XR806 BLE Mesh

开发指南

版本号：1.0

发布时间：2021-04-03

版本历史

版本	日期	责任人	版本描述
1.0	2021-04-03	AWA1426	创建文档

目录

表格目录

1 前言

1.1 文档简介

本文档基于 XR806 的 SDK，介绍了 BLE Mesh 模块接口函数的功能、使用说明及使用示例，以指导读者使用 XR806 BLE Mesh 模块提供的接口进行应用开发。

1.2 目标读者

XR806 BLE Mesh 相关开发人员。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

2 概述

2.1 背景说明

XR806 通过 Zephyr 协议栈提供 BLE Mesh 功能的支持，提供 API 供用户进行调用。API 所提供的功能主要有 Provisioner 的 Provisioning 功能、Node 的 Friend、Relay、LPN 功能等。

2.2 规格特性

XR806 BLE Mesh 功能基于 Zephyr V2.5.0 提供的开源 Bluetooth Mesh 协议栈进行二次开发。

XR806 BLE Mesh 主要规格如表 2-1 所示，详细规格可参考 Zephyr 官网(https://zephyrproject.org/)介绍。

表 2-1 XR806 BLE Mesh 主要规格表

模块	场景	规格描述
Mesh	Node	支持通过 PB-ADV 和 PB-GATT 方式配网，支持 Proxy、Friend、Low Power Node、Relay 功能，支持 config cli/srv model，支持 health cli/srv model，支持 on/off cli/srv model
	Provisioner	支持 PB-ADV 方式配网，支持 config cli model，支持 health cli model，支持 on/off cli model

2.3 文件位置

XR806 BLE Mesh 接口 API 由 BLE Host 协议栈提供，其代码位置如表 2-2 所示。

表 2-2 XR806 BLE Mesh 接口代码位置

模块	文件类型	代码位置
Mesh	header	./include/ble/bluetooth/mesh
	source	./src/ble/mesh
	demo	./project/demo/bluetooth

说明

XR806 SDK 可在以下 Github 仓库获取：https://github.com/XradioTech/xr806_sdk.git

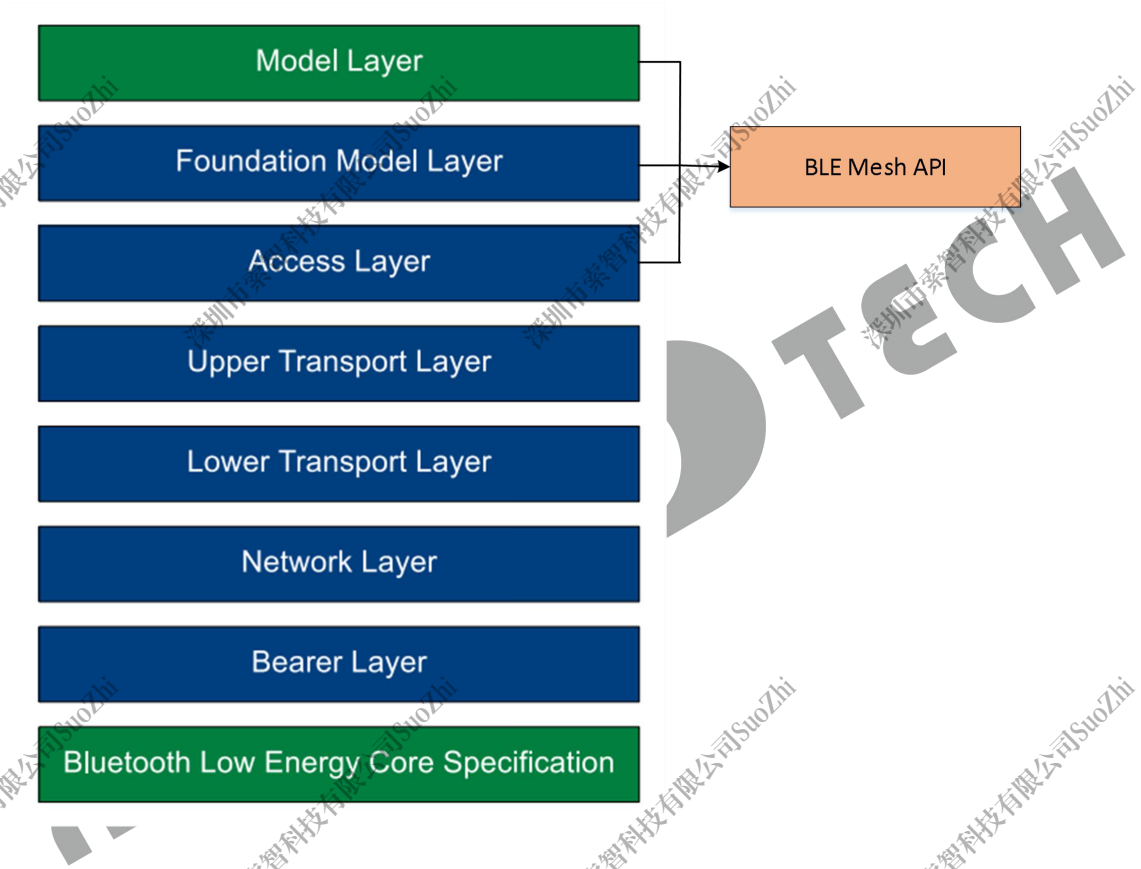
3 技术说明

3.1 模块框架

XR806 BLE Mesh 模块的系统层次如下图 3-1 所示，是基于 BLE 模块的一个模块。

XR806 BLE Mesh 接口位于 BLE Mesh 模块的上层，是 BLE Mesh 协议提供给用户开发的接口。用户通过调用 BLE Mesh API，即可使用 BLE Mesh 相关功能。

图 3-1 BLE Mesh 系统层次



4 应用说明

4.1 配置说明

启用 BLE Mesh 功能需要使能 BLE Mesh 的相关配置，通过工程配置即可启用，如表 4-1 所示。需要注意的是启用 Mesh 功能需要同时开启 BLE 的 Pripheral 和 Central 功能。

表 4-1 XR806 BLE 配置列表

配置项	配置说明
BLE 功能使能	设置说明： 此项配置用于在 SDK 中启用 BLE 功能。 设置方法： 1.编译之前执行命令“make menuconfig”，进入 BLE 功能项下，将以下功能项打开： **[*] BLE Controller** **[*] Ble Host** **[*] Peripheral Role support** **[*] Central Role support** 2.进入 BLE Host 功能项下，将以下功能项打开： **[*] Bluetooth Mesh support**

4.2 接口说明

XR806 BLE Mesh 提供了各种功能的接口，主要功能涵括了初始化、配网、CDB(Configuration Database)、模型配置等。但需要注意的是，应用可使用的接口还取决于 Kconfig 选项，因为大多数 BLE Mesh 功能都是在编译时已经确定好的。例如，任何与 Provisioner 相关的 API 都需要配置 CONFIG_BT_MESH_PROVISIONER 宏，其在 menuconfig 中对应的选项为 Provisioner support。

各接口按照功能可分为 12 类，其简要说明如表 4-2 所示。

表 4-2 BLE 接口功能分类说明

接口名	简要说明
基础接口	本接口共有 4 个接口函数，主要用于 BLE Mesh 初始化、重置等。
配网接口	本接口共有 12 个接口函数，主要用于 BLE Mesh 的配网。
Friend/LPBN 接口	本接口共有 3 个接口函数，主要用于 BLE Mesh 的 Friendship 功能。
CDB 接口	本接口共有 15 个接口函数，主要用于 Provisioner 对 Mesh 网络的管理。
模型通用接口	本接口共有 9 个接口函数，主要用于 Model 模型的基础操作。
配置模型接口	本接口共有 80 个接口函数，主要用于 Client 对 Server 端的配置，如 Beacon、TTL 等功能的配置。

接口名	简要说明
On/Off 模型接口	本接口共有 9 个接口函数，主要用于 On/Off 模型的操作。
Health 模型接口	本接口共有 11 个接口函数，主要用于元素的 Health 状态检查。
Heartbeat 接口	本接口共有 2 个接口函数，主要用于节点的 Heartbeat 状态配置。
Mesh Common 接口	本接口共有 12 个接口函数，主要用于节点的应用自由模型的基础开发。
Proxy 接口	本接口共有 1 个接口函数，主要用于 Proxy 功能的开启。
Transition 接口	本接口共有 14 个接口函数，主要用于渐变时间模型的开发。

4.2.1 基础接口

4.2.1.1 bt_mesh_init

表 4-3 bt_mesh_init 接口函数说明

信息项	说明
原型	int bt_mesh_init(const struct bt_mesh_prov *prov, const struct bt_mesh_comp *comp);
功能	BLE Mesh 的初始化，协议栈初始化及资源配置等

参数	struct bt_mesh_prov *prov
	含义解释：BLE Mesh 当前节点配置信息
	使用说明：结构体定义见表 4-4
	
	struct bt_mesh_comp *comp
	含义解释：BLE Mesh 当前节点组成信息
	使用说明：其使用如下：
	struct bt_mesh_model root_models[] = {
	BT_MESH_MODEL_CFG_SRV,
	BT_MESH_MODEL_CFG_CLI(&cfg_cli),
	BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
	BT_MESH_MODEL_HEALTH_CLI(&health_cli),
	BT_MESH_MODEL_GEN_ONOFF_SRV(&onoff_srv_user_data, &gen_onoff_pub_srv),
	BT_MESH_MODEL_GEN_ONOFF_CLI(&onoff_cli_user_data, &gen_onoff_pub_cli),
	};
	struct bt_mesh_elem elements[] = {
	BT_MESH_ELEM(0, root_models, BT_MESH_MODEL_NONE),
	};
	struct bt_mesh_comp comp = {
	.cid = BT_COMP_ID_LF,
	.elem = elements,
	.elem_count = ARRAY_SIZE(elements),
	};
	结构体定义见表 4-5

返回值	int 类型 0：初始化成功 非 0：初始化失败

表 4-4 bt_mesh_prov 结构体成员说明

成员项	说明
uint8_t *uuid	含义解释：指向未配网广播中使用的 UUID 的指针 使用说明：UUID 为 128bit
char *uri	含义解释：与 URI AD Type 关联的 URI Hash 值 使用说明：未配网广播包中包含有该 URI
bt_mesh_prov_oob_info_t oob_info	含义解释：OOB 配网信息 使用说明：枚举型变量，每 bit 代表一种形式，具体定义见 Mesh Spec，以下为常用取值： BT_MESH_PROV_OOB_URI：BIT(1)，URI Hash 域 BT_MESH_PROV_OOB_NUMBER：BIT(5)，数字形式 OOB BT_MESH_PROV_OOB_STRING：BIT(6)，字符串形式 OOB
uint8_t *static_val	含义解释：静态 OOB 值 使用说明：用于存储静态 OOB 值，可用于节点配网
uint8_t static_val_len	含义解释：静态 OOB 值长度 使用说明：与静态 OOB 值搭配使用
uint8_t output_size	含义解释：输出式 OOB 的最大信息长度 使用说明：用于节点配网，显示 OOB 值的最大数值长度
uint16_t output_actions	含义解释：输出式 OOB 的输出动作类型，用于节点配网 使用说明：枚举型变量，具体取值如下： BT_MESH_NO_OUTPUT：不支持输出 OOB 操作 BT_MESH_BLINK：若存在灯可使用，闪烁式输出 BT_MESH_BEEP：若存在蜂鸣器可使用，蜂鸣式输出 BT_MESH_VIBRATE：若存在马达之类可使用，震动式输出 BT_MESH_DISPLAY_NUMBER：屏幕显示数字方式 BT_MESH_DISPLAY_STRING：屏幕显示字符串方式
uint8_t input_size	含义解释：输入式 OOB 的最大信息长度 使用说明：用于节点配网，输入 OOB 值的最大数值长度
uint16_t input_actions	含义解释：输入式 OOB 的输出动作类型，用于节点配

成员项	说明
	网 使用说明：枚举型变量，具体取值如下： BT_MESH_NO_OUTPUT：不支持输入 OOB 操作 BT_MESH_PUSH：按键方式进行输入 BT_MESH_TWIST：旋钮方式进行输入 BT_MESH_DISPLAY_NUMBER：数字输入方式 BT_MESH_DISPLAY_STRING：字符串输入方式
void (*capabilities)(const struct bt_mesh_dev_capabilities *cap)	含义解释：节点所具备的能力回调函数 使用说明：用于通知应用程序已接收到未配网设备的配置功能。当此回调函数返回时，配网流程将使用所选方法开始身份验证 参数说明： struct bt_mesh_dev_capabilities *cap：设备所支持的能力
int (*output_number)(bt_mesh_output_action_t act, uint32_t num)	含义解释：请求输出数字回调函数 使用说明：此回调函数通知应用应使用给定的操作输出给定的数字 参数说明： bt_mesh_output_action_t act：枚举型变量 BT_MESH_NO_OUTPUT：不支持输出 OOB 操作 BT_MESH_BLINK：若存在灯可使用，闪烁式输出 BT_MESH_BEEP：若存在蜂鸣器可使用，蜂鸣式输出 BT_MESH_VIBRATE：若存在马达之类可使用，震动式输出 BT_MESH_DISPLAY_NUMBER：屏幕显示数字方式 BT_MESH_DISPLAY_STRING：屏幕显示字符串方式 uint32_t num：需要输出的数字
int (*output_string)(const char *str)	含义解释：请求输出字符串回调函数 使用说明：此回调函数通知应用显示给定的字符串给用户 参数说明： const char *str：需要显示的字符串
int (*input)(bt_mesh_input_action_t act, uint8_t size)	含义解释：输入回调函数

成员项	说明
	使用说明：此回调函数通知应用应使用给定的操作请求用户输入，请求的输入为字符串或数字，因此一旦从用户获取到了数据，应用需要调用 bt_mesh_input_string()或 bt_mesh_input_number()函数 参数说明： 枚举型变量，具体取值如下： BT_MESH_NO_OUTPUT：不支持输入 OOB 操作 BT_MESH_PUSH：push 方式进行输入 BT_MESH_TWIST：扭方式进行输入 BT_MESH_DISPLAY_NUMBER：数字输入方式 BT_MESH_DISPLAY_STRING：字符串输入方式 uint8_t size：输入数据的最大长度
void (*input_complete)(void)	含义解释：对端设备完成输入回调函数 使用说明：此回调函数通知应用对端设备已完成了 OOB 输入，停止显示其输出的 OOB 值
void (*unprovisioned_beacon)(uint8_t uuid[16], bt_mesh_prov_oob_info_t oob_info, uint32_t *uri_hash)	含义解释：收到未配网 beacon 回调函数 使用说明：此回调函数通知应用收到了未配网的 beacon 参数说明： uint8_t uuid[16]：收到的 beacon 中的 UUID bt_mesh_prov_oob_info_t oob_info：收到的 beacon 中的 oob 信息，枚举型变量，每 bit 代表一种形式，具体定义见 Mesh Spec，以下为常用取值： BT_MESH_PROV_OOB_URI: BIT(1)，URI Hash 域 BT_MESH_PROV_OOB_NUMBER:BIT(5)，数字形式 OOB BT_MESH_PROV_OOB_STRING:BIT(6)，字符串形式 OOB uint32_t *uri_hash：指向 URI 哈希值的指针，如果 beacon 中不存在哈希，则为 NULL
void (*link_open)(bt_mesh_prov_bearer_t bearer)	含义解释：配网链路已打开回调函数 使用说明：此回调通知应用已在给定的配网承载层上打开配网链路 参数说明： bt_mesh_prov_bearer_t bearer：枚举变量 BT_MESH_PROV_ADV：ADV 承载层

成员项	说明
	BT_MESH_PROV_GATT：GATT 承载层
void (*link_close)(bt_mesh_prov_bearer_t bearer)	含义解释：配网链路已关闭回调函数 使用说明：此回调通知应用已在给定的配网承载层上关闭配网链路 参数说明： bt_mesh_prov_bearer_t bearer：枚举变量 BT_MESH_PROV_ADV：ADV 承载层 BT_MESH_PROV_GATT：GATT 承载层
void (*complete)(uint16_t net_idx, uint16_t addr)	含义解释：配网完成回调函数 使用说明：此回调函数通知应用配网已经完成，并且已为本地节点分配了指定的 NetKey Index 和主元素地址 参数说明： uint16_t net_idx：配网过程中给定的 NetKey Index uint16_t addr：分配的主元素地址
void (*node_added)(uint16_t net_idx, uint8_t uuid[16], uint16_t addr, uint8_t num_elem);	含义解释：新节点已添加到配网数据库回调函数 使用说明：此回调函数通知应用已完成配网，并且已为节点分配了指定的 NetKey Index 和主元素地址，一般用于 Provisioner 参数说明： uint16_t net_idx：配网过程中给定的 NetKey Index uint8_t uuid[16]：所添加节点的 UUID uint16_t addr：所添加节点的主元素地址 uint8_t num_elem：所添加节点中的元素个数
void (*reset)(void)	含义解释：节点重置回调函数 使用说明：此回调函数通知应用本地节点已被重置，需要重新配网。该节点不会自动在进行未配网广播包的发送，因此需要调用 bt_mesh_prov_enable()函数才能再次进行未配网广播包的发送

表 4-5 bt_mesh_comp 结构体成员说明

成员项	说明
uint16_t cid	含义解释：公司 ID

成员项	说明
	使用说明：包含由蓝牙 SIG 分配的 16 位公司标识符
uint16_t pid	含义解释：产品 ID 使用说明：包含一个 16 位厂商分配的产品标识符
uint16_t vid	含义解释：版本 ID 使用说明：包含一个 16 位厂商分配的产品版本标识符
size_t elem_count	含义解释：元素数量 使用说明：此设备中的元素数
struct bt_mesh_elem *elem	含义解释：元素列表 使用说明：此设备中的元素列表

4.2.1.2 bt_mesh_reset

表 4-6 bt_mesh_reset 接口函数说明

信息项	说明
原型	void bt_mesh_reset(void);
功能	重置本地 Mesh 节点的状态
参数	无
返回值	无

4.2.1.3 bt_mesh_suspend

表 4-7 bt_mesh_suspend 接口函数说明

信息项	说明
原型	int bt_mesh_suspend(void);
功能	暂时挂起 Mesh 网络，以此来节约功耗，但是需要注意的是，长时间保持本地节点挂起状态可能会导致其永久与 Mesh 网络断开连接。所以最好使用 Friend 功能，使本地节点成为低功耗节点
参数	无
返回值	int 类型 0：挂起成功 非 0：挂起失败

4.2.1.4 bt_mesh_resume

表 4-8 bt_mesh_resume 接口函数说明

信息项	说明
原型	int bt_mesh_resume(void);
功能	恢复挂起的 Mesh 网络，通常与 bt_mesh_suspend()结合使用
参数	无
返回值	int 类型 0：恢复成功 非 0：恢复失败

4.2.2 配网接口

4.2.2.1 bt_mesh_input_string

表 4-9 bt_mesh_input_string 接口函数说明

信息项	说明
原型	int bt_mesh_input_string(const char *str);
功能	配网 OOB 字符串输入，配网 action 配置为 BT_MESH_ENTER_STRING，在调用 bt_mesh_prov 输入回调(即表 4-4 中 input 回调函数)之后进行调用
参数	const char *str 含义解释：输入的 OOB 字符串 使用说明：需要的字符串存储在该变量中，用于配网使用
返回值	int 类型 0：输入成功 非 0：输入失败

4.2.2.2 bt_mesh_input_number

表 4-10 bt_mesh_input_number 接口函数说明

信息项	说明
原型	int bt_mesh_input_number(uint32_t num);
功能	配网 OOB 数字输入，配网 action 配置为 BT_MESH_ENTER_NUMBER，在调用 bt_mesh_prov 输入回调(即表 4-4 中 input 回调函数)之后进行调用
参数	uint32_t num 含义解释：输入的 OOB 数字 使用说明：需要的数字存储在该变量中，用于配网使用
返回值	int 类型 0：输入成功 非 0：输入失败

4.2.2.3 bt_mesh_prov_remote_pub_key_set

表 4-11 bt_mesh_prov_remote_pub_key_set 接口函数说明

信息项	说明
原型	int bt_mesh_prov_remote_pub_key_set(const uint8_t public_key[64]);
功能	设置配网设备公钥
参数	const uint8_t public_key[64] 含义解释：设备公钥 使用说明：将输入的设备公钥用于配网产生密钥
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.2.4 bt_mesh_auth_method_set_input

表 4-12 bt_mesh_auth_method_set_input 接口函数说明

信息项	说明
原型	int bt_mesh_auth_method_set_input(bt_mesh_input_action_t action, uint8_t size);
功能	设置使用输入 OOB 身份验证方式，仅 Provisioner 使用。指示未配网设备使用指定的输入 OOB 验证方式进行配网
参数	bt_mesh_input_action_t action 含义解释：未配网设备选用的配网认证方式 使用说明：枚举型变量，具体取值如下： BT_MESH_NO_OUTPUT：不支持输入 OOB 操作 BT_MESH_PUSH：push 方式进行输入 BT_MESH_TWIST：扭方式进行输入 BT_MESH_DISPLAY_NUMBER：数字输入方式 BT_MESH_DISPLAY_STRING：字符串输入方式 uint8_t size 含义解释：输入数据的最大长度 使用说明：一般为 6 位数字的长度；
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.2.5 bt_mesh_auth_method_set_output

表 4-13 bt_mesh_auth_method_set_output 接口函数说明

信息项	说明
原型	int bt_mesh_auth_method_set_output(bt_mesh_output_action_t action, uint8_t size);
功能	设置使用输出 OOB 身份验证方式，仅 Provisioner 使用。指示未配网设备使用指定的输出 OOB 验证方式进行配网
参数	bt_mesh_input_action_t action 含义解释：未配网设备选用的配网认证方式 使用说明：枚举型变量，具体取值如下： BT_MESH_NO_OUTPUT：不支持输出 OOB 操作 BT_MESH_BLINK：若存在灯可使用，闪烁式输出 BT_MESH_BEEP：若存在蜂鸣器可使用，蜂鸣式输出 BT_MESH_VIBRATE：若存在马达之类可使用，震动式输出 BT_MESH_DISPLAY_NUMBER：屏幕显示数字方式 BT_MESH_DISPLAY_STRING：屏幕显示字符串方式 uint8_t size 含义解释：输出数据的最大长度 使用说明：一般为 6 位数字的长度；
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.2.6 bt_mesh_auth_method_set_static

表 4-14 bt_mesh_auth_method_set_static 接口函数说明

信息项	说明
原型	int bt_mesh_auth_method_set_static(const uint8_t *static_val, uint8_t size);
功能	设置使用静态 OOB 配网方式，仅 Provisioner 使用，标识未配网设备使用静态 OOB 配网方式，并在配网时使用指定的静态 OOB 验证值
参数	const uint8_t *static_val 含义解释：静态 OOB 配网值 使用说明：由应用提供的静态 OOB 配网值，通常为 6 位 uint8_t size 含义解释：静态 OOB 值长度 使用说明：一般静态 OOB 值设为 6 位
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.2.7 bt_mesh_auth_method_set_none

表 4-15 bt_mesh_auth_method_set_none 接口函数说明

信息项	说明
原型	int bt_mesh_auth_method_set_none(void);
功能	不使用 OOB 身份验证进行配网，仅 Provisioner 使用，即在配网新设备时不使用任何身份验证，为默认行为
参数	无
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.2.8 bt_mesh_prov_enable

表 4-16 bt_mesh_prov_enable 接口函数说明

信息项	说明
原型	int bt_mesh_prov_enable(bt_mesh_prov_bearer_t bearers);
功能	启用特定的配网承载层
参数	bt_mesh_prov_bearer_t bearers 含义解释：配网承载层 使用说明：枚举变量，可同时置起 bit 位，标识支持两种方式承载层 BT_MESH_PROV_ADV：BIT(0)，使用 ADV 承载层 BT_MESH_PROV_GATT：BIT(1)，使用 GATT 承载层
返回值	int 类型 0：启用成功 非 0：启用失败

4.2.2.9 bt_mesh_prov_disable

表 4-17 bt_mesh_prov_disable 接口函数说明

信息项	说明
原型	int bt_mesh_prov_disable(bt_mesh_prov_bearer_t bearers);
功能	关闭特定的配网承载层
参数	bt_mesh_prov_bearer_t bearers 含义解释：配网承载层 使用说明：枚举变量，可同时置起 bit 位，标识支持两种方式承载层 BT_MESH_PROV_ADV：BIT(0)，使用 ADV 承载层 BT_MESH_PROV_GATT：BIT(1)，使用 GATT 承载层

信息项	说明
返回值	int 类型 0：关闭成功 非 0：关闭失败

4.2.2.10 bt_mesh_provision

表 4-18 bt_mesh_provision 接口函数说明

信息项	说明
原型	int bt_mesh_provision(const uint8_t net_key[16], uint16_t net_idx, uint8_t flags, uint32_t iv_index, uint16_t addr, const uint8_t dev_key[16]);
功能	将本地设备节点配入 Mesh 网中，该接口一般只有 Provisioner 使用，将自身节点配入网中，其他节点应该均由 Provisioner 配入网中
参数	const uint8_t net_key[16] 含义解释：网络密钥 使用说明：Mesh 网络需要使用的网络密钥，为 128bit 数据 uint16_t net_idx 含义解释：网络 index 使用说明：该网络密钥所组建的 Mesh 网络的 index uint8_t flags 含义解释：配网标志 使用说明：配网中所使用的一些功能的标识，如 key refresh 和 IV Update 功能 uint32_t iv_index 含义解释：iv 序列号 使用说明：表示本地节点在 Mesh 网络中的包从哪个序列号开始发 uint16_t addr 含义解释：配入网后本地设备节点的地址 使用说明：为主元素的地址，其他元素地址依次往后增加，不能配置为 0 const uint8_t dev_key[16] 含义解释：设备密钥 使用说明：用于对包中的数据进行加密
返回值	int 类型 0：配网成功 非 0：配网失败

4.2.2.11 bt_mesh_provision_adv

表 4-19 bt_mesh_provision_adv 接口函数说明

信息项	说明
原型	int bt_mesh_provision_adv(const uint8_t uuid[16], uint16_t net_idx, uint16_t addr, uint8_t attention_duration);
功能	使用 PB-ADV 方式将节点配入 Mesh 网中，该接口一般只有 Provisioner 使用，将其他未配网设备配入 Mesh 网络中
参数	const uint8_t uuid[16] 含义解释：未配网设备的 UUID 使用说明：Provisioner 通过 UUID 来区别各个未配网设备 uint16_t net_idx 含义解释：网络 index 使用说明:将目标节点配置到 Mesh 网络中，并分配到指定的 net_idx 的子网中， uint16_t addr 含义解释：节点地址 使用说明：将节点配入 Mesh 网络中时分配给对端设备的地址，如果 addr 设置为 0，则将自动选择最低的可用地址 uint8_t attention_duration 含义解释：持续关注时间 使用说明：告知对端设备将持续进行扫描接收的时间
返回值	int 类型 0：配网成功 非 0：配网失败

4.2.2.12 bt_mesh_is_provisioned

表 4-20 bt_mesh_is_provisioned 接口函数说明

信息项	说明
原型	bool bt_mesh_is_provisioned(void);
功能	检查本地设备节点是否已经配网，如可确定是否从 flash 中恢复了存储的网络
参数	无
返回值	bool 类型 True：已配网 False：未配网

4.2.3 Friend/LPN 接口

4.2.3.1 bt_mesh_lpn_set

表 4-21 bt_mesh_lpn_set 接口函数说明

信息项	说明
原型	int bt_mesh_lpn_set(bool enable);
功能	启用/禁用本地设备的低功耗功能，在一些场景下需要启用低功耗动能，如电池供电情况下等
参数	bool enable 含义解释：开启/关闭标志 使用说明：True 时开启 LPN 功能，False 时关闭 LPN 功能
返回值	int 类型 0：开启成功 非 0：开启失败

4.2.3.2 bt_mesh_lpn_poll

表 4-22 bt_mesh_lpn_poll 接口函数说明

信息项	说明
原型	int bt_mesh_lpn_poll(void);
功能	向朋友节点发送 Poll 包，以从朋友节点获取缓存的消息。如果没有建立 friendship，则该函数返回错误
参数	无
返回值	int 类型 0：发送成功 非 0：发送失败

4.2.3.3 bt_mesh_friend_terminate

表 4-23 bt_mesh_friend_terminate 接口函数说明

信息项	说明
原型	int bt_mesh_friend_terminate(uint16_t lpn_addr);
功能	与给定的 LPN 终止 friendship 关系，Friend 节点调用
参数	uint16_t lpn_addr 含义解释：LPN 地址 使用说明：Friend 存储在本地设备中的已建立 Friendship 的 LPN 地址
返回值	int 类型 0：终止成功 非 0：终止失败

4.2.4 CDB 接口

CDB(Configuration Database)，即配网数据库，用于存储 Mesh 网络的节点、子网、App Key 等信息。在一个 Mesh 网络中一般只有一个节点会存在配网数据库，默认该节点为 Provisioner。在使用上，先调用 4.2.4.1 接口进行 CDB 的创建，然后 Provisioner 再将自身配入网中。

4.2.4.1 bt_mesh_cdb_create

表 4-24 bt_mesh_cdb_create 接口函数说明

信息项	说明
原型	int bt_mesh_cdb_create(const uint8_t key[16]);
功能	创建并初始化 Mesh 网络配网数据库，将添加一个主子网，即 NetIdx 为 0 的一个子网，并将提供的密钥用作该子网的 Netkey，一般只有 Provisioner 拥有配网数据库
参数	const uint8_t key[16] 含义解释：用于主子网的 Netkey 使用说明：一般为随机产生的随机数，128bit
返回值	int 类型 0：创建成功 非 0：创建失败

4.2.4.2 bt_mesh_cdb_clear

表 4-25 bt_mesh_cdb_clear 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_clear(void);
功能	清除配网数据库，删除存储在数据库中的所有节点，子网和 App Key，并将数据库标记为无效，如果启用了存储功能，则将从 flash 中清除数据
参数	无
返回值	无

4.2.4.3 bt_mesh_cdb_node_alloc

表 4-26 bt_mesh_cdb_node_alloc 接口函数说明

信息项	说明
原型	struct bt_mesh_cdb_node *bt_mesh_cdb_node_alloc(const uint8_t uuid[16], uint16_t addr, uint8_t num_elem, uint16_t net_idx);
功能	在 CDB 中分配一个新的节点
参数	const uint8_t uuid[16] 含义解释：节点的 UUID 使用说明：所要添加节点的 UUID

信息项	说明
	uint16_t addr 含义解释：节点主元素的地址， 使用说明：如果设置为 0，则将最低的可用地址分配给该节点 uint8_t num_elem 含义解释：节点包含元素个数 使用说明：节点各个元素也会依次分配一个地址 uint16_t net_idx 含义解释：网络 Index 使用说明：节点被配网到 Mesh 中的子网的 index
返回值	bt_mesh_cdb_node 指针类型 NULL：分配失败 非 NULL：分配成功

4.2.4.4 bt_mesh_cdb_node_del

表 4-27 bt_mesh_cdb_node_del 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_node_del(struct bt_mesh_cdb_node *node, bool store);
功能	在 CDB 中删除一个指定的节点
参数	struct bt_mesh_cdb_node *node 含义解释：将要被删除的节点 使用说明：可通过函数 bt_mesh_cdb_node_get()从 addr 查找到对应的 node bool store 含义解释：是否从存储设备中删除信息标志 使用说明：若为 True，则会从 flash 中删除对应的存储信息，否则不会删除
返回值	无

4.2.4.5 bt_mesh_cdb_node_get

表 4-28 bt_mesh_cdb_node_get 接口函数说明

信息项	说明
原型	struct bt_mesh_cdb_node *bt_mesh_cdb_node_get(uint16_t addr);
功能	通过指定地址查找获取对应的节点
参数	uint16_t addr 含义解释：节点地址 使用说明：需要查找的元素的地址

信息项	说明
返回值	bt_mesh_cdb_node 指针类型 NULL：查找失败 非 NULL：查找成功

4.2.4.6 bt_mesh_cdb_node_store

表 4-29 bt_mesh_cdb_node_store 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_node_store(const struct bt_mesh_cdb_node *node);
功能	将指定节点存储到 flash 中，用于应用调用 bt_mesh_cdb_node_alloc()接口创建节点后，调用此接口将创建的信息进行存储
参数	const struct bt_mesh_cdb_node *node 含义解释：需要存储的节点 使用说明：需要存储的节点，在开启存储功能情况下才能使用
返回值	无

4.2.4.7 bt_mesh_cdb_subnet_alloc

表 4-30 bt_mesh_cdb_subnet_alloc 接口函数说明

信息项	说明
原型	struct bt_mesh_cdb_subnet *bt_mesh_cdb_subnet_alloc(uint16_t net_idx);
功能	在 CDB 中分配一个新的子网
参数	uint16_t net_idx 含义解释：子网的 net index 使用说明：需要申请的子网的 net index
返回值	struct bt_mesh_cdb_subnet 指针类型 NULL：分配失败 非 NULL：分配成功

4.2.4.8 bt_mesh_cdb_subnet_del

表 4-31 bt_mesh_cdb_subnet_del 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_subnet_del(struct bt_mesh_cdb_subnet *sub, bool store);
功能	从 CDB 中删除一个子网
参数	struct bt_mesh_cdb_subnet *sub 含义解释：将要被删除的子网 使用说明：可通过函数 bt_mesh_cdb_subnet_get()从 netidx 查找到对应的 subnet

信息项	说明
	bool store 含义解释：是否从存储设备中删除信息标志 使用说明：若为 True，则会从 flash 中删除对应的存储信息，否则不会删除
返回值	无

4.2.4.9 bt_mesh_cdb_subnet_get

表 4-32 bt_mesh_cdb_subnet_get 接口函数说明

信息项	说明
原型	struct bt_mesh_cdb_subnet *bt_mesh_cdb_subnet_get(uint16_t net_idx);
功能	通过指定 Net Index 查找获取对应的子网
参数	uint16_t net_idx 含义解释：节点的网络的 net index 使用说明：需要查找的节点的 net index
返回值	bt_mesh_cdb_subnet 指针类型 NULL：查找失败 非 NULL：查找成功

4.2.4.10 bt_mesh_cdb_subnet_store

表 4-33 bt_mesh_cdb_subnet_store 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_subnet_store(const struct bt_mesh_cdb_subnet *sub);
功能	将指定子网存储到 flash 中
参数	const struct bt_mesh_cdb_subnet *sub 含义解释：需要存储的子网 使用说明：需要存储的子网，在开启存储功能情况下才能使用
返回值	无

4.2.4.11 bt_mesh_cdb_subnet_flags

表 4-34 bt_mesh_cdb_subnet_flags 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_cdb_subnet_flags(const struct bt_mesh_cdb_subnet *sub);
功能	获取子网的 flag
参数	const struct bt_mesh_cdb_subnet *sub 含义解释：需要查询的子网指针

信息项	说明
	使用说明：获取子网的 flag，有 key refresh、iv update 等功能
返回值	无

4.2.4.12 bt_mesh_cdb_app_key_alloc

表 4-35 bt_mesh_cdb_app_key_alloc 接口函数说明

信息项	说明
原型	struct bt_mesh_cdb_app_key *bt_mesh_cdb_app_key_alloc(uint16_t net_idx, uint16_t app_idx);
功能	在 CDB 中分配一个新的 app key
参数	uint16_t net_idx 含义解释：子网的 net index 使用说明：app key 绑定到的 NetKey 的 NetIdx uint16_t app_idx 含义解释：子网的 app index 使用说明：需要申请的子网的 app index
返回值	bt_mesh_cdb_app_key 指针类型 NULL：分配失败 非 NULL：分配成功

4.2.4.13 bt_mesh_cdb_app_key_del

表 4-36 bt_mesh_cdb_app_key_del 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_app_key_del(struct bt_mesh_cdb_app_key *key, bool store);
功能	从 CDB 中删除一个 app key
参数	struct bt_mesh_cdb_app_key *key 含义解释：将要被删除的 app key 使用说明：可通过函数 bt_mesh_cdb_app_key_get()从 appidx 查找到对应的 appkey bool store 含义解释：是否从存储设备中删除信息标志 使用说明：若为 True，则会从 flash 中删除对应的存储信息，否则不会删除
返回值	无

4.2.4.14 bt_mesh_cdb_app_key_get

表 4-37 bt_mesh_cdb_app_key_get 接口函数说明

信息项	说明
原型	struct bt_mesh_cdb_app_key *bt_mesh_cdb_app_key_get(uint16_t app_idx);
功能	通过指定 App Index 查找获取对应的 app key
参数	uint16_t app_idx 含义解释：节点的网络的 app index 使用说明：需要查找的节点的 app index
返回值	bt_mesh_cdb_app_key 指针类型 NULL：查找失败 非 NULL：查找成功

4.2.4.15 bt_mesh_cdb_app_key_store

表 4-38 bt_mesh_cdb_app_key_store 接口函数说明

信息项	说明
原型	void bt_mesh_cdb_app_key_store(const struct bt_mesh_cdb_app_key *key);
功能	将指定 app key 存储到 flash 中
参数	const struct bt_mesh_cdb_app_key *key 含义解释：需要存储的 app key 使用说明：需要存储的 app ley，在开启存储功能情况下才能使用
返回值	无

4.2.5 模型通用接口

4.2.5.1 bt_mesh_model_msg_init

表 4-39 bt_mesh_model_msg_init 接口函数说明

信息项	说明
原型	void bt_mesh_model_msg_init(struct net_buf_simple *msg, uint32_t opcode);
功能	初始化模型消息，清除消息缓存区的内容，并对给定的操作码进行编码，消息缓存区将准备好填充有效数据
参数	struct net_buf_simple *msg 含义解释：消息缓存区 使用说明：放置消息的实际数据区域 uint32_t opcode 含义解释：进行编码的操作码 使用说明：部分为协议规定的固定的操作码，可自由定义的为厂商操作码

信息项	说明
返回值	无

4.2.5.2 bt_mesh_model_send

表 4-40 bt_mesh_model_send 接口函数说明

信息项	说明
原型	int bt_mesh_model_send(struct bt_mesh_model *model, struct bt_mesh_msg_ctx *ctx, struct net_buf_simple *msg, const struct bt_mesh_send_cb *cb, void *cb_data);
功能	发送一个访问层消息
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：消息所属的 Mesh(客户端)模型，相关结构体定义见表 4-41 struct bt_mesh_msg_ctx *ctx 含义解释：Mesh 网络消息 使用说明：包含密钥、TTL 等，相关结构体定义见表 4-42 struct net_buf_simple *msg 含义解释：访问层有效负载 使用说明：要发送的实际消息 const struct bt_mesh_send_cb *cb 含义解释：已发送消息回调函数 使用说明：可置为 NULL，不进行回调，相关结构体定义见表 4-43 void *cb_data 含义解释：要传递给回调的用户数据 使用说明：当已发送消息回调函数为 NULL 时，也置为 NULL
返回值	int 类型 0：发送成功 非 0：发送失败

表 4-41 bt_mesh_model 结构体成员说明

成员项	说明
const uint16_t id	含义解释：SIG 模型 ID 使用说明：Mesh 标准对一些常用的模型有固定的 ID
uint8_t elem_idx	含义解释：元素 Index 使用说明：表征该模型属于设备中哪个元素
uint8_t mod_idx	含义解释：模型 Index

成员项	说明
	使用说明：表征模型属于元素中第几个模型
uint16_t flags	含义解释：模型标记 使用说明：用于内部标记的模型标记
struct bt_mesh_model_pub * const pub	含义解释：模型发布 使用说明：用于向固定地址发布消息
uint16_t keys[]	含义解释：App Key 列表 使用说明：模型可有多个 App Key，存储在此，可进行大小配置
uint16_t groups[]	含义解释：订阅列表 使用说明：可能存在多个节点订阅该模型，可配置大小
const struct bt_mesh_model_op * const op	含义解释：操作码处理列表 使用说明：对各个接收到的操作码对应消息的处理
const struct bt_mesh_model_cb * const cb	含义解释：模型回调结构 使用说明：模型提供给处理中调用的回调函数
struct bt_mesh_model *next	含义解释：模型指针 使用说明：指向模型扩展列表中下一个模型的指针
struct bt_mesh_model *extends	含义解释：模型指针 使用说明：指向该模型扩展列表的第一个模型的指针
void *user_data	含义解释：用户数据 使用说明：特定提供给模型的用户数据

表 4-42 bt_mesh_msg_ctx 结构体成员说明

成员项	说明
uint16_t net_idx	含义解释：NetKey Index 使用说明：在指定子网中发送消息的 Netkey 索引
uint16_t app_idx	含义解释：AppKey Index 使用说明：App key 索引，用于加密消息

成员项	说明
uint16_t addr	含义解释：对端设备地址 使用说明：配网时分配给对端设备的地址
uint16_t recv_dst	含义解释：收到的消息的发送地址 使用说明：不用于发送
int8_t recv_rssi	含义解释：RSSI 值 使用说明：所接收数据包的 RSSI，不用于发送
uint8_t recv_ttl	含义解释：TTL 值 使用说明：所接收数据包的 TTL 值，不用于发送
bool send_rel	含义解释：是否以可靠形式发送 使用说明：通过使用包确认的方式强制使用分段包发送使发送可靠
uint8_t send_ttl	含义解释：发送 TTL 使用说明：一般使用默认 TTL 值，通过 BT_MESH_TTL_DEFAULT 宏进行配置

表 4-43 bt_mesh_send_cb 结构体成员说明

成员项	说明
void (*start)(uint16_t duration, int err, void *cb_data)	含义解释：包发送开始时回调处理函数 使用说明：在真正开始发送包时的回调处理函数 参数说明： uint16_t duration：完整传输的持续时间 int err：发送时启动广播时的状态码 void *cb_data：回调数据，传回给发送接口
void (*end)(int err, void *cb_data)	含义解释：包发送结束时回调处理函数 使用说明：在真正发包结束时的回调处理函数 参数说明： int err：发送时广播停止时的状态码 void *cb_data：回调数据，传回给发送接口

4.2.5.3 bt_mesh_model_publish

表 4-44 bt_mesh_model_publish 接口函数说明

信息项	说明
原型	int bt_mesh_model_publish(struct bt_mesh_model *model);
功能	发送模型发布消息，在调用此函数之前，应用需确保模型发布消息包含要发送的有效信息。需注意，此接口仅用于非定期发布。对于定期发布，需应用自行实现 bt_mesh_model_pub.update 回调函数，以此触发周期性发布功能，且需在 bt_mesh_model_pub.update 回调函数中对发送数据进行更新
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：消息所属的 Mesh(客户端)模型，相关结构体定义见表 4-41
返回值	int 类型 0：发送成功 非 0：发送失败

4.2.5.4 bt_mesh_model_elem

表 4-45 bt_mesh_model_elem 接口函数说明

信息项	说明
原型	struct bt_mesh_elem *bt_mesh_model_elem(struct bt_mesh_model *mod);
功能	获取模型所属的元素
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：要查找的 Mesh 模型，相关结构体定义见表 4-41
返回值	bt_mesh_elem 指针类型 NULL：查找失败 非 NULL：查找成功，查找到对应元素，相关结构体定义见表 4-46

表 4-46 bt_mesh_elem 结构体成员说明

成员项	说明
uint16_t addr	含义解释：单播地址 使用说明：在配网期间分配到的单播地址
const uint16_t loc	含义解释：位置描述符 使用说明：GATT 蓝牙命名空间描述符，可直接设置为 0
const uint8_t model_count	含义解释：模型数量 使用说明：此元素中 SIG 模型的数量

成员项	说明
const uint8_t vnd_model_count	含义解释：厂商模型数量 使用说明：此元素中厂商模型数量
struct bt_mesh_model * const models	含义解释：模型列表 使用说明：为链表，表征此模型中的 SIG 模型列表
struct bt_mesh_model * const vnd_models	含义解释：厂商模型列表 使用说明：为链表，此元素中的供应商模型列表

4.2.5.5 bt_mesh_model_find

表 4-47 bt_mesh_model_find 接口函数说明

信息项	说明
原型	struct bt_mesh_model *bt_mesh_model_find(const struct bt_mesh_elem *elem, uint16_t id);
功能	在指定元素中查找指定的 SIG 模型
参数	const struct bt_mesh_elem *elem 含义解释：需要搜索的元素 使用说明：在该元素中搜索指定的模型，相关结构体说明见表 4-46 uint16_t id 含义解释：模型 ID 使用说明：需要查找的模型的 ID
返回值	bt_mesh_model 指针类型 NULL：查找失败 非 NULL：查找成功，查找到对应模型，相关模型定义见表 4-41

4.2.5.6 bt_mesh_model_find_vnd

表 4-48 bt_mesh_model_find_vnd 接口函数说明

信息项	说明
原型	struct bt_mesh_model *bt_mesh_model_find_vnd(const struct bt_mesh_elem *elem, uint16_t company, uint16_t id);
功能	在指定元素中查找指定的厂商模型
参数	const struct bt_mesh_elem *elem 含义解释：需要搜索的元素 使用说明：在该元素中搜索指定的模型，相关结构体说明见表 4-46 uint16_t company 含义解释：厂商 ID

信息项	说明
	使用说明：需要查找模型的厂商 ID uint16_t id 含义解释：模型 ID 使用说明：需要查找的模型的 ID
返回值	bt_mesh_model 指针类型 NULL：查找失败 非 NULL：查找成功，查找到对应模型，相关模型定义见表 4-41

4.2.5.7 bt_mesh_model_in_primary

表 4-49 bt_mesh_model_in_primary 接口函数说明

信息项	说明
原型	static inline bool bt_mesh_model_in_primary(const struct bt_mesh_model *mod);
功能	判断模型是否在主元素中
参数	const struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：要查找的 Mesh 模型，相关结构体定义见表 4-41
返回值	bool 类型 True：该模型在主元素中 False：该模型不在主元素中

4.2.5.8 bt_mesh_model_data_store

表 4-50 bt_mesh_model_data_store 接口函数说明

信息项	说明
原型	int bt_mesh_model_data_store(struct bt_mesh_model *mod, bool vnd, const char *name, const void *data, size_t data_len);
功能	立即将模型的用户数据存储到 flash 中，但若需要重启后将数据恢复，需在对应模型中实现 bt_mesh_model_cb 结构体中 settings_set 回调函数
参数	const struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：要存储的 Mesh 模型，相关结构体定义见表 4-41 bool vnd 含义解释：是否是厂商模型表示 使用说明：True 表示存储的模型为厂商模型，False 表示不为厂商模型 const char *name 含义解释：存储名称

信息项	说明
	使用说明：以什么名称存储到 flash 中 const void *data 含义解释：要存储的模型数据 使用说明：当设置为 NULL 时表征删除所有模型数据 size_t data_len 含义解释：模型数据长度 使用说明：与 const void *data 参数搭配使用
返回值	int 类型 0：存储成功 非 0：存储失败

4.2.5.9 bt_mesh_model_extend

表 4-51 bt_mesh_model_extend 接口函数说明

信息项	说明
原型	int bt_mesh_model_extend(struct bt_mesh_model *mod, struct bt_mesh_model *base_mod);
功能	将一个模型扩展到另一个，Mesh 的模型可以被扩展以复用其功能，从而形成更复杂的模型。一组相互扩展的模型形成一个模型扩展树，扩展树中的所有模型每个元素共享一个订阅地址。接入层利用扩展树和元素中所有模型的订阅列表组合，从而为模型提供扩展的订阅列表容量
参数	const struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：扩展 Mesh 模型，相关结构体定义见表 4-41 struct bt_mesh_model *base_mod 含义解释：Mesh 网络模型 使用说明：被扩展的 Mesh 模型，将 model 扩展到 base_mod 上
返回值	int 类型 0：扩展成功成功 -EALREADY：base_mod 模型已经被扩展

4.2.6 配置模型接口

配置模型接口主要指的是 Foundation Model 下的 Cfg Client 和 Cfg Server 使用的接口。

1) 定义一个 Cfg Client 模型使用宏：

```
#define BT_MESH_MODEL_CFG_CLI(cli_data)                                  \
        BT_MESH_MODEL_CB(BT_MESH_MODEL_ID_CFG_CLI, bt_mesh_cfg_cli_op, NULL,  \
                         cli_data, &bt_mesh_cfg_cli_cb)
```

2) 定义一个 Cfg Server 模型使用宏：

```
#define BT_MESH_MODEL_CFG_SRV                                          \
    BT_MESH_MODEL_CB(BT_MESH_MODEL_ID_CFG_SRV, bt_mesh_cfg_srv_op, NULL,    \
                     NULL, &bt_mesh_cfg_srv_cb)
```

4.2.6.1 bt_mesh_beacon_set

表 4-52 bt_mesh_beacon_set 接口函数说明

信息项	说明
原型	void bt_mesh_beacon_set(bool beacon);
功能	启用/禁用本地 Secure Network Beacon 的发送
参数	bool beacon 含义解释：新的 Secure Network Beacon 状态 使用说明：为 True 时开启发送，为 False 是关闭发送
返回值	无

4.2.6.2 bt_mesh_beacon_enabled

表 4-53 bt_mesh_beacon_enabled 接口函数说明

信息项	说明
原型	bool bt_mesh_beacon_enabled(void);
功能	获取本地当前 Secure Network Beacon 的发送状态
参数	无
返回值	bool 类型 True：该功能已使能 False：该功能未使能

4.2.6.3 bt_mesh_default_ttl_set

表 4-54 bt_mesh_default_ttl_set 接口函数说明

信息项	说明
原型	int bt_mesh_default_ttl_set(uint8_t default_ttl);
功能	设置本地默认的 TTL 值，当未设置任何 TTL 值时，将使用 SDK 配置的默认 TTL 值
参数	uint8_t default_ttl 含义解释：要设置的 TTL 值 使用说明：新的默认 TTL 值，有效值为 0x00 和 0x02~BT_MESH_TTL_MAX
返回值	无

4.2.6.4 bt_mesh_default_ttl_get

表 4-55 bt_mesh_default_ttl_get 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_default_ttl_get(void);
功能	获取本地当前使用的默认的 TTL 值
参数	无
返回值	uint8_t 类型 获取到的默认的 TTL 值

4.2.6.5 bt_mesh_net_transmit_set

表 4-56 bt_mesh_net_transmit_set 接口函数说明

信息项	说明
原型	void bt_mesh_net_transmit_set(uint8_t xmit);
功能	设置本地 Mesh 网络传输参数，Mesh 网络传输参数决定了本地消息传输参数
参数	uint8_t xmit 含义解释：新的网络传输参数 使用说明：使用 BT_MESH_TRANSMIT 宏将 count 和 interval 参数进行拼接得到 xmit
返回值	无

4.2.6.6 bt_mesh_net_transmit_get

表 4-57 bt_mesh_net_transmit_get 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_default_ttl_get(void);
功能	获取本地当前的网络传播参数，使用宏 BT_MESH_TRANSMIT_COUNT 和宏 BT_MESH_TRANSMIT_INT 宏对获取到的值进行解码，获取 count 和 interval 值
参数	无
返回值	uint8_t 类型 获取当前的网络传播参数

4.2.6.7 bt_mesh_relay_set

表 4-58 bt_mesh_relay_set 接口函数说明

信息项	说明
原型	int bt_mesh_relay_set(enum bt_mesh_feat_state relay, uint8_t xmit);
功能	配置启用/禁用本地中继功能，并配置参数以进行消息的中继传输，必须通过

信息项	说明
	CONFIG_BT_MESH_RELAY 配置选项启用中继功能
参数	enum bt_mesh_feat_state relay 含义解释：新的中继状态 使用说明：枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED：不支持状态，不能被启用 uint8_t xmit 含义解释：新的网络传输参数 使用说明：中继发送使用该组参数
返回值	int 类型 0：设置成功 -ENOTSUP：中继功能不支持 -EINVAL：输入参数无效 -EALREDY：正在使用给定参数

4.2.6.8 bt_mesh_relay_get

表 4-59 bt_mesh_relay_get 接口函数说明

信息项	说明
原型	enum bt_mesh_feat_state bt_mesh_relay_get(void);
功能	获取本地当前中继功能状态
参数	无
返回值	bt_mesh_feat_state 类型 枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED：不支持状态，不能被启用

4.2.6.9 bt_mesh_relay_retransmit_get

表 4-60 bt_mesh_relay_retransmit_get 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_relay_retransmit_get(void);
功能	获取本地当前中继重传参数，使用宏 BT_MESH_TRANSMIT_COUNT 和宏 BT_MESH_TRANSMIT_INT 宏对获取到的值进行解码，获取 count 和 interval 值
参数	无

信息项	说明
返回值	uint8_t 类型 0：当前不支持中继功能 其他：当前中继重传参数

4.2.6.10 bt_mesh_gatt_proxy_set

表 4-61 bt_mesh_gatt_proxy_set 接口函数说明

信息项	说明
原型	int bt_mesh_gatt_proxy_set(enum bt_mesh_feat_state gatt_proxy);
功能	启用/禁用本地 GATT 代理功能，需通过 CONFIG_BT_MESH_GATT_PROXY 配置选项启用对 GATT 代理功能的支持。GATT 代理功能仅能控制代理节点消息中继到 Mesh 网络的能力，即使禁用了该功能，支持 GATT 代理的节点仍能广播可连接代理广播包，只能通过编译时配置完全禁用代理功能来停止代理广播包的发送
参数	enum bt_mesh_feat_state gatt_proxy 含义解释：新的 GATT 代理状态 使用说明：枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED：不支持状态，不能被启用
返回值	int 类型 0：设置成功 -ENOTSUP：GATT 代理功能不支持 -EINVAL：输入参数无效 -EALREDY：已经处于设置状态

4.2.6.11 bt_mesh_gatt_proxy_get

表 4-62 bt_mesh_gatt_proxy_get 接口函数说明

信息项	说明
原型	enum bt_mesh_feat_state bt_mesh_gatt_proxy_get(void);
功能	获取本地当前 GATT 代理服务状态
参数	无
返回值	bt_mesh_feat_state 类型 枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED:不支持状态，不能被启用

4.2.6.12 bt_mesh_friend_set

表 4-63 bt_mesh_friend_set 接口函数说明

信息项	说明
原型	int bt_mesh_friend_set(enum bt_mesh_feat_state friendship);
功能	启用/禁用本地 Friend 功能，如果禁用该功能，任何与本设备已经建立的 Friendship 将断开终止，必须通过 CONFIG_BT_MESH_FRIEND 配置选项启用对 Friend 功能的支持
参数	enum bt_mesh_feat_state friendship 含义解释：新的 Friend 状态 使用说明：枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED：不支持状态，不能被启用
返回值	int 类型 0：设置成功 -ENOTSUP：GATT 代理功能不支持 -EINVAL：输入参数无效 -EALREDY：已经处于设置状态

4.2.6.13 bt_mesh_friend_get

表 4-64 bt_mesh_friend_get 接口函数说明

信息项	说明
原型	enum bt_mesh_feat_state bt_mesh_friend_get(void);
功能	获取本地当前 Friend 功能状态
参数	无
返回值	bt_mesh_feat_state 类型 枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED：不支持状态，不能被启用

4.2.6.14 bt_mesh_subnet_add

表 4-65 bt_mesh_subnet_add 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_add(uint16_t net_idx, const uint8_t key[16]);
功能	向本地 Mesh 子网添加一个子网，将具有给定 Mesh 子网 Index 和网络密钥的子网添加到已知子网列表中。在给定子网上发送的所有消息都将有该节点处理，

信息项	说明
	并且该节点可以在给定子网上发送和接收 Network Beacon
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要新添加的子网的 Index const uint8_t key[16] 含义解释：子网的根网络密钥 使用说明：所有其他的密钥都是通过根网络密钥产生的
返回值	int 类型 STATUS_SUCCESS：添加成功 STATUS_INSUFF_RESOURCES：没有空间添加子网 STATUS_UNSPECIFIED：子网由于一些原因无法添加

4.2.6.15 bt_mesh_subnet_update

表 4-66 bt_mesh_subnet_update 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_update(uint16_t net_idx, const uint8_t key[16]);
功能	更新本地给定的子网密钥，通过添加第二组加密密钥来启动此子网的密钥刷新过程，子网将继续使用旧密钥发送(但同时使用这两个密钥接收消息)，直到子网进入密钥刷新阶段 2。Provisioner 为整个 Mesh 网络更新旧的网络密钥和 App 密钥，从而有效从 Mesh 网络中移除未获得新密钥的所有节点，通常与 bt_mesh_subnet_kr_phase_set()搭配使用
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要刷新密钥的子网的 Index const uint8_t key[16] 含义解释：子网的根网络密钥 使用说明：需要添加的新的网络密钥
返回值	int 类型 STATUS_SUCCESS：添加成功 STATUS_INVALID_NETKEY：子网 Index 未知 STATUS_IDX_ALREADY_STORED：添加的密钥和原网络密钥相同 STATUS_CANNOT_UPDATE：子网由于一些原因无法更新密钥

4.2.6.16 bt_mesh_subnet_del

表 4-67 bt_mesh_subnet_del 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_del(uint16_t net_idx);
功能	从本地节点中删除给定的子网，该节点将停止发送给定子网的 Network Beacon，并且不再处理该子网中的消息，绑定到该子网中的所有 App 也会被删除
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要删除的子网的 Index
返回值	int 类型 STATUS_SUCCESS：删除成功 STATUS_INVALID_NETKEY：子网 Index 未知

4.2.6.17 bt_mesh_subnet_exists

表 4-68 bt_mesh_subnet_exists 接口函数说明

信息项	说明
原型	bool bt_mesh_subnet_exists(uint16_t net_idx);
功能	检查本地子网是否已经存在
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要检查的子网的 Index
返回值	bool 类型 True：给定 Index 的子网存在 False：给定 Index 的子网不存在

4.2.6.18 bt_mesh_subnet_kr_phase_set

表 4-69 bt_mesh_subnet_kr_phase_set 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_kr_phase_set(uint16_t net_idx, uint8_t *phase);
功能	设置本地子网的密钥刷新阶段，通过 bt_mesh_subnet_update()函数来更新子网密钥来启动密钥刷新过程，会将子网置于密钥刷新阶段 1，一旦所有的节点都收到了新的子网密钥，就可以通过此函数进入密钥刷新阶段 2，以开始使用新的网络密钥进行传输。最后，要取消旧的网络密钥，需将密钥刷新阶段设置为 3，将使节点删除旧密钥，并使用新的密钥进行接收和发送
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要刷新密钥的子网的 Index

信息项	说明
	uint8_t *phase 含义解释：新的密钥刷新阶段指针 使用说明：更新后将返回实际的密钥刷新阶段，枚举型变量，取值如下： BT_MESH_KR_NORMAL：正常密钥刷新状态 BT_MESH_KR_PHASE_1：密钥刷新阶段 1 BT_MESH_KR_PHASE_2：密钥刷新阶段 2 BT_MESH_KR_PHASE_3：密钥刷新阶段 3
返回值	int 类型 STATUS_SUCCESS：密钥刷新成功 STATUS_INVALID_NETKEY：子网 Index 未知 STATUS_CANNOT_UPDATE：给定的密钥刷新阶段无效

4.2.6.19 bt_mesh_subnet_kr_phase_get

表 4-70 bt_mesh_subnet_kr_phase_get 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_kr_phase_get(uint16_t net_idx, uint8_t *phase);
功能	获取本地指定子网密钥刷新阶段
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要获取密钥刷新阶段的子网的 Index uint8_t *phase 含义解释：密钥刷新阶段指针 使用说明：获取到的密钥刷新阶段填充到该变量中
返回值	int 类型 STATUS_SUCCESS：密钥刷新成功 STATUS_INVALID_NETKEY：子网 Index 未知

4.2.6.20 bt_mesh_subnet_node_id_set

表 4-71 bt_mesh_subnet_node_id_set 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_node_id_set(uint16_t net_idx, enum bt_mesh_feat_state node_id);
功能	设置本地子网的节点身份状态，子网的节点身份状态确定了子网是否为代理客户端发送可连接的节点身份 Beacon。一旦启动，该 Beacon 将发送 60s，或者手动停止。此功能的作用与 bt_mesh_proxy_identity_enable()相同，但仅作用于单个子网。必须通过 CONFIG_BT_MESH_GATT_PROXY 启用 GATT 代理功能

信息项	说明
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要设置的子网的 Index enum bt_mesh_feat_state node_id 含义解释：新的节点身份状态 使用说明：枚举型变量，其取值如下： BT_MESH_FEATURE_DISABLED：禁止状态 BT_MESH_FEATURE_ENABLED：启用状态 BT_MESH_FEATURE_NOT_SUPPORTED：不支持状态，不能被启用
返回值	int 类型 STATUS_SUCCESS：设置成功 STATUS_INVALID_NETKEY：子网 Index 未知 STATUS_FEAT_NOT_SUPP：节点身份功能不支持 STATUS_CANNOT_SET：无法设置节点身份状态

4.2.6.21 bt_mesh_subnet_node_id_get

表 4-72 bt_mesh_subnet_node_id_get 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_subnet_node_id_get(uint16_t net_idx, enum bt_mesh_feat_state *node_id);
功能	获取本地子网的节点身份状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：需要获取身份状态的子网的 Index enum bt_mesh_feat_state *node_id 含义解释：节点身份状态 使用说明：获取到的节点身份状态填充到该变量中
返回值	int 类型 STATUS_SUCCESS：获取节点身份状态成功 STATUS_INVALID_NETKEY：子网 Index 未知

4.2.6.22 bt_mesh_subnets_get

表 4-73 bt_mesh_subnets_get 接口函数说明

信息项	说明
原型	ssize_t bt_mesh_subnets_get(uint16_t net_idxs[], size_t max, off_t skip);
功能	获取本地所有已知子网索引的列表，在两次调用此函数之间对子网列表进行的

信息项	说明
	任何更改都可能更改列表中条目的顺序和数量
参数	uint16_t net_idxs[] 含义解释：存储子网索引的数组 使用说明：在该数组中生成所有已知子网索引的列表，如果数组小于已知子网列表，则此函数将填充所有可用条目并返回-ENOMEM，在这种情况下，可以通过再次调用此函数来读取列表中的下一个 max 数量的条目 size_t max 含义解释：需要读取的条目数量 使用说明：需要读取的最达条目数量 off_t skip 含义解释：要跳过的索引数 使用说明：当读取多次时，将 skip 设置为 max 值则跳过上一次读取数量从下一个开始读
返回值	ssize_t 类型 -ENOMEM：已知子网条目数据超过 max 参数设置大小 其他：成功读取到的条目数量

4.2.6.23 bt_mesh_app_key_add

表 4-74 bt_mesh_app_key_add 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_app_key_add(uint16_t app_idx, uint16_t net_idx, const uint8_t key[16]);
功能	向本地添加一个 App Key，将具有给定索引的 App 添加到 App 列表中，允许节点发送和接收使用此 App Key 加密的模型消息。每个 App 都绑定到特定的子网，节点必须先知道 App 绑定到的子网，然后才能添加 App
参数	uint16_t app_idx 含义解释：需要添加的 App Index 使用说明：需要添加的 App Index uint16_t net_idx 含义解释：子网 Index 使用说明：需要添加 App key 到该子网的 Index const uint8_t key[16] 含义解释：实际的 App Key 使用说明：需要添加的 App Key
返回值	uint8_t 类型 STATUS_SUCCESS：App Key 成功添加 STATUS_INVALID_NETKEY：子网 Index 未知

信息项	说明
	STATUS_INSUFF_RESOURCES：无可用空间存储 App Key STATUS_INVALID_BINDING：该 App 已经绑定到其他子网 STATUS_IDX_ALREADY_STORED：该 App 已经存储了一个不同的 Key STATUS_CANNOT_SET：由于一些原因无法设置新的 App Key

4.2.6.24 bt_mesh_app_key_update

表 4-75 bt_mesh_app_key_update 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_app_key_update(uint16_t app_idx, uint16_t net_idx, const uint8_t key[16]);
功能	向本地更新 App Key，使用第二个 App Key 来更新 App，是密钥刷新过程的一部分，节点将继续使用旧的 App Key 进行传输，但同时在两者上进行接收，直到子网进入密钥刷新阶段 2，一旦子网进入密钥刷新阶段 3，旧的 App Key 将被删除。仅当绑定的子网处于密钥刷新阶段 1 时，才能更新 App Key
参数	uint16_t app_idx 含义解释：需要更新的 App Index 使用说明：需要更新的 App Index uint16_t net_idx 含义解释：子网 Index 使用说明：App 绑定到的子网 Index，或使用 BT_MESH_KEY_ANY 来绕过子网 Index 检查 const uint8_t key[16] 含义解释：实际的 App Key 使用说明：将要更新的 App Key
返回值	uint8_t 类型 STATUS_SUCCESS：App Key 更新添加 STATUS_INVALID_NETKEY：子网 Index 未知 STATUS_INVALID_BINDING：该 App 未绑定到给定的子网 STATUS_CANNOT_UPDATE：由于一些原因无法更新 App Key STATUS_IDX_ALREADY_STORED：该 App 已经更新了一个不同的 Key

4.2.6.25 bt_mesh_app_key_del

表 4-76 bt_mesh_app_key_del 接口函数说明

信息项	说明
原型	uint8_t bt_mesh_app_key_del(uint16_t app_idx, uint16_t net_idx);
功能	本地删除 App Key，绑定到此 App 的所有模型都将删除此绑定，使用此 App 发

信息项	说明
	布的所有模型都将停止发布
参数	uint16_t app_idx 含义解释：App Index 使用说明：需要删除的 App Index uint16_t net_idx 含义解释：子网 Index 使用说明：需要删除的子网的 Index
返回值	int 类型 STATUS_SUCCESS：删除成功 STATUS_INVALID_NETKEY：子网 Index 未知 STATUS_INVALID_BINDING：该 App 未绑定到给定的子网

4.2.6.26 bt_mesh_app_key_exists

表 4-77 bt_mesh_app_key_exists 接口函数说明

信息项	说明
原型	bool bt_mesh_app_key_exists(uint16_t app_idx);
功能	检查本地该 App Key 是否已经存在
参数	uint16_t app_idx 含义解释：App Index 使用说明：需要检查的 App Index
返回值	bool 类型 True：给定 Index 的 App Key 存在 False：给定 Index 的 App Key 不存在

4.2.6.27 bt_mesh_app_keys_get

表 4-78 bt_mesh_app_keys_get 接口函数说明

信息项	说明
原型	ssize_t bt_mesh_app_keys_get(uint16_t net_idx, uint16_t app_idxs[], size_t max, off_t skip)
功能	获取本地所有已知 App Key 的列表，在两次调用此函数之间对子网列表进行的任何更改都可能更改列表中条目的顺序和数量
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：App 所属子网的 Index uint16_t app_idxs[] 含义解释：存储 App Key 的数组

信息项	说明
	使用说明：在该数组中生成所有已知 App Key 的列表，如果数组小于已知子网列表，则此函数将填充所有可用条目并返回-ENOMEM，在这种情况下，可以通过再次调用此函数来读取列表中的下一个 max 数量的条目 size_t max 含义解释：需要读取的条目数量 使用说明：需要读取的最达条目数量 off_t skip 含义解释：要跳过的索引数 使用说明：当读取多次是，将 skip 设置为 max 值则调过上一次读取数量从下一个开始读
返回值	ssize_t 类型 -ENOMEM：已知 App Key 条目数据超过 max 参数设置大小 其他：成功读取到的条目数量

4.2.6.28 bt_mesh_cfg_node_reset

表 4-79 bt_mesh_cfg_node_reset 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_node_reset(uint16_t net_idx, uint16_t addr, bool *status);
功能	重置目标节点，并从 Mesh 网络中移除该节点
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：将要重置的节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：将要重置的节点的地址 bool *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：重置成功 非 0：重置失败

4.2.6.29 bt_mesh_cfg_comp_data_get

表 4-80 bt_mesh_cfg_comp_data_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_comp_data_get(uint16_t net_idx, uint16_t addr, uint8_t page, uint8_t *status, struct net_buf_simple *comp);
功能	获取目标节点的组成数据
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t page 含义解释：组成数据页面 使用说明：设置为 0xff 时为请求第一个可用页面 bool *status 含义解释：操作状态 使用说明：状态响应参数 struct net_buf_simple *comp 含义解释：组成数据 使用说明：获取到的目标节点的组成数据存储在该变量中
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.30 bt_mesh_cfg_beacon_get

表 4-81 bt_mesh_cfg_beacon_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_beacon_get(uint16_t net_idx, uint16_t addr, uint8_t *status);
功能	获取目标节点的 Network Beacon 状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址

信息项	说明
	bool *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_BEACON_DISABLED 或 BT_MESH_BEACON_ENABLED 表示成功
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.31 bt_mesh_cfg_beacon_set

表 4-82 bt_mesh_cfg_beacon_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_beacon_set(uint16_t net_idx, uint16_t addr, uint8_t val, uint8_t *status);
功能	设置目标节点的 Network Beacon 状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t val 含义解释：新的 Network Beacon 状态 使用说明：需要设置为 BT_MESH_BEACON_DISABLED 或 BT_MESH_BEACON_ENABLED bool *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_BEACON_DISABLED 或 BT_MESH_BEACON_ENABLED 表示成功
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.32 bt_mesh_cfg_ttl_get

表 4-83 bt_mesh_cfg_ttl_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_ttl_get(uint16_t net_idx, uint16_t addr, uint8_t *ttl);

信息项	说明
功能	获取目标节点的 TTL 值
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t *ttl 含义解释：TTL 响应 Buffer 使用说明：获取到的 TTL 的值存放到该值中
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.33 bt_mesh_cfg_ttl_set

表 4-84 bt_mesh_cfg_ttl_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_ttl_set(uint16_t net_idx, uint16_t addr, uint8_t val, uint8_t *ttl);
功能	设置目标节点的 ttl 值
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t val 含义解释：新的 TTL 值 使用说明：需要设置到目标节点的 TTL 的值 uint8_t *ttl 含义解释：TTL 响应 Buffer 使用说明：设置完成后的响应值放在该值中
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.34 bt_mesh_cfg_friend_get

表 4-85 bt_mesh_cfg_friend_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_friend_get(uint16_t net_idx, uint16_t addr, uint8_t *status);
功能	获取目标节点的 Friend 功能的状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_FRIEND_DISABLED、BT_MESH_FRIEND_ENABLED 或 BT_MESH_FRIEND_NOT_SUPPORTED 表示从对端获取状态成功，返回其他值表示从对端获取状态失败
返回值	int 类型 0：命令执行成功 非 0：命令执行失败

4.2.6.35 bt_mesh_cfg_friend_set

表 4-86 bt_mesh_cfg_friend_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_friend_set(uint16_t net_idx, uint16_t addr, uint8_t val, uint8_t *status);
功能	设置目标节点的 Friend 功能状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t val 含义解释：新的 Friend 功能状态 使用说明：需要设置为 BT_MESH_FRIEND_DISABLED 或 BT_MESH_FRIEND_ENABLED uint8_t *status

信息项	说明
	含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_FRIEND_DISABLED 或 BT_MESH_FRIEND_ENABLED 或 BT_MESH_FRIEND_NOT_SUPPORTED 表示成功
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.36 bt_mesh_cfg_gatt_proxy_get

表 4-87 bt_mesh_cfg_gatt_proxy_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_gatt_proxy_get(uint16_t net_idx, uint16_t addr, uint8_t *status);
功能	获取目标节点的 Proxy 功能的状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_GATT_PROXY_DISABLED、BT_MESH_GATT_PROXY_ENABLED 或 BT_MESH_GATT_PROXY_NOT_SUPPORTED 表示成功
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.37 bt_mesh_cfg_gatt_proxy_set

表 4-88 bt_mesh_cfg_gatt_proxy_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_gatt_proxy_set(uint16_t net_idx, uint16_t addr, uint8_t val, uint8_t *status);
功能	设置目标节点的 Proxy 功能状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index

信息项	说明
	uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t val 含义解释：新的 Proxy 功能状态 使用说明：需要设置为 BT_MESH_GATT_PROXY_DISABLED 或 BT_MESH_GATT_PROXY_ENABLED uint8_t *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_GATT_PROXY_DISABLED 或 BT_MESH_GATT_PROXY_ENABLED 或 BT_MESH_GATT_PROXY_NOT_SUPPORTED 表示成功
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.38 bt_mesh_cfg_net_transmit_get

表 4-89 bt_mesh_cfg_net_transmit_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_net_transmit_get(uint16_t net_idx, uint16_t addr, uint8_t *transmit);
功能	获取目标节点的 Network Transmit 功能的状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t *transmit 含义解释：网络传输响应参数 使用说明：获取到目标节点的响应数据，存储在该参数中，可通过宏 BT_MESH_TRANSMIT_COUNT 和 BT_MESH_TRANSMIT_INT 计算实际的 count 和 interval
返回值	int 类型 0：命令执行成功 非 0：命令执行失败

4.2.6.39 bt_mesh_cfg_net_transmit_set

表 4-90 bt_mesh_cfg_net_transmit_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_net_transmit_set(uint16_t net_idx, uint16_t addr, uint8_t val, uint8_t *transmit);
功能	设置目标节点的 Network Transmit 参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t val 含义解释：新的 Network Transmit 参数 使用说明：通过宏 BT_MESH_TRANSMIT 将 count 与 interval 连接获取 uint8_t *transmit 含义解释：网络传输响应参数 使用说明：获取到值表示成功，可通过宏 BT_MESH_TRANSMIT_COUNT 和 BT_MESH_TRANSMIT_INT 获取到实际的 count 和 interval 表示成功
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.40 bt_mesh_cfg_relay_get

表 4-91 bt_mesh_cfg_relay_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_relay_get(uint16_t net_idx, uint16_t addr, uint8_t *status, uint8_t *transmit);
功能	获取目标节点的 Relay 功能的状态
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_RELAY_DISABLED、

信息项	说明
	BT_MESH_RELAY_ENABLED 或 BT_MESH_RELAY_NOT_SUPPORTED 表示成功 uint8_t *transmit 含义解释：网络传输响应参数 使用说明：获取到值表示成功，可通过宏 BT_MESH_TRANSMIT_COUNT 和 BT_MESH_TRANSMIT_INT 获取到实际的 count 和 interval
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.41 bt_mesh_cfg_relay_set

表 4-92 bt_mesh_cfg_relay_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_relay_set(uint16_t net_idx, uint16_t addr, uint8_t new_relay, uint8_t new_transmit, uint8_t *status, uint8_t *transmit);
功能	设置目标节点的 Relay 参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint8_t new_relay 含义解释：新的 Relay 状态 使用说明:需要设置为 BT_MESH_RELAY_DISABLED 或 BT_MESH_RELAY_ENABLED uint8_t new_transmit 含义解释：新的 Network Transmit 参数 使用说明：通过宏 BT_MESH_TRANSMIT 将 count 与 interval 进行组合 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数，返回 BT_MESH_GATT_PROXY_DISABLED 或 BT_MESH_GATT_PROXY_ENABLED 或 BT_MESH_GATT_PROXY_NOT_SUPPORTED 表示成功 uint8_t *transmit 含义解释：网络传输响应参数 使用说明：获取到值表示成功，可通过宏 BT_MESH_TRANSMIT_COUNT 和 BT_MESH_TRANSMIT_INT 获取到实际的 count 和 interval

信息项	说明
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.42 bt_mesh_cfg_net_key_add

表 4-93 bt_mesh_cfg_net_key_add 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_net_key_add(uint16_t net_idx, uint16_t addr, uint16_t key_net_idx, const uint8_t net_key[16], uint8_t *status);
功能	给目标节点添加一个网络密钥
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t key_net_idx 含义解释：网络密钥的 Index 使用说明：需要添加的网络密钥的 Index const uint8_t net_key[16] 含义解释：实际的网络密钥 使用说明：实际将要添加的网络密钥 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：添加成功 非 0：添加失败

4.2.6.43 bt_mesh_cfg_net_key_get

表 4-94 bt_mesh_cfg_net_key_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_net_key_get(uint16_t net_idx, uint16_t addr, uint16_t *keys, size_t *key_cnt);
功能	获取目标节点的网络密钥索引列表

信息项	说明
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t *keys 含义解释：网络密钥索引列表响应参数 使用说明：获取到的目标节点的网络密钥索引列表填充在该变量中 size_t *key_cnt 含义解释：网络密钥索引列表长度 使用说明：获取到的目标节点的网络密钥索引列表长度，和 keys 参数搭配使用
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.44 bt_mesh_cfg_net_key_del

表 4-95 bt_mesh_cfg_net_key_del 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_net_key_del(uint16_t net_idx, uint16_t addr, uint16_t key_net_idx, uint8_t *status);
功能	删除目标节点的一个网络密钥
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t key_net_idx 含义解释：网络密钥的 Index 使用说明：将要删除的网络密钥的 Index uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：删除成功

信息项	说明
	非 0：删除失败

4.2.6.45 bt_mesh_cfg_net_key_update

表 4-96 bt_mesh_cfg_net_key_update 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_net_key_update(uint16_t net_idx, uint16_t addr, uint16_t key_net_idx, const uint8_t net_key[16], uint8_t *status);
功能	更新目标节点的一个网络密钥
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t key_net_idx 含义解释：网络密钥的 Index 使用说明：将要更新的网络密钥的 Index const uint8_t net_key[16] 含义解释：网络密钥 使用说明：将要更新的网络密钥 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：更新成功 非 0：更新失败

4.2.6.46 bt_mesh_cfg_key_refresh_phase_get

表 4-97 bt_mesh_cfg_key_refresh_phase_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_key_refresh_phase_get(uint16_t net_idx, uint16_t addr, uint16_t idx, uint16_t *netkey_idx, uint8_t *phase, uint8_t *status);
功能	获取目标节点密钥刷新阶段
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index

信息项	说明
	uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t idx 含义解释：网络密钥的 Index 使用说明：需要设置密钥刷新阶段的网络密钥的 Index uint16_t *netkey_idx 含义解释：网络密钥的 Index 使用说明：获取到的响应中指示的网络密钥 Index 填充在该变量中 uint8_t *phase 含义解释：密钥刷新阶段 使用说明：获取到的密钥刷新阶段填充在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.47 bt_mesh_cfg_key_refresh_phase_set

表 4-98 bt_mesh_cfg_key_refresh_phase_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_key_refresh_phase_set(uint16_t net_idx, uint16_t addr, uint16_t idx, uint8_t transition, uint16_t *netkey_idx, uint8_t *phase, uint8_t *status);
功能	设置目标节点密钥刷新阶段
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t idx 含义解释：网络密钥的 Index 使用说明：需要获取密钥刷新阶段的网络密钥的 Index uint8_t transition

信息项	说明
	含义解释：密钥刷新阶段 使用说明：将要把目标节点密钥刷新阶段设置到的值 uint16_t *netkey_idx 含义解释：网络密钥的 Index 使用说明：获取到的响应中指示的网络密钥 Index 填充在该变量中 uint8_t *phase 含义解释：密钥刷新阶段 使用说明：获取到的密钥刷新阶段填充在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.48 bt_mesh_cfg_app_key_add

表 4-99 bt_mesh_cfg_app_key_add 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_app_key_add(uint16_t net_idx, uint16_t addr, uint16_t key_net_idx, uint16_t key_app_idx, const uint8_t app_key[16], uint8_t *status);
功能	给目标节点添加一个 App Key
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t key_net_idx 含义解释：网络密钥的 Index 使用说明：需要添加的 App Key 所对应的网络密钥的 Index uint16_t key_app_idx 含义解释：App Key 的 Index 使用说明：即将添加的 App Key 的 Index const uint8_t app_key[16] 含义解释：实际的 App Key

信息项	说明
	使用说明：实际将要添加的 App Key uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：添加成功 非 0：添加失败

4.2.6.49 bt_mesh_cfg_app_key_get

表 4-100 bt_mesh_cfg_app_key_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_app_key_get(uint16_t net_idx, uint16_t addr, uint16_t key_net_idx, uint8_t *status, uint16_t *keys, size_t *key_cnt);
功能	获取特定网络密钥的目标节点的 App Key 索引列表
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t key_net_idx 含义解释：网络密钥的 Index 使用说明：需要获取的 App Key 索引列表所对应的网络密钥的 Index uint8_t *status 含义解释：操作状态 使用说明：状态响应参 uint16_t *keys 含义解释：App Key 索引列表响应参数 使用说明：获取到的目标节点的 App Key 索引列表填充在该变量中 size_t *key_cnt 含义解释：App Key 索引列表长度 使用说明：获取到的目标节点的 App Key 索引列表长度，和 keys 参数搭配使用
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.50 bt_mesh_cfg_app_key_del

表 4-101 bt_mesh_cfg_app_key_del 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_app_key_del(uint16_t net_idx, uint16_t addr, uint16_t key_net_idx, uint16_t key_app_idx, uint8_t *status);
功能	删除目标节点的一个 App Key
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t key_net_idx 含义解释：网络密钥的 Index 使用说明：将要删除的 App Key 所属的网络密钥的 Index uint16_t key_app_idx 含义解释：App Key 的 Index 使用说明：将要删除的 App Key 的 Index uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：删除成功 非 0：删除失败

4.2.6.51 bt_mesh_cfg_mod_app_bind

表 4-102 bt_mesh_cfg_mod_app_bind 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_app_bind(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_app_idx, uint16_t mod_id, uint8_t *status);
功能	将目标节点的一个 SIG 模型与一个 App Key 进行绑定
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址

信息项	说明
	使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要绑定的模型所在的元素地址 uint16_t mod_app_idx 含义解释：App Key 的 Index 使用说明：将要绑定的 App 的 Index uint16_t mod_id 含义解释：模型 ID 使用说明：将要进行绑定的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：绑定成功 非 0：绑定失败

4.2.6.52 bt_mesh_cfg_mod_app_unbind

表 4-103 bt_mesh_cfg_mod_app_unbind 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_app_unbind(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_app_idx, uint16_t mod_id, uint8_t *status);
功能	将目标节点的一个 SIG 模型与一个 App Key 进行解绑
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要解绑的模型所在的元素地址 uint16_t mod_app_idx 含义解释：App Key 的 Index 使用说明：将要解绑的 App 的 Index

信息项	说明
	uint16_t mod_id 含义解释：模型 ID 使用说明：将要进行解绑的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：解绑成功 非 0：解绑失败

4.2.6.53 bt_mesh_cfg_mod_app_bind_vnd

表 4-104 bt_mesh_cfg_mod_app_bind_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_app_bind_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_app_idx, uint16_t mod_id, uint16_t cid, uint8_t *status);
功能	将目标节点的一个厂商模型与一个 App Key 进行绑定
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要绑定的厂商模型所在的元素地址 uint16_t mod_app_idx 含义解释：App Key 的 Index 使用说明：将要绑定的 App 的 Index uint16_t mod_id 含义解释：模型 ID 使用说明：将要进行绑定的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status

信息项	说明
	含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：绑定成功 非 0：绑定失败

4.2.6.54 bt_mesh_cfg_mod_app_unbind_vnd

表 4-105 bt_mesh_cfg_mod_app_unbind_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_app_unbind_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_app_idx, uint16_t mod_id, uint16_t cid, uint8_t *status);
功能	将目标节点的一个厂商模型与一个 App Key 进行解绑
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要解绑的模型所在的元素地址 uint16_t mod_app_idx 含义解释：App Key 的 Index 使用说明：将要解绑的 App 的 Index uint16_t mod_id 含义解释：模型 ID 使用说明：将要进行解绑的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：解绑成功

信息项	说明
	非 0：解绑失败

4.2.6.55 bt_mesh_cfg_mod_app_get

表 4-106 bt_mesh_cfg_mod_app_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_app_get(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, uint8_t *status, uint16_t *apps, size_t *app_cnt);
功能	获取目标节点上绑定到 SIG 模型的所有 App Key 列表
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要获取的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要获取的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数 uint16_t *apps 含义解释：App 索引列表响应参数 使用说明：获取到的目标节点绑定 SIG 模型的 App Key 索引列表填充在该变量中 size_t *app_cnt 含义解释：App Key 索引列表长度 使用说明：获取到的目标节点绑定 SIG 模型的 App Key 索引列表长度，和 keys 参数搭配使用
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.56 bt_mesh_cfg_mod_app_get_vnd

表 4-107 bt_mesh_cfg_mod_app_get_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_app_get_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, uint16_t cid, uint8_t *status, uint16_t *apps, size_t *app_cnt);
功能	获取目标节点上绑定到厂商模型的所有 App 的列表
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要获取的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要获取的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数 uint16_t *apps 含义解释：App 索引列表响应参数 使用说明：获取到的目标节点绑定厂商模型的 App Key 索引列表填充在该变量中 size_t *app_cnt 含义解释：App Key 索引列表长度 使用说明：获取到的目标节点绑定厂商模型的 App Key 索引列表长度，和 keys 参数搭配使用
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.57 bt_mesh_cfg_mod_pub_get

表 4-108 bt_mesh_cfg_mod_pub_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_pub_get(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, struct bt_mesh_cfg_mod_pub *pub, uint8_t *status);
功能	获取目标节点上 SIG 模型的发布参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要获取的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要获取的模型 ID struct bt_mesh_cfg_mod_pub *pub 含义解释：发布参数 Buffer 使用说明：获取到的目标节点的发布参数存储到该变量中，结构体说明见表 4-109 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：获取成功 非 0：获取失败

表 4-109 bt_mesh_cfg_mod_pub 结构体成员说明

成员项	说明
uint16_t addr	含义解释：发布的目标地址 使用说明：发布的消息将会发往的地址
uint16_t app_idx	含义解释：App 索引 使用说明：该模型使用的 App 索引
bool cred_flag	含义解释：Friendship 证书标志

成员项	说明
	使用说明：标识是使用 Friendship 密钥加密还是使用 App Key 进行加密
uint8_t ttl	含义解释：发布的默认 TTL 值 使用说明：发布时将默认使用该 TTL 值进行发送
uint8_t period	含义解释：发布周期 使用说明：通过宏 BT_MESH_PUB_PERIOD_100MS、BT_MESH_PUB_PERIOD_SEC、BT_MESH_PUB_PERIOD_10SEC 和 BT_MESH_PUB_PERIOD_10MIN 来确定周期
uint8_t transmit	含义解释：发布发送参数 使用说明：通过宏 BT_MESH_TRANSMIT 将 count 和 interval 进行组合

4.2.6.58 bt_mesh_cfg_mod_pub_get_vnd

表 4-110 bt_mesh_cfg_mod_pub_get_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_pub_get_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, uint16_t cid, struct bt_mesh_cfg_mod_pub *pub, uint8_t *status);
功能	获取目标节点上厂商模型的发布参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要获取的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要获取的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID

信息项	说明
	struct bt_mesh_cfg_mod_pub *pub 含义解释：发布参数结构体 使用说明：获取到的目标节点的发布参数存储到该变量中，结构体说明见表 4-109 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.59 bt_mesh_cfg_mod_pub_set

表 4-111 bt_mesh_cfg_mod_pub_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_pub_set(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, struct bt_mesh_cfg_mod_pub *pub, uint8_t *status);
功能	设置目标节点上 SIG 模型的发布参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID struct bt_mesh_cfg_mod_pub *pub 含义解释：发布参数 Buffer 使用说明：需要设置的参数存储在该变量中，结构体说明见表 4-109 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数

信息项	说明
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.60 bt_mesh_cfg_mod_pub_set_vnd

表 4-112 bt_mesh_cfg_mod_pub_set_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_pub_set_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, uint16_t cid, struct bt_mesh_cfg_mod_pub *pub, uint8_t *status);
功能	设置目标节点上厂商模型的发布参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID struct bt_mesh_cfg_mod_pub *pub 含义解释：发布参数 Buffer 使用说明：设置到目标节点的发布参数存储到该变量中，结构体说明见表 4-109 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.61 bt_mesh_cfg_mod_sub_add

表 4-113 bt_mesh_cfg_mod_sub_add 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_add(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t sub_addr, uint16_t mod_id, uint8_t *status);
功能	将组播地址添加到 SIG 模型的订阅列表中
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t sub_addr 含义解释：组播地址 使用说明：将要添加到订阅列表中的组播地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：添加成功 非 0：添加失败

4.2.6.62 bt_mesh_cfg_mod_sub_add_vnd

表 4-114 bt_mesh_cfg_mod_sub_add_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_add_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t sub_addr, uint16_t mod_id, uint16_t cid, uint8_t *status);
功能	将组播地址添加到厂商模型的订阅列表中
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index

信息项	说明
	uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t sub_addr 含义解释：组播地址 使用说明：将要添加到订阅列表中的组播地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：添加成功 非 0：添加失败

4.2.6.63 bt_mesh_cfg_mod_sub_del

表 4-115 bt_mesh_cfg_mod_sub_del 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_del(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t sub_addr, uint16_t mod_id, uint8_t *status);
功能	删除 SIG 模型的订阅列表中的指定的组播地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr

信息项	说明
	含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t sub_addr 含义解释：组播地址 使用说明：将要从订阅列表中删除的组播地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：删除成功 非 0：删除失败

4.2.6.64 bt_mesh_cfg_mod_sub_del_vnd

表 4-116 bt_mesh_cfg_mod_sub_del_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_del_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t sub_addr, uint16_t mod_id, uint16_t cid, uint8_t *status);
功能	删除厂商模型的订阅列表中的指定的组播地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t sub_addr 含义解释：组播地址 使用说明：将要从订阅列表中删除的组播地址 uint16_t mod_id 含义解释：模型 ID

信息项	说明
	使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：删除成功 非 0：删除失败

4.2.6.65 bt_mesh_cfg_mod_sub_overwrite

表 4-117 bt_mesh_cfg_mod_sub_overwrite 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_overwrite(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t sub_addr, uint16_t mod_id, uint8_t *status);
功能	用指定组播地址覆盖 SIG 模型的订阅列表中的所有地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t sub_addr 含义解释：组播地址 使用说明：将要覆盖到订阅列表中的组播地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数

信息项	说明
返回值	int 类型 0：覆盖成功 非 0：覆盖失败

4.2.6.66 bt_mesh_cfg_mod_sub_overwrite_vnd

表 4-118 bt_mesh_cfg_mod_sub_overwrite_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_overwrite_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t sub_addr, uint16_t mod_id, uint16_t cid, uint8_t *status);
功能	用指定组播地址覆盖厂商模型的订阅列表中的所有地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 uint16_t sub_addr 含义解释：组播地址 使用说明：将要覆盖到订阅列表中的组播地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：覆盖成功 非 0：覆盖失败

4.2.6.67 bt_mesh_cfg_mod_sub_va_add

表 4-119 bt_mesh_cfg_mod_sub_va_add 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_va_add(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, const uint8_t label[16], uint16_t mod_id, uint16_t *virt_addr, uint8_t *status);
功能	将虚拟地址添加到 SIG 模型的订阅列表中
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 const uint8_t label[16] 含义解释：虚拟地址标签 使用说明：要添加到订阅列表中的虚拟地址标签 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t *virt_addr 含义解释：虚拟地址响应参数 使用说明：设置完后获取到的虚拟地址存储在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：添加成功 非 0：添加失败

4.2.6.68 bt_mesh_cfg_mod_sub_va_add_vnd

表 4-120 bt_mesh_cfg_mod_sub_va_add_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_va_add_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, const uint8_t label[16], uint16_t mod_id, uint16_t cid, uint16_t *virt_addr, uint8_t *status);

信息项	说明
功能	将虚拟地址添加到厂商模型的订阅列表中
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 const uint8_t label[16] 含义解释：虚拟地址标签 使用说明：要添加到订阅列表中的虚拟地址标签 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint16_t *virt_addr 含义解释：虚拟地址响应参数 使用说明：设置完后获取到的虚拟地址存储在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：添加成功 非 0：添加失败

4.2.6.69 bt_mesh_cfg_mod_sub_va_del

表 4-121 bt_mesh_cfg_mod_sub_va_del 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_va_del(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, const uint8_t label[16], uint16_t mod_id, uint16_t *virt_addr, uint8_t *status);

信息项	说明
功能	删除 SIG 模型的订阅列表中的指定的虚拟地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 const uint8_t label[16] 含义解释：虚拟地址标签 使用说明：要从订阅列表中删除的虚拟地址标签 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t *virt_addr 含义解释：虚拟地址响应参数 使用说明：设置完后获取到的虚拟地址存储在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：删除成功 非 0：删除失败

4.2.6.70 bt_mesh_cfg_mod_sub_va_del_vnd

表 4-122 bt_mesh_cfg_mod_sub_va_del_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_va_del_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, const uint8_t label[16], uint16_t mod_id, uint16_t cid, uint16_t *virt_addr, uint8_t *status);
功能	删除厂商模型的订阅列表中的指定的虚拟地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index

信息项	说明
	uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 const uint8_t label[16] 含义解释：虚拟地址标签 使用说明：要从订阅列表中删除的虚拟地址标签 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint16_t *virt_addr 含义解释：虚拟地址响应参数 使用说明：设置完后获取到的虚拟地址存储在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：删除成功 非 0：删除失败

4.2.6.71 bt_mesh_cfg_mod_sub_va_overwrite

表 4-123 bt_mesh_cfg_mod_sub_va_overwrite 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_va_overwrite(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, const uint8_t label[16], uint16_t mod_id, uint16_t *virt_addr, uint8_t *status);
功能	用指定虚拟地址覆盖 SIG 模型的订阅列表中的所有地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index

信息项	说明
	uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 const uint8_t label[16] 含义解释：虚拟地址标签 使用说明：将要覆盖到订阅列表中的虚拟地址标签 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t *virt_addr 含义解释：虚拟地址响应参数 使用说明：设置完后获取到的虚拟地址存储在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：覆盖成功 非 0：覆盖失败

4.2.6.72 bt_mesh_cfg_mod_sub_va_overwrite_vnd

表 4-124 bt_mesh_cfg_mod_sub_va_overwrite_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_va_overwrite_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, const uint8_t label[16], uint16_t mod_id, uint16_t cid, uint16_t *virt_addr, uint8_t *status);
功能	用指定虚拟地址覆盖厂商模型的订阅列表中的所有地址
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr

信息项	说明
	含义解释：元素地址 使用说明：将要设置的模型所在的元素地址 const uint8_t label[16] 含义解释：虚拟地址标签 使用说明：将要覆盖到订阅列表中的虚拟地址标签 uint16_t mod_id 含义解释：模型 ID 使用说明：将要设置的模型 ID uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint16_t *virt_addr 含义解释：虚拟地址响应参数 使用说明：设置完后获取到的虚拟地址存储在该变量中 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：覆盖成功 非 0：覆盖失败

4.2.6.73 bt_mesh_cfg_mod_sub_get

表 4-125 bt_mesh_cfg_mod_sub_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_get(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, uint8_t *status, uint16_t *subs, size_t *sub_cnt);
功能	获取目标节点上 SIG 模型的订阅列表
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址

信息项	说明
	使用说明：将要获取的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要获取的模型 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数 uint16_t *subs 含义解释：订阅列表 Buffer 使用说明：获取到的目标节点的订阅列表存储到该变量中 size_t *sub_cnt 含义解释：订阅列表元素个数 使用说明：获取到的订阅列表中的元素个数，与 subs 参数搭配使用
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.74 bt_mesh_cfg_mod_sub_get_vnd

表 4-126 bt_mesh_cfg_mod_sub_get_vnd 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_mod_sub_get_vnd(uint16_t net_idx, uint16_t addr, uint16_t elem_addr, uint16_t mod_id, uint16_t cid, uint8_t *status, uint16_t *subs, size_t *sub_cnt);
功能	获取目标节点上厂商模型的订阅列表
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 uint16_t elem_addr 含义解释：元素地址 使用说明：将要获取的模型所在的元素地址 uint16_t mod_id 含义解释：模型 ID 使用说明：将要获取的模型 ID

信息项	说明
	uint16_t cid 含义解释：公司 ID 使用说明：厂商模型的公司 ID uint8_t *status 含义解释：操作状态 使用说明：状态响应参数 uint16_t *subs 含义解释：订阅列表 Buffer 使用说明：获取到的目标节点的订阅列表存储到该变量中 size_t *sub_cnt 含义解释：订阅列表元素个数 使用说明：获取到的订阅列表中的元素个数，与 subs 参数搭配使用
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.75 bt_mesh_cfg_hb_sub_set

表 4-127 bt_mesh_cfg_hb_sub_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_hb_sub_set(uint16_t net_idx, uint16_t addr, struct bt_mesh_cfg_hb_sub *sub, uint8_t *status);
功能	设置目标节点的 Heartbeat 订阅参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 struct bt_mesh_cfg_hb_sub *sub 含义解释：新的 Heartbeat 订阅参数 使用说明：将要设置到目标节点的 Heartbeat 订阅参数，相关结构体定义见表 4-128 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数

信息项	说明
返回值	int 类型 0：设置成功 非 0：设置失败

表 4-128 bt_mesh_cfg_hb_sub 结构体成员说明

成员项	说明
uint16_t src	含义解释：源地址 使用说明：接收到的 Heartbeat 消息中的源地址字段
uint16_t dst	含义解释：目的地址 使用说明：接收到的 Heartbeat 消息中的目的地址字段
uint8_t period	含义解释：对数订阅周期 使用说明：实际的订阅周期为(1<<(period - 1))秒，如果 period 设置为 0，则为 0 秒
uint8_t count	含义解释：对数 Heartbeat 订阅接收计数 使用说明：若计数在 1~0xFE 之间，则 Heartbeat 计数为(1<<(count-1))；若计数为 0，则 Heartbeat 计数为 0；若计数为 0xFF，则 Heartbeat 计数为 0xFFFF；但在 Heartbeat Set 中该参数会被忽略
uint8_t min	含义解释：接收到的消息中的最小 hop 使用说明：从发布节点到订阅节点的最短路径，与相邻节点的 hop count=1；在 Heartbeat Set 中该参数会被忽略
uint8_t max	含义解释：接收到的消息中的最大 hop 使用说明：从发布节点到订阅节点的最长路径，与相邻节点的 hop count=1；在 Heartbeat Set 中该参数会被忽略

4.2.6.76 bt_mesh_cfg_hb_sub_get

表 4-129 bt_mesh_cfg_hb_sub_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_hb_sub_get(uint16_t net_idx, uint16_t addr, struct bt_mesh_cfg_hb_sub *sub, uint8_t *status);
功能	获取目标节点的 Heartbeat 订阅参数

信息项	说明
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 struct bt_mesh_cfg_hb_sub *sub 含义解释：新的 Heartbeat 订阅参数 使用说明：获取到的目标节点的 Heartbeat 订阅参数存储在该变量中，相关结构体定义见表 4-128 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.6.77 bt_mesh_cfg_hb_pub_set

表 4-130 bt_mesh_cfg_hb_pub_set 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_hb_pub_set(uint16_t net_idx, uint16_t addr, const struct bt_mesh_cfg_hb_pub *pub, uint8_t *status);
功能	设置目标节点的 Heartbeat 发布参数
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 const struct bt_mesh_cfg_hb_pub *pub 含义解释：新的 Heartbeat 发布参数 使用说明：将要设置到目标节点的 Heartbeat 发布参数，相关结构体定义见表 4-131 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数

信息项	说明
返回值	int 类型 0：设置成功 非 0：设置失败

表 4-131 bt_mesh_cfg_hb_pub 结构体成员说明

成员项	说明
uint16_t dst	含义解释：目的地址 使用说明：Heartbeat 消息发送的目的地址
uint8_t count	含义解释：对数 Heartbeat 计数 使用说明：若计数在 1~0x11 之间，则 Heartbeat 计数为(1<<(count-1))；若计数为 0，则 Heartbeat 计数为 0；若计数为 0xFF，则 Heartbeat 计数为不确定；在 Heartbeat Set 中使用时，此参数表示要发送的 Heartbeat 消息数；在获取 Heartbeat 发布后，此参数表示剩余要发送的心跳消息数
uint8_t period	含义解释：对数 Heartbeat 发布传输间隔(单位秒) 使用说明：如果 period 在 1~0x11 之间，则实际的发布周期为(1<<(period - 1))秒，如果 period 设置为 0，则禁用 Heartbeat 发布
uint8_t ttl	含义解释：发布消息 TTL 值 使用说明：发布消息所使用的默认的 TTL 值
uint16_t feat	含义解释：触发 Heartbeat 发布的功能的位图 使用说明：合理值的值有：BT_MESH_FEAT_RELAY、BT_MESH_FEAT_PROXY、BT_MESH_FEAT_FRIEND 和 BT_MESH_FEAT_LOW_POWER
uint16_t net_idx	含义解释：子网 Index 使用说明：要发布到的子网索引

4.2.6.78 bt_mesh_cfg_hb_pub_get

表 4-132 bt_mesh_cfg_hb_pub_get 接口函数说明

信息项	说明
原型	int bt_mesh_cfg_hb_pub_get(uint16_t net_idx, uint16_t addr, struct bt_mesh_cfg_hb_pub *pub, uint8_t *status);
功能	获取目标节点的 Heartbeat 发布参数

信息项	说明
参数	uint16_t net_idx 含义解释：子网 Index 使用说明：目标节点所在的子网 Index uint16_t addr 含义解释：节点地址 使用说明：目标节点的地址 struct bt_mesh_cfg_hb_pub *pub 含义解释：新的 Heartbeat 发布参数 使用说明：获取到的目标节点的 Heartbeat 发布参数存储在该变量中，相关结构体定义见表 4-131 uint8_t *status 含义解释：操作状态 使用说明：状态响应参数
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.6.79 bt_mesh_cfg_cli_timeout_get

表 4-133 bt_mesh_cfg_cli_timeout_get 接口函数说明

信息项	说明
原型	int32_t bt_mesh_cfg_cli_timeout_get(void);
功能	获取当前 Cfg Client 的消息响应超时时间
参数	无
返回值	int32_t 类型 配置的传输超时值，以 ms 为单位

4.2.6.80 bt_mesh_cfg_cli_timeout_set

表 4-134 bt_mesh_cfg_cli_timeout_set 接口函数说明

信息项	说明
原型	void bt_mesh_cfg_cli_timeout_set(int32_t timeout);
功能	设置当前 Cfg Client 的消息响应超时时间
参数	int32_t timeout 含义解释：传输超时值 使用说明：将要设置的传输超时值，ms 为单位

信息项	说明
返回值	无

4.2.7 On/Off 模型接口

对于 On/Off 模型，存在 Client 和 Server 两端，其模型定义分别如下：

1) Client 端：

Generic On/Off Client 模型的 publish 内容定义：

```
#define BT_MESH_GEN_ONOFF_CLI_PUB_DEFINE(_name) \
    BT_MESH_MODEL_PUB_DEFINE(_name, NULL, 2 + 4)
```

Generic On/Off Client 模型定义：

```
#define BT_MESH_MODEL_GEN_ONOFF_CLI(cli, pub) \
        BT_MESH_MODEL_CB(BT_MESH_MODEL_ID_GEN_ONOFF_CLI,      \
                gen_onoff_cli_op, pub, cli, &bt_mesh_gen_onoff_cb)
```

2) Server 端

Generic On/Off Server 模型的 publish 内容定义：

```
#define BT_MESH_GEN_ONOFF_SRV_PUB_DEFINE(_name) \
    BT_MESH_MODEL_PUB_DEFINE(_name, NULL, 2 + 3)
```

Generic On/Off Server 模型定义：

```
#define BT_MESH_MODEL_GEN_ONOFF_SRV(srv, pub)     \
        BT_MESH_MODEL_CB(BT_MESH_MODEL_ID_GEN_ONOFF_SRV,\
                gen_onoff_srv_op, pub, srv, &bt_mesh_gen_onoff_srv_cb)
```

4.2.7.1 bt_mesh_gen_onoff_cli_set

表 4-135 bt_mesh_gen_onoff_cli_set 接口函数说明

信息项	说明
原型	int bt_mesh_gen_onoff_cli_set(struct bt_mesh_model *model, uint16_t addr, uint8_t onoff, struct bt_mesh_transition_params *opt);
功能	客户端设置服务端 On/Off 状态，有确认(即目标节点回复 Status 包)
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint16_t addr 含义解释：节点地址 使用说明：目标节点地址

信息项	说明
	uint8_t onoff 含义解释：OnOff 状态 使用说明：将要设置服务端的目标 OnOff 状态 struct bt_mesh_transition_params *opt 含义解释：渐变参数 使用说明：由当前状态转变为目标状态的参数，为可选字段，具体说明见表 4-136
返回值	int 类型 0：设置成功 非 0：设置失败

表 4-136 bt_mesh_transition_params 结构体成员说明

成员项	说明
uint8_t trans_time	含义解释：渐变时间 使用说明：由当前状态转到目标状态所需要的时间
uint8_t delay	含义解释：延迟时间 使用说明：设备从接收到消息到开始执行变化的时间。延迟时间和渐变时间必须一起设置，如果渐变时间存在，则延迟时间必须存在，反之亦然

4.2.7.2 bt_mesh_gen_onoff_cli_set_unack

表 4-137 bt_mesh_gen_onoff_cli_set_unack 接口函数说明

信息项	说明
原型	int bt_mesh_gen_onoff_cli_set_unack(struct bt_mesh_model *model, uint16_t addr, uint8_t onoff, struct bt_mesh_transition_params *opt);
功能	客户端设置服务端 On/Off 状态，无确认(即目标节点不回复 Status 包)
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint16_t addr 含义解释：节点地址 使用说明：目标节点地址 uint8_t onoff 含义解释：On/Off 状态 使用说明：将要设置服务端的目标 On/Off 状态

信息项	说明
	struct bt_mesh_transition_params *opt 含义解释：渐变参数 使用说明：由当前状态转变为目标状态的参数，为可选字段，具体说明见表 4-136
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.7.3 bt_mesh_gen_onoff_cli_get

表 4-138 bt_mesh_gen_onoff_cli_get 接口函数说明

信息项	说明
原型	int bt_mesh_gen_onoff_cli_get(struct bt_mesh_model *model, uint16_t addr, uint8_t *present, uint8_t *target, uint8_t *remain);
功能	客户端获取服务端 On/Off 状态
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint16_t addr 含义解释：节点地址 使用说明：目标节点地址 uint8_t *present 含义解释：On/Off 当前状态 使用说明：目标节点当前 On/Off 状态 uint8_t *target 含义解释：On/Off 目标状态 使用说明：目标节点的目标 On/Off 状态 uint8_t *remain 含义解释：剩余时间 使用说明：目标节点切换到目标 On/Off 状态还剩余的时间
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.7.4 bt_mesh_gen_onoff_cli_set_net_idx

表 4-139 bt_mesh_gen_onoff_cli_set_net_idx 接口函数说明

信息项	说明
原型	static inline void bt_mesh_gen_onoff_cli_set_net_idx(struct bt_mesh_model *model, uint16_t net_idx);
功能	设置客户端网络 Index
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint16_t net_idx 含义解释：网络 Index 使用说明：指定的 Mesh 网络模型需要设置的网络 Index
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.7.5 bt_mesh_gen_onoff_cli_set_app_idx

表 4-140 bt_mesh_gen_onoff_cli_set_app_idx 接口函数说明

信息项	说明
原型	static inline void bt_mesh_gen_onoff_cli_set_app_idx(struct bt_mesh_model *model, uint16_t app_idx);
功能	设置客户端 App Index
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint16_t app_idx 含义解释：App Index 使用说明：指定的 Mesh 网络模型需要设置的 App Index
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.7.6 bt_mesh_gen_onoff_cli_set_send_rel

表 4-141 bt_mesh_gen_onoff_cli_set_send_rel 接口函数说明

信息项	说明
原型	static inline void bt_mesh_gen_onoff_cli_set_send_rel(struct bt_mesh_model *model, bool send_rel);
功能	设置客户端是否以可靠形式发送消息

信息项	说明
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 bool send_rel 含义解释：是否可靠形式发送标志 使用说明：True 为使用可靠形式发送，即以分段包形式发送，False 为使用正常形式发送
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.7.7 bt_mesh_gen_onoff_cli_set_ttl

表 4-142 bt_mesh_gen_onoff_cli_set_ttl 接口函数说明

信息项	说明
原型	static inline void bt_mesh_gen_onoff_cli_set_ttl(struct bt_mesh_model *model, uint8_t send_ttl);
功能	设置客户端发送消息的默认 TTL
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint8_t send_ttl 含义解释：发送 TTL 使用说明：将要设置到模型中的 TTL 值
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.7.8 bt_mesh_gen_onoff_cli_set_timeout

表 4-143 bt_mesh_gen_onoff_cli_set_timeout 接口函数说明

信息项	说明
原型	static inline void bt_mesh_gen_onoff_cli_set_timeout(struct bt_mesh_model *model, int32_t timeout)
功能	设置客户端发送消息的超时时间
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41

信息项	说明
	int32_t timeout 含义解释：发送超时时间 使用说明：将要设置到模型中的发送超时时间
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.7.9 bt_mesh_gen_onoff_srv_set

表 4-144 bt_mesh_gen_onoff_srv_set 接口函数说明

信息项	说明
原型	int bt_mesh_gen_onoff_srv_set(struct bt_mesh_model *model, uint8_t onoff, struct bt_mesh_transition_params *opt);
功能	服务端设置自身 On/Off 状态，有确认，如对于一个 Mesh 灯来说，需要能通过本地按钮控制自身的开关状态
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41 uint16_t addr 含义解释：节点地址 使用说明：目标节点地址 uint8_t onoff 含义解释：OnOff 状态 使用说明：将要设置服务端的目标 OnOff 状态 struct bt_mesh_transition_params *opt 含义解释：渐变参数 使用说明：由当前状态转变为目标状态的参数，为可选字段，具体说明见表 4-136
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.8 Health 模型接口

4.2.8.1 bt_mesh_health_cli_set

表 4-145 bt_mesh_health_cli_set 接口函数说明

信息项	说明
原型	int bt_mesh_health_cli_set(struct bt_mesh_model *model);

信息项	说明
功能	设置 Health Client 模型实例以用于通信
参数	struct bt_mesh_model *model 含义解释：Mesh 网络模型 使用说明：将要操作的 Mesh 网络模型，结构体相关说明见表 4-41，为组成数据中的 Health Client 模型实例
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.8.2 bt_mesh_health_fault_get

表 4-146 bt_mesh_health_fault_get 接口函数说明

信息项	说明
原型	int bt_mesh_health_fault_get(uint16_t addr, uint16_t app_idx, uint16_t cid, uint8_t *test_id, uint8_t *faults, size_t *fault_count);
功能	获取给定公司 ID 的已注册故障状态
参数	uint16_t addr 含义解释：元素地址 使用说明：目标节点的元素地址 uint16_t app_idx 含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint16_t cid 含义解释：公司 ID 使用说明：公司 ID，用于清除已注册的故障 uint8_t *test_id 含义解释：测试 ID 响应 Buffer 使用说明：获取到的测试 ID 存储在该变量中 uint8_t *faults 含义解释：故障列表响应 Buffer 使用说明：获取到的故障列表存储在该变量中 size_t *fault_count 含义解释：故障数量响应 Buffer 使用说明：获取到的故障数量存储到该变量中
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.8.3 bt_mesh_health_fault_clear

表 4-147 bt_mesh_health_fault_clear 接口函数说明

信息项	说明
原型	int bt_mesh_health_fault_clear(uint16_t addr, uint16_t app_idx, uint16_t cid, uint8_t *test_id, uint8_t *faults, size_t *fault_count);
功能	清除给定公司 ID 的已注册故障
参数	uint16_t addr 含义解释：元素地址 使用说明：目标节点的元素地址 uint16_t app_idx 含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint16_t cid 含义解释：公司 ID 使用说明：公司 ID，用于清除已注册的故障 uint8_t *test_id 含义解释：测试 ID 响应变量 使用说明：获取到的测试 ID 存储在该变量中 uint8_t *faults 含义解释：故障列表响应数组 使用说明：获取到的故障列表存储在该变量中 size_t *fault_count 含义解释：故障数量响应变量 使用说明：获取到的故障数量存储到该变量中
返回值	int 类型 0：清除成功 非 0：清除失败

4.2.8.4 bt_mesh_health_fault_test

表 4-148 bt_mesh_health_fault_test 接口函数说明

信息项	说明
原型	int bt_mesh_health_fault_test(uint16_t addr, uint16_t app_idx, uint16_t cid, uint8_t test_id, uint8_t *faults, size_t *fault_count);
功能	使用给定公司 ID 的自检程序
参数	uint16_t addr 含义解释：元素地址

信息项	说明
	使用说明：目标节点的元素地址 uint16_t app_idx 含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint16_t cid 含义解释：公司 ID 使用说明：公司 ID，用以调用测试 uint8_t *faults 含义解释：故障列表响应数组 使用说明：获取到的故障列表存储在该变量中 size_t *fault_count 含义解释：故障数量响应变量 使用说明：获取到的故障数量存储到该变量中
返回值	int 类型 0：调用成功 非 0：调用失败

4.2.8.5 bt_mesh_health_period_get

表 4-149 bt_mesh_health_period_get 接口函数说明

信息项	说明
原型	int bt_mesh_health_period_get(uint16_t addr, uint16_t app_idx, uint8_t *divisor);
功能	获取目标节点的 Health 快速周期除数。该除数用于记录故障是的发布率，通常，Health Server 将使用配置的发布参数进行发布。当记录了故障后，发布周期将更改为 period/(1<<divisor)
参数	uint16_t addr 含义解释：元素地址 使用说明：目标节点的元素地址 uint16_t app_idx 含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint8_t *divisor 含义解释：快速周期除数响应变量 使用说明：获取到的快速周期除数存放在该变量中
返回值	int 类型 0：获取成功

信息项	说明
	非 0：获取失败

4.2.8.6 bt_mesh_health_period_set

表 4-150 bt_mesh_health_period_set 接口函数说明

信息项	说明
原型	int bt_mesh_health_period_set(uint16_t addr, uint16_t app_idx, uint8_t divisor, uint8_t *updated_divisor);
功能	设置目标节点的 Health 快速周期除数。该除数用于记录故障是的发布率，通常，Health Server 将使用配置的发布参数进行发布。当记录了故障后，发布周期将更改为 period/(1<<divisor)
参数	uint16_t addr 含义解释：元素地址 使用说明：目标节点的元素地址 uint16_t app_idx 含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint8_t *divisor 含义解释：快速周期除数 使用说明：需要设置到目标节点的快速周期除数 uint8_t *updated_divisor 含义解释：快速周期除数响应变量 使用说明：设置完成后获取到的快速周期除数存放在该变量中
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.8.7 bt_mesh_health_attention_get

表 4-151 bt_mesh_health_attention_get 接口函数说明

信息项	说明
原型	int bt_mesh_health_attention_get(uint16_t addr, uint16_t app_idx, uint8_t *attention);
功能	获取目标节点当前 Health 模型的提示时间值
参数	uint16_t addr 含义解释：元素地址 使用说明：目标节点的元素地址 uint16_t app_idx

信息项	说明
	含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint8_t *attention 含义解释：提示时间响应变量 使用说明：获取到的提示时间存放在该变量中
返回值	int 类型 0：获取成功 非 0：获取失败

4.2.8.8 bt_mesh_health_attention_set

表 4-152 bt_mesh_health_attention_set 接口函数说明

信息项	说明
原型	int bt_mesh_health_attention_set(uint16_t addr, uint16_t app_idx, uint8_t attention, uint8_t *updated_attention);
功能	设置目标节点的 Health 提示时间
参数	uint16_t addr 含义解释：元素地址 使用说明：目标节点的元素地址 uint16_t app_idx 含义解释：模型 App 的 Index 使用说明：Health 模型使用的 App uint8_t attention 含义解释：提示时间 使用说明：需要设置到目标节点的提示时间 uint8_t *updated_attention 含义解释：提示时间响应变量 使用说明：设置完成后获取到的提示时间存放在该变量中
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.8.9 bt_mesh_health_cli_timeout_get

表 4-153 bt_mesh_health_cli_timeout_get 接口函数说明

信息项	说明
原型	int32_t bt_mesh_health_cli_timeout_get(void);

信息项	说明
功能	获取目标节点当前 Health 模型的传输超时时间
参数	无
返回值	int32_t 类型 配置的传输超时时间，单位 ms

4.2.8.10 bt_mesh_health_cli_timeout_set

表 4-154 bt_mesh_health_cli_timeout_set 接口函数说明

信息项	说明
原型	void bt_mesh_health_cli_timeout_set(int32_t timeout);
功能	设置目标节点的 Health 传输响应超时时间
参数	int32_t timeout 含义解释：传输超时时间 使用说明：将要设置到目标节点 Health 模型的传输超时时间
返回值	无

4.2.8.11 bt_mesh_fault_update

表 4-155 bt_mesh_fault_update 接口函数说明

信息项	说明
原型	int bt_mesh_fault_update(struct bt_mesh_elem *elem);
功能	通知协议栈给定元素的故障列表状态已更新，如果禁用了定期发布功能，则此接口将提示此元素上的 Health Server 发布当前的故障列表
参数	struct bt_mesh_elem *elem 含义解释：指定元素 使用说明：更新指定元素的故障列表，结构体说明见表 4-46
返回值	int 类型 0：更新成功 非 0：更新失败

4.2.9 Heartbeat 接口

4.2.9.1 bt_mesh_hb_pub_get

表 4-156 bt_mesh_hb_pub_get 接口函数说明

信息项	说明
原型	void bt_mesh_hb_pub_get(struct bt_mesh_hb_pub *get);
功能	获取目标节点当前的 Heartbeat 发布参数

信息项	说明
参数	struct bt_mesh_hb_pub *get 含义解释：Heartbeat 发布参数响应结构体 使用说明：获取到的目标节点的 Heartbeat 发布参数存储在该变量中，结构体定义见表 4-157
返回值	无

表 4-157 bt_mesh_hb_pub 结构体成员说明

成员项	说明
uint16_t dst	含义解释：目的地址 使用说明：Heartbeat 消息发送的目的地址
uint16_t count	含义解释：Heartbeat 计数 使用说明：此参数表示剩余要发送的心跳消息数
uint8_t ttl	含义解释：发布消息 TTL 值 使用说明：发布消息所使用的默认的 TTL 值
uint16_t feat	含义解释：触发 Heartbeat 发布的功能的位图 使用说明：合理值的值有：BT_MESH_FEAT_RELAY、BT_MESH_FEAT_PROXY、BT_MESH_FEAT_FRIEND 和 BT_MESH_FEAT_LOW_POWER
uint16_t net_idx	含义解释：子网 Index 使用说明：要发布到的子网索引
uint32_t period	含义解释：Heartbeat 发布传输间隔 使用说明：单位为秒

4.2.9.2 bt_mesh_hb_sub_get

表 4-158 bt_mesh_hb_sub_get 接口函数说明

信息项	说明
原型	void bt_mesh_hb_sub_get(struct bt_mesh_hb_sub *get);
功能	获取目标节点当前的 Heartbeat 订阅参数
参数	struct bt_mesh_hb_sub *get 含义解释：Heartbeat 订阅参数响应结构体 使用说明：获取到的目标节点的 Heartbeat 订阅参数存储在该变量中，结构体定义见表 4-159
返回值	无

表 4-159 bt_mesh_hb_sub 结构体成员说明

成员项	说明
uint32_t period	含义解释：订阅周期 使用说明：实际订阅周期，单位为秒
uint32_t remaining	含义解释：剩余订阅时间 使用说明：单位为秒
uint16_t src	含义解释：源地址 使用说明：接收到 Heartbeat 消息的源地址
uint16_t dst	含义解释：目的地址 使用说明：接收 Heartbeat 消息的目的地址
uint16_t count	含义解释：Heartbeat 订阅接收计数 使用说明：到目前为止收到的 Heartbeat 消息数
uint8_t min_hops	含义解释：接收到的消息中的最小 hop 使用说明：从发布节点到订阅节点的最短路径，与相邻节点的 hop count=1
uint8_t max_hops	含义解释：接收到的消息中的最大 hop 使用说明：从发布节点到订阅节点的最长路径，与相邻节点的 hop count=1

4.2.10 Mesh Common 接口

Mesh Common 接口主要用于应用自己设计的模型的开发。

4.2.10.1 model_cli_internal_get_msg_ctx

表 4-160 model_cli_internal_get_msg_ctx 接口函数说明

信息项	说明
原型	void model_cli_internal_get_msg_ctx(struct model_cli_internal *cli, struct bt_mesh_msg_ctx *ctx);
功能	从 Client Model 获取发送消息的有用参数
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 struct bt_mesh_msg_ctx *ctx 含义解释：Mesh 网络消息

信息项	说明
	使用说明：包含密钥、TTL 等，相关结构体定义见表 4-42
返回值	无

表 4-161 model_cli_internal 结构体成员说明

成员项	说明
uint16_t net_idx	含义解释：网络 Index 使用说明：在其上发送消息的子网的 NetKey 索引
uint16_t app_idx	含义解释：App Index 使用说明：App Key 索引，用于加密消息
bool send_rel	含义解释：是否可靠发送标志 使用说明：True：使用分段形式进行发送，False：正常形式发送
uint8_t send_ttl	含义解释：TTL 值 使用说明：默认使用 BT_MESH_TTL_DEFAULT

4.2.10.2 model_cli_internal_set_net_idx

表 4-162 model_cli_internal_set_net_idx 接口函数说明

信息项	说明
原型	static inline void model_cli_internal_set_net_idx(struct model_cli_internal *cli, uint16_t net_idx);
功能	设置 Client Model 的网络 Index
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 uint16_t net_idx 含义说明：网络 Index 使用说明：将要设置到模型中的网络 Index
返回值	无

4.2.10.3 model_cli_internal_set_app_idx

表 4-163 model_cli_internal_set_app_idx 接口函数说明

信息项	说明
原型	static inline void model_cli_internal_set_app_idx(struct model_cli_internal *cli,

信息项	说明
	uint16_t app_idx);
功能	设置 Client Model 的 App Index
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 uint16_t app_idx 含义说明：App Index 使用说明：将要设置到模型中的 App Index
返回值	无

4.2.10.4 model_cli_internal_set_send_rel

表 4-164 model_cli_internal_set_send_rel 接口函数说明

信息项	说明
原型	static inline void model_cli_internal_set_send_rel(struct model_cli_internal *cli, bool send_rel);
功能	设置 Client Model 的可靠发送标志
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 bool send_rel 含义说明：可靠发送标志 使用说明：True 时使用分段包进行可靠发送，False 时使用正常方式发送
返回值	无

4.2.10.5 model_cli_internal_set_ttl

表 4-165 model_cli_internal_set_ttl 接口函数说明

信息项	说明
原型	static inline void model_cli_internal_set_ttl(struct model_cli_internal *cli, uint8_t send_ttl);
功能	设置 Client Model 的 TTL
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 uint8_t send_ttl 含义说明：需要设置的 TTL 值 使用说明：将要设置到模型中的 TTL 值

信息项	说明
返回值	无

4.2.10.6 model_cli_internal_release

表 4-166 model_cli_internal_release 接口函数说明

信息项	说明
原型	void model_cli_internal_release(struct model_cli_internal *cli);
功能	释放 Client Model 的空间
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161
返回值	无

4.2.10.7 model_cli_internal_param_get

表 4-167 model_cli_internal_param_get 接口函数说明

信息项	说明
原型	int model_cli_internal_param_get(struct model_cli_internal *cli, void **param, uint32_t op);
功能	获取 Client Model 参数
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 void **param 含义解释：参数结构体 使用说明：获取到的参数存储在该变量中，与 op 搭配使用 uint32_t op 含义解释：opcode 使用说明：不同 opcode 对应不同的结构体，将对应 opcode 的结构体赋值给变量 param
返回值	无

4.2.10.8 model_cli_internal_prepare

表 4-168 model_cli_internal_prepare 接口函数说明

信息项	说明
原型	int model_cli_internal_prepare(struct model_cli_internal *cli, void *param, uint32_t op);

信息项	说明
功能	给指定 opcode 分配 param，在收到对应包后将内容放到变量 param 中
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161 void **param 含义解释：参数结构体 使用说明：获取到的参数存储在该变量中，与 op 搭配使用 uint32_t op 含义解释：opcode 使用说明：不同 opcode 对应不同的结构体，将对应 opcode 的结构体赋值给变量 param
返回值	无

4.2.10.9 model_cli_internal_reset

表 4-169 model_cli_internal_reset 接口函数说明

信息项	说明
原型	void model_cli_internal_reset(struct model_cli_internal *cli);
功能	重置模型内部参数
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161
返回值	无

4.2.10.10 model_cli_internal_wait

表 4-170 model_cli_internal_wait 接口函数说明

信息项	说明
原型	int model_cli_internal_wait(struct model_cli_internal *cli);
功能	等待模型获取到响应
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161
返回值	无

4.2.10.11 model_cli_internal_get_timeout

表 4-171 model_cli_internal_get_timeout 接口函数说明

信息项	说明
原型	static inline int32_t model_cli_internal_get_timeout(struct model_cli_internal *cli);
功能	获取模型等待响应超时时间
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161
返回值	int32_t 类型 模型的超时时间

4.2.10.12 model_cli_internal_init

表 4-172 model_cli_internal_init 接口函数说明

信息项	说明
原型	int model_cli_internal_init(struct model_cli_internal *cli);
功能	初始化模型内容参数
参数	struct model_cli_internal *cli 含义解释：模型内部参数 使用说明：用于跟踪消息响应的内部参数，结构体说明见表 4-161
返回值	int 类型 0：初始化成功 非 0：初始化失败

4.2.11 Proxy 接口

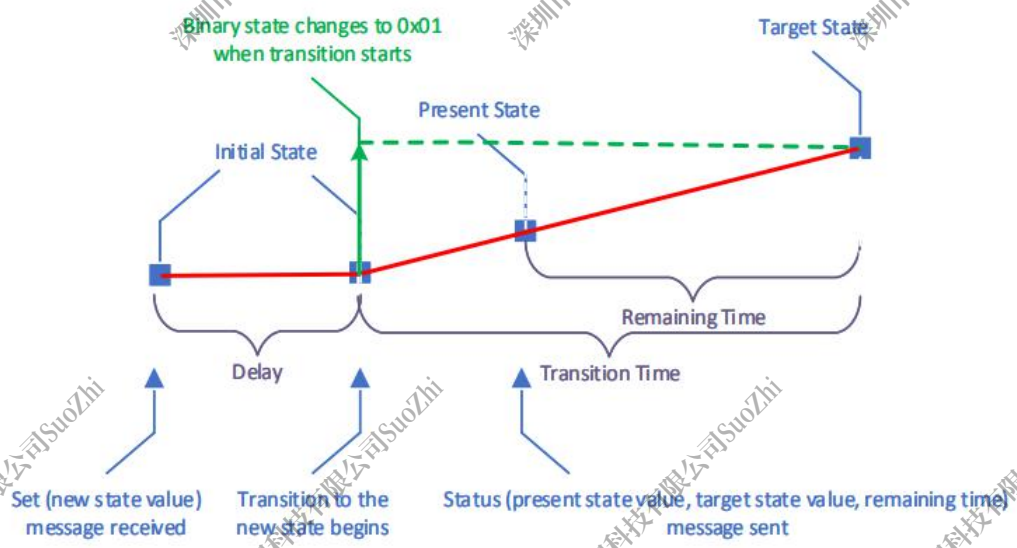
4.2.11.1 bt_mesh_proxy_identity_enable

表 4-173 bt_mesh_proxy_identity_enable 接口函数说明

信息项	说明
原型	int bt_mesh_proxy_identity_enable(void);
功能	使能节点身份启用广播，此接口要求启用 GATT 代理功能。一旦被调用，每个子网将在接下来的 60 秒内使用节点标识进行广播
参数	无
返回值	int 类型 0：使能成功 非 0：使能失败

4.2.12 Transition 接口

图 4-1 Transition 模型示意图



所谓 Transition Time(渐变时间)指的是设备从一个状态变换到另一个状态所需要的时间。该字段长度为 1 个字节，低 6bit 表示渐变时间的具体数值，即渐变步数；高 2bit 表示渐变时间的单位为渐变步长。通过该模型的操作接口，可设置在渐变时间范围内产生若干次的渐变超时。在每次产生的渐变超时，可要求 Server 报告当前的状态和 Remaining Time，如设置当前渐变步数为 7，则将当前渐变时间均分为 7 段，会产生 7 次超时。

4.2.12.1 transition_init

表 4-174 transition_init 接口函数说明

信息项	说明
原型	int transition_init(struct transition *tt, transition_cb_t cb, void *arg);
功能	渐变时间模型初始化
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数 transition_cb_t cb 含义解释：渐变时间模型回调函数 使用说明：用户实现，在达到渐变时间超时时通知用户 void *arg 含义解释：回调函数参数 使用说明：与回调函数联合使用，作为参数传入
返回值	int 类型 0：初始化成功 非 0：初始化失败

4.2.12.2 transition_deinit

表 4-175 transition_deinit 接口函数说明

信息项	说明
原型	void transition_deinit(struct transition *tt);
功能	渐变时间模型关闭
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数
返回值	无

4.2.12.3 transition_set_cb

表 4-176 transition_set_cb 接口函数说明

信息项	说明
原型	static inline void transition_set_cb(struct transition *tt, transition_cb_t cb);
功能	设置渐变时间模型回调函数
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数 Transition_cb_t cb 含义解释：回调函数 使用说明：赋值给 transition 参数的回调函数，在渐变超时时使用
返回值	无

4.2.12.4 transition_set_steps

表 4-177 transition_set_steps 接口函数说明

信息项	说明
原型	static inline void transition_set_steps(struct transition *tt, uint16_t new_steps);
功能	设置渐变时间模型的步长
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数 uint16_t new_steps 含义解释：渐变时间模型步长 使用说明：模型设置给自身的步长，优先级较 client 设置高，取值为非 0 时有效，为 0 时步长使用 client 设置值
返回值	无

4.2.12.5 transition_set_steps

表 4-177 transition_set_steps 接口函数说明

信息项	说明
原型	static inline void transition_set_steps(struct transition *tt, uint16_t new_steps);
功能	设置渐变时间模型的步长，针对定制化使用，例如 Client 端设置为 7，通过此接口设置为 5，则会优先使用 5
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数 uint16_t new_steps 含义解释：渐变时间模型步长 使用说明：模型设置给自身的步长，优先级较 client 设置高，取值为非 0 时有效，为 0 时步长使用 client 设置值
返回值	无

4.2.12.6 transition_set_steps_clear

表 4-179 transition_set_steps_clear 接口函数说明

信息项	说明
原型	static inline void transition_set_steps_clear(struct transition *tt)
功能	清除渐变时间模型自身设置的步长，即采用 client 设置的步长值
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数
返回值	无

4.2.12.7 transition_get_remain_time

表 4-180 transition_get_remain_time 接口函数说明

信息项	说明
原型	void transition_get_remain_time(struct transition *tt, uint8_t *rt);
功能	获取渐变时间模型剩余渐变时间
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数 uint8_t *rt 含义解释：渐变模型剩余转变时间

信息项	说明
	使用说明：获取到的渐变模型剩余转变时间存储在该变量中，获取到的值一般需通过 transition_time_to_ms()接口转换为实际时间
返回值	无

4.2.12.8 transition_time_to_ms

表 4-181 transition_time_to_ms 接口函数说明

信息项	说明
原型	uint32_t transition_time_to_ms(uint8_t trans_time);
功能	渐变时间转换为实际时间
参数	uint8_t trans_time 含义解释：渐变时间 使用说明：获取到的渐变时间是一个组合参数，需要通过转换，获取到实际的剩余时间
返回值	uint32_t 类型 转换后获取到的实际的渐变时间模型的转换剩余时间，单位毫秒

4.2.12.9 transition_ms_to_time

表 4-182 transition_ms_to_time 接口函数说明

信息项	说明
原型	uint8_t transition_ms_to_time(uint32_t ms);
功能	实际时间转换为渐变时间
参数	uint32_t ms 含义解释：实际时间 使用说明：实际的渐变时间模型的剩余转换时间，单位毫秒，通过该接口转换为渐变时间
返回值	uint8_t 类型 转换后的渐变时间模型的渐变时间

4.2.12.10 transition_is_instantaneous

表 4-183 transition_is_instantaneous 接口函数说明

信息项	说明
原型	static inline bool transition_is_instantaneous(struct transition *tt, uint8_t trans_time);
功能	判断当前转变是否为瞬间转变
参数	struct transition *tt 含义解释：模型内 transition 参数

信息项	说明
	使用说明：模型内 transition 参数 uint8_t trans_time 含义解释：渐变时间 使用说明：Client 端设置的渐变时间，通过该时间来判断当前次转换是否需要渐变
返回值	bool 类型 True：非渐变型，不需要渐变，直接进行转换 False：有渐变时间，需要渐变

4.2.12.11 transition_is_started

表 4-184 transition_is_started 接口函数说明

信息项	说明
原型	static inline bool transition_is_started(struct transition *tt);
功能	判断当前渐变时间模型是否处于渐变状态转换中
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数
返回值	bool 类型 True：处于渐变状态转换中 False：未处于渐变状态转换中

4.2.12.12 transition_prepare

表 4-185 transition_prepare 接口函数说明

信息项	说明
原型	int transition_prepare(struct transition *tt, uint8_t trans_time, uint8_t delay);
功能	根据 Client 端参数设置渐变时间模型
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数 uint8_t trans_time 含义解释：渐变时间 使用说明：Client 端设置渐变时间参数 uint8_t delay 含义解释：延迟时间 使用说明：Client 端设置延迟时间参数

信息项	说明
返回值	int 类型 0：设置成功 非 0：设置失败

4.2.12.13 transition_stop

表 4-186 transition_stop 接口函数说明

信息项	说明
原型	void transition_stop(struct transition *tt);
功能	渐变时间模型停止计时
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数
返回值	无

4.2.12.14 transition_start

表 4-187 transition_start 接口函数说明

信息项	说明
原型	int transition_start(struct transition *tt);
功能	渐变时间模型启动计时
参数	struct transition *tt 含义解释：模型内 transition 参数 使用说明：模型内 transition 参数
返回值	无

5 示例说明

本章提供了一个示例，对 Mesh 配网及 On/Off 控制进行演示说明。

5.1 Mesh 配网及 On/Off 控制示例

5.1.1 示例简介

本示例通过延时 Mesh 配网及 On/Off 控制场景，简要介绍了 Mesh 配网接口的基本使用方法。

5.1.2 获取方法

通过 mesh_demo 工程和 mesh_provisioner_demo 工程可以完成 Mesh 配网过程演示，mesh_demo 工程和 mesh_provisioner_demo 工程位于 XR806 SDK 的/project/demo/bluetooth 目录。

说明

XR806 SDK 可在以下 Github 仓库获取：https://github.com/XradioTech/xr806_sdk.git

5.1.3 准备工作

示例运行的硬件准备如下：

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 Uart0 插针，用于镜像的烧录和串口的输入和输出。
3. PC 机：用于镜像烧录和串口的输入和输出。

5.1.4 操作步骤

1. 编译 mesh_demo 工程，获取到 Node 对应 image。
2. 编译 mesh_provisioner_demo 工程，获取到 Provisioner 对应的 image。
3. 将 image 分别烧录到两个评估板中，完成烧写后，复位即可，示例代码会自动运行。

5.1.5 代码解析

1) Node 端代码解析

1. 初始化

```
static void bt_ready(int err)
{
	struct bt_le_oob oob;
	if (err) {
		printf("bt_ready : Bluetooth init failed (err %d)\n", err);
		return;
	}
	/*Mesh 协议栈初始化*/
	err = bt_mesh_init(&prov, &comp);
	if (err) {
		printf("bt_ready : Initializing mesh failed (err %d)\n", err);
		return;
	}
	/*从 flash 恢复数据*/
	if (IS_ENABLED(CONFIG_SETTINGS)) {
		settings_load();
	}
	/*  根据 BLE 地址设置设备 UUID */
	if (bt_le_oob_get_local(BT_ID_DEFAULT, &oob)) {
		printf("bt_ready : Identity Address unavailable\n");
	} else {
		memcpy(dev_uuid, oob.addr.a.val, 6);
	}
	/*判断设备是否已经配网，若未配网则进行 Mesh 广播*/
	if (bt_mesh_is_provisioned()) {
		printf("Mesh network restored from flash\n");
	} else {
		bt_mesh_prov_enable(BT_MESH_PROV_GATT | BT_MESH_PROV_ADV);
	}
	printf("bt_ready : Mesh initialized\n");
}

int main(void)
{
	int err;
	/*SDK 初始化*/
	platform_init();
	/*BLE Controller 初始化*/
	err = bt_ctrl_enable();
	if (err) {
		printf("Bluetooth controller init failed (err %d)\n", err);
```

```
        return 0;
    }
    /* BLE 协议栈初始化 */
    err = bt_enable(bt_ready);
    if (err) {
        printf("main : Bluetooth init failed (err %d)\n", err);
    }
    return 0;
}
```

2. Mesh 配网结构体

Mesh 配网结构体主要用于配网，结构体中参数和回调函数由应用自行实现，在 Mesh 协议栈初始化时注册进协议栈中。在配网过程中会由协议栈在不同阶段对参数或是回调函数进行调用。

```
/*配网完成回调函数，以此通知应用配网已经完成，Node 已被配入网中*/
static void prov_complete(uint16_t net_idx, uint16_t addr)
{
#if (CONFIG_BT_MESH_FRIEND)
#if (MESH_FRIEND_FUNC_ENABLE)
    int err;
    uint8_t frnd;
#endif
#endif
    primary_addr = addr;
    primary_net_idx = net_idx;
    printf("***** Provisioning - complete\n");
    printf("net_idx 0x%04x\naddr 0x%04x\n",    net_idx, addr);
#if (CONFIG_BT_MESH_FRIEND)
#if (MESH_FRIEND_FUNC_ENABLE)
    /*  在启用 Friend 功能情况下设置本设备的 Friend 功能启动 */
    err = bt_mesh_cfg_friend_set(primary_net_idx, primary_addr, 1, &frnd);
    if (err) {
        printf("Unable to send Friend Get/Set (err %d)\n", err);
    } else {
        printf("Friend is set to 0x%02x\n", frnd);
    }
#endif
#endif
}
/*节点 Reset 回调函数，以此通知节点设备已被 Reset，已不再 Mesh 网内*/
static void prov_reset(void)
{
```

```
    /*再次启动 Mesh 广播*/
    bt_mesh_prov_enable(BT_MESH_PROV_ADV | BT_MESH_PROV_GATT);
}
/*配网结构体声明和填充*/
static const struct bt_mesh_prov prov = {
    .uuid = dev_uuid,
    .oob_info = (BT_MESH_PROV_OOB_NUMBER |BT_MESH_PROV_OOB_STRING),
    .output_size = 6,
    .output_actions = (BT_MESH_DISPLAY_NUMBER | BT_MESH_DISPLAY_STRING),
    .output_number = output_number,
    .output_string = output_string,
    .complete = prov_complete,
    .reset = prov_reset,
};
```

3. Mesh 元素模型

Mesh Demo 中声明的元素模型共有 1 个元素，元素包含有 6 个模型，分别为基础必备的 4 个模型：cfg server、cfg client、health server 以及 health client；2 个通用模型：gen onoff server 和 gen onoff client。以此组成了 Demo 中使用的 Mesh 节点。

```
/*cfg client 模型声明*/
static struct bt_mesh_cfg_cli cfg_cli = {
};
/*On/Off 设置回调函数，以此通知应用收到了其他 On/Off client 的设置请求*/
static void app_onoff_srv_set_cb(const struct bt_mesh_model *model, uint8_t onoff,
                 uint8_t target_onoff, const struct bt_mesh_transition_status *opt)
{
    g_onoff_value = onoff;
    printf("[app] onoff set(%d)", onoff);
    if (opt) {
        printf("target onoff(%d), total_steps(%d), steps(%d)",
                target_onoff, opt->total_steps, opt->present_steps);
    }
    printf("\n");
}
/*On/Off 读取回调函数，以此通知应用收到了其他 On/Off client 的读取请求*/
static void app_onoff_srv_get_cb(const struct bt_mesh_model *model,
                                 uint8_t *onoff)
{
    printf("[app] onoff get(%d)\n", g_onoff_value);
    *onoff = g_onoff_value;
}
```

```
/*On/Off server 回调函数集，由应用实现，注册到 Mesh 协议栈中*/
static struct bt_mesh_gen_onoff_srv onoff_srv_user_data = {
	.set_cb = app_onoff_srv_set_cb,
	.get_cb = app_onoff_srv_get_cb,
};
/*On/Off 状态获取回调函数，以此通知应用获取到了指定 Server 的 On/Off 状态*/
void app_onoff_cli_status_cb(const struct bt_mesh_model *model, uint8_t present,
			     uint8_t target, const struct bt_mesh_transition_remain_time *opt)
{
	printf("present onoff(%d), target onoff(%d)", present, target);
	if (opt)
		printf(", remain time(%d)", opt->remain_time);
	printf("\n");
}
/*On/Off client 回调函数集，由应用实现，注册到 Mesh 协议栈中*/
static struct bt_mesh_gen_onoff_cli onoff_cli_user_data = {
	.status_cb = app_onoff_cli_status_cb
};
/*节点的根元素中的根模型声明*/
static struct bt_mesh_model root_models[] = {
	BT_MESH_MODEL_CFG_SRV,
	BT_MESH_MODEL_CFG_CLI(&cfg_cli),
	BT_MESH_MODEL_HEALTH_SRV(&health_srv, &health_pub),
	BT_MESH_MODEL_HEALTH_CLI(&health_cli),
	BT_MESH_MODEL_GEN_ONOFF_SRV(&onoff_srv_user_data, &gen_onoff_pub_srv),
	BT_MESH_MODEL_GEN_ONOFF_CLI(&onoff_cli_user_data, &gen_onoff_pub_cli),
};
/*节点的根元素声明*/
static struct bt_mesh_elem elements[] = {
	BT_MESH_ELEM(0, root_models, BT_MESH_MODEL_NONE),
};
/*节点的组成元素声明*/
static const struct bt_mesh_comp comp = {
	.cid = BT_COMP_ID_LF,
	.elem = elements,
	.elem_count = ARRAY_SIZE(elements),
};
```

2) Provisioner 端代码解析

1. 初始化

```
static void bt_ready(int err)
{
	uint8_t net_key[16], dev_key[16];
	if (err) {
		printf("bt_ready : Bluetooth init failed (err %d)\n", err);
		return;
	}
	/*Mesh 协议栈初始化*/
	err = bt_mesh_init(&prov, &comp);
	if (err) {
		printf("bt_ready : Initializing mesh failed (err %d)\n", err);
		return;
	}
	/*从 flash 恢复数据*/
	if (IS_ENABLED(CONFIG_SETTINGS)) {
		settings_load();
	}
	/*创建 CDB*/
	bt_rand(net_key, 16);
	err = bt_mesh_cdb_create(net_key);
	if (err == -EALREADY) {
		printf("Using stored CDB\n");
	} else if (err) {
		printf("Failed to create CDB (err %d)\n", err);
		return;
	} else {
		printf("Created CDB\n");
		/*为 Provisioner 添加 App Key*/
		setup_cdb();
	}
	/*将 Provisioner 自身配入网中*/
	bt_rand(dev_key, 16);
	err = bt_mesh_provision(net_key, BT_MESH_NET_PRIMARY, 0, 0, self_addr, dev_key);
	if (err == -EALREADY) {
		printf("Using stored settings\n");
	} else if (err) {
		printf("Provisioning local failed (err %d)\n", err);
		return;
	} else {
		/*将 Provisioner 添加到 CDB 中*/
		cdb_local_node_add(dev_key);
```

```
        printf("Provisioning local completed\n");
    }
    /*设置命令响应超时时间为 8s*/
    bt_mesh_cfg_cli_timeout_set(8000);
    printf("bt_ready : Mesh initialized\n");
}
int main(void)
{
    int err;
    /*SDK 初始化*/
    platform_init();
    /*BLE Controller 初始化*/
    err = bt_ctrl_enable();
    if (err) {
        printf("Bluetooth controller init failed (err %d)\n", err);
        return 0;
    }
    /*BLE 协议栈初始化*/
    err = bt_enable(NULL);
    if (err) {
        printf("main : Bluetooth init failed (err %d)\n", err);
        return 0;
    }
    bt_ready(err);
    /*Mesh 配网任务初始化*/
    if (prov_task_init() != 0) {
        printf("prov task init fail\n");
        return -1;
    }
    /*启动 Mesh 配网任务*/
    prov_task_start();
    return 0;
}
```

2. Mesh 配网结构体

Mesh 配网结构体主要用于配网，结构体中参数和回调函数由应用自行实现，在 Mesh 协议栈初始化时注册进协议栈中。在配网过程中会由协议栈在不同阶段对参数或是回调函数进行调用。Provisioner 的配网结构体应用所关注的功能与 Node 不同。

```
/*收到 Mesh 未配网 Beacon 回调函数，以通知应用收到了未配网 Beacon 包，应用进行区别是否需要对该设备进行配网*/
static void unprovisioned_beacon(uint8_t uuid[16],
                 bt_mesh_prov_oob_info_t oob_info,
                 uint32_t *uri_hash)
{
    if (!node_provising && memcmp(uuid, node_uuid_match, sizeof(node_uuid_match)) == 0) {
        memcpy(node_uuid, uuid, 16);
        OS_SemaphoreRelease(&sem_unprov_beacon);
    }
}
/*配网完成回调函数，以此通知应用配网已完成*/
static void node_added(uint16_t net_idx, uint8_t uuid[16], uint16_t addr, uint8_t num_elem)
{
    node_provising_addr = addr;
    OS_SemaphoreRelease(&sem_node_added);
}
/*配网结构体声明和填充*/
static const struct bt_mesh_prov prov = {
    .uuid = dev_uuid,
    .unprovisioned_beacon = unprovisioned_beacon,
    .node_added = node_added,
};
```

3. Mesh 元素模型

同 5.1.5 节 Node 端 Mesh 元素模型

4. 运行配网任务

```
/*配网任务初始化*/
static int prov_task_init(void)
{
    OS_SemaphoreCreate(&sem_unprov_beacon, 0, 1);
    OS_SemaphoreCreate(&sem_node_added, 0, 1);
    /*从 CDB 中读取已配网节点*/
    cdb_subnet_restore();
    /*  创建 On/Off 任务  */
    if (onoff_task_init() != 0) {
        OS_SemaphoreDelete(&sem_unprov_beacon);
        OS_SemaphoreDelete(&sem_node_added);
        return -1;
```

```
    }
    return 0;
}
/*配网任务启动*/
static void prov_task_start(void)
{
    int err, cnt;
    char uuid_hex_str[32 + 1];
    while (1) {
        /*遍历各个 CDB 中的几点，并进行配置检查*/
        bt_mesh_cdb_node_foreach(check_unconfigured, NULL);
        /*获取 CDB 中 node 的总个数*/
        cnt = cdb_node_cnt_get();
        if (!cnt || (cnt && (cnt - 1) >= PROVISION_NODE_CNT_MAX))
            break;
        printf("Waiting for unprovisioned beacon...\n");
        /*等待收到未配网 Beacon*/
        err = OS_SemaphoreWait(&sem_unprov_beacon, K_SECONDS(10));
        if (err == OS_E_TIMEOUT) {
            continue;
        }
        node_provising = true;
        bin2hex(node_uuid, 16, uuid_hex_str, sizeof(uuid_hex_str));
        printf("Provisioning for node %s\n", uuid_hex_str);
        /*对符合条件的设备进行配网*/
        err = bt_mesh_provision_adv(node_uuid, net_idx, 0, 0);
        if (err < 0) {
            printf("Provisioning failed (err %d)\n", err);
            node_provising = false;
            continue;
        }
        printf("Waiting for node to be added...\n");
       /*等待配网完成*/
        err = OS_SemaphoreWait(&sem_node_added, K_SECONDS(50));
        if (err == OS_E_TIMEOUT) {
            printf("Timeout waiting for node to be added\n");
            node_provising = false;
            continue;
        }
        printf("Added node 0x%04x\n", node_provising_addr);
        node_addr[cnt - 1] = node_provising_addr;
        node_provising = false;
```

```
    }
}
```

5. 对配入 Mesh 网内的节点进行配置

```
/*对 Provisioner 自身进行配置*/
static void configure_self(struct bt_mesh_cdb_node *self)
{
    struct bt_mesh_cdb_app_key *key;
    uint8_t status = 0;
    int err, i;
    printf("Configuring self 0x%04x...\n", self->addr);
    /*从 CDB 中获取 App Key*/
    key = bt_mesh_cdb_app_key_get(app_idx);
    if (key == NULL) {
        printf("No app-key 0x%04x\n", app_idx);
        return;
    }
    /*若未获取到 App Key，则进行添加*/
    err = bt_mesh_cfg_app_key_add(self->net_idx, self->addr, self->net_idx,
                                  app_idx, key->keys[0].app_key, &status);
    if (err || status) {
        printf("Failed to add app-key (err %d, status %d)\n", err,
                status);
        return;
    }
    /*遍历所有的 Model，对部分 Model 进行 App Key 的绑定*/
    for (i = 0; i < ARRAY_SIZE(root_models); i++) {
        if (root_models[i].id == BT_MESH_MODEL_ID_CFG_CLI ||
            root_models[i].id == BT_MESH_MODEL_ID_CFG_SRV) {
            continue;
        }
        printf("Binding AppKey to model 0x%03x:%04x\n",
                    self->addr, root_models[i].id);
        /*将模型与 App Key 进行绑定*/
        err = bt_mesh_cfg_mod_app_bind(self->net_idx, self->addr, self->addr,
                                       app_idx, root_models[i].id,
                                       &status);
        if (err || status) {
            printf("Failed to bind app-key (err %d, status %d)\n", err,
                    status);
            return;
```

```
        }
    }
    atomic_set_bit(self->flags, BT_MESH_CDB_NODE_CONFIGURED);
    if (IS_ENABLED(CONFIG_BT_SETTINGS)) {
        /*将节点信息进行存储*/
        bt_mesh_cdb_node_store(self);
    }
    printf("Configuration complete\n");
}
/*对 Node 进行配置*/
static void configure_node(struct bt_mesh_cdb_node *node)
{
    NET_BUF_SIMPLE_DEFINE(buf, BT_MESH_RX_SDU_MAX);
    struct bt_mesh_cdb_app_key *key;
    uint8_t status;
    int err, elem_addr;
    printf("Configuring node 0x%04x...\n", node->addr);
    /*从 CDB 中获取 App Key*/
    key = bt_mesh_cdb_app_key_get(app_idx);
    if (key == NULL) {
        printf("No app-key 0x%04x\n", app_idx);
        return;
    }
    /*若未获取到 App Key，则进行添加*/
    err = bt_mesh_cfg_app_key_add(net_idx, node->addr, net_idx, app_idx,
                                  key->keys[0].app_key, &status);
    if (err || status) {
        printf("Failed to add app-key (err %d status %d)\n", err, status);
        return;
    }
    /*获取 Node 的组成数据*/
    err = bt_mesh_cfg_comp_data_get(net_idx, node->addr, 0, &status, &buf);
    if (err || status) {
        printf("Failed to get Composition data (err %d, status: %d)\n",
                    err, status);
        return;
    }
    ...
    elem_addr = node->addr;
    while (buf.len > 4) {
        uint8_t sig, vnd;
        uint16_t loc;
```

```
int i;
loc = net_buf_simple_pull_le16(&buf);
sig = net_buf_simple_pull_u8(&buf);
vnd = net_buf_simple_pull_u8(&buf);
printf("\tElement loc 0x%04x\n", loc);
if (buf.len < ((sig * 2U) + (vnd * 4U))) {
    printf("\t\t...truncated data!\n");
    break;
}
if (sig) {
    printf("\t\tSIG Models:\n");
} else {
    printf("\t\tNo SIG Models\n");
}
for (i = 0; i < sig; i++) {
    uint16_t mod_id = net_buf_simple_pull_le16(&buf);
    if (mod_id == BT_MESH_MODEL_ID_CFG_CLI ||
        mod_id == BT_MESH_MODEL_ID_CFG_SRV) {
        continue;
    }
    printf("Binding AppKey to model 0x%03x:%04x\n",
           elem_addr, mod_id);
    /*将获取到的节点数据中的一些模型与 App Key 进行绑定*/
    err = bt_mesh_cfg_mod_app_bind(net_idx, node->addr,
                                   elem_addr, app_idx, mod_id,
                                   &status);
    if (err || status) {
        printf("Failed (err: %d, status: %d)\n", err,
               status);
    }
}
if (vnd) {
    printf("\t\tVendor Models:\n");
} else {
    printf("\t\tNo Vendor Models\n");
}
for (i = 0; i < vnd; i++) {
    uint16_t cid = net_buf_simple_pull_le16(&buf);
    uint16_t mod_id = net_buf_simple_pull_le16(&buf);
    printf("Binding AppKey to model 0x%03x:%04x:%04x\n",
           elem_addr, cid, mod_id);
    /*若存在厂商模型，则将厂商模型与 App Key 进行绑定*/
    err = bt_mesh_cfg_mod_app_bind_vnd(net_idx, node->addr,
```

```
                                        elem_addr, app_idx,
                                        mod_id, cid,
                                        &status);
                if (err || status) {
                    printf("Failed (err: %d, status: %d)\n", err,
                           status);
                }
            }
            elem_addr++;
        }
        atomic_set_bit(node->flags, BT_MESH_CDB_NODE_CONFIGURED);
        if (IS_ENABLED(CONFIG_BT_SETTINGS)) {
            /*将节点信息进行存储*/
            bt_mesh_cdb_node_store(node);
        }
        printf("Configuration complete\n");
}
/*对 CDB 中的节点进行遍历*/
static uint8_t check_unconfigured(struct bt_mesh_cdb_node *node, void *data)
{
        if (!atomic_test_bit(node->flags, BT_MESH_CDB_NODE_CONFIGURED)) {
            if (node->addr == self_addr) {
                configure_self(node);
            } else {
                configure_node(node);
            }
        }
        return BT_MESH_CDB_ITER_CONTINUE;
}
```

6. On/Off 操作

在上述操作全部完成后，Node 端与 Provisioner 端均已完成配网，且对应 On/Off 模型也已进行了 App Key 的绑定，因此可以直接由 Provisioner 端对 Node 端的 Gen On/Off Server 进行操作。

```
/*On/Off 任务*/
static void onoff_task(void *arg)
{
        while (1) {
            int err, i;
            uint32_t keys_cnt, node_cnt;
            uint16_t keys[16];
            static uint8_t onoff_value = 0;
```

```
            keys_cnt = ARRAY_SIZE(keys);
            node_cnt = cdb_node_cnt_get();
            onoff_value = onoff_value ? 0 : 1;
            for (i = 0; i < (node_cnt - 1) && node_addr[i] != 0; i++) {
                /*获取对应节点的 net key*/
                err = bt_mesh_cfg_net_key_get(net_idx, node_addr[i], keys, &keys_cnt);
                if (err)
                    continue;
                /*对节点进行 On/Off 操作*/
                bt_mesh_gen_onoff_cli_set(&root_models[5], node_addr[i], onoff_value, NULL);
            }
            OS_MSleep(ONOFF_CHANGE_PERIOD);
        }
        OS_ThreadDelete(&onoff_task_thread);
    }
    /*On/Off 操作初始化*/
    static int onoff_task_init(void)
    {
        /*创建 On/Off 线程*/
        if (OS_ThreadCreate(&onoff_task_thread,
                            "onoff_task",
                            onoff_task,
                            NULL,
                            OS_THREAD_PRIO_APP,
                            ONOFF_THREAD_STACK_SIZE) != OS_OK) {
            printf("thread create error\n");
            return -1;
        }
        return 0;
    }
```

5.1.6 效果展示

1. Node 端运行效果

```
ble controller open
version       : 9.1.17
build sha1 : v9.1.17-1-g37fcb80
build date : Mar 15 2021
build time : 09:24:00
platform      : xr806
```

```
Device A not active,waking up!
ble rf_init done!
BLE INIT ALL DONE!
BT Coex. Init. OK.
== XRadio BLE HOST V2.5.0 ==
[bt] [WRN] : Controller to host flow control not supported
[bt] [INF] : No ID address. App must call settings_load()
[bt] [INF] : Identity: D1:86:28:47:2A:9A (random)
[bt] [INF] : HCI: version 5.0 (0x09) revision 0x0111, manufacturer 0x063d
[bt] [INF] : LMP: version 5.0 (0x09) subver 0x0111
[bt] [INF] : Device UUID: 00000000-0000-0000-0000-0000dcdc5258
************************************************
[RandomAddress 54:B1:CA:AE:41:D7 ]
************************************************
bt_ready : Mesh initialized
[bt] [INF] : Saving ID
[bt] [INF] : Primary Element: 0x0002
***** Provisioning - complete
net_idx 0x0000
addr 0x0002
Friend is set to 0x01
[app] onoff get(0)
[app] onoff set(1)
[app] onoff get(1)
[app] onoff get(1)
[app] onoff set(0)
[app] onoff get(0)
[app] onoff get(0)
[app] onoff set(1)
[app] onoff get(1)
[app] onoff get(1)
[app] onoff set(0)
[app] onoff get(0)
[app] onoff get(0)
[app] onoff set(1)
[app] onoff get(1)
[app] onoff get(1)
[app] onoff set(0)
[app] onoff get(0)
[app] onoff get(0)
[app] onoff set(1)
[app] onoff get(1)
[app] onoff get(1)
```

```
[app] onoff set(0)
[app] onoff get(0)
[app] onoff get(0)
[app] onoff set(1)
[app] onoff get(1)
```

2. Provisioner 端运行效果

```
ble controller open
version       : 9.1.17
build sha1 : v9.1.17-1-g37fcb80
build date : Mar 15 2021
build time : 09:24:00
platform      : xr806


Device A not active,waking up!
ble rf_init done!
BLE INIT ALL DONE!
BT Coex. Init. OK.
== XRadio BLE HOST V2.5.0 ==
[bt] [WRN] : Controller to host flow control not supported
[bt] [INF] : No ID address. App must call settings_load()
[bt] [INF] : Identity: C7:A3:80:48:00:90 (random)
[bt] [INF] : HCI: version 5.0 (0x09) revision 0x0111, manufacturer 0x063d
[bt] [INF] : LMP: version 5.0 (0x09) subver 0x0111
[bt] [INF] : Saving ID
Created CDB
[bt] [INF] : Primary Element: 0x0001
**********************************************
[RandomAddress 6A:AB:A2:4D:E5:DD ]
**********************************************
Provisioning local completed
bt_ready : Mesh initialized
Configuring self 0x0001...
Binding AppKey to model 0x001:0002
Binding AppKey to model 0x001:0003
Binding AppKey to model 0x001:1000
Binding AppKey to model 0x001:1001
Configuration complete
Waiting for unprovisioned beacon...
Provisioning for node 5852dcdc00000000000000000000000
Waiting for node to be added...
Added node 0x0002
```

```
Configuring node 0x0002...
        CID         0x05f1
        PID         0x0000
        VID         0x0000
        CRPL        0x000a
        Features 0x0007
        Element loc 0x0000
                SIG Models:
Binding AppKey to model 0x002:0002
Binding AppKey to model 0x002:0003
Binding AppKey to model 0x002:1000
Binding AppKey to model 0x002:1001
                No Vendor Models
Configuration complete
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
Waiting for unprovisioned beacon...
```

著作权声明

商标声明

免责声明