哎呀研究这个数据的发送和收发研究了 2 天了。今天终于把困扰我很久的一个问题给解决了。

问题：终端节点启动会为什么会自动的发送数据呢？

解决过程：这个过程可是异常的艰辛。要解决这个问题。咱们先聊聊这个整个

zigbee 协议栈的工作流程。

程序肯定都是从 main 函数开始的，这个肯定也不例外。大家查看一下 main 函数主要就是关闭中断，检查电源电压是否够高，还有就是初始化了，什么物理层，mac 层等等。。而我们在这里关注 2 个函数就好了。第一个是：osal_init_system();第二个：osal_start_system();

第一个 osal_init_system()函数就是初始化与系统运行相关的一些东西如：初始化内存分配系统，初始化消息队列，初始化定时器，初始化电源管理系统，初始化第一块堆，最后一个就是我们要讲的一个非常重要的函数：osalInitTasks();初始化任务函数

```
void osalInitTasks( void )//系统任务初始化函数
{
  uint8 taskID = 0;
  //这个指针指向了所有任务空间的首地址
  tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);//这个 tasksEvents 指针总共有多少个数据空间，其实总共有多少任务就有多少个空间。
  osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));
  macTaskInit( taskID++ );    //mac 层的任务是 0
  nwk_init( taskID++ );    //网络层的任务是 1
  Hal_Init( taskID++ );    //物理层的任务号是 2
#if defined( MT_TASK )
  MT_TaskInit( taskID++ );//串口的任务
#endif
  APS_Init( taskID++ );
#if defined ( ZIGBEE_FRAGMENTATION )
  APSF_Init( taskID++ );
#endif
  ZDApp_Init( taskID++ );
#if defined ( ZIGBEE_FREQ_AGILITY ) || defined ( ZIGBEE_PANID_CONFLICT )
  ZDNwkMgr_Init( taskID++ );
#endif
  GenericApp_Init( taskID );//应用程序的初始化。并且得到这个任务的 ID 号。
}
```

大家可以看到有 2 个非常重要的变量：taskID,tasksEvents[sizeof(tasksArr) / sizeof(tasksArr[0]);

第 1 个 taskID 被初始化为 0，然后赋值给 mac 层，以后就赋值一次加 1，所以每一个任务都会得到属于自己 taskID 的值。而那个值就是自己在系统中对应的任务号。而有必要谈一下就是每一个任务对应下面可以有 16 个事件，但是有一个事件被系统给强制应用了，就是 SYS_EVENT_MSG = 0x8000.其它 15 个事件就可以自己定义了。

SYS_EVENT_MSG：消息等待事件。比如自己这一层，或者说这个任务有对应的消息传来。就会把这个事件给置位。或者干脆说这个事件是一直被置位的，而你在你的任务处理函数中可以选择是否检测这个事件。如果你检测了，而又想得到属于自己层的消息你可以调用函数 MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive(GenericApp_TaskID);来获取自己的消息，在通过判断 MSGpkt->hdr.event 这个事件类型来判断进行相应的处理。

第 2 个 taskEvents[]这个数组的容量其实等于你总任务的个数。而每一个对应的变量存放的就是这个任务里面对应的事件。比如 taskEvents[0]=0x8000;就是任务 0 里面的消息等待事件。这里有个东西提一下，这个 16 为的数每一位表示一个事件，一般不进行组合，所以最多有 16 个事件。

这个函数的目地就是给每一个任务分配任务 ID 然后创建一个任务数组存放各自任务现在对应的事件。

第二个函数 osal_start_system();这个函数才是操作系统的核心。里面有个死循环 for(;;)

```
  for(;;)   // Forever Loop
#endif
  {
    uint8 idx = 0;

    osalTimeUpdate();
    Hal_ProcessPoll();   // This     replaces MT_SerialPoll() and osal_check_timer().

    do {
      if (tasksEvents[idx])   // Task is highest priority that is ready.
      {
        break;//当不是 0  的时候就直接返回。
      }
    } while (++idx < tasksCnt);

    if (idx < tasksCnt)//说明有任务发生了，要去做
    {
      uint16 events;
      halIntState_t intState;
①
      HAL_ENTER_CRITICAL_SECTION(intState);
      events = tasksEvents[idx];       //当是传感器的发送报道的时候传送的肯定是 MY_REPORT_EVT
      tasksEvents[idx] = 0;   // Clear the Events for this task.
      HAL_EXIT_CRITICAL_SECTION(intState);
②
      events = (tasksArr[idx])( idx, events );//调用任务处理函数
③
      HAL_ENTER_CRITICAL_SECTION(intState);
      tasksEvents[idx] |= events;   // Add back unprocessed events to the current task.
      HAL_EXIT_CRITICAL_SECTION(intState);
    }
#if defined( POWER_SAVING )
    else   // Complete pass through all task events with no activity?
    {
      osal_pwrmgr_powerconserve();   // Put the processor/system into sleep
    }
#endif
  }
```

在 do while 里面去检测(tasksEvents[idx]是不是 0，也就是现在有没有事件发生事件，当然顺序就是从 idx=0 开始。也就是说从 0 开始。当有事件发生就跳出循环执行下面的语句。下面的语句很简单就是①先获取这个事件。②调去任务处理函数。③传出来的 events 这个变量已经把刚做过的那个事件取消掉了。但是任然要把这个值给当前任务。因为这个任务里面可能还有其它事件。就这样循环把第一个任务的所有事件做完，接着做第二个任务的事件。。。知道把所有任务做一遍，然后不停的循环。这也即使操作系统的工作机制。

接下来我们要讲第三个比较重要的东西：events = (tasksArr[idx])(idx, events);

第三个：tasksArr[]数组。

下面来看看这个数组的定义：

```
const pTaskEventHandlerFn tasksArr[] = {
  macEventLoop,     //mac 层的事件循环      你去追踪一下你就知道他没有被定义
  nwk_event_loop,//网络层的事件循环      你去追踪一下你就知道他没有被定义
  Hal_ProcessEvent,//物理层的事件处理事件
#if defined( MT_TASK )
  MT_ProcessEvent,
#endif
  APS_event_loop,
#if defined ( ZIGBEE_FRAGMENTATION )
  APSF_ProcessEvent,
#endif
  ZDApp_event_loop,
#if defined ( ZIGBEE_FREQ_AGILITY ) || defined ( ZIGBEE_PANID_CONFLICT )
  ZDNwkMgr_event_loop,
#endif
  GenericApp_ProcessEvent    //这个是应用程序自己定义的任务事件。
};
```

```
typedef unsigned short (*pTaskEventHandlerFn)( unsigned char task_id, unsigned short event );
```

可以看出来它里面都是回调函数，传递的参数那就是任务 ID 和事件的集合。这样大家就知道调去任务处理函数的机制了吧。

讲完上面的这 3 个函数哪里。那么我们再来讲讲数据的发送和接收就比较简单了。

点对点数据发送和接收的机制

由上面的操作系统和任务处理函数的运行机制。我们再来理解这个数据发送和接收的机制就很容易了。

准备工作：

首先我们的数据发送放在了应用层这个任务里面了。所以我们就知道需要 2 个函数了。一个是应用层的初始化函数，应用层任务处理函数。我们可以在 Enddevice.c 定义，并且把 GenericApp_Init(byte task_id)函数放在 osalInitTasks(void)函数下，例如这样：

```
void osalInitTasks( void )
 {
      。。。。。
      GenericApp_Init( taskID );//放在最后即可
}
```

再把任务处理函数放在 tasksArr[]数组下：例如：

```
const pTaskEventHandlerFn tasksArr[]= {
。。。
GenericApp_ProcessEvent//放在最后即可
} ;
```

运行：

运行的时候不是我们已经讲了吗。因为有个事件是一定有的就是 SYS_EVENT_MSG 消息等待事件。所以我们的策略就是，在任务处理函数里面检测 SYS_EVENT_MSG 这个事件，然后获取对应的消息。找到你想要的事件来启动数据的发送。对应代码：

```
UINT16 GenericApp_ProcessEvent(byte task_id,UINT16 events)
{
  afIncomingMSGPacket_t *MSGpkt;
  ①if(events&SYS_EVENT_MSG)
  {
    ②MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive(GenericApp_TaskID);
    while(MSGpkt)
    {
      switch(MSGpkt->hdr.event)
      {
    ③ case ZDO_STATE_CHANGE://ZDO 改变了设备的网络状态   这样做会不会意味着只能进行一次发送呢？
        /*需要修改的地方*/
        GenericApp_NwkState = (devStates_t)(MSGpkt->hdr.status);//读取节点的设备类型
        if(GenericApp_NwkState == DEV_END_DEVICE)//对节点设备判定，如果是终端节点，执行下一行。
        {
        ④   GenericApp_SendTheMessage();//实现 90 无线数据的发送。
        }
        /**/
       break;
      default:break;
      }
     osal_msg_deallocate((uint8 *)MSGpkt);
     MSGpkt = (afIncomingMSGPacket_t*)osal_msg_receive(GenericApp_TaskID);//循环的接收设备的消息。知道通过
switch 是网络状态的改变
    }
    return (events^SYS_EVENT_MSG);
  }  return 0;
}
```

所以我们需要关心的就是这四行语句：

①来查看消息等待事件是否发生。发生做下面代码。

②获取消息。在下面 while 循环里检测消息的事件类型

③当消息事件类型是网络状态的改变时。也就是刚加入 zigbee 无线网的时候。下面的就是检测是不是端点设备如果是做④

④调去数据发送函数完成发送。

接下来我们就看看数据发送函数吧：

```
/*消息发送*/
void GenericApp_SendTheMessage(void)
{
    unsigned int x,y;
  unsigned char theMessageData[4] = "LED";//用于存放要发送的数据
  /*定义一个 afAddrType_t 类型的变量你，因为数据发送函数 AF_DataRequest
    的第一个参数就是这种类型的变量。
    afAddrType_t 类型定义如下：
    typedef   struct
  {
    union
    {
      uint16   shortAddr;   //短地址，也是网络地址      常用
      ZLongAddr_t extAddr;//
    )addr;
    afAddrMode_t addrMode;//发送数据用的方式是单播，广播或者多播的方式    常用
    byte endPoint;      //常用
    uint16 panId;
  }afAddrType_t;
  */
  afAddrType_t my_DstAddr;
  my_DstAddr.addrMode = (afAddrMode_t)Addr16Bit;//Addr16Bit 表示的是 2，枚举变量，单播
  my_DstAddr.endPoint = GENERICAPP_ENDPOINT;//初始化端口号   10
  my_DstAddr.addr.shortAddr = 0x0000;//协调器的网络地址是固定的，为 0x0000，因此向协调器发送时，可以直接指定协调器的网络地址。

 /*添加的测试函数*/
  led1 = 0;
   led2 = 0;
   for(x=1000;x>0;x--)   //恰当的延时
     for(y=110;y>0;y--);
   led1 = 1;
   led2 = 1;
   for(x=1000;x>0;x--)
     for(y=110;y>0;y--);
 /*测试结束
   测试结果，从测试开始只进行了一次数据的发送。所以可以证明 ZDO_STATE_CHANGE 这个状态只改变了一次，就是加入网络后。*/

  /*数据发送函数*/
  AF_DataRequest(&my_DstAddr, //对于地址和那种模式方式发送
                 &GenericApp_epDesc,//是端口描述符。
                 GENERICAPP_CLUSTERID,//我想的是节点发送的簇的个数？
                 3,//发送数据的长度
                 theMessageData,//发送数据缓冲区的指针
                 &GenericApp_TransID,//数据发送序列号
                 AF_DISCV_ROUTE,   //?可能是表示的一些数据的类型
                 AF_DEFAULT_RADIUS//?
                 );
//   HalLedBlink(HAL_LED_2,0,50,500);//调用 HalLedBlink 函数，使终端节点 LED2 闪烁。这个地方其实你可以改成其它的代码。甚至可以直接不管协议栈。用你会的直接解决。就是对端口的设置。
//   HalLedBlink(HAL_LED_1,0,50,500);//这个只是让实验板上不同的灯量。
 //    HalLedBlink1();
}
```

刚开始都是最一些消息的初始化。最后调用原语发送

AF_DataRequest(&my_DstAddr,

```
                    &GenericApp_epDesc,//是端口描述符。
                    GENERICAPP_CLUSTERID,//我想的是节点发送的簇的个数？
                    3,//发送数据的长度
                    theMessageData,//发送数据缓冲区的指针
                    &GenericApp_TransID,//数据发送序列号
                    AF_DISCV_ROUTE,   //?可能是表示的一些数据的类型
                    AF_DEFAULT_RADIUS//?
                    );
```

这就是发送的机制。接收的机制雷同只不过消息里面检测的不 ZDO_STATE_CHANGE: 事件而是 AF_INCOMING_MSG_CMD = 0x1A 接收到新数据封装的消息的事件。然后调去处理函数进行相应的处理。

刚才我们只是介绍了数据能够发送和接收的机制。没有具体到发送数据的一些数据格式和消息包的格式。下面我们就来看看这些东西。

数据发送的具体内容

```
/*数据发送函数*/
  AF_DataRequest(&my_DstAddr, //对于地址和那种模式方式发送
                    &GenericApp_epDesc,//是端口描述符。
                    GENERICAPP_CLUSTERID,//我想的是节点发送的簇的个数？
                    3,//发送数据的长度
                    theMessageData,//发送数据缓冲区的指针
                    &GenericApp_TransID,//数据发送序列号
                    AF_DISCV_ROUTE,   //?可能是表示的一些数据的类型
                    AF_DEFAULT_RADIUS//?
                    );
```

```
afStatus_t AF_DataRequest(
                    afAddrType_t *dstAddr, //目地节点的网络地址以及数据发送格式，例广播
                    endPointDesc_t *srcEP,//端口 ，具体的端口，需要对应的端口描述符
                    uint16 cID,//命令号， 比如是 1 亮 LED，0 关闭 LED
                    uint16 len, //发送数据的长度
                    uint8 *buf, //数据缓冲区的指针
                    uint8 *transID,//发送序列的指针，就是对应的序列号
                    uint8 options, //默认？
                    uint8 radius //默认？
                    )
```

既然我们知道了，发送数据的数据格式，那么接收端会把它封装成什么样子呢？

收据接收的具体内容

我想说的是你看啊，接收的数据是放在消息里面了，对吧，所以我们肯定去消息的结构体中去找了，下面这个是消息的结构体：

```
typedef struct
{
  osal_event_hdr_t hdr;        /* OSAL Message header OSAL 的消息头 */
/*typedef struct
{
  uint8   event;
  uint8   status;
} osal_event_hdr_t;*/
```

```
  uint16 groupId;              /* Message's group ID - 0 if not set 消息的组 ID 如果没有被设置默认为 0*/
  uint16 clusterId;            /* Message's cluster ID 消息的簇（集群）ID */
  afAddrType_t srcAddr;        /* Source Address, if endpoint is STUBAPS_INTER_PAN_EP,源地址，如果端点号是 ，地址类型   源地址类型
                                  it's an InterPAN message */
  uint16 macDestAddr;          /* MAC header destination short address 目的地 MAC 短地址 */
  uint8 endPoint;              /* destination endpoint 目标端点 */
  uint8 wasBroadcast;          /* TRUE if network destination was a broadcast address 如果网络的目的地是一个广播地址 */
  uint8 LinkQuality;           /* The link quality of the received data frame 接收到的数据帧的链接质量 */
  uint8 correlation;           /* The raw correlation value of the received data frame 接收到的数据帧的原始关联值*/
  int8  rssi;                  /* The received RF power in units dBm 接收到的射频功率*/
  uint8 SecurityUse;           /* deprecated 弃用*/
  uint32 timestamp;            /* receipt timestamp from MAC 从 MAC 收到时间戳*/  收到的时间标记
  afMSGCommandFormat_t cmd; /* Application Data 应用数据
// Generalized MSG Command Format
typedef struct
{
  byte    TransSeqNumber;
  uint16 DataLength;                  // Number of bytes in TransData
  byte   *Data;
} afMSGCommandFormat_t;*/

} afIncomingMSGPacket_t;    //无线数据包格式结构体。
```

把 afAddrType_t srcAddr 摘取出来，看看表示什么：

```
typedef struct
{
  union
  {
    uint16        shortAddr;//短地址 16 位网络地址
    ZLongAddr_t extAddr; // byte ZLongAddr_t[Z_EXTADDR_LEN];Z_EXTADDR_LEN=8  总共 64 位 IEEE 地址
  } addr;
  afAddrMode_t addrMode; //地址模式，
  byte endPoint;      //端点
  uint16 panId;   // used for the INTER_PAN feature 网络的 ID
} afAddrType_t;

同样给出地址模式：
typedef enum
{
  afAddrNotPresent = AddrNotPresent,   //无地址
  afAddr16Bit         = Addr16Bit,          //短地址
  afAddr64Bit         = Addr64Bit,          //64 位地址
  afAddrGroup         = AddrGroup,          //组播地址
  afAddrBroadcast   = AddrBroadcast      //广播地址
} afAddrMode_t;
```

大家如果仔细的去看，就知道最主要的就是我把它们编程蓝色的部分了。

osal_event_hdr_t hdr; 这个是 OSAL 消息的头了，这个结构体如下：

```
typedef struct
{
  uint8   event;       //这个是表示的事件
  uint8   status;      //这个是表示的是状态了。
} osal_event_hdr_t;
```

大家可以看到这个结构里面一个表示事件，一个表示状态。而如果是无线网接收到的数据封装成的消息，系统会把该事件标志成 AF_INCOMING_MSG_CMD = 0x1A 它的值是不可以改变的。

uint16 clusterId; 这个大家肯定不陌生，这个因为我们在发送的时候

afStatus_t AF_DataRequest 里面的 uint16 cID,这一项。表示的是一个命令是吧。而我们再确定是接收到的事件是 AF_INCOMING_MSG_CMD 之后。我们就可以根据不同的命令去做不同的动作。所以这个东西也很重要。

afMSGCommandFormat_t cmd; 这个是你的核心部分。它的结构体如下：

```
// Generalized MSG Command Format
typedef struct
{
  byte     TransSeqNumber; //表示的是发送的数据序列号
  uint16 DataLength;              // Number of bytes in TransData     数据长度单位字节
  byte    *Data;       //数据的指针
} afMSGCommandFormat_t;
```

大家看看啊，这里面的

接收	发送
TransSeqNumber	uint8 *transID,//发送序列的指针
DataLength	uint16 len,
*Data	uint8 *buf,

所以如果你需要上面的数据进行判断时可以这样干。

所以通过上面的讲解，总结出来的数据解析的步骤：

第一步：判断 MSGpkt->hdr.event 代码如下：代码只做参考

```
afIncomingMSGPacket_t *MSGpkt;//定义一个指向接收消息结构体的指针 MSGpkt
 if(events & SYS_EVENT_MSG)//消息等待事件。说明现在有消息被传过来了。
 {
   /*从消息队列中接收一个消息。*/
   MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive(GenericApp_TaskID);//用 osal_msg_receive 接收消息队列接收消息（无线数据包的指针）
   while(MSGpkt) //不空即可
   {
     switch(MSGpkt->hdr.event)//消息头的事件类型
     {
     case  AF_INCOMING_MSG_CMD:GenericApp_MessageMSGCB(MSGpkt);break;//完成对接收数据的处理，是核心函数  如果接收到来消息，则调用函数处理
     default:break;
     }
```

第二步：去判断 clusterId 是哪个具体的命令：代码只做参考

```
switch(pkt->clusterId)//消息的簇 ID GENERICAPP_CLUSTERID 这个其实就是一个命令吧。
  {
      case GENERICAPP_CLUSTERID:osal_memcpy(buffer,pkt->cmd.Data,3);
       if((buffer[0]=='L')||(buffer[1]=='E')||(buffer[2]=='D'))/   {
            HalLedBlink(HAL_LED_2,0,50,500);//闪烁。
          }
       else
       {
            HalLedSet(HAL_LED_2,HAL_LED_MODE_ON);//打开就好了
        }
       break;
  }
```

第三步：去接收具体的数据了，如数据，长度你，序列号。在上一步绿色部分。

这些东西我想已经完全理解数据的发送和接收吧。