

ESP8266

Mesh 用户手册



版本 1.2

版权 © 2016

关于本手册

本手册介绍了 ESP8266 Mesh 网络，包含以下章节：

章	标题	主题
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 Header	介绍 mesh header 的格式、字段和代码。
第 3 章	API 索引	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.....	1
1.1. 概念	1
1.2. 网络结构	3
1.2.1. 组网原理.....	3
1.2.2. 组网示意图	3
1.2.3. 网络节点.....	4
2. Mesh Header	5
2.1. Mesh Header 格式.....	5
2.2. Mesh 选项	7
2.2.1. 结构	7
2.2.2. 示例	8
3. API 索引	10
3.1. 数据结构.....	10
3.1.1. Mesh Header 格式	10
3.1.2. Mesh 选项 Header 格式.....	10
3.1.3. Mesh 选项格式	10
3.1.4. Mesh 选项分片格式	11
3.1.5. Mesh 回调格式	11
3.1.6. Mesh 扫描回调格式	11
3.1.7. Mesh 扫描用户回调格式.....	11
3.2. 数据包 APIs.....	11
4. 示例代码	12
4.1. 设备	12
4.2. 手机或服务器.....	12
4.3. 获取拓扑结构.....	13
4.4. 解析拓扑结构包	14
4.5. Dev-App.....	15



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 组网：物联网节点之间可以组成网络，并可以转发数据包。

ESP8266 利用的就是 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。

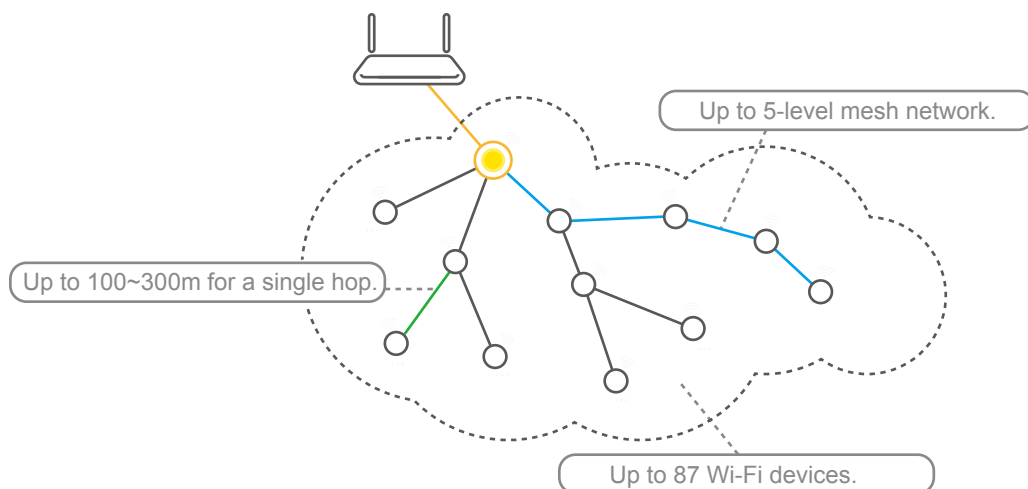


图 1-1. ESP-Mesh 组网

1.1. 概念

IOT Espressif App

IOT 乐鑫 App（以下简称 IOT app）由乐鑫研发，用来就地控制和遥控 Wi-Fi 设备，包括智能灯和智能开关等。

ESP-Touch

ESP-Touch 由乐鑫研发，用来将 Wi-Fi 设备连接到路由器。



ESP-Touch 的智能配置模式

设备只有在智能配置模式下才能由 ESP-Touch 配置。这种状态称为 ESP-Touch 状态。关于如何配置，请参考第 2 章 Mesh 组网。

本地设备

如图 1-2 所示，若 ESP-Touch 只将设备配置在路由器上，而不在服务器端激活，则该设备为本地设备。

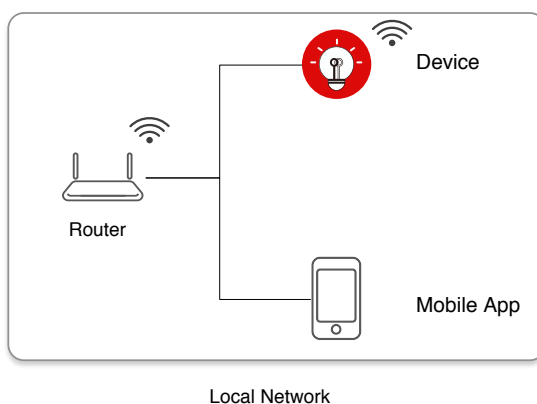


图 1-2. 本地网络

云端设备

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

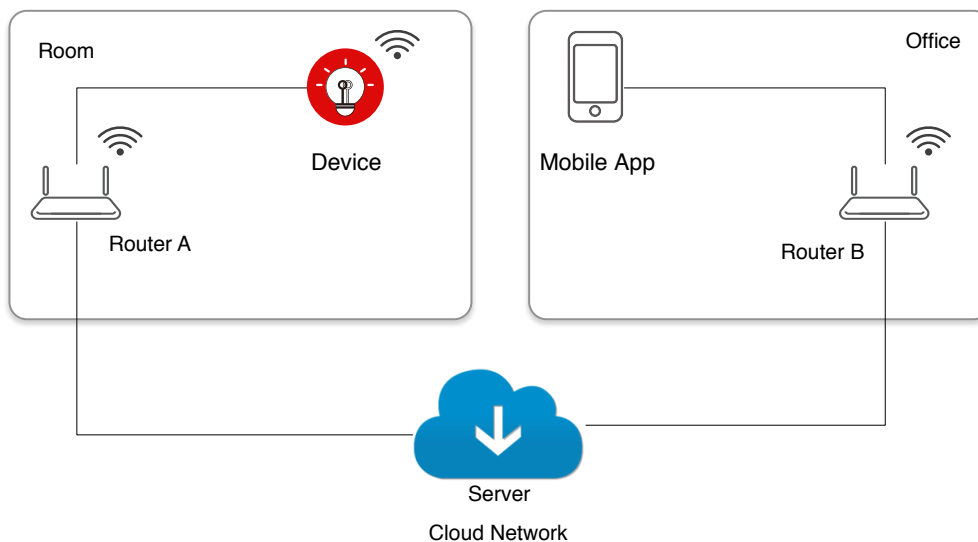


图 1-3. 云端网络



节点在手机 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，未连接路由器。

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 AP。

1.2.2. 组网示意图

组网示意图如图 1-4 所示。

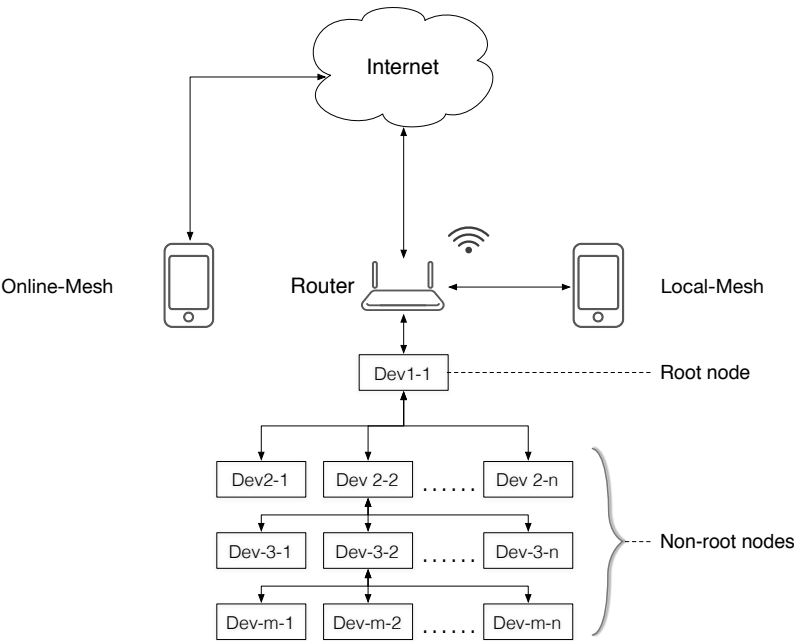


图 1-4. Mesh 组网示意图



- 直接接入路由器的节点是根节点，其他为非根节点。更多详情请参考章节 1.2.3 网络节点。
- 在线 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。
- 本地 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，节点可以分为：

根节点

- 接收和发送数据包。
- 转发来自服务器、手机 app 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 Header

2.1. Mesh Header 格式

Mesh Header 格式如图 2-1 所示。

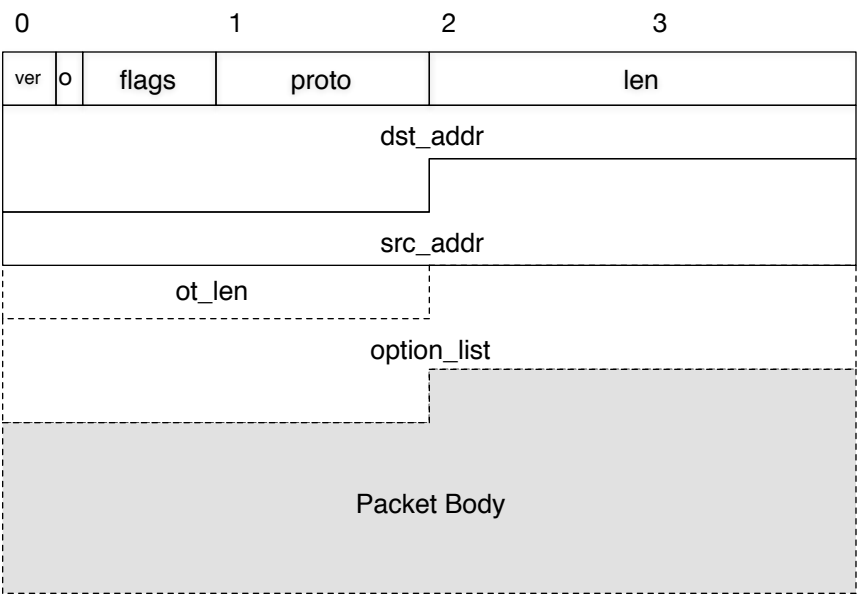


图 2-1. Mesh Header 格式

Mesh Header 的字段如表 2-1 所示。

表 2-1. Mesh Header 格式

字段名	长	说明			
ver	2 bits	mesh 的版本。			
o	1 bit	选项标志。			
flags	5 bits	<div>bit 0 1 2 3 4</div> <table><tr><td>CP</td><td>CR</td><td>resv</td></tr></table>	CP	CR	resv
	CP	CR	resv		
	CP	数据包捎带流量应答。			
	CR	数据包捎带流量请求。			
resv	保留。				



字段名	长	说明
proto	8 bits	<div><div>bit 0 1 2 3 4 5 6 7</div><div><div>D</div><div>P2P</div><div>protocol</div></div></div>
	D	数据包流向： <ul style="list-style-type: none">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 Bytes	mesh 数据包的长度（包括 mesh header）。
dst_addr	6 Bytes	目的地址 <ul style="list-style-type: none">proto.D = 0 or proto.P2P = 1：dst_addr 代表目的设备的 mac 地址。Bcast or mcast packet：dst_addr 代表 bcast 或者 mcast 的 mac 地址。proto.D = 1 and proto.P2P = 0：dst_addr 代表手机或者服务器的目的 IP 和端口。
src_addr	6 Bytes	源地址 <ul style="list-style-type: none">proto.P2P = 1：src_addr 代表源设备的 mac 地址。Bcast or mcast packet：src_addr 代表源设备的 mac 地址。proto.D = 1：src_addr 代表源设备的 mac 地址。proto.D = 0 and forward packet into mesh：src_addr 代表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）。
option_list		选项的元素列表 <div><div><div>option-1</div><div>option-2</div><div>.....</div><div>option-n</div></div><div><div>otype</div><div>olen</div><div>ovalue</div></div></div>
otype	1 Byte	选项
olen	1 Byte	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. 结构

Mesh 选项由 mesh_option_type 的结构来定义。

```
enum mesh_option_type {  
    M_O_CONGEST_REQ = 0, //flow request option  
    M_O_CONGEST_RESP,    //flow response option  
    M_O_ROUTER_SPREAD,    //router information spread option  
    M_O_ROUTE_ADD,        //route table update (node joins mesh) option  
    M_O_ROUTE_DEL,        //route table update (node exits mesh) option  
    M_O_TOPO_REQ,         //topology request option  
    M_O_TOPO_RESP,        //topology response option  
    M_O_MCAST_GRP,        //group list of mcast  
    M_O_MESH_FRAG,        //mesh management fragmentation option  
    M_O_USR_FRAG,         //user data fragmentation  
    M_O_USR_OPTION,       //user option  
};
```

表 2-2. Mesh Header 类型

字段名	长	说明	格式		
M_O_CONGEST_REQ	2 Bytes	用于流量请求。	otype	olen	ovalue
			0x00	0x02	
M_O_CONGEST_RESP	6 Bytes	用于流量应答。	otype	olen	ovalue
			0x01	0x06	congest capacity
M_O_ROUTER_SPREAD	106 Bytes	用于传播路由信息。	otype	olen	ovalue
			0x02	0x6A	Router information
M_O_ROUTE_ADD	6*n+2 Bytes	用于当新节点连接 mesh 网络时更新路由表。	otype	olen	ovalue
			0x03	length	MAC address list
M_O_ROUTE_DEL	6*n+2 Bytes	用于当节点退出 mesh 网络时更新路由表。	otype	olen	ovalue
			0x04	length	MAC address list
M_O_TOPO_REQ	1 Byte	用于获取 mesh 网络的拓扑结构。	otype	olen	ovalue
			0x05	0x06	MAC address of the device searched



字段名	长	说明	格式		
M_O_TOPO_RESP	6*n+2 Bytes	用于应答 mesh 网络的拓扑结构。	otype	olen	ovalue
			0x06	length	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.CP	0	无捎带流量应答
head.flags.CR	0	无捎带流量请求
head.flags.resv	000	保留。
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。
head.len	0x0014	数据包长度为 20 Bytes。
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_CONGEST_REQ.
head.option_list[0].olen	0x02	选项长度为 0x02。

流量应答包

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	04	00	18	00	18	FE	34	A2	C7	76	18	FE	34	A5	3B	AD
00000010h:	08	00	01	06	01	00	00	00								



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.CP	0	无捎带流量应答
head.flags.CR	0	无捎带流量请求
head.flags.resv	000	保留。
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 Bytes。
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_CONGEST_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 索引

3.1. 数据结构

3.1.1. Mesh Header 格式

```
struct mesh_header_format {
    uint8_t ver:2;           // version of mesh
    uint8_t oe: 1;           // option flag
    uint8_t cp: 1;           // piggyback flow permit in packet
    uint8_t cr: 1;           // piggyback flow request in packet
    uint8_t rsv:3;           // reserved;
    struct {
        uint8_t d: 1;        // direction, 1:upwards, 0:downwards
        uint8_t p2p:1;       // node to node packet
        uint8_t protocol:6;   // protocol used by user data;
    } proto;
    uint16_t len;             // packet total length (mesh header included)

    uint8_t dst_addr[ESP_MESH_ADDR_LEN]; // destination address
    uint8_t src_addr[ESP_MESH_ADDR_LEN]; // source address
    struct mesh_header_option_header_type option[0]; // mesh option
} __packed;
```

3.1.2. Mesh 选项 Header 格式

```
struct mesh_header_option_header_type {
    uint16_t ot_len;          // option total length;
    struct mesh_header_option_format olist[0]; // option list
} __packed;
```

3.1.3. Mesh 选项格式

```
struct mesh_header_option_format {
    uint8_t otype;           // option type
```



```
uint8_t olen;           // current option length
uint8_t ovalue[0];      // option value
} __packed;
```

3.1.4. Mesh 选项分片格式

```
struct mesh_header_option_frag_format {
    uint16_t id;          // identity of fragmentation
    struct {
        uint16_t resv:1;  // reserved
        uint16_t mf:1;    // more fragmentation
        uint16_t idx:14;  // fragmentation offset;
    } 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 APIs

注意：

更多信息请参考 2C-ESP8266_SDK_Programming Guide。



4.

示例代码

4.1. 设备

更多详情请参考 mesh_demo.c。

4.2. 手机或服务器

```
void controller_entrance(Parameter list)
{
    /*Add your codes to check status*/
    /*Add your codes to build control packet*/
    uint8_t json_control_data[] = { /*Add your codes*/ };
    uint16_t control_data_len = sizeof(json_control_data);
    struct mesh_header_format *mesh_header = NULL;
    /* src_addr should be the combination of IP and port of
    Mobile or Server. You can set the address to zero, then the
    root device will fill in the section. If you fill in the
    section by yourself, please make sure the value is right.*/
    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;
    }
    // sent control packet
    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)
{
    /*Add your codes to check status*/
    /*Add your codes to build getting topology packet*/
    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}; // mac address of root
device
    uint8_t dev_mac[6] = {xx,xx,xx,xx,xx,xx}; // zero represents
topology of all devices
    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;
        }
        // send packet of getting topology
        espconn_mesh_sent(esp, mesh_header, mesh_header->len);
        free(mesh_header);
    }
}
```

4.4. 解析拓扑结构包

```
void topology_parser_entrance(uint8_t *topo_resp, uint16_t len)
{
    /*Add your codes to check parameter*/
    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;
    }
    /* you need parse all the options one by one in the packet header
    */
    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，请参考：

- `user_config.h`
- `mesh_demo.c`



乐鑫科技
IOT 团队

<http://bbs.espressif.com>

免责声明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

版权归© 2016 乐鑫信息科技（上海）有限公司所有。保留所有权利。