南

4.1 库配置步骤

- 1、调用基本函数初始化协议栈库和系统时间
- 2、调用 GAP 函数初始化广播参数, 角色, 连接相关参数, 设备名称, 绑定相关参数
- 3、如果是连接通信, 初始化 GATT 函数, 注册 GATT 服务, 读写回调函数
- 4、如果是从机模式, 启动设备进入广播态, 等待连接。如果是主机模式, 则开始发现设备, 主动发起连接
- 5、进入连接态完成枚举过程, 通过回调通知应用层
- 6、通过读写属性回调来收发数据, 或者通过通知或指示来主动发送数据

4.2 系统管理

协议栈库提供了丰富的系统管理的编程接口, 主要包括任务管理和消息管理。

任务管理——多任务管理方式可以实现数据共享, 合理分配各任务来提高效率, 多任务运行实际上只有一个任务在运行, 但是可以使用任务调度的策略将多个任务进行调度, 每个任务占用一定的时间, 所有的任务通过时间分片的方式处理。

消息管理——消息是一个带有数据的事件, 用于协议栈各层之间传递数据, 支持同时添加多个消息。而任务只是去执行一个相应的操作 (调用事件处理函数去处理)。

协议栈系统使用方法

任务管理层 ID 号——taskID

事件——events

- (1) 通过 TMOS\_ProcessEventRegister 将自定义的任务添加到系统中, 添加成功则返回 taskID, 参数为任务处理函数的入口地址 (processEventCB)。每一层事件 (events) 包括 15 个任务和一个消息。当有对应层的任务触发或者有消息处理时, 系统调用 processEventCB 函数,

传入 taskID 和

```

/*****
 * @fn          tmos_set_event
 *
 * @brief       start a event immediately
 *
 * input parameters
 *
 * @param       taskID - task ID of event
 * @param       event - event value
 *
 * output parameters
 *
 * @param       None.
 *
 * @return      0 - success.
 */

```

```
extern bStatus_t tmos_set_event( tmosTaskID taskID, tmosEvents event );
```

```

/*****
 * @fn          tmos_clear_event
 *
 * @brief       clear a event already timeout, cannot be used in it own event function.
 *
 * input parameters
 *
 * @param       taskID - task ID of event
 * @param       event - event value
 *
 * output parameters
 *
 * @param       None.
 *
 * @return      0 - success.
 */

```

```
extern bStatus_t tmos_clear_event( tmosTaskID taskID, tmosEvents event );
```

```

/*****
 * @fn          tmos_start_task
 *
 * @brief       start a event after period of time
 *
 * input parameters

```

```
*
* @param      taskID - task ID of event
* @param      event - event value
* @param      time - period of time
*
* output parameters
*
* @param      None.
*
* @return     TRUE - success.
*/
extern bStatus_t tmos_start_task( tmosTaskID taskID, tmosEvents event, tmosTimer time );

/*****
* @fn          tmos_stop_task
*
* @brief       stop a event
*
* input parameters
*
* @param      taskID - task ID of event
* @param      event - event value
*
* output parameters
*
* @param      None.
*
* @return     0 - success.
*/
extern bStatus_t tmos_stop_task( tmosTaskID taskID, tmosEvents event );

/*****
* @fn          tmos_get_task_timer
*
* @brief       get last period of time for this event
*
* input parameters
*
* @param      taskID - task ID of event
* @param      event - event value
*
* output parameters
*
* @param      None.
*
* @return     the timer's tick count if found, zero otherwise.
*/
extern tmosTimer tmos_get_task_timer( tmosTaskID taskID, tmosEvents event );

/*****/>
```

---

```

* @fn          tmos_msg_send
*
* @brief       send msg to a task, callback events&SYS_EVENT_MSG
*
* input parameters
*
* @param       taskID - task ID of task need to send msg
* @param       *msg_ptr - point of msg
*
* output parameters
*
* @param       None.
*
* @return      0 - success.
*/
extern bStatus_t tmos_msg_send( tmosTaskID taskID, u8 *msg_ptr );

/*****
* @fn          tmos_msg_deallocate
*
* @brief       delete a msg
*
* input parameters
*
* @param       *msg_ptr - point of msg
*
* output parameters
*
* @param       None.
*
* @return      0 - success.
*/
extern bStatus_t tmos_msg_deallocate( u8 *msg_ptr );

/*****
* @fn          tmos_msg_receive
*
* @brief       receive a msg
*
* input parameters
*
* @param       taskID - task ID of task need to receive msg
*
* output parameters
*
* @param       None.
*
* @return      *u8 - message information or NULL if no message
*/
extern u8 *tmos_msg_receive( tmosTaskID taskID );

```

```

/*****
 * @fn          tmos_msg_allocate
 *
 * @brief       allocate buffer for msg when need to send msg
 *
 * input parameters
 *
 * @param       len  - length of msg
 *
 * output parameters
 *
 * @param       None.
 *
 * @return      pointer to allocated buffer or NULL if allocation failed.
 */
extern u8  *tmos_msg_allocate( u16 len );

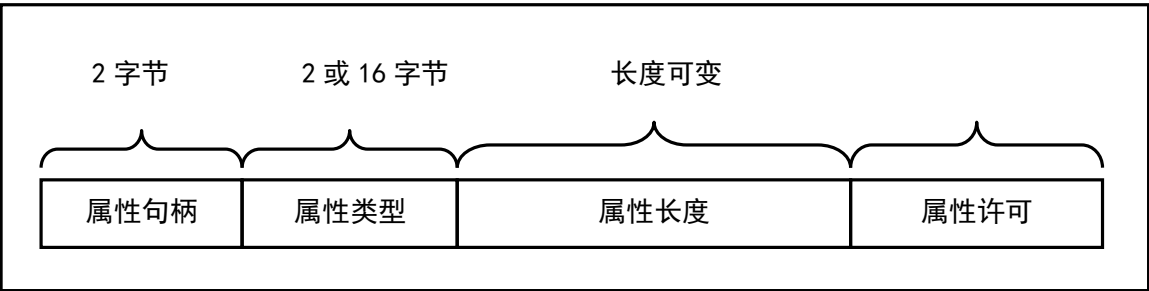
```

4.3 属性、特性和服务的关系

一个配置文件最少要包含一个服务，一个服务必须包含一个服务声明，这里可以是主服务或者次服务。可能包含一个或者多个引用声明，必须最少包含一个特性。

4.3.1 属性

- (1) 属性句柄：一个索引值，可以用来区别不同的属性，句柄值有效范围是 0x0001~0xFFFF.
- (2) 属性类型：2 字节或者 16 字节的 UUID。
- (3) 属性值：属性值和长度是根据属性类型来确定的。例如：属性类型是 0x2800，属性值就是一个服务的 UUID；如果属性类型是自定义的 0xFFFF0，属性值就是应用层自定义的数据。



4.3.2 特性

- (1) 特性定义：
  - 必须包含：一个特性声明和一个特性值声明。特性值声明必须紧跟在特性声明之后，也就是这 2 个属性句柄值是连续的。
  - 可能包含：特性扩展属性声明；特性用户描述声明；客户端特性配置声明；特性格式声明。
- (2) 特性声明：

由属性句柄(2 字节、属性类型(2 或 16 字节)、 属性值(5 或 19 字节)构成。这里属性类型被蓝牙联盟组织分配的 UUID 是 0X2803.

属性句柄	属性类型	属性值			属性许可
0XNNNN	0x2803 《特性》	特性性质	特性值句柄	特 性 值 UUID	可读

属性值：  
1 字节特性性质；  
2 字节特性值属性句柄，也就是后面的特性值声明的属性句柄；  
2 字节特性 UUID，也就是后面的特性值声明的属性类型。

属性值	大小	描述
特性性质	1 字节	特性各个位的定义
特性值句柄	2 字节	特性值属性的句柄值
特性值 UUID	2 或者 16 字节	特性值属性的类型

属性值中的 1 字节性质定义：

性质	值	描述
广播	0x01	如果此位设置，需要用到服务器特性配置描述符来使能
读取	0x02	如果此位设置，特性值允许被读取
写命令(无应答写)	0x04	如果此位设置，特性值允许写入, 没有应答
写入(有应答)	0x08	如果此位设置，特性值允许被写入，有应答
通知	0x10	如果此位设置，特性值允许主动上报给客户端，客户端没有应答，需要用到客户端特性配置描述符来使能
指示	0x20	如果此位设置，特性值允许主动上报给客户端，客户端有应答，需要用到客户端特性配置描述符来使能
加密写命令	0x40	如果此位设置，加密后写命令
扩展属性	0x80	如果此位设置，说明是个扩展属性

(3) 特性值声明：  
由属性句柄(2 字节、属性类型(2 或 16 字节)、 属性值(长度可变)构成。

属性句柄	属性类型	属性值	属性许可
0xNNNN	0xuuuu, 16 或 128 位特性 UUID	特性值	由上层定义

(4) 特性用户描述声明：

属性句柄	属性类型	属性值	属性许可
0xNNNN	0x2901 《特性用户描述》	UTF-8 字符串	由上层定义

(5) 客户端特性配置声明：

属性句柄	属性类型	属性值	属性许可
0xNNNN	0x2902 《客户端特性配置》	端特性配置位	由上层定义

配置	值	描述
通知	0x0001	服务器可以主动通知属性值给客户端
指示	0x0002	服务器将属性值指示给客户端，并得到客户端的确认
保留值	0xFFFF	

如果特性声明中的性质定义了通知，客户端特性配置声明就需要添加到特性中。

4.3.3 服务

(1) 服务声明：

主要服务声明属性类型是 0X2800；  
次要服务声明属性类型是 0X2801；  
属性值是 2 字节或 16 字节的 UUID。

属性句柄	属性类型	属性值	属性许可
0xNNNN	0x2800 《首要服务》，或 0x2801 《次要服务》	16 或 128 位服务 UUID	可读

(2) 引用声明：

引用声明必须在服务声明之后，特性声明之前；  
属性类型被蓝牙联盟组织分配的 UUID 是 0X2802；  
属性值是 2 字节或 16 字节的 UUID；

服务的引用声明不能形成环型，如：服务 A 中定义了一个引用声明引用到服务 B，服务 B 中也定义一个引用声明引用到服务 A。

属性句柄	属性类型	属性值			属性许可
0xNNNN	0x2802 《引用》	引用的服务 属性句柄	组结束句柄	服务的 UUID	可读