ESP8266

Mesh 用户手册

版本 1.2

关于本手册

本手册介绍了 ESP8266 Mesh 网络，包含以下章节：

章	标题	内容
第 1 章	概述	概括描述 ESP-Mesh，介绍一些概念和网络结构。
第 2 章	Mesh 头信息	介绍 Mesh 头信息的格式、字段和代码。
第 3 章	API 参考	介绍数据结构和 API。
第 4 章	示例代码	为 Mesh 开发提供示例代码。

发布说明

日期	版本	发布说明
2015.07	V1.0	首次发布。
2015.09	V1.1	更新第 3 章。
2016.01	V1.2	增加第 2 章和第 4 章， 更新第 1 章和第 3 章。

📖 **说明：**

现行版本为支持初期产品开发者的早期发布版本。内容如有变更，恕不另行通知。

目录

1. 概述

随着物联网的发展，物联网节点规模迅速扩张，但路由器可供直接接入的节点数有限（通常少于 32 个），因此在大规模的物联网应用中，不可能把所有物节点都直接接入路由器。针对此问题，目前有两种解决方法。

- 超级路由：增强路由的功能，使其可以直接接入更多的节点。
- Mesh 网络：物联网节点之间可以组成网络，并可以转发数据包。

ESP8266 使用的是 Mesh 组网，如图 1-1 所示。这样，不需要改变路由就可以实现大量节点连接到网络。

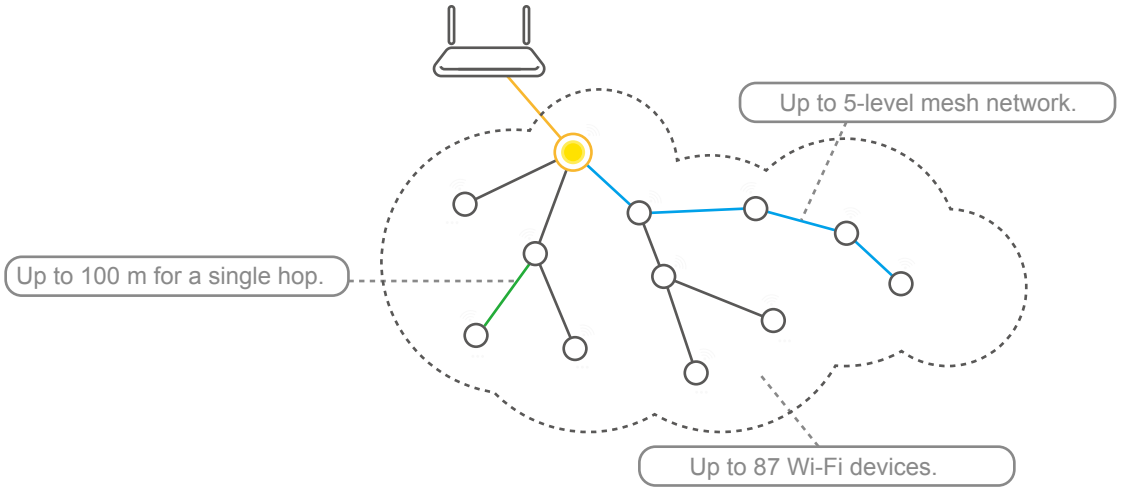


图 1-1. ESP-Mesh 组网

1.1. 概念介绍

IOT Espressif App

IOT Espressif App（以下简称 IOT App）是乐鑫自主研发的手机 App，可以实现对 Wi-Fi 设备的就地控制和遥控，这些 Wi-Fi 设备包括智能灯和智能开关等。

ESP-Touch

ESP-Touch 由乐鑫自主研发，用来将 Wi-Fi 设备连接到路由器。

ESP-Touch Smart Config 模式

设备只有在 Smart Config 模式下才能由 ESP-Touch 配置。这种状态称为 ESP-Touch 状态。关于如何配置，请参考“1.2 网络结构”。

本地设备

如图 1-2 所示，若设备仅通过 ESP-Touch 配置在路由器端，而并未在服务器端激活，则该设备为本地设备。

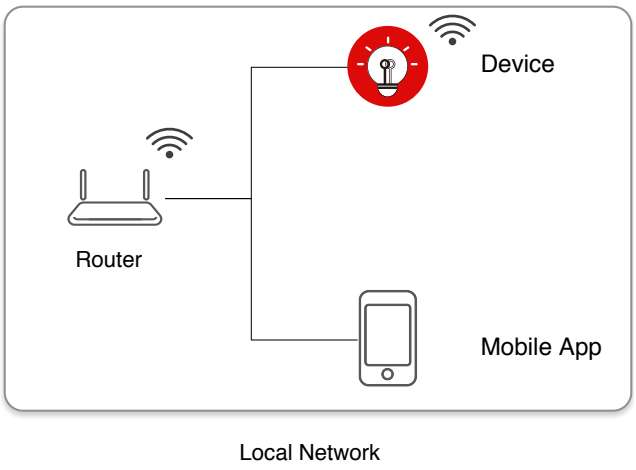


图 1-2. 本地网络

云端设备

如图 1-3 所示，若设备通过 ESP-Touch 配置在路由器端，并在服务器上激活，则该设备为云端设备。

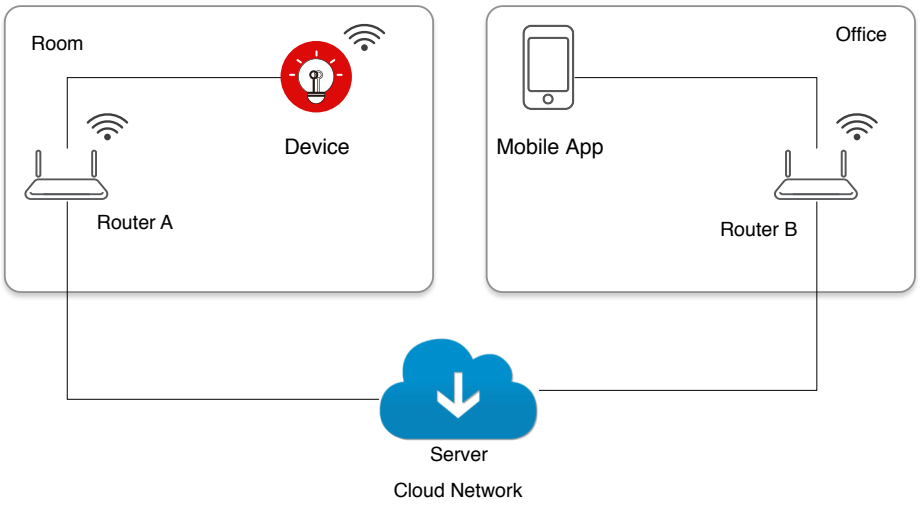


图 1-3. 云端网络

节点在 IOT App 上有三种状态：

- 云端状态：设备为云端设备，设备和 IOT App 连接到不同路由器。

- 在线状态：设备为本地设备或云端设备，设备和 IOT App 连接到同一路由器。
- 离线状态：设备为云端设备，且未接入路由器。

设备类型和状态

设备状态	云端状态	在线状态	离线状态
云端设备	✓	✓	✓
本地设备	✗	✓	✗

1.2. 网络结构

1.2.1. 组网原理

Mesh 组网支持自动组网。当用户使用 ESP-Touch 配置 Mesh 网络时，设备会自动扫描周围的 Wi-Fi 接入点（AP）。

1.2.2. 组网示意图

Mesh 组网示意图如图 1-4 所示。

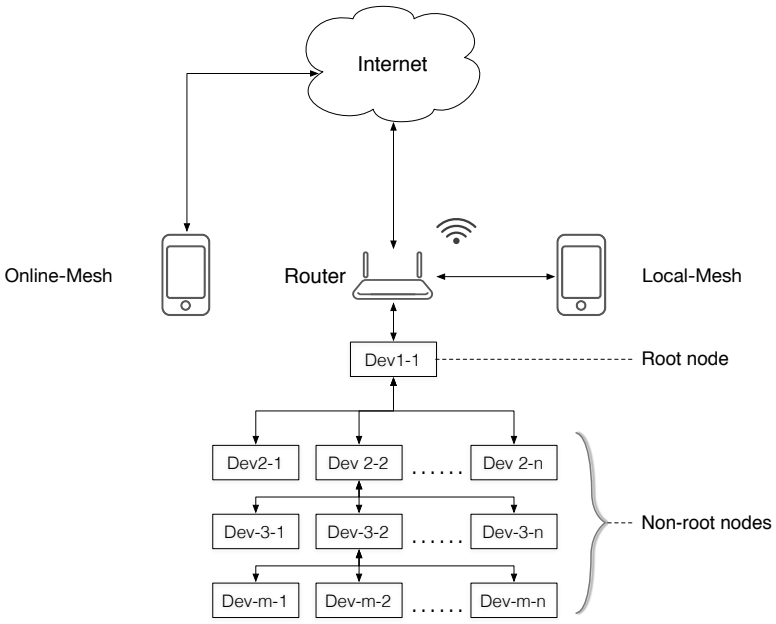


图 1-4. Mesh 组网示意图

- 直接接入路由器的节点是根节点，其他为非根节点。更多详情请参考“1.2.3 网络节点”。

- Online-Mesh：当路由器连接网络，用户可以使用手机通过网络控制云端设备。
- Local-Mesh：用户通过路由器只能控制本地设备。

1.2.3. 网络节点

根据在 Mesh 网络中的位置，节点可分为：

根节点

- 接收和发送数据包。
- 转发来自服务器、手机和其子节点的数据包。

非根节点

- 非叶节点：接收和发送数据包，转发来自其父节点和子节点的数据包。
- 叶子节点：只接收或发送数据包，不转发数据包。

2. Mesh 头信息

2.1. Mesh 头信息格式

Mesh 头信息格式如图 2-1 所示。

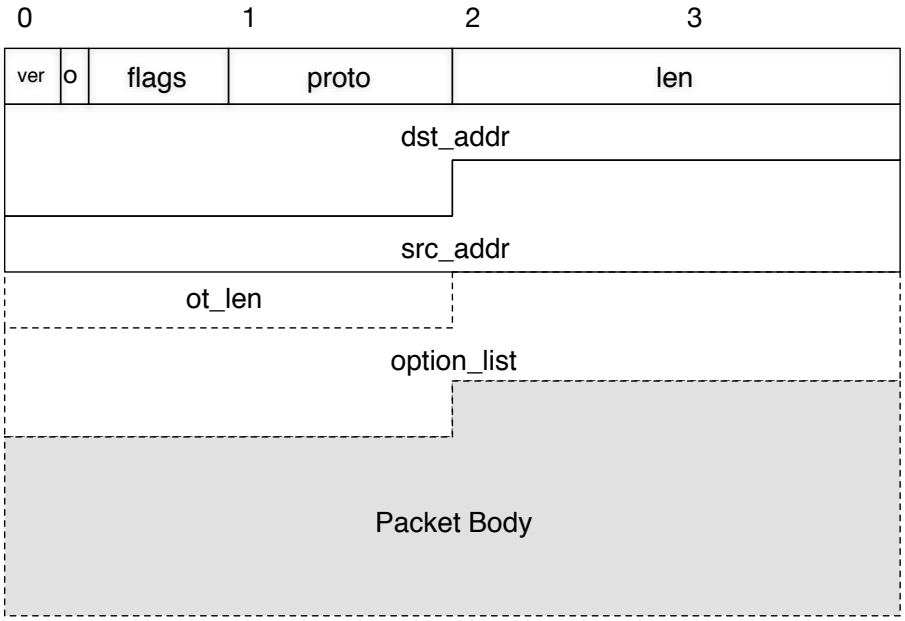
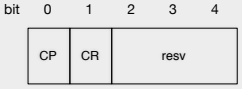
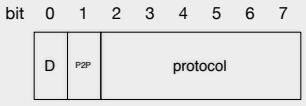


图 2-1. Mesh 头信息格式

Mesh 头信息的字段如表 2-1 所示。

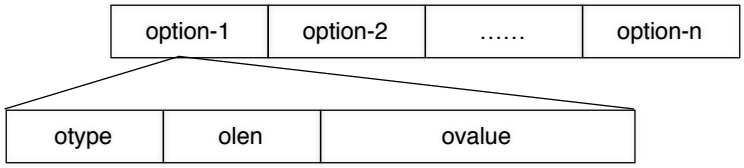
表 2-1. Mesh 头信息格式

字段名	长度	说明
ver	2 比特位	Mesh 的版本。
o	1 比特位	选项标志。
flags	5 比特位	
	FP	数据包捎带流量应答。
	FR	数据包捎带流量请求。

字段名	长度	说明
	resv	保留字段。
proto	8 比特位	
	D	数据包流向： • 0：向下 • 1：向上
	P2P	节点到节点数据包。
	协议	用户数据协议。

	mesh_usr_proto_type 的定义如下：
	enum mesh_usr_proto_type {
	M_PROTO_NONE = 0, // 用于发送 Mesh 管理数据包
	M_PROTO_HTTP, // HTTP 格式用户数据
	M_PROTO_JSON, // JSON 格式用户数据
	M_PROTO_MQTT, // MQTT 格式用户数据
	M_PROTO_BIN, // 用户数据为二进制流
	};	

len	2 字节	Mesh 数据包的长度（包括 Mesh 头信息）。
dst_addr	6 字节	目的地址 • **proto.D = 0 或 proto.P2P = 1**：dst_addr 表示目标设备的 MAC 地址。 • **Bcast 或 mcast packet**：dst_addr 表示 bcast 或者 mcast 的 MAC 地址。 • **proto.D = 1 且 proto.P2P = 0**：dst_addr 表示手机或者服务器的目的 IP 和端口。
src_addr	6 字节	源地址 • **proto.P2P = 1**：src_addr 表示源设备的 MAC 地址。 • **Bcast 或 mcast packet**：src_addr 表示源设备的 MAC 地址。 • **proto.D = 1**：src_addr 表示源设备的 MAC 地址。 • **proto.D = 0 且 forward packet into mesh**：src_addr 表示手机或者服务器的目的 IP 和端口。
ot_len		选项的总长（包括自身）。

字段名	长度	说明
option_list		选项的元素列表。 
otype	1 字节	选项的类型。
olen	1 字节	当前选项的长度。
ovlaue	用户定义	当前选项的值。

2.2. Mesh 选项

2.2.1. 数据结构

Mesh 选项的类型由 `mesh_option_type` 的结构来定义。

```
enum mesh_option_type {
  M_O_FLOW_REQ = 0,// 流量请求选项
  M_O_FLOW_RESP,   // 流量应答选项
  M_O_ROUTER_SPREAD,  // 路由信息传播选项
  M_O_ROUTE_ADD,      // 路由表更新（节点连接 Mesh）选项
  M_O_ROUTE_DEL,      // 路由表更新（节点退出 Mesh）选项
  M_O_TOPO_REQ,       // 拓扑结构请求选项
  M_O_TOPO_RESP,      // 拓扑结构应答选项
  M_O_MCAST_GRP,      // 组播成员列表
  M_O_MESH_FRAG,      // Mesh 管理分片选项
  M_O_USR_FRAG,       // 用户数据分片
  M_O_USR_OPTION,     // 用户选项
};
```

表 2-2. Mesh 头信息类型

字段名	选项长度	说明	格式
M_O_FLOW_REQ	2 字节	用于流量请求。	otype: 0x00; olen: 0x02; ovalue
M_O_FLOW_RESP	6 字节	用于流量应答。	otype: 0x01; olen: 0x06; ovalue: congest capacity
M_O_ROUTER_SPREAD	106 字节	用于传播路由信息。	otype: 0x02; olen: 0x6A; ovalue: Router information
M_O_ROUTE_ADD	6*n+2 字节	用于新增节点连接 Mesh 网络时更新路由表。	otype: 0x03; olen: length; ovalue: MAC address list
M_O_ROUTE_DEL	6*n+2 字节	用于节点退出 Mesh 网络时更新路由表。	otype: 0x04; olen: length; ovalue: MAC address list
M_O_TOPO_REQ	8 字节	用于获取 Mesh 网络的拓扑结构。	otype: 0x05; olen: 0x06; ovalue: MAC address of the device searched
M_O_TOPO_RESP	6*n+2 字节	用于应答 Mesh 网络的拓扑结构。	otype: 0x06; olen: length; ovalue: MAC address list

2.2.2. 示例

流量请求包

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 04 01 14 00 18 FE 34 A5 3B AD 18 FE 34 A2 C7 76
00000010h: 04 00 00 02
```

表 2-3. 流量请求包

字段名	值	说明
head.ver	00	Mesh 的当前版本为 00。
head.O	1	选项存在于数据包中。
head.flags.FP	0	无捎带流量应答。
head.flags.FR	0	无捎带流量请求。
head.flags.resv	000	保留字段。
head.proto.D	1	向上。
head.proto.P2P	0	无节点到节点数据包。
head.proto.protocol	000000	Mesh 管理数据包。

字段名	值	说明
head.len	0x0014	数据包长度为 20 字节。
head.dst_addr	18 FE 34 A5 3B AD	目的设备的 MAC 地址。
head.src_addr	18 FE 34 A2 C7 76	源设备的 MAC 地址。
head.ot_len	0x0004	选项长度为 0x0004。
head.option_list[0].otype	0x00	M_FLOW_REQ。
head.option_list[0].olen	0x02	选项长度为 0x02。

流量应答包

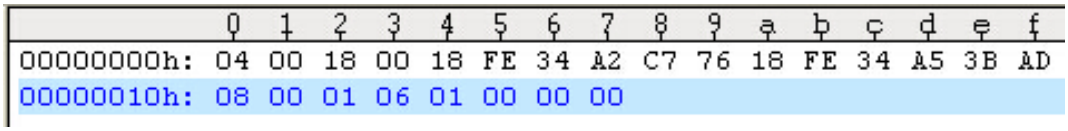


表 2-4. 流量应答包

字段名	值	说明
head.ver	00	Mesh 的当前版本为 00。
head.O	1	选项存在于数据包中。
head.flags.FP	0	无捎带流量应答。
head.flags.FR	0	无捎带流量请求。
head.flags.resv	000	保留字段。
head.proto.D	0	向下。
head.proto.P2P	0	无节点到节点数据包。
head.proto.protocol	000000	Mesh 管理数据包。
head.len	0x0015	数据包长度为 21 字节。
head.dst_addr	18 FE 34 A2 C7 76	目的设备的 MAC 地址。
head.src_addr	18 FE 34 A5 3B AD	源设备的 MAC 地址。
head.ot_len	0x0008	选项长度为 0x0008。
head.option_list[0].otype	0x01	M_FLOW_RESP。
head.option_list[0].olen	0x06	选项长度为 0x06。
head.option_list[0].ovalue	0x01	选项值为 0x00000001，流量容量为 0x00000001。

3. API 参考

3.1. 数据结构

3.1.1. Mesh 头信息格式

```
struct mesh_header_format {
    uint8_t ver:2;                              // Mesh 的版本
    uint8_t oe: 1;                              // 选项标志
    uint8_t fp: 1;                              // 数据包捎带流量应答
    uint8_t fr: 1;                              // 数据包捎带流量请求
    uint8_t rsv:3;                              // 保留字段
    struct {
        uint8_t d:  1;                          // 方向：1，向上；2，向下
        uint8_t p2p:1;                          // 节点到节点数据包
        uint8_t protocol:6;                     // 用户数据使用的协议
    } proto;
    uint16_t len;                               // 数据包总长（包括 Mesh 头
信息）
    uint8_t dst_addr[ESP_MESH_ADDR_LEN];        // 目的地址
    uint8_t src_addr[ESP_MESH_ADDR_LEN];        // 源地址
    struct mesh_header_option_header_type option[0];  // Mesh 选项
} __packed;
```

3.1.2. Mesh 选项头信息格式

```
struct mesh_header_option_header_type {
  uint16_t ot_len;                               // 选项总长
  struct mesh_header_option_format olist[0]; // 选项列表
} __packed;
```

3.1.3. Mesh 选项格式

```
struct mesh_header_option_format {
    uint8_t otype;          // 选项的类型
    uint8_t olen;           // 当前选项长度
    uint8_t ovalue[0];      // 选项值
} __packed;
```

3.1.4. Mesh 选项分片格式

```
struct mesh_header_option_frag_format {
    uint16_t id;              // 分片 ID
    struct {
        uint16_t resv:1;      // 保留字段
        uint16_t mf:1;        // 更多分片
        uint16_t idx:14;      // 分片偏移
    } offset;
} __packed;
```

3.1.5. Mesh 回调格式

```
typedef void (* espconn_mesh_callback)(int8_t result);
```

3.1.6. Mesh 扫描回调格式

```
typedef void (* espconn_mesh_scan_callback)(void *arg, int8_t
status);
```

3.1.7. Mesh 扫描用户回调格式

```
typedef void (* espconn_mesh_usr_callback)(void *arg);
```

3.2. 数据包 API

> 📖 **说明：**
>
> 更多信息请参考《*ESP8266 Non-OS SDK API Guide*》，链接如下：
>
> *http://www.espressif.com/zh-hans/support/download/documents*。

4. 示例代码

4.1. 设备

更多详情请参考：

ESP8266_MESH_DEMO/blob/master/mesh_demo/demo/mesh_demo.c。

4.2. 手机或服务器

```
void controller_entrance(Parameter list)
{
    /*添加代码以检查状态*/
    /*添加代码以创建控制包*/
    uint8_t json_control_data[] = {/*添加代码*/};
    uint16_t control_data_len = sizeof(json_control_data)
    struct mesh_header_format *mesh_header = NULL;
    /* src_addr 应为手机或服务器的 IP 和端口的组合。用户可以设置地址为
    0，则根设备会填充此字段。如果用户自行填充此字段，请确保填入正确的值。*/
    uint8_t src_addr[] = {0,0,0,0,0,0},
    dst_addr[] = {xx,xx,xx,xx,xx,xx};
    mesh_header = (struct mesh_header_format
*)espconn_mesh_create_packet(dst_addr, src_addr, false, true,
M_PROTO_JSON, control_data_len,
    false, 0, false, 0, false, 0, 0);
    if (!mesh_header)
    {
        printf("alloc resp packet fail\n");
        return;
    }
    if (espconn_mesh_set_usr_data(mesh_header,
resp_json_packet_body, resp_data_len))
        {
        printf("set user data fail\n");
```

```
                free(mesh_header);
                return;
            }
            // 发送控制包
            espconn_mesh_sent(esp, mesh_header, mesh_header->len);
            free(mesh_header);
        }
```

4.3. 获取拓扑结构

```
void topology_entrance(Parameter list)
{
    /*添加代码以检查状态*/
    /*添加代码以创建控制包*/
    bool res;
    struct mesh_header_format *mesh_header = NULL;
    struct mesh_header_option_format *topo_option = NULL;
    uint8_t src_addr[] = {0,0,0,0,0,0};
    uint8_t dst_addr[] = {xx,xx,xx,xx,xx,xx}; //根设备的 MAC 地址
    uint8_t dev_mac[6] = {xx,xx,xx,xx,xx,xx}; //0 表示所有设备的拓扑结构
    uint16_t ot_len = sizeof(*topo_option) + sizeof(struct
mesh_header_option_header_type) + sizeof(dev_mac);
    mesh_header = (struct mesh_header_format
*)espconn_mesh_create_packet(
    dst_addr, src_addr, false, true, M_PROTO_NONE, 0,
    true, ot_len, false, 0, false, 0, 0);
    if (!mesh_header) {
        printf("alloc resp packet fail\n");
        return;
    }
    topo_option = (struct mesh_header_option_format
*)espconn_mesh_create_option(
    M_O_TOPO_REQ, dev_mac, sizeof(dev_mac));
```

```
        if (!topo_option) {
            printf("alloc topo option fail\n");
            free(mesh_header);
            return;
        }
        res = espconn_mesh_add_option(mesh_header, topo_option);
        free(topo_option);
        if (res) {
            printf("add topo option fail\n");
            free(mesh_header);
            return;
        }
        // 发送获取拓扑结构包
        espconn_mesh_sent(esp, mesh_header, mesh_header->len);
        free(mesh_header);
    }
```

4.4. 解析拓扑结构

```
    void topology_parser_entrance(uint8_t *topo_resp, uint16_t len)
    {
        /*添加代码以检查参数*/
        uint16_t oidx = 1;
        struct mesh_header_format *mesh_header = NULL;
        struct mesh_header_option_format *topo_option = NULL;
        mesh_header = (struct mesh_header_format *)topo_resp;
        if (!mesh_header->oe) {
            printf("no option exist\n");
            return;
        }
        /* 用户需要在数据包头信息中逐个解析所有选项 */
        while(espconn_mesh_get_option(mesh_header, M_O_TOPO_RESP,
        oidx++, &topo_option)) {
```

```
                uint16_t dev_count = topo_option->olen/6;
                process_dev_list(topo_option->ovalue, dev_count);
            }
        }
```

4.5. Dev-App

有关示例代码的详细信息参考：

- *ESP8266_MESH_DEMO/blob/master/mesh_demo/include/user_config.h*
- *ESP8266_MESH_DEMO/blob/master/mesh_demo/demo/mesh_demo.c*

乐鑫 IOT 团队

www.espressif.com