

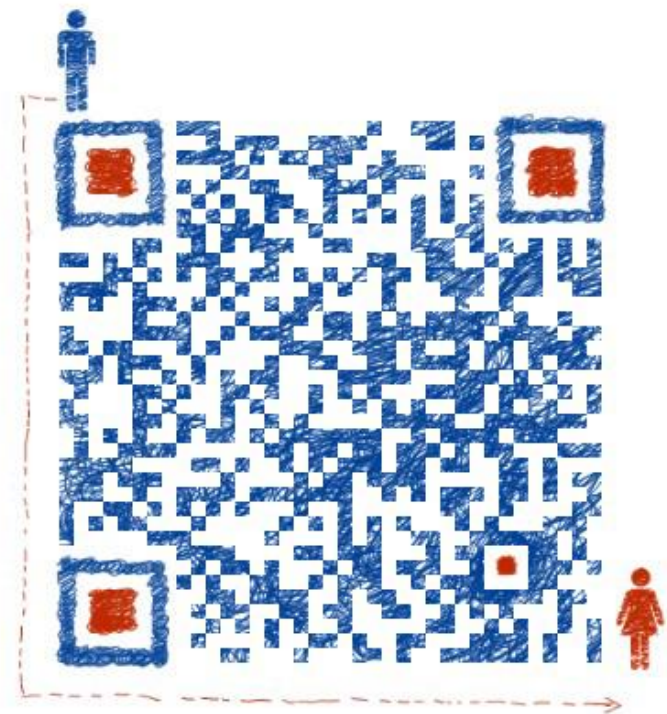


— Linux

无线传感器网项目实战

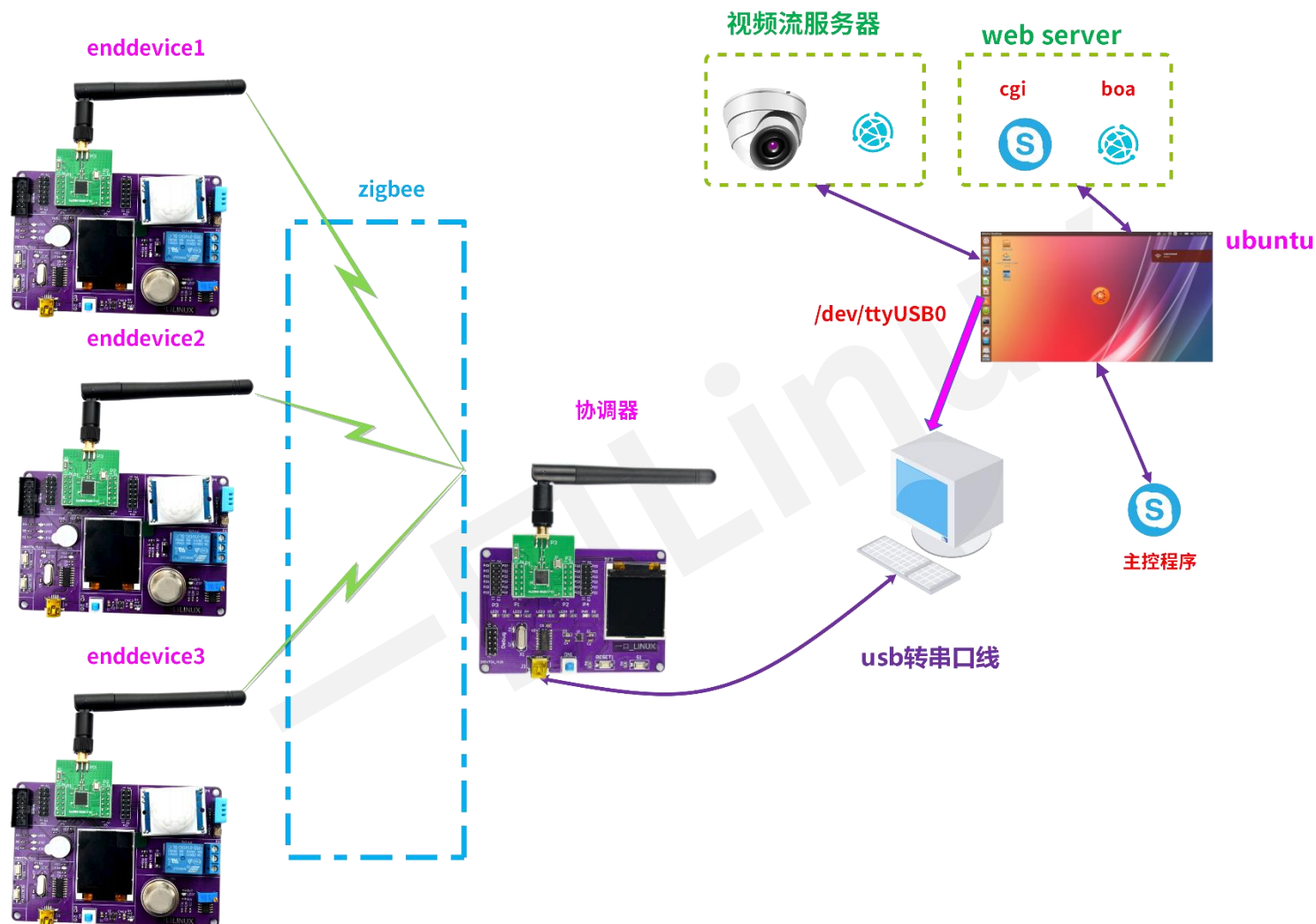


公众号:一口Linux



彭老师个人微信号

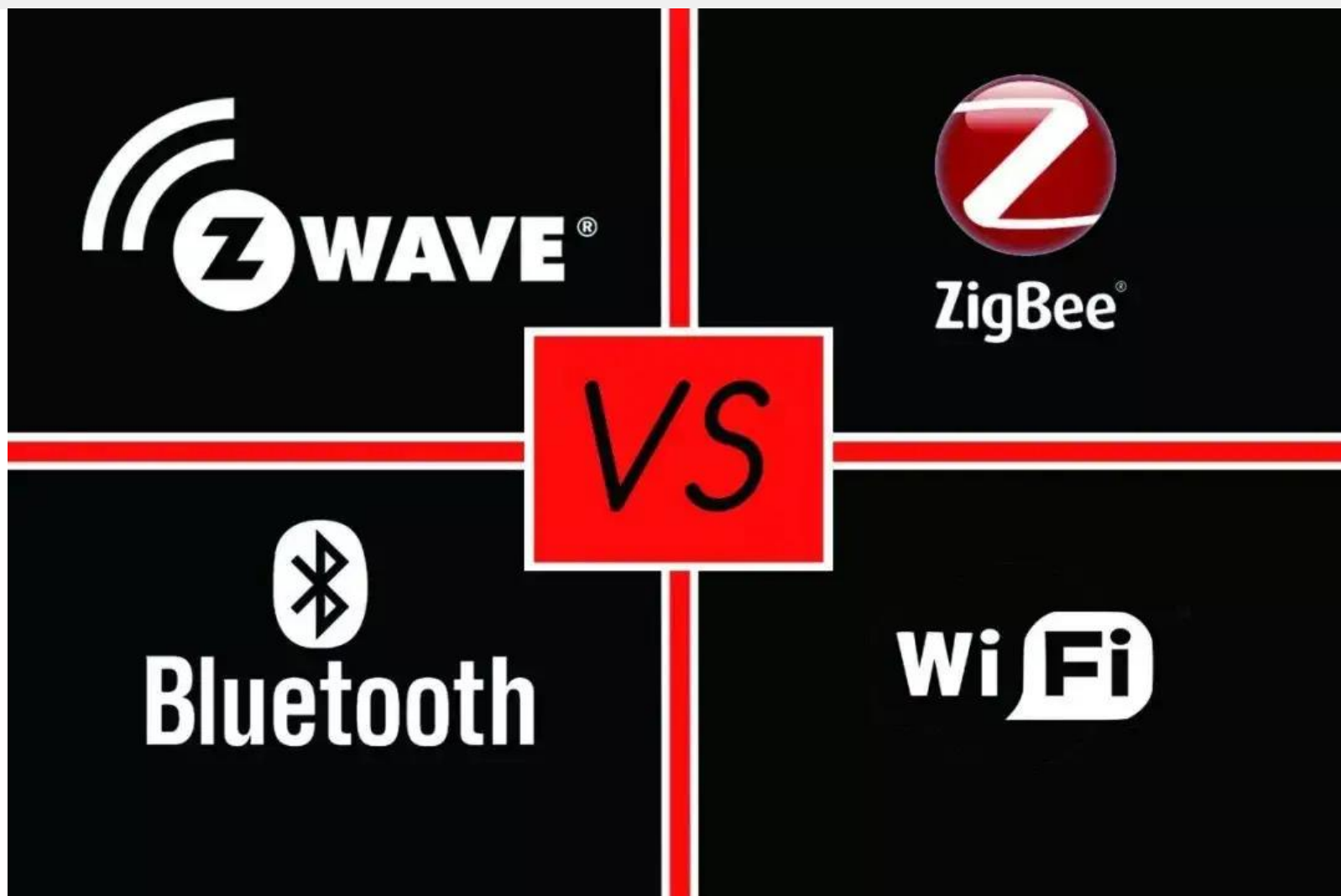
Zigbee在该项目中的作用



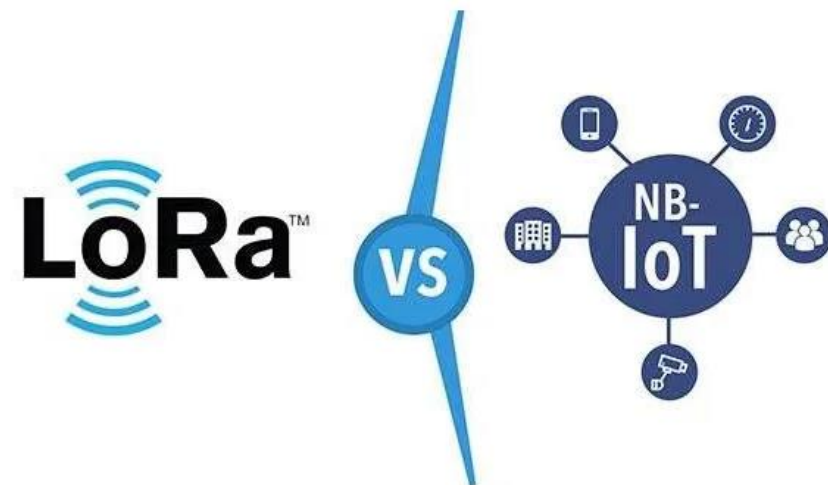
01

物联网通信协议

常见短距离无线通信协议



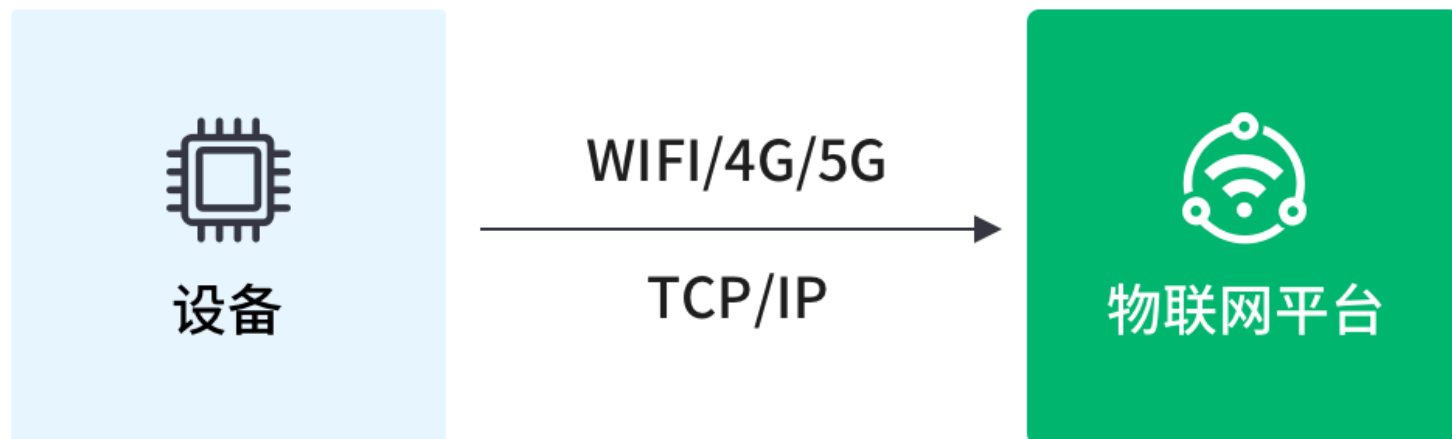
物联网通信协议



关注公众号：一口Linux

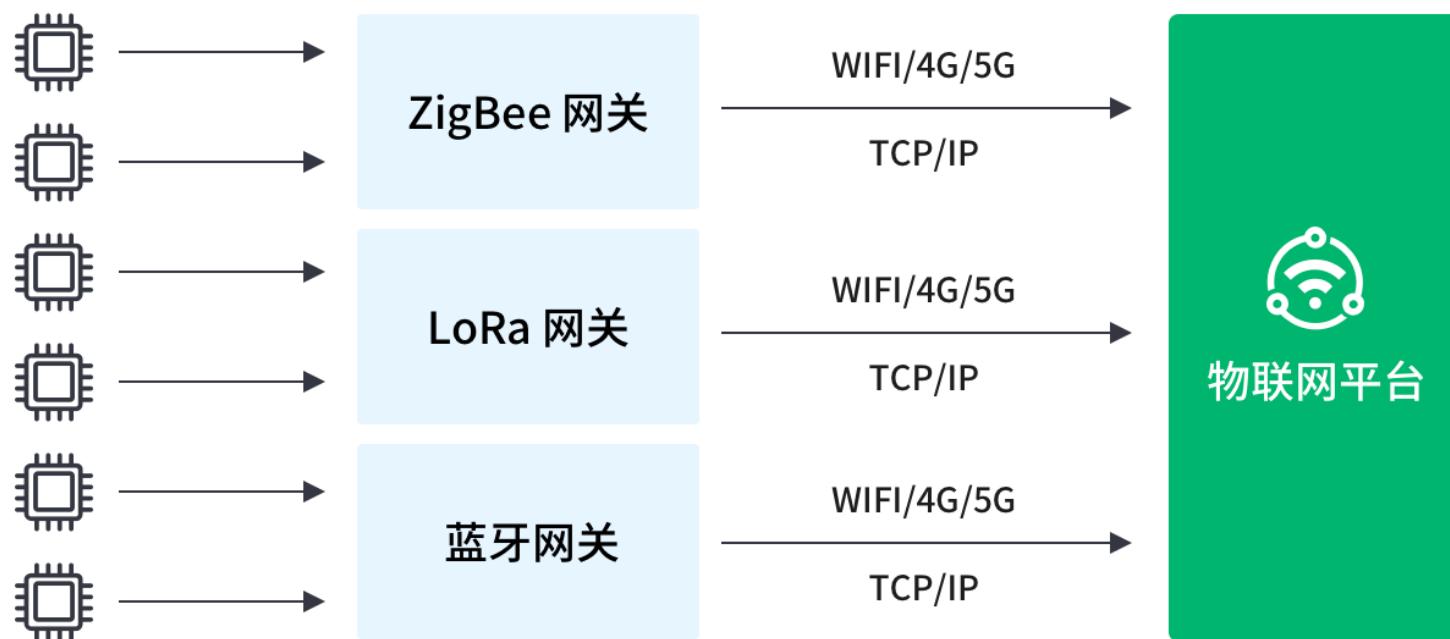
接入方式-云端协议

- 支持 TCP/IP 的物联网设备，可以通过 WIFI、蜂窝网络以及以太网，使用 HTTP、MQTT、CoAP、LwM2M 以及 XMPP 等应用层协议接入云端



接入方式-网关协议

- 网关协议是适用于短距通信无法直接上云的协议，比如**蓝牙、ZigBee、LoRa**等。
- 此类设备需要接入网关转换之后，通过TCP/IP 协议进行上云。





02

Zigbee协议简介

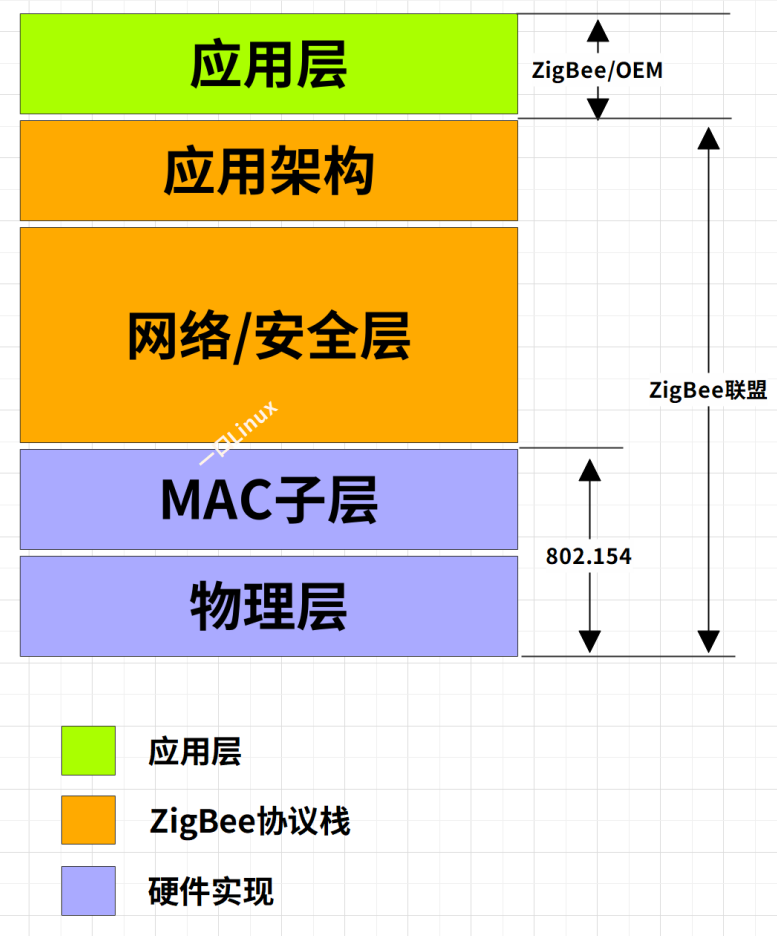
Zigbee历史

- ZigBee，这个名字来源于蜂群使用的赖以生存和发展的通信方式：蜜蜂通过跳Zig-Zag形状的舞蹈来分享新发现的食物源的位置，距离和方向等资讯。
- ZigBee的前身是1998年由INTEL、IBM等产业巨头发起的“HomeRFLite”技术。
- 2002年下半年，英国Invensys公司、日本三菱电气公司、美国摩托罗拉公司以及荷兰飞利浦半导体公司四大巨头共同宣布加盟“Zigbee联盟”，以研发名为“Zigbee”的下一代无线通信标准



Zigbee与IEEE 802.15.4

- ZigBee 是一种开放式的基于IEEE 802.15.4 协定的无线个人局域网(Wireless Personal Area Networks)标准。
- IEEE 802.15.4定义了物理层和媒体接入控制层
- 而ZigBee则定义了更高层如网路层及应用层等。



频带	使用范围	数据传输率	信道数
2.4 GHz	ISM 全世界	250 kbps	16
868 MHz	欧洲	20 kbps	1
915 MHz	ISM 北美	40 kbps	10

ZigBee技术特点

- 1. 低功耗
由于ZigBee的传输速率低,发射功率仅为 1mW,而且采用了休眠模式,功耗低,因此ZigBee设备非常省电。
- 据估算,ZigBee设备仅靠两节5号电池就可以维持长达6个月到2年左右的使用时间。
- 2. 低成本
由于ZigBee模块的复杂度不高,ZigBee协议免专利费,再加之使用的频段无需付费,所以它的成本较低。
- 3. 时延短
通信时延和从休眠状态激活的时延都非常短,典型的搜索设备时延30ms,休眠激活的时延是15ms,活动设备信道接入的时延为15ms。
- 4. 网络容量大
一个星型结构的ZigBee网络最多可以容纳254个从设备和一个主设备,一个区域内可以同时存在最多100个ZigBee网络,而且网络组成灵活。网状结构的ZigBee网络中可有65000多个节点。
- 5. 可靠
采取了碰撞避免策略,同时为需要固定带宽的通信业务预留了专用时隙,避开了发送数据的竞争和冲突。MAC层采用了完全确认的数据传输模式,每个发送的数据包都必须等待接收方的确认信息。如果传输过程中出现问题可进行重发。
- 6. 安全
ZigBee提供了基于循环冗余校验(CRC)的数据包完整性检查功能,支持鉴权和认证,采用了AES-128的加密算法,各个应用可以灵活确定其安全属性。

ZigBee联盟的部分会员

Honeywell

CompXs



关注公众号：一口Linux

二、Zigbee设备类型

- 在 ZigBee 无线传感器网络中有三种设备类型：
 - 协调器、
 - 路由器、
 - 终端节点

1. ZigBee协调器 (Coordinator)

- 它包含所有的网络信息，是3种设备中最复杂的，存储容量大、计算能力最强。
- 它主要用于发送网络信标、建立一个网络、管理网络节点、存储网络节点信息、寻找一对节点间的路由信息并且不断的接收信息。
- 一旦网络建立完成,这个协调器的作用就像路由器节点。



2. ZigBee路由器 (Router)

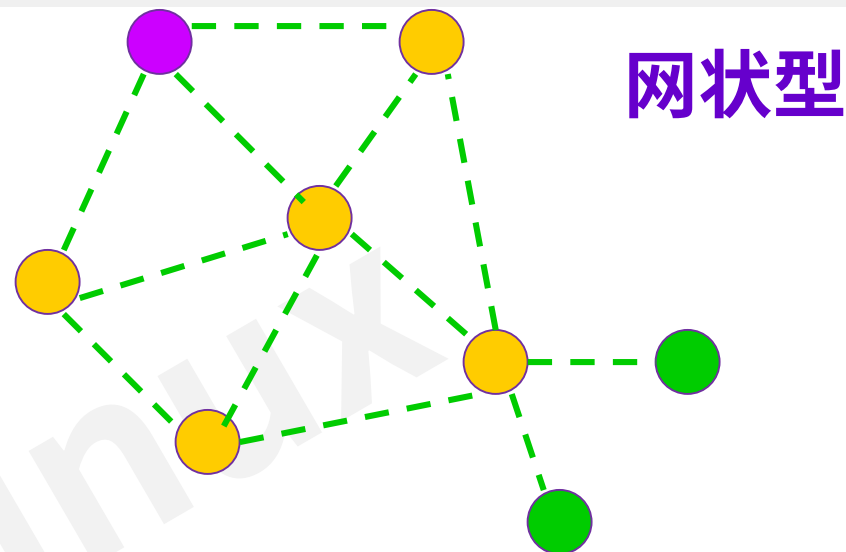
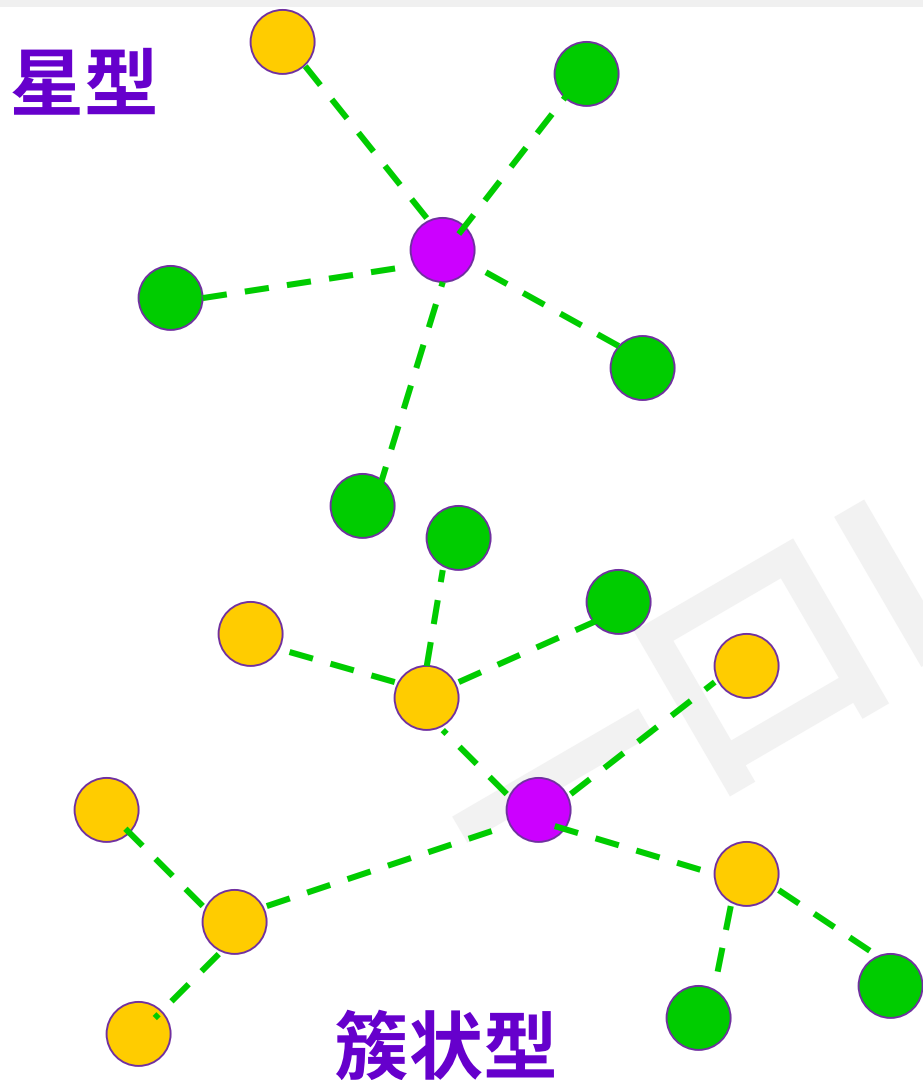
- 它执行的功能包括允许其它设备加入这个网络，跳跃路由，辅助子树下电池供电终端的通信。

3. ZigBee终端设备 (End-device)

- 一个终端设备对于维护这个网络设备没有具体的责任,所以它可以睡眠和唤醒,看它自己的选择。
- 因此它能作为电池供电节点。



三、ZigBee网络拓扑

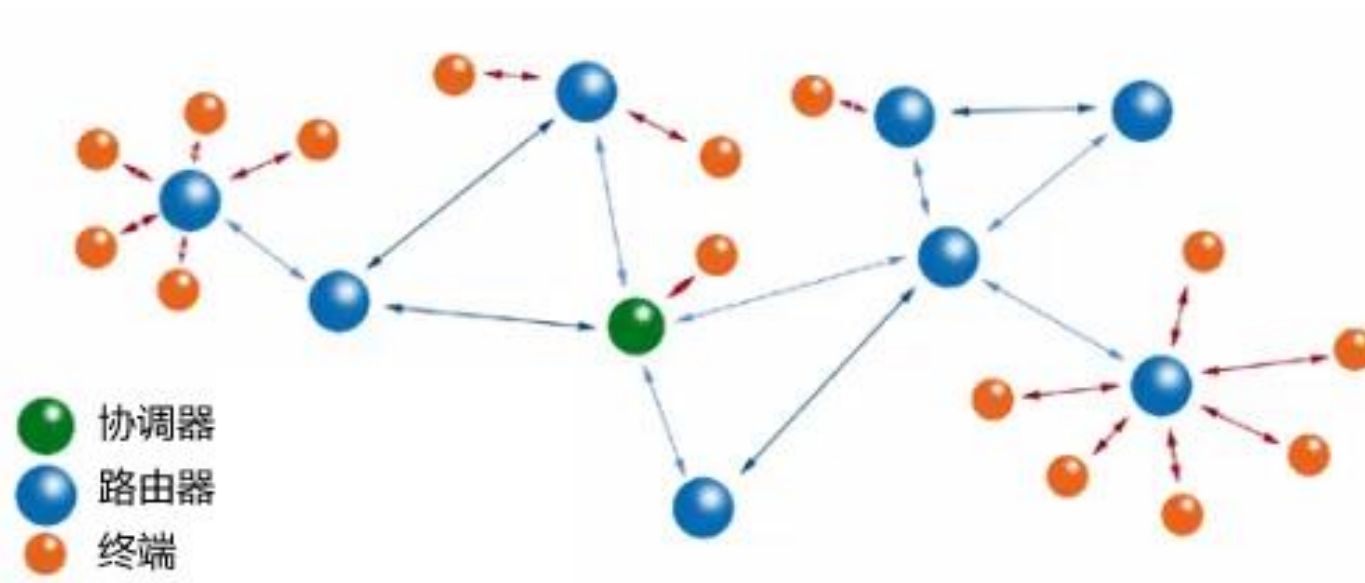


- 网络协调器
- 全功能设备 (FFD, Router):
可以支持任何一种拓扑结构, 可以作为网络协商者和普通协商者, 并且可以和任何一种设备进行通信
- 精简功能设备 (RFD):
只支持星型结构, 不能成为任何协商者, 可以和网络协商者进行通信, 实现简单。

设备类型与拓扑关系

设备类型	拓扑类型	是否成为协调器	通话对象
全功能设备	星型、树状、网状	可以	与任何ZigBee设备通话
简化功能设备	星型	不可以	与协调器、路由器通话，不能与终端设备通话

ZigBee网状（MESH）网络



设备之间
连接关系

MESH网状网络拓扑结构的网络具有强大的功能，
网络可以通过**多级跳**的方式来通信；
该拓扑结构还可以组成极为复杂的网络；
网络还具备**自组织、自愈功能**

四、ZigBee网络建立-协调器建立一个新网络的流程

- 1) 检测协调器
节点必须具备两个条件：
 - 1) 具有ZigBee协调器功能，
 - 2) 没有加入到其它网络中。
- 任何不满足这两个条件的节点发起建立一个新网络的进程都会被网络层管理实体终止，
- 2) 信道扫描
信道扫描包括能量扫描和主动扫描两个过程。
- 3) 配置网络参数
网络层管理实体将为新网络选择一个PAN描述符，必须满足PAN描述符小于或等于0x3fff，不等于0xffff，并且在所选信道内是唯一的PAN描述符
- 4) 运行新网络
- 5) 允许设备加入网络
只有ZigBee协调器或路由器才能通过NLME_PERMIT_JOINING.request原语来设置节点处于允许设备加入网络的状态。

ZigBee网络建立-节点加入网络

- 1) 通过MAC层关联加入网络
- 2) 通过与先前指定父节点连接加入网络

通过MAC层关联加入网络

- 1、子节点发起信道扫描
- 2、子节点存储各PAN信息
- 3、子节点选择PAN
- 4、子节点选择父节点
- 5、子节点请求MAC关联
- 6、父节点响应MAC关联
- 7、子节点响应连接成功
- 8、父节点响应连接成功

组网操作均由
协议栈来实现

五、ZigBee应用领域

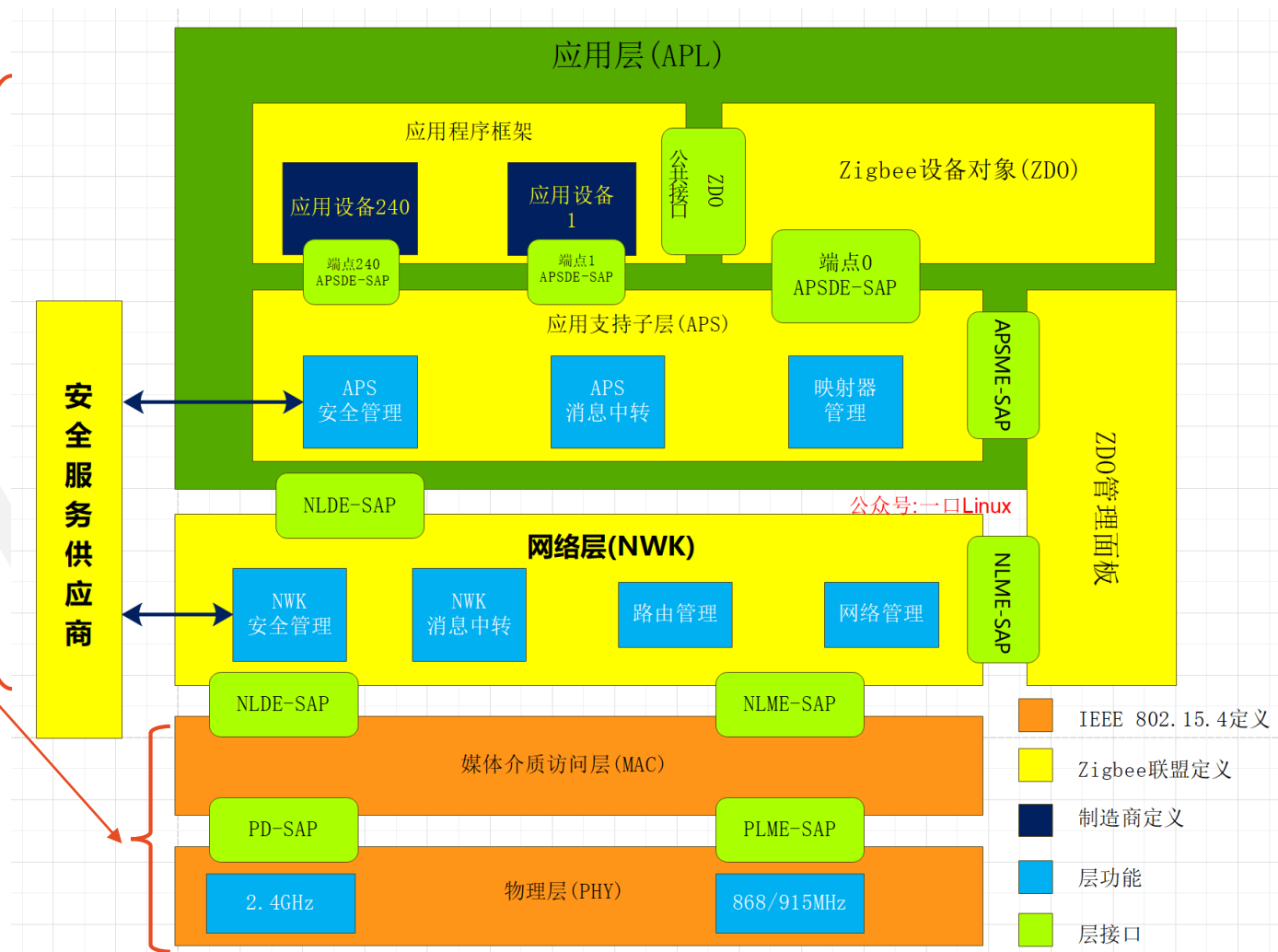
- ❑ 消费性电子设备
- ❑ 家庭和楼宇自动化设备
- ❑ 工业控制装置
- ❑ 农业自动化
- ❑ 电脑外设
- ❑ 医用传感器
- ❑ 玩具和游戏机等设备
- ❑ 支持小范围的基于无线通信的控制和自动化等领域

03

Zigbee协议栈

Zigbee协议栈

- ZigBee 的协议分为两部分：
 - ZigBee 联盟定义了网络层、安全层和应用层技术规范，
 - IEEE802.15.4 定义了物理层和 MAC 层技术规范，
- ZigBee 协议栈就是将各个层定义的协议都集合在一起，以函数的形式实现，并给用户提供一些应用层 API,供用户调用。



Zigbee协议栈

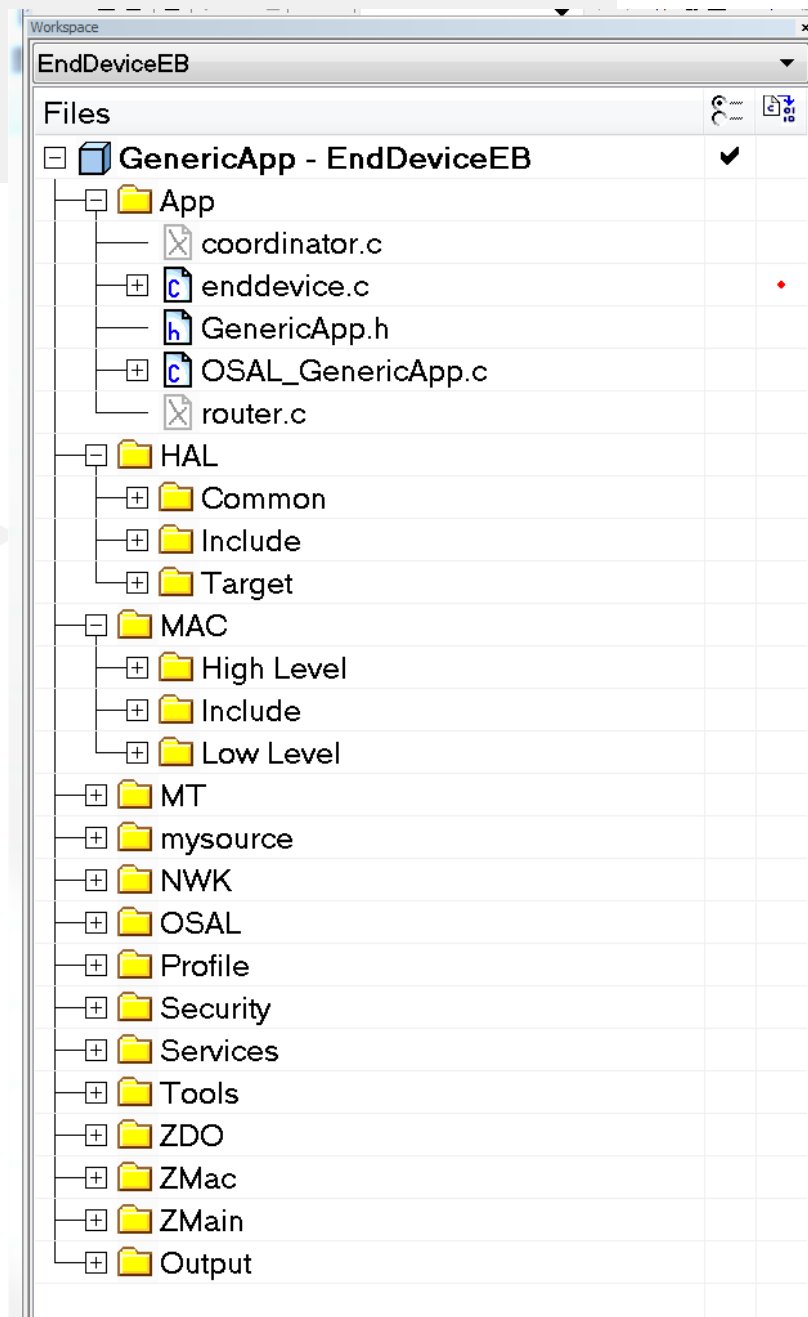
- ZigBee 协议栈具有很多版本，不同厂商提供的 ZigBee 协议栈有一定的区别。
- 虽然协议是统一的，但是协议的具体实现形式是变化的，即不同厂商提供的协议栈是有区别的
 - 函数名称和参数列表有区别
 - 学习厂商提供的 Demo 演示程序和说明文档学习

Zigbee协议栈安装

- ZigBee 协议栈 ZStack-CC2530-2.5.1a 要安装以后才能使用。
- 协议栈安装文件目录：
 - 物联网实训项目所有资料\环境工具\ZStack-CC2530-2.5.1a\ZStack-CC2530-2.5.1a.exe
- 双击 ZStack-CC2530-2.5.1a.exe,即可进行协议栈的安装，默认是安装到 c 盘。
- 然后在路径：
 - C:\Texas Instruments\ZStack-CC2530-2.5.1a\Projects\zstack\Samples\GenericApp\CC2530DB
- 下找到 GenericApp.eww打开该工程

Zigbee协议栈

- 一口君已经给出的实例都已经包括了协议栈的代码，找到如下目录：
 - ZStack-CC2530-2.5.1a\Projects\zstack\Samples\GenericApp\CC2530DB



Zigbee协议栈使用

- 使用 ZigBee 协议栈进行开发的基本思路可以概括为如下三点：
 - 1、用户对于 ZigBee 无线网络的开发就简化为应用层的 c 语言程序开发，不需要深入研究复杂的 ZigBee 协议栈；
 - 2、 ZigBee 无线传感器网络中数据采集，只需用户在应用层加入传感器的读取函数即可；
 - 3、如果考虑节能，可以根据数据采集周期进行定时，定时时间到就唤醒 ZigBee 的终端节点，终端节点唤醒后，自动采集传感器数据，然后将数据发送给路由器或者直接发给协调器。

04

Zigbee协议栈函数

Zigbee协议栈初始化

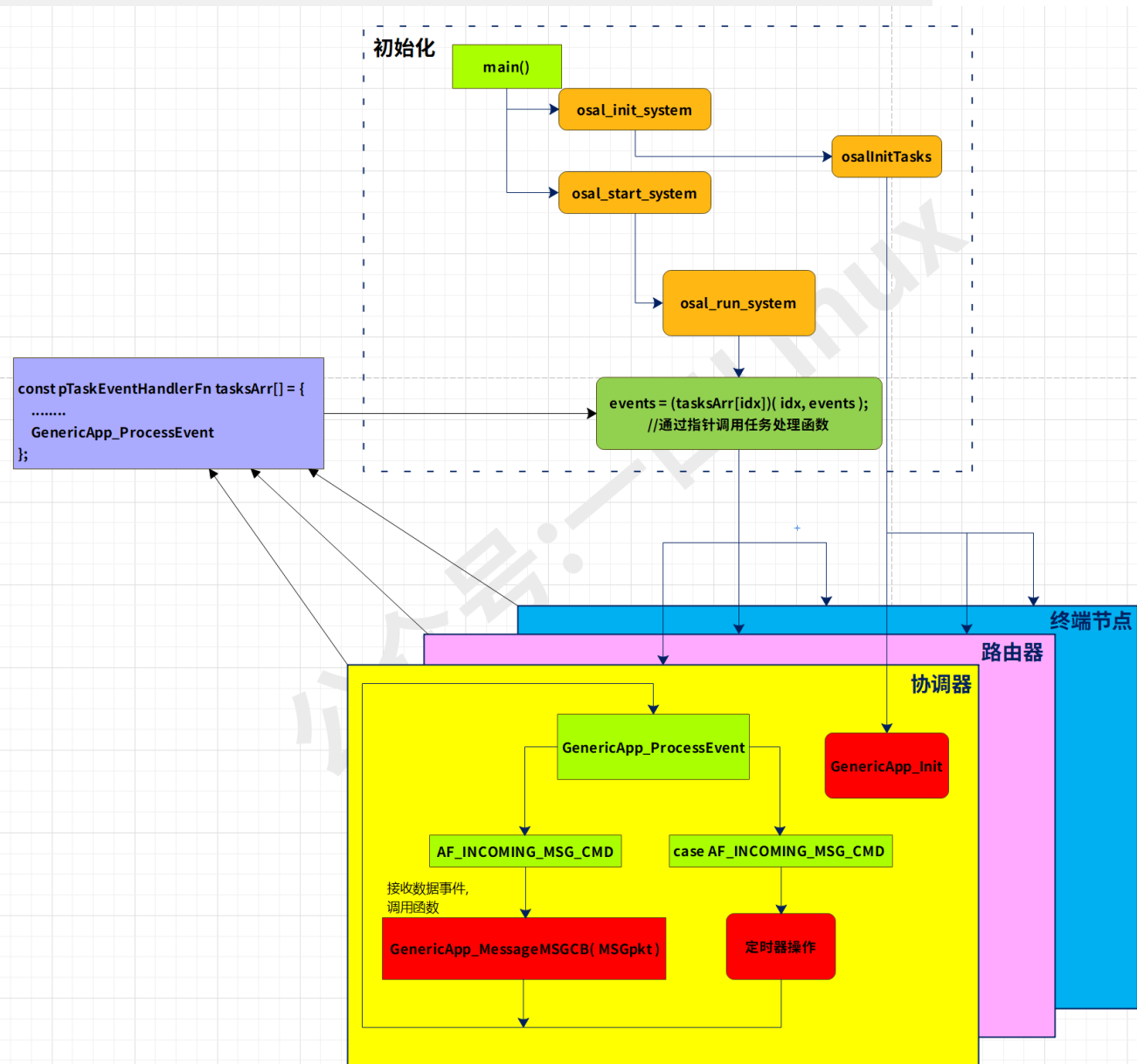
- 物联网实训项目所有资料\4. 知识点相关资料\3.zigbee\1.zigbee协议栈入门.pdf

Main()
Projects\zstack\ZMain\TI2530DB\ZMain.c

osalInitTasks()
Projects\zstack\Samples\GenericApp\Source\OSAL_GenericApp.c

tasksArr()
Projects\zstack\Samples\GenericApp\Source\OSAL_GenericApp.c

关注公众号



AF_DataRequest

- 无线发送函数
 - 向zigbee网络发送函数都调用该函数
 - 其中cID对应的就是函数GenericApp_MessageMSGCB()接收到的数据pkt->clusterId

```
if(AF_DataRequest( &GenericApp_DstAddr, &GenericApp_epDesc,
    GENERICAPP_CLUSTERID,
    1,
    &cmd,
    &GenericApp_TransID,  //0
    AF_DISCV_ROUTE, AF_DEFAULT_RADIUS )==afStatus_SUCCESS)
```

```
/* ***** */
* @fn      AF_DataRequest
*
* @brief   Common functionality for invoking APSDE_DataReq() for both
*          SendMulti and MSG-Send.
*
* input parameters
*
* @param   *dstAddr - Full ZB destination address: Nwk Addr + End Point.
* @param   *srcEP - Origination (i.e. respond to or ack to) End Point Descr.
* @param   cID - A valid cluster ID as specified by the Profile.
* @param   len - Number of bytes of data pointed to by next param.
* @param   *buf - A pointer to the data bytes to send.
* @param   *transID - A pointer to a byte which can be modified and which will
*                   be used as the transaction sequence number of the msg.
* @param   options - Valid bit mask of Tx options.
* @param   radius - Normally set to AF_DEFAULT_RADIUS.
*
* output parameters
*
* @param   *transID - Incremented by one if the return value is success.
*
* @return  afStatus_t - See previous definition of afStatus_... types.
*/
uint8 AF_DataRequestDiscoverRoute = TRUE;
afStatus_t AF_DataRequest( afAddrType_t *dstAddr, endPointDesc_t *srcEP,
    uint16 cID, uint16 len, uint8 *buf, uint8 *transID,
    uint8 options, uint8 radius )
{
    .
}
```

变量	含义
afAddrType_t *dstAddr	发送目的地址 + 端点地址和传送模式
endPointDesc_t *srcEP	源(答复或确认)终端的描述（比如操作系统中任务 ID 等）源 EP
uint16 cID	被 Profile 指定的有效的集群号
uint16 len	发送数据长度
uint8 *buf	发送数据缓冲区
uint8 *transID	任务 ID 号
uint8 options	有效位掩码的发送选项
uint8 radius	传送跳数，通常设置为 AF_DEFAULT_RADIUS

GenericApp_MessageMSGCB()

• 接收数据函数

- 无线信道接收到的数据都会调用到该函数

```
typedef struct
{
    osal_event_hdr_t hdr; /* OS event header */
    uint16 groupId; /* Message group ID */
    uint16 clusterId; /* Message cluster ID */
    afAddrType_t srcAddr; /* Source address */
    int16 destAddr; /* Destination address */
    uint16 macDestAddr; /* MAC destination address */
    uint8 endPoint; /* Destination endpoint */
    uint8 wasBroadcast; /* Was broadcast */
    uint8 LinkQuality; /* Link quality */
    uint8 correlation; /* Correlation */
    int8 rssi; /* Received signal strength indicator */
    uint8 SecurityUse; /* Security use */
    uint32 timestamp; /* Timestamp */
    uint8 nwkSeqNum; /* Network sequence number */
    afMSGCommandFormat_t cmd; /* Command format */
} afIncomingMSGPacket_t;
```

```
typedef struct
{
    uint8 TransSeqNumber;
    uint16 DataLength;
    uint8 *Data;
} afMSGCommandFormat_t;
```

```
#define GENERICAPP_CLUSTERID 0
#define GENERICAPP_CLUSTERID1 1 // 节点1
#define GENERICAPP_CLUSTERID2 2 // 节点2
#define GENERICAPP_CLUSTERID3 3
#define GENERICAPP_CLUSTERID4 4
```

```
// 接收数据，参数为接收到的数据
void GenericApp_MessageMSGCB( afIncomingMSGPacket_t *pkt )
{
    uint16 flashTime;
    byte buf[3];
    switch ( pkt->clusterId ) // 设备ID
    {
        case GENERICAPP_CLUSTERID: // 收到广播数据
            .....
            break;
        case GENERICAPP_CLUSTERID1:
        case GENERICAPP_CLUSTERID2:
        case GENERICAPP_CLUSTERID3:
        case GENERICAPP_CLUSTERID4:
            HalUARTWrite(0, pkt->cmd.Data, 24);
            break;
    }
}
```

如何给串口注册回调函数

- 作为协调器，需要通过串口从上位机读取数据，可以通过右边方法注册回调函数

- 波特率：115200
- 流控：无
- 收发缓冲区：255
- 回调函数：rxCB

void GenericApp_Init(uint8 task_id)

关注公众号

```
halUARTCfg_t uartConfig;  
uartConfig.configured = TRUE;  
uartConfig.baudRate = HAL_UART_BR_115200;  
uartConfig.flowControl = FALSE;  
uartConfig.flowControlThreshold = 1;  
uartConfig.rx.maxBufSize = 255;  
uartConfig.tx.maxBufSize = 255;  
uartConfig.idleTimeout = 1;  
uartConfig.intEnable = TRUE;  
uartConfig.callBackFunc = rxCB;  
HalUARTOpen (HAL_UART_PORT_0, &uartConfig);
```

```
static void rxCB(uint8 port,uint8 event)  
{  
    uint8 uartbuf[4] = {0};  
    uint8 cmd;  
    HalUARTRead(0,&cmd,3); //从串口读取3个字节的数据到cmd中  
    sprintf(uartbuf,"cmd:%x",cmd);  
    //显示到Lcd屏幕  
    HalLcdWriteString(uartbuf,HAL_LCD_LINE_4);  
    if(AF_DataRequest( &GenericApp_DstAddr, &GenericApp_epDesc,  
        GENERICAPP_CLUSTERID,  
        3, //发送两个字节的数据  
        &cmd, //发送的数据内容，其实就是从串口收到的3个字节的数据  
        &GenericApp_TransID,  
        AF_DISCV_ROUTE, AF_DEFAULT_RADIUS )==afStatus_SUCCESS)  
    {  
    }  
}
```

如何添加定时器功能

- 我们希望终端节点可以定时上传传感器数据，这就需要定时器操作。

- 思路如下：

- 1. ZDO层更改网络状态后就启动定时器
- 2. 主任务处理函数
GenericApp_ProcessEvent()中，增加超时事件处理逻辑，上传传感器数据
- 3. 通过函数osal_start_timerEx()重新启动定时器

开启定时器函数

```
uint8 osal_start_timerEx( uint8 taskID, uint16 event_id, uint16 timeout_value )
taskID      : 启动定时器的任务ID
event_id    : 需要通知的事件
timeout_value : 超时时间
```

实例

```
osal_start_timerEx( GenericApp_TaskID,
                    GENERICAPP_SEND_MSG_EVT,
                    GENERICAPP_SEND_MSG_TIMEOUT );
```

05

命令下发

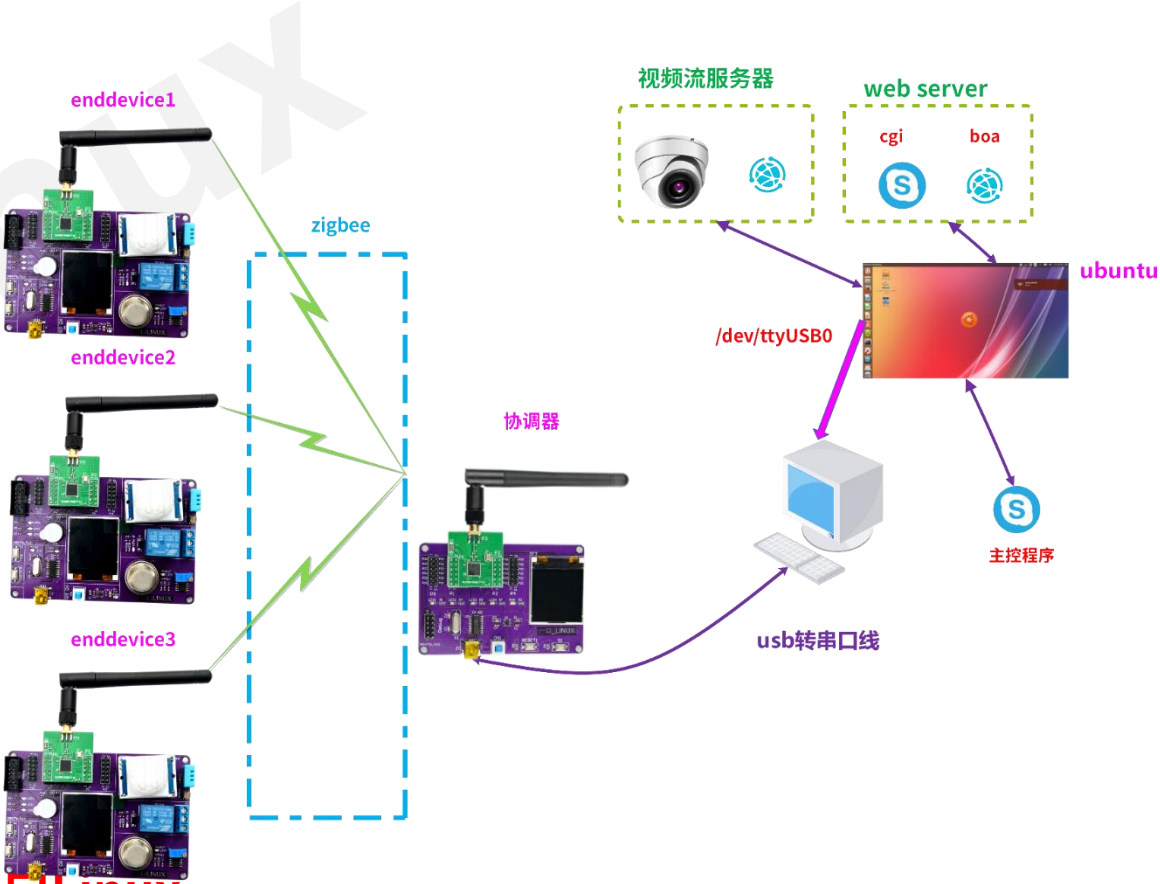
串口命令格式

7	6	5	4	3	2	1	0
节点编号		设备编号		操作设备			

设备编号	设备名字	操作掩码	操作
0x00	继电器	0x00	关闭
		0x01	打开
0x10	蜂鸣器	0x00	关闭
		0x01	打开
		0x02	自动报警关闭
		0x03	自动报警打开
0x20	LED	0x00	关闭
		0x01	打开

举例：设置节点1
0100 0001 0x41 rely on
0100 0000 0x40 rely off
0101 0001 0x51 beep on
0101 0000 0x50 beep off
0110 0001 0x61 led on
0110 0000 0x60 led off

节点编号	节点名
0x40	1号节点
0x80	2号节点
0xc0	3号节点



终端节点收到命令后的处理

```
static void GenericApp_MessageMSGCB( afIncomingMSGPacket_t *pkt )
{
    uint8_t cmd = 0;
    char buf[10];
    switch ( pkt->clusterId )
    {
        case GENERICAPP_CLUSTERID: //协调器发来数据

            cmd = *(pkt->cmd.Data);
            //显示到lcd屏幕

            if((cmd & STO_MASK) == (ENDDEVICE_ID << 6)) // 仓库号正确
            {
                sprintf(buf,"cmd:%x",cmd);

                HalUARTWrite(0, buf,strlen(buf));
                HalLcdWriteString(buf,HAL_LCD_LINE_4);
                switch(cmd & DEV_MASK) // 判断设备号
                {
                    case RELY: // 排风扇命令字
                        // 风扇命令处理函数
                        RelyCtrl(cmd & CMD_MASK);
                        //HalLcdWriteString("rely",HAL_LCD_LINE_5);
                        break;
                    case BEEP: // 蜂鸣器命令字
                        // 蜂鸣器命令处理函数
                        BeepCtrl(cmd & CMD_MASK);
                        break;
                    case LED: // 照明灯命令字
                        // 照明灯命令处理函数
                        LedCtrl(cmd & CMD_MASK);
                        break;
                    default :
                        break;
                }
            }
            break;
    }
}
```






















关注公众号：一口Linux

06

数据采集

传感器采集驱动代码

ZStack-CC2530-2.5.1a\Projects\zstack\Samples\GenericApp\Source

 beep.c	2022/5/6 21:31
 beep.h	2022/5/6 20:01
 coordinator.c	2022/5/28 12:06
 dht11.C	2022/5/16 14:37
 dht11.H	2022/5/7 21:12
 enddevice.c	2022/5/25 20:52
 Gas.c	2022/5/6 20:22
 Gas_head.h	2015/1/9 15:10
 GenericApp.h	2022/5/25 20:45
 head.h	2022/5/6 20:50
 Human_Body_infrared.c	2022/5/6 20:18
 Infrare_head.h	2015/1/8 19:19
 led.c	2022/5/6 19:56
 led.h	2015/1/8 19:19
 light.c	2022/5/6 20:22
 light.h	2015/1/9 15:16
 OSAL_GenericApp.c	2022/5/28 11:36
 relay.c	2022/5/6 20:48
 relay.h	2022/5/6 20:47
 router.c	2012/3/7 2:04
 Temperature_head.h	2022/5/6 20:08

定时上传

```
375:
376:     case ZDO_STATE_CHANGE:
377:         GenericApp_NwkState = (devStates_t)(MSGpkt->hdr.status);
378:         if ( (GenericApp_NwkState == DEV_ZB_COORD)
379:             || (GenericApp_NwkState == DEV_ROUTER)
380:             || (GenericApp_NwkState == DEV_END_DEVICE) )
381:         {
382:             // Start sending "the" message in a regular interval.
383:             osal_start_timerEx( GenericApp_TaskID,
384:                                GENERICAPP_SEND_MSG_EVT,
385:                                GENERICAPP_SEND_MSG_TIMEOUT );
386:             show_title();
387:         }
```

GenericApp_ProcessEvent ()

```
407: if ( events & GENERICAPP_SEND_MSG_EVT )
408: { // 定时器，每隔1秒发送一次传感器数据信息
409:     SyncData(ENDDEVICE_ID);
410:
411:     memcpy(sendbuf, &EnvMsg, 24);
412:     AF_DataRequest( &GenericApp_DstAddr, &GenericApp_epDesc,
413:                    GENERICAPP_CLUSTERID1,
414:                    24, |
415:                    (uchar *)&EnvMsg,
416:                    &GenericApp_TransID,
417:                    AF_DISCV_ROUTE, AF_DEFAULT_RADIUS );
418:
419:
420:     // Setup to send message again
421:     osal_start_timerEx( GenericApp_TaskID,
422:                        GENERICAPP_SEND_MSG_EVT,
423:                        ENDDEVICE_SEND_MSG_TIMEOUT );
424:
425:     // return unprocessed events
426:     return (events ^ GENERICAPP_SEND_MSG_EVT);
427: } « end if events&GENERICAPP_SEN... »
```

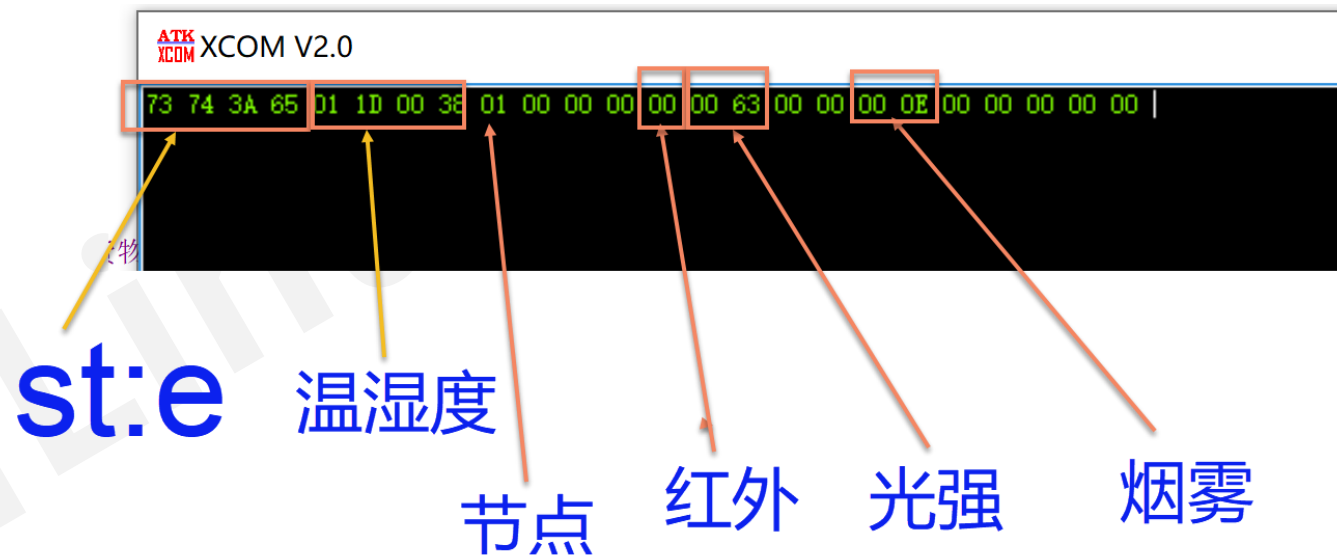
采集数据格式说明

```
struct _EnvMsg
{
    uint8_t head[3];    //标识位st:73 74 3A 65
    uint8_t type;        //数据类型

    uint8_t temp[2];    //温度
    uint8_t hum[2];     //湿度
    uint8_t ep_no;      //节点编号01

    int8_t x;           //三轴信息
    int8_t y;
    int8_t z;

    unsigned int hongwai; //01 00
    unsigned int lux;     //光照2个字节
    unsigned int rsv2;
    unsigned int gas;     //烟雾
    unsigned int rsv3;
    unsigned int adc;
};
```

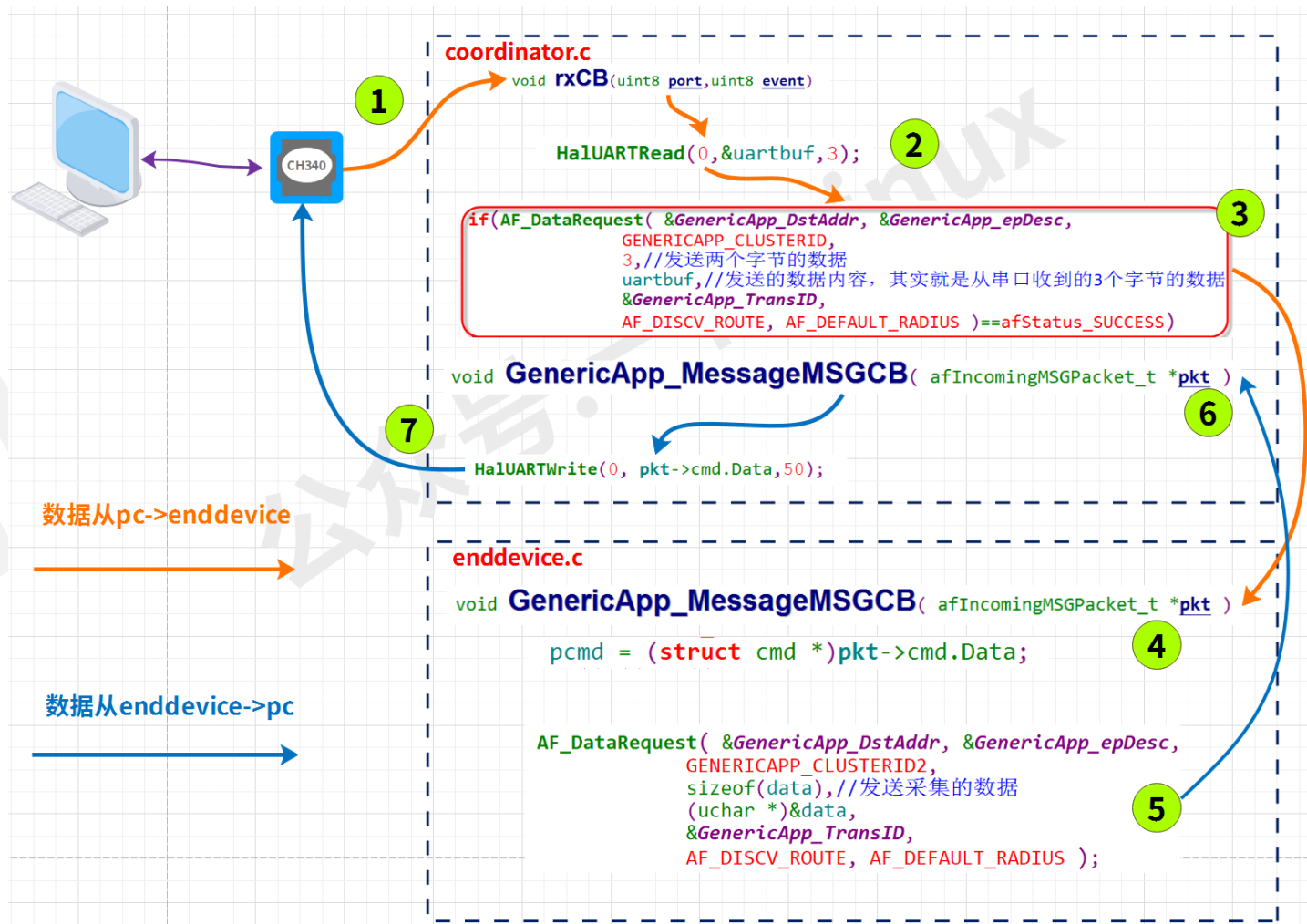


07

完整数据流程

上位机与无线节点通信函数调用流程

- 1.上位机通过串口发送数据给协调器;
- 2.触发协调器串口回调函数`rxCB()`,通过`HalUARTRead()`从串口提取出数据;
- 3.协调器通过`AF_DataRequest()`将数据发送给网络中其他设备,通过`id:GENERICAPP_CLUSTERID`,表明改数据是协调器发送;
- 4.触发终端节点回调函数`GenericApp_MessageMSGCB()`,协调器`id`存放在`pkt->clusterId`中,发送的数据存放在`pkt->cmd.data`中;
- 5.终端节点可以采集传感器数据,发送数据在缓冲区`data`中,然后通过函数`AF_DataRequest()`发送数据,填充`id:GENERICAPP_CLUSTERID2`,表示数据是节点2发送;
- 6.触发协调器的回调函数`GenericApp_MessageMSGCB()`;
- 7.协调器提取出终端节点发送的数据`pkt->cmd.data`,通过`HalUARTWrite()`发送给上位机。





更多嵌入式Linux知识
请关注一口君的公众号：一口Linux

公众号：一口Linux