TinyWSN 常见问题及回答

tzhong
2023/5/15

目录

TinyWSN 技术有什么特点？

TinyWSN 从设计初期就是面向无线传感器网络，从团队多年工程的实践经验，设定以下几个优先的设计目标：

- ✓ 支持节点数目多，节点数目可达 65536
- ✓ 支持多种组网方式，例如星型，树型和链型
- ✓ 节点支持低功耗，电池可以支持 10 年以上
- ✓ 支持双向通信，可由网络侧和节点发起通信
- ✓ 可靠通信质量，不随节点数目增加而恶化

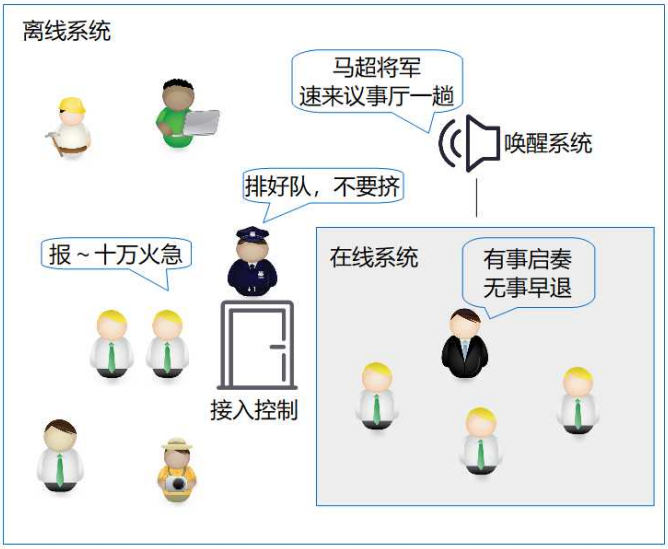
目前存在的一些多节点组网方式的优缺点

类型	示例	优缺点
同步	主从结构星型网，主节点周期轮询	优点是避免了冲突，但随着节点数目增加，节点的上报周期变动越来越大，而且大部分节点空闲的，并无新数据需要上报，浪费了带宽和电池电量，而需要上报的节点又被延迟了。
异步	主从结构星型网，从节点可以随机发送，有的完全没有冲突检查，例如 LORAWAN，有的会在发送前会加上一些冲突检查机制，常见的采用 先听后说，通过检查信号强度来推测信道的空闲	优点是通信高效，但随着节点数目增加，以及冲突检查的不可靠，冲突可能性大大增加，尤其是大量节点同时被告警触发后。而且这类网络往往无法直接由网络侧发起即时通信，只能缓存下行消息，等待节点自己主动接入。或者要求从节点长期处于接受状态，主要原因是由于主节点无法精确知道从节点的接收窗口，这都将导致从节点的功耗高。例如在 NB-IoT 设置的三种工作模式（DRX、eDRx 和 PSM），也是为了平衡下行通信和节点功耗而设置的。

而 TinyWSN 综合了上面两种方式的优点，采取分层的设计，在外围是一个离线的异步系统，大部分空闲的节点都处于这个系统中，只有当收到外部告警或无线唤醒时，才进入在线的同步系统，在同步系统中由路由模块负责无线资源的分配，当传输结束后，就退出恢复成离线节点，节省宝贵的无线资源。通过这样的设计，在数目，效率和功耗间取得一个很好的平衡。

下面举一个我们熟悉的一个日常生活的场景图的，它就很好地诠释的 TinyWSN 的工作模式。

TinyWSN 的场景图



下面就是场景图的一些说明，以及在 TinyWSN 系统中角色的类比：

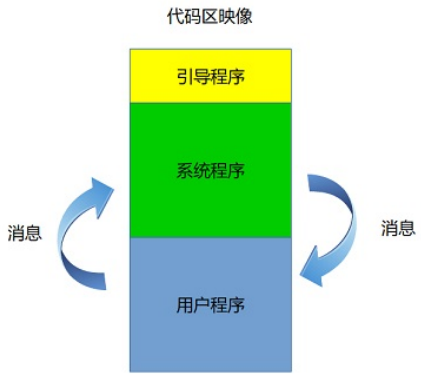
- ✓ 议事厅就是一个交流场地，里面有一个主持人（路由模块）主持会议，进入会议厅的人员（在线节点）人人都有位置，相互没有冲突
- ✓ 进入议事厅需要经过一个通道，有一个警察（接入控制）管理次序，在紧急状态下通道可能会冲突，但是这个冲突不会影响到议事厅的人员
- ✓ 外部的人员（离线节点）可以自由休息，所以他们消耗的能量（节点功耗）很低
- ✓ 当外部发现异常情况（告警唤醒），它们可以要求进入议事厅汇报
- ✓ 主持人如果有要紧事，可以通过喇叭把外部人员叫进来（无线唤醒）
- ✓ 议事厅是一个宝贵的资源，原则上是快进快出，尽快腾出空位给有急事的人

如果理解和熟悉了上面的场景图，就能更好地理解和使用 TinyWSN 系统，它也是从类似的场景出发，解决上面设定的技术目标。《TinyWSN 快速入门指南》系列文章中有具体的实战教程，可以动手实践上面描述的场景。

TinyWSN 的内嵌模式程序如何调试？

节点模块支持三种开发模式，其中一种是内嵌模式，如下图所示，系统程序和应用程序同时运行在在模块的控制器内，它们之间通过共享内存的消息接口互相通信，应用程序可以单独编译和加载，但是应用程序设计需要符合一定的规范，避免干扰到系统程序的时序， 具体

可以参见**《TinyWSN 内嵌模式手册》**。



下面是链接中提供的应用的开发例程的源码，可供参考：

https://gitee.com/tinywsn/fw-stm32l1-wbed-usr

✓ 调试输出

利用模块的串口，输出调试信息，由于串口设备是和系统程序共用，应用程序需要使用消息接口来使用串口，例如下面就是把字符串“hello,world” 发送到串口上:

net_msg_send(WIPC_MSG_LOG_TEXT, “hello,world”, 0, ctx);

✓ 调试命令

模块本身是支持交互命令，详见**《TinyWSN 交互命令手册》**，其中包括下面数据收发命令，

$addr$data,payload

当使能内嵌模式后，数据会传入到内嵌应用中，可以利用这条命令扩充内嵌应用的交互命令，例如我们要实现一个查询命令，获取内嵌应用的程序内部状态，命令格式如下

$addr$data,qry,sys

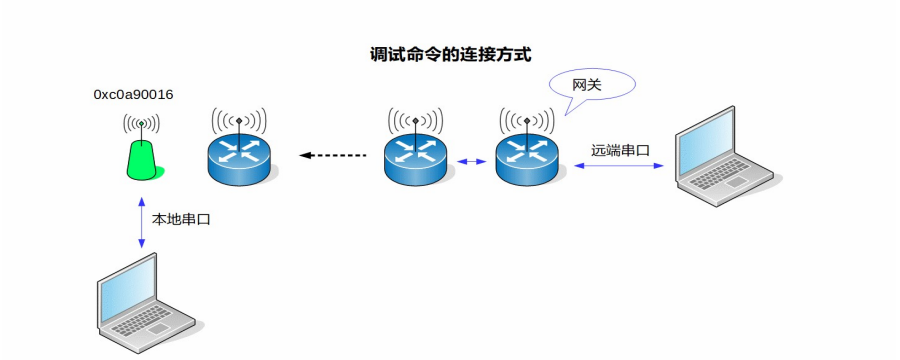
当内嵌应用接受到字符串“qry,sys”进行识别和回应，具体实现可以参见在文件 s_utils.c 中的函数 rx_usr_data，它负责处理接受到的数据:

https://gitee.com/tinywsn/fw-stm32l1-wbed-usr/blob/master/lib/s_util.c

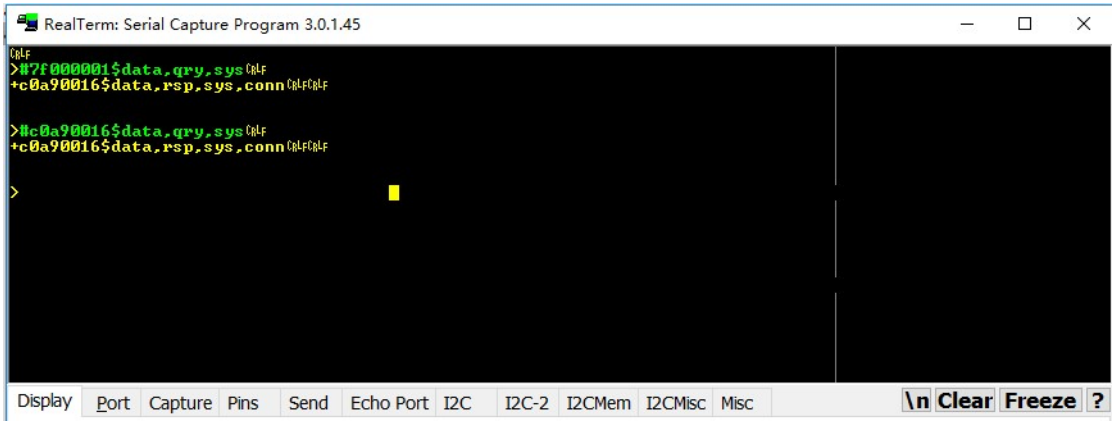
它识别出系统查询命令，就返回当前程序状态机的名称

```
s = "qry,sys";
if(!strncmp((char*)p, s, strlen(s))){
   stringf((char*)p, "rsp,sys,%s", ctx->cur_fsm_name);
   tx_usr_data(p, strlen((char*)p), ((wipc_pkt_hdr_t*)in)->src2, TX_F_PRIO, ctx);
}
```

通过上面这种方法，就很容易扩展内嵌应用的调试命令，通过这些调试命令可以观察内部状态，控制程序的运行。
如下图所示，可以通过两种连接方式发送调试命令，一种是本地串口直连，另外一种也可以通过网关的远端串口，发送调试命令到节点模块 0xc0a90016

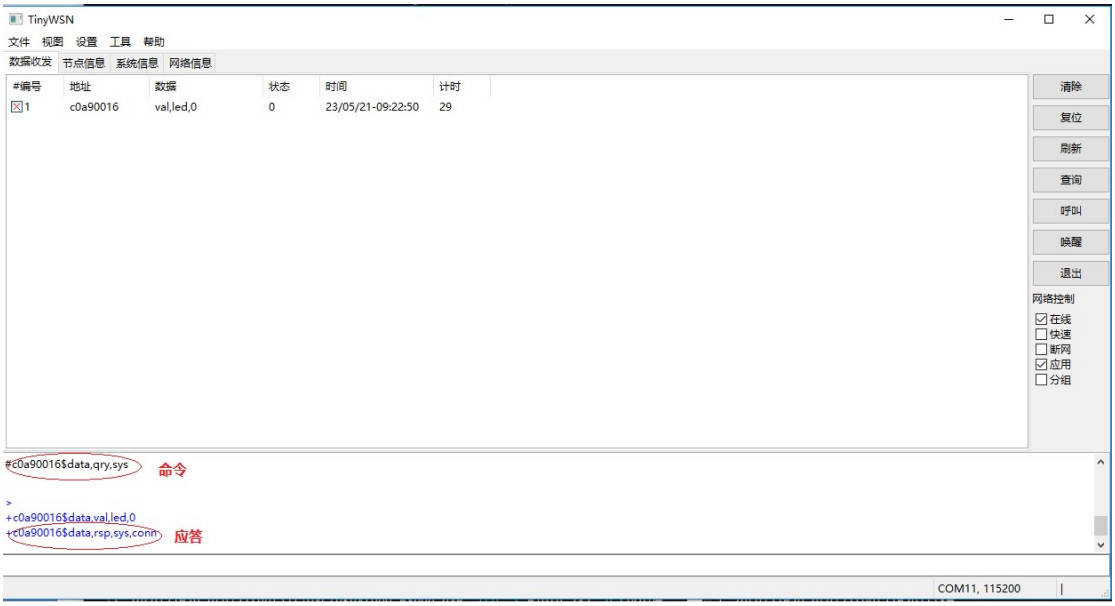


本地串口



从上图看到，首先使用通用的对端地址 7f000001 发出命令，模块给出应答“rsp,sys,conn”，显示当前是 conn 状态机，当然也可以直接使用模块的地址 c0a90016 发送命令。

远端串口



上图是从网关下发调试命令给模块 #c0a90016$data,qry,sys，内嵌应用的应答通过无线返回到网关。

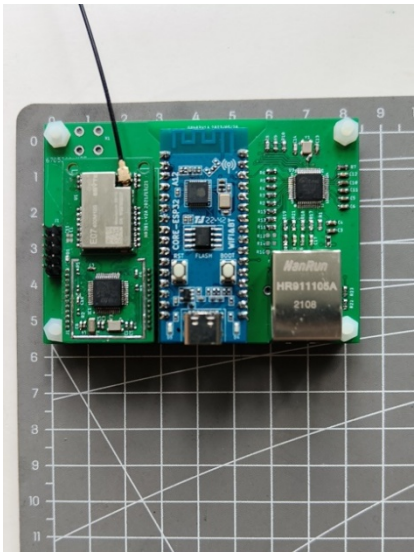
通过上面的示例可以看出，调试输出和调试命令组合可以成为一种有效的调试手段，为内嵌应用的开发提供便利。

TinyWSN 的透传型网关如何组网？

目前 TinyWSN 的网关分两类

类型	操心系统	说明
MQTT	Linux	网关支持 wifi，以太网以及 4G，网关连接到 MQTT 服务器，可以直接使用 MQTT 客户端对网络进行管理，而参考的网关软件 TinyGW 使用 python 开发，它使用的系统报文接口，源码链接 https://gitee.com/tinywsn/tinygw，可以自行修改和定制。
透传型	freertos	无需开发，只需简单配置，就把网关的命令接口透传至远端

下面介绍一个迷你型透传网关 G0B03，它是基于 ESP32C3 开发的，160M 主频的 RISC-V 处理器，支持 WIFI，以太网以及 BLE，支持 USB 接口（串口，JTAG），有很高性价比，



它可以完成以下透传类型

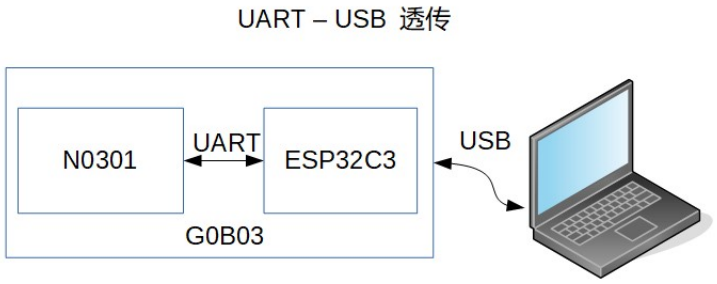
类型	说明
USB	网关命令接口透传至 USB 的串口，可以无法拔下节点模块，直接对节点配置
网络	网关命令接口透传 TCP 服务器，可以连接到网络管理程序 TinyWSN
BLE	网关命令接口透传远端的 BLE 的模块，连接到网络管理程序 TinyWSN

TinyNMS 是一个简单的网络管理程序，它基于交互命令接口，它支持串口连接，TCP Client 和 TCP Server

下面展示几种常见透传型网关的组网示例

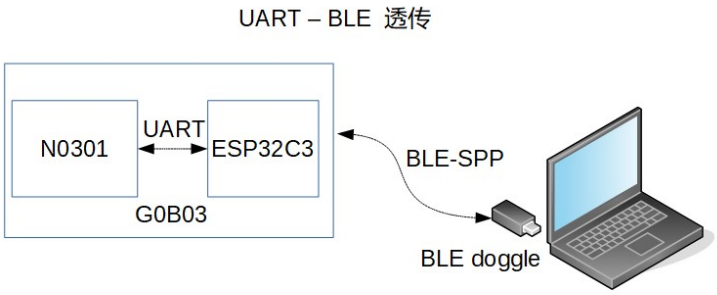
1. UART-USB 透传

   如下图所示，节点模块的命令接口透传至 USB，管理或者配置软件（TinyWSN 或 TinyCFG），可以直接打开 USB 的串口，就可以管理和配置网关上的节点模块



2. UART-BLE 透传

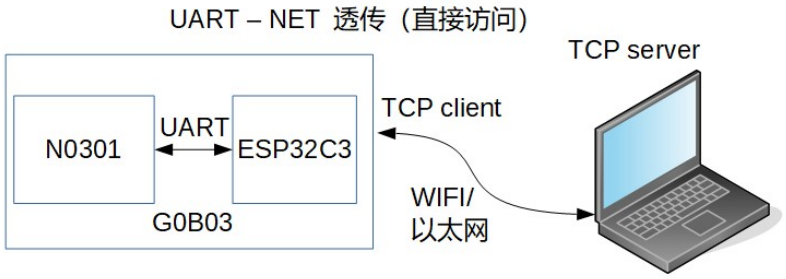
   如下图所示，节点模块的命令接口透传至 BLE，网关作为 BLE 服务器，主机插一个 BLE 模块（也可以用另外一块网关），扫描并连接至网关模块，使用自定义的 BLE-SPP 协议，管理软件打开 BLE 模块的 USB 串口，就可以直接管理整个网络。



3. UART-NET 透传（直接访问）

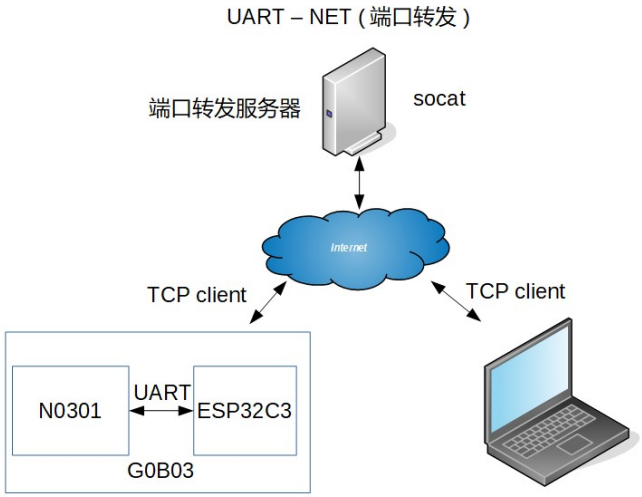
如下图所示，网关和主机处于同一网段或者连通的网段，节点模块的命令接口透传至主机，网络可以是 WIFI 或者以太网，网络管理软件 TinyWSN 启动 TCP 服务器，网关作为 TCP 客户端进行连接，连接后就可以直接管理整个网络了。

了



4. UART-NET 透传（端口转发）

如下图所示，网关和主机无法直接访问，例如处于不同的的局域网内，需要借助公网服务器进行互联，服务器可以运行端口转发软件，例如 socat, netcat 等等，网关和主机作为 TCP 客户端，连接服务器的配对的转发端口，

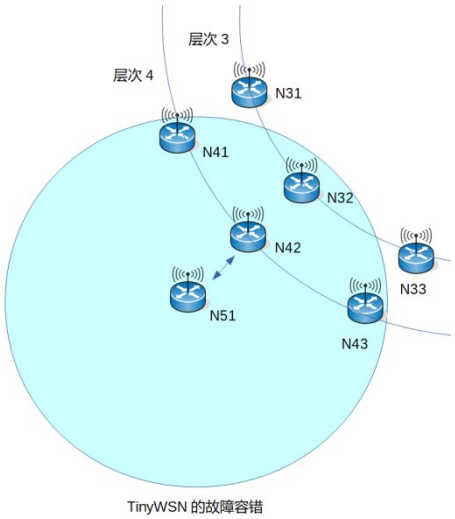


TinyWSN 的故障容错机制?

TinyWSN 本质上层次网，路由节点组成核心传输网，而终端节点不参与转发功能，如果路由节点发生故障，会影响它的子网的所以节点，所以一定的故障容错能力非常重要。
路由节点有两个重要的参数

名称	数值	说明
初始层次	5	主要用来决定路由节点在网络中的层次，0 表示根节点，节点模块只能接入层次比自己小的上级节点
层次范围	2	用来节点控制可接入范围，只有符合（上级层次+层次范围 >= 自己层次），默认值表示可以接入任何层次

如上例所示，如果一个路由节点 N51 的初始层次是 5，可接入的层次范围是 2，那么就意味它可以接入层次为 3,4 的上层节点，如下图所示



初始上电后路由节点 N51 根据可用频点表，扫描周围可用的上层接入点，如上图所示它会发现有四个可用的接入点 N41，N42，N43 和 N32，根据接收的信号强度和质量，以及其他一些因素，它优选 N42 作为接入点。如果节点 N42 发生故障时，N51 会重新扫描，从其他三个可用的节点，优选其中一个作为接入点，使得故障得到自动恢复。

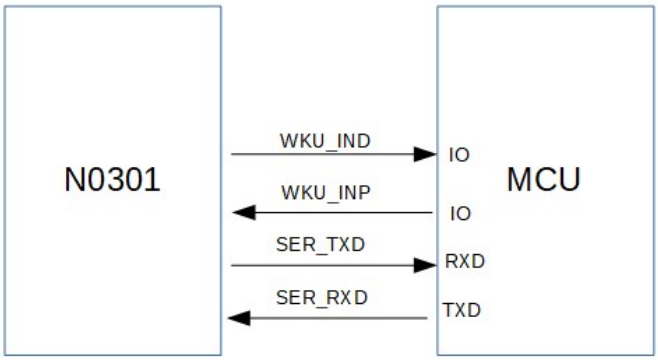
而终端节点的默认层次和范围都是 255，意味它可以从任何路由节点接入，当一个路由节点故障时，它就会重新选择优选新的接入点。

无线模块和外部 MCU 之间的相互唤醒?

TinyWSN 的节点模块支持多种用户号使用方式，用户可根据需要进行选择

方式	接口	说明
内嵌模式	共享内存	模块的二次开发，节省一个外部 MCU
交互命令	串口连接	类似 AT 的交互命令接口，简单易用
系统报文	串口连接	使用报文帧格式，安全可靠，功能强大

无线模块和外部 MCU 之间的硬件连接关系，如下图所示

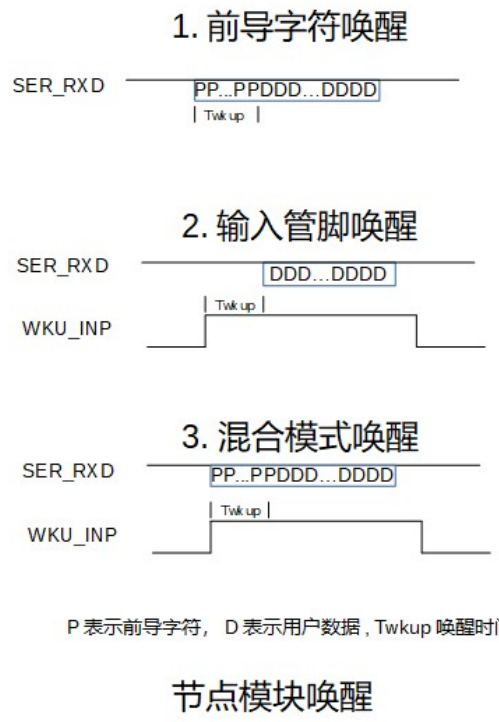


无线模块和外部 MCU 硬件连接

连接管脚说明

名称	方向（MCU）	说明
WKU_IND	输入	唤醒外部 MCU
WKU_INP	输出	唤醒节点模块
SER_TXD	输入	节点串口输出
SER_RXD	输出	节点串口输入

唤醒无线模块，主要有三种方式，如下图所示



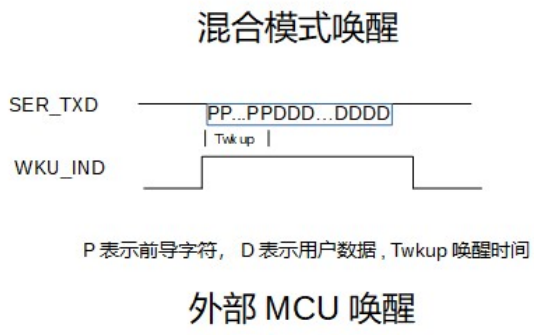
P 表示前导字符， D 表示用户数据，Twkup 唤醒时间

节点模块唤醒

节点模块唤醒，其中 Twkup=10ms，默认前导字符 0xff

模式	说明
前导字符唤醒	连续输入前导字符，满足唤醒时间 Twkup，然后紧接着输入用户数据，可以节省一个 WKU_INP 管脚
输入管脚唤醒	拉高 WKU_INP 电平 Twkup 时间后，然后输入用户数据，等输入结束后再拉低 WKU_INP 电平
混合模式唤醒	就是既有前导字符，也有输入管脚同时工作

唤醒外部 MCU，采用的混合唤醒模式，如下图所示



其中前导字符和个数都是可以用户配置的，可以利用前导个数来控制 Twkup 时间，

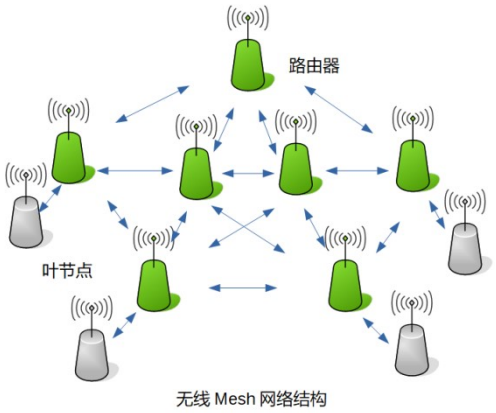
配置项	默认值	说明
前导字符	0xff	用户可根据具体的应用需要进行修改
前导个数	0x00	用户可根据 MCU 唤醒时间进行修改

由于节点模块会输出前导字符，MCU 需要过滤掉这些字符，无论是交互命令还是系统报文模式，都有特定的数据头，所以很容易过滤前导字符和识别出数据头，然后进入数据接受模式。

模式	数据头
交互命令	\r\n+
系统报文	STX

TinyWSN 支持 Mesh 组网吗？

首先让我们先了解一下无线 Mesh 网的特点，如下图所示



无线 Mesh 网络结构

从中可以看出它的一些特点

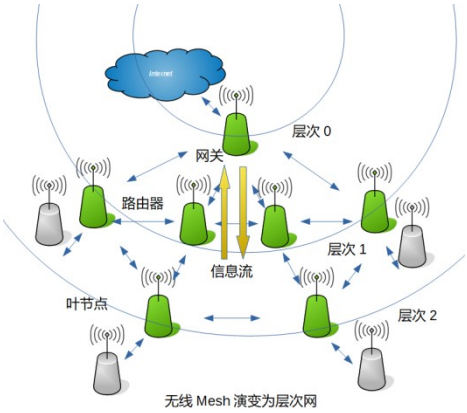
✓ 无中心节点，根据相邻节点广播的邻接表，动态构建路由表

✓ 有多条连接路径，当一条失效时，可以选择其他备选的路径

✓ 任何两个节点间可以通信

它非常适合一些分布应的边缘计算应用，节点根据收集到信息（可能是其他节点的），独立运行自己的处理算法，由于没有中心节点的协调，网络通常采取的异步通信模式，依赖于无线冲突检测的效果，但是随着节点数目的增加，发生无线冲突概率大大增加，特别某个紧急告警触发全网节点上报时，而且 LORA 可以在信噪比很低（-140dbm）情况下通信，这时它的接受到的信号强度非常小，这也使得冲突检测更容易失效，所以 Mesh 网络的节点数目通常会做出限制，而且多径也可能造成节点重复收到相同的消息。

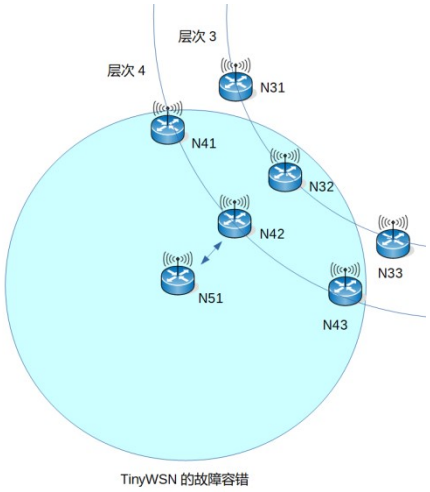
通常传感器网络的特点是节点数量多，但是功能都非常简单，只需完成信号的采集和执行下行控制命令，单个节点不会收集其他节点的信息，信息往往汇集到网关，集中在服务器进行处理，这样实际就从原来的 Mesh 无中心网演变成为以网关为中心的层次网。信息路由就变成简单了，一条是从网关向层次数高的叶节点方向流出的，一条是从各个叶节点向层次低的网关汇集的。如下图所示



无线 Mesh 演变为层次网

而且传感器网络节点数量多，容易导致无线干扰和冲突问题，为了从设计上解决这个问题，最终 TinyWSN 采取的是同步的层次网络架构，网关作为整个网络的根节点，同时也作为整个网络的同步源，所有节点的收发均以此作为基准，按分配的时隙进行收发，这样网络节点数目增加不再带来更多的无线冲突和干扰了，通信质量可以得到有效的保障。当然一个无线小区划分的时隙数目不能过多，否则就会造成延时多大，而且时隙是由节点共享的资源，节点从离线状态变成在线状态，需要一个接入和时隙分配过程，这也会带来一些额外的延时。

TinyWSN 引入了 Mesh 的多路由的能力，以提高网络的容错，在前面介绍过它的容错机制，如下图所示的层次网络中，层次间的节点并无固定的绑定关系，节点只能选定层次小的节点作为上级父节点，根据实际的网络规划，可能会有多个候选节点



TinyWSN 的故障容错

初始上电后路由节点 N51 根据可用频点表，扫描周围可用的上层接入点，如上图所示它会发现有四个可用的接入点 N41，N42，N43 和 N32，根据接收的信号强度和质量，以及其他一些因素，它优选 N42 作为接入点。如果节点 N42 发生故障时，N51 会重新扫描，从其他三个可用的节点，优选其中一个作为接入点，使得故障得到自动恢复。

TinyWSN 引入了 Mesh 的自组网的能力，以简化节点的配置，层次网的由于路由简单，只有上下行两种路径，所以它的自动网络层次分配也是相对简单的，从根节点开始周期广播自己的信息（层次为 0），周围节点接受到这个信息，如上图所示，当然可能会收到多个上级节点的广播信息，择优选择其中一个作为自己的父节点，然后更新自己的网络层次（父节点层次+1），继续广播，直至所有路由节点选定自己层次，这样以根节点为起点的树状的层次网自动组成了，具体的过程可以参见《TinyWSN 快速入门指南》中的《实战教程-自动组网》章节。

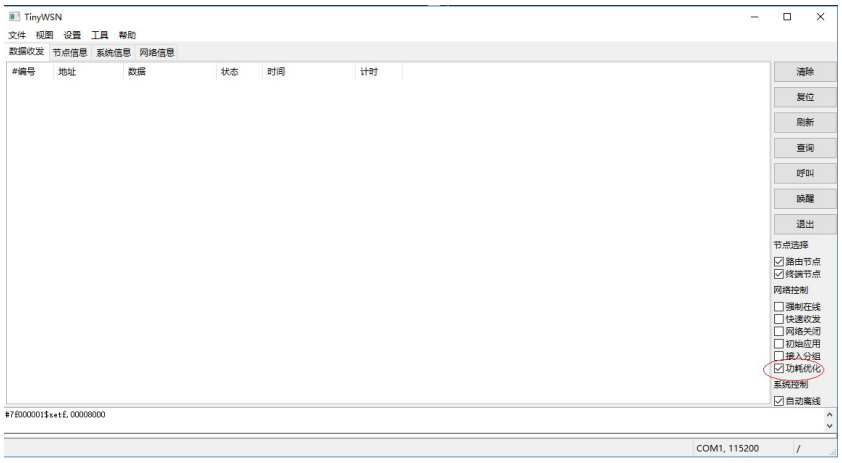
总之，TinyWSN 采用的是同步的层次网，虽然不是 Mesh 网络架构，但是它引入了 Mesh 中的多路由和自组网能力。

如何降低路由节点的功耗?

在 TinyWSN 网络中，路由节点构成核心的传输网络，通常它们是一直在线的，为终端节点提供接入服务，它们只在空闲时，才能进入低功耗状态，由于它需要工作在多个时隙上，导致它的功耗远远高于终端节点，如何降低路由节点的功耗呢？下面从三个方面或层次来讨论这个问题。

1) 使能动态时隙

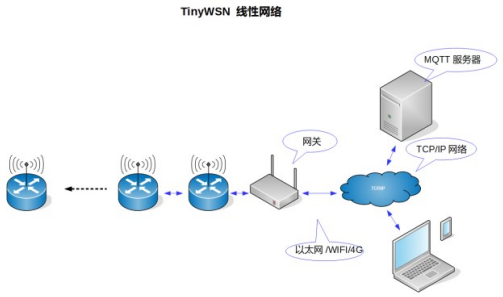
路由节点需要提供多时隙的服务，当时隙已分配时，就在这个时隙上为节点提供数据传输服务，如果时隙空闲时，就在这个时隙上提供接入服务。当大部分终端节点都处于离线状态，这时路由节点的时隙也大部分处于接入服务的状态，除了需要发射 beacon 信息，还需要接收可能的接入请求， 这也会导致增大路由节点的功耗，这时可以打开功耗优化开关，如下图所示



这时接入时隙的数量是会动态变化的，有点类似“潮汐车道”的概念，当接入请求变少时，就降低接入时隙的数目，当有新的接入请求时，就增大接入时隙的数目。当大部分终端节点都处于离线状态，只需维持最小数量的接入时隙，这可以有效地降低路由节点的功耗。

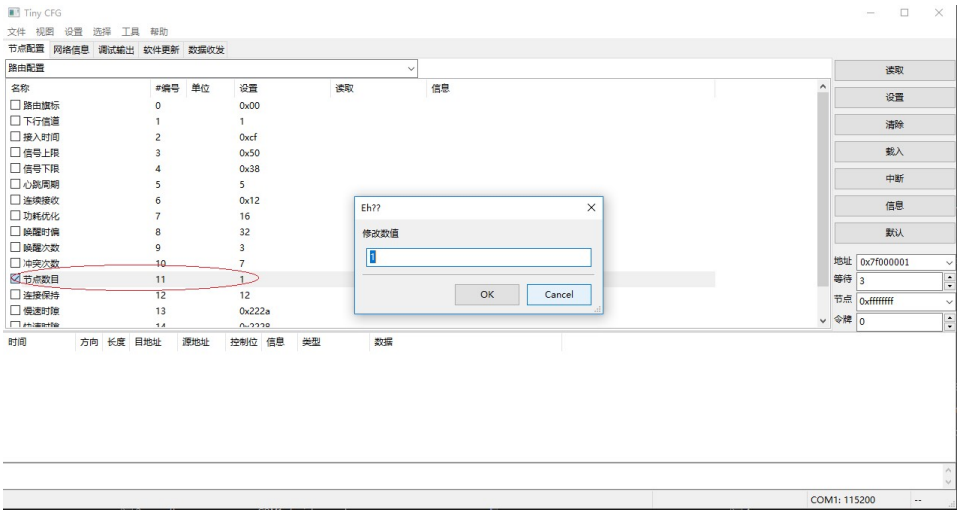
2) 配置节点数目

在一些应用场景中，特别是在如下图所示的链型组网中，其中路由节点又充当终端节点的角色，但是这些路由节点的子节点的数据基本上固定的，如下图所示，每个节点只带一个子节点



这时除了象上例打开上面的功耗优化开关，还可以使用配置每个路由节点的子节点数目，

如下图所示：



当路由节点的接入子节点数目等于配置的数目后，路由节点就不再提供接入服务了，只对已接入的节点提供数据传输服务，这也将进一步降低路由节点的功耗。

3) 开启全网静默

在一些应用场景中，除了对终端节点的功耗，对路由节点的功耗也有严格的要求，这些应用场景中通信的流量比较低，而且数据通常是周期采集的，比如几个小时一次，但是偶尔也会有突发状况，比如终端节点受到外部触发需要主动上报告警消息，全网静默就非常适应这个场景，终端节点和路由节点都处于 uA 级别的功耗（由睡眠周期决定，通常 2~3uA 的功耗），同时它支持自顶向下（top-down）和自下向上(bottom-up)发起的的通信方式。具体的操作过程可以参见《TinyWSN 快速入门指南》中的全网静默的实战教程。

总之通过上面介绍了解了如何降低路由节点的功耗，当然对功耗的优化会带来额外的节点接入时延，所以这是一个功耗和速度之间权衡问题。

TinyWSN 如何分配无线信道？

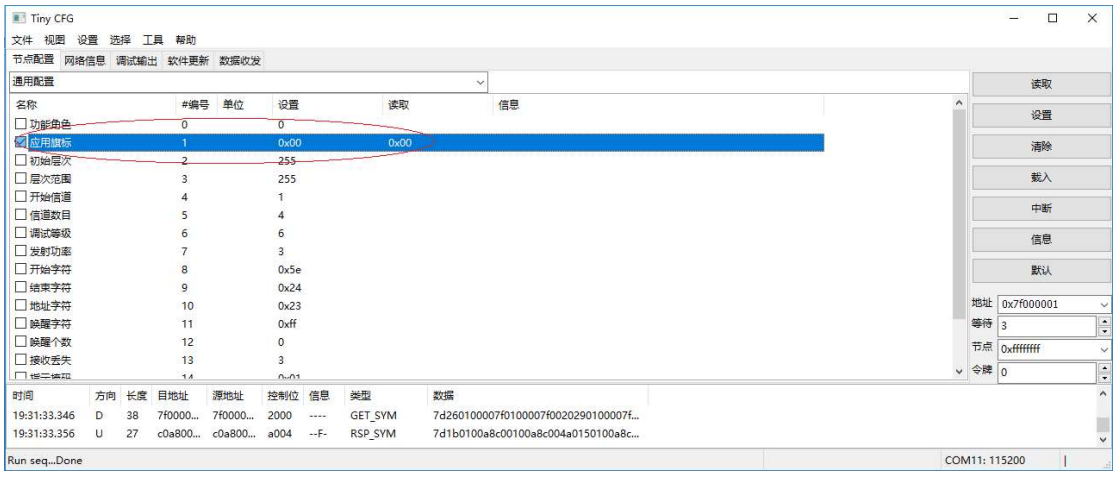
TinyWSN 模块组成类似如下的蜂窝网络，每个蜂窝中心就是一个路由模块，它的下行信道（D_LINK）就是这个蜂窝的信道，它为蜂窝范围内的节点提供接入服务

但是这个路由模块本身又作为上层网络的子节点，它有处在父节点的蜂窝范围内，它的上行信道（U_LINK）就是父节点的下行信道，这两个蜂窝的覆盖范围是有重叠的，启动路由节点是有上下行两个信道，终端节点只有上行信道，根节点只有下行信道。如上图所示，信道在不相邻的蜂窝是可以重复使用的，达到频分复用的效果，如上图的蓝色标注的信道 4，这也是蜂窝移动通信 GSM 采取的方式。

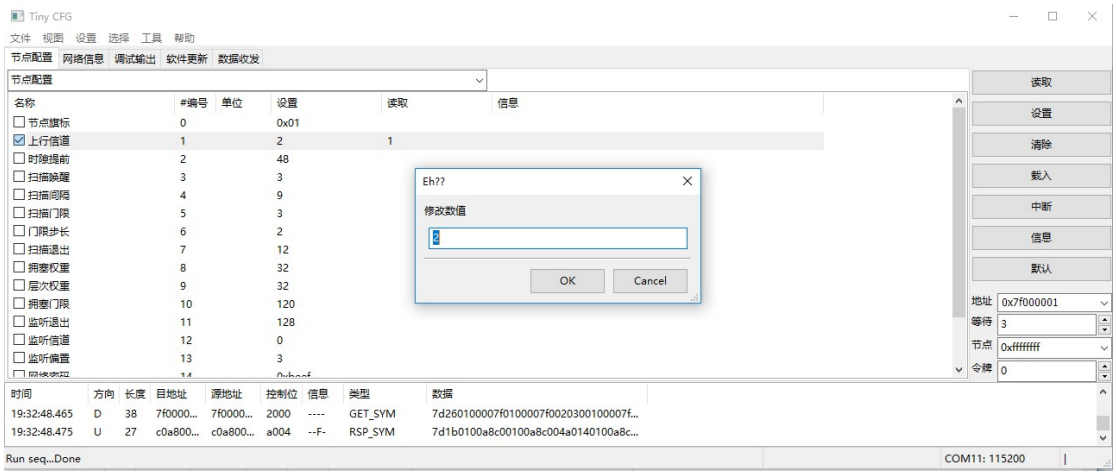
在 TinyWSN 中支持静态配置，自动配置和组合配置等分配方式，可以通过 TinyCFG 进行配置

✓ 静态配置

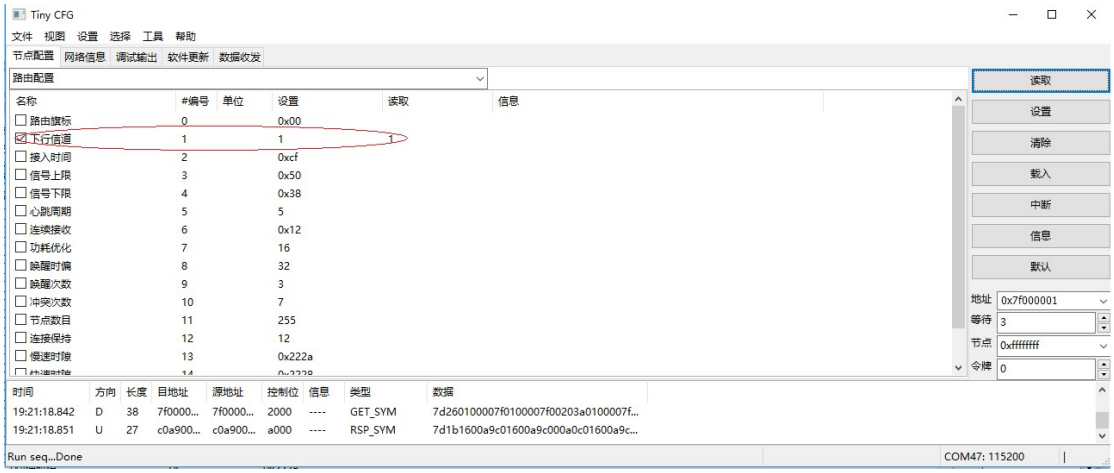
清除自动配置项，如下图所示，其中 bit0 为 0 时表示清除自动配置



指定上行信道



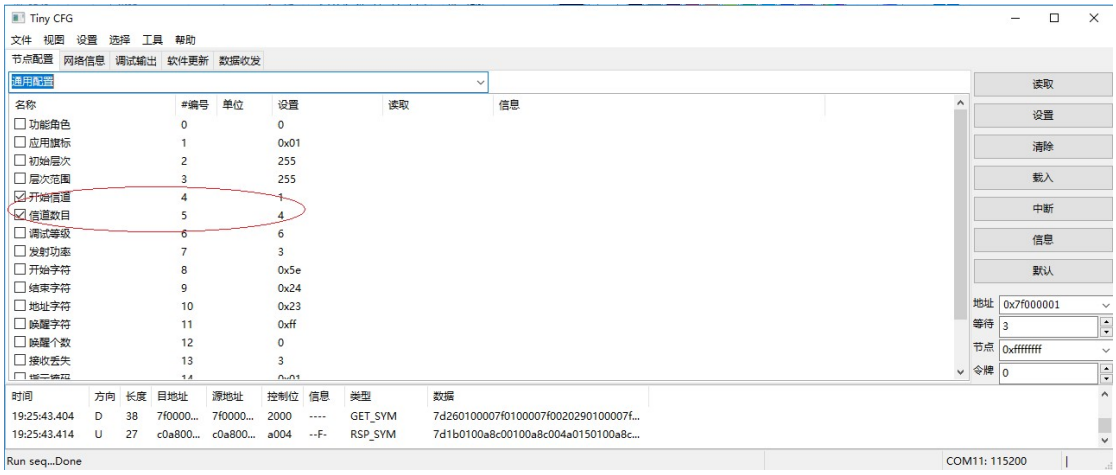
指定下行信道



上电后各个模块就在指定的信道上开始工作，网络建立和接入速度快

✓ 自动配置

如上图所示，首先需要使能自动配置，就是把应用旗标设置成 0x01，然后为每个模块配置相同的一组信道，如下图所示，选择从信道 1 开始的 4 个信道作为可用的信道组



上电后首先根节点扫描信道组中的信道，从中择优选择一条干净的信道作为自己的下行

信道开始广播信标，路由节点从信道组搜索可接入的信道，作为自己的上行信道，同时还需根据扫描结果确定自己的下行信道，终端节点只需搜索可接入的上行信道。通过这种方式，最终完成整个网络构建。

✓ 组合配置

就是前两种配置方式的结合，常见的是所有的路由节点是静态配置，可以提高响应速度，还比较容易进行网络规划，而数量多的终端节点是动态配置，这样可以简化配置和维护，而且可以自由移动，当需要接入时就开启扫描，搜索附近可用的信道。

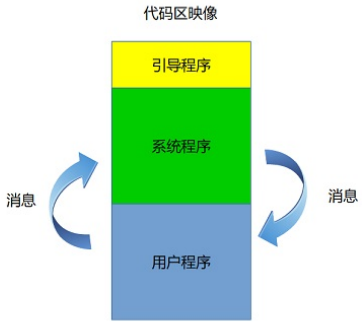
最后总结一下这几种信道分配方式的优缺点：

名称	特点
静态配置	由于没有信道扫描过程，网络建立和接入的速度快，缺点就是配置和修改繁琐，无法有效地应对外界无线环境的变化
自动配置	能够动态适应外界无线环境的变化，信道的选择是根据扫描择优选取，配置过程简单，缺点是信道扫描会给网络建立也接入带来时延
组合配置	前两种配置方式的结合，同时具有它们的优点和缺点

TinyWSN 如何进行二次开发?

最常见的模块的使用方式是通过串口的交互命令接口，这时需要有一个外部 MCU 完成传感器数据采集，然后通过命令接口完成数据收发，这种方式的好处应用程序独立开发，相互之间互不影响，容易开发和调试，缺点就是增加一个外部 MCU，带来系统成本的增加。

TinyWSN 的无线模块是支持二次开发的，也就是内嵌应用的开发，应用程序和系统程序分别在系统不同的存储区域内，它们各自独立开发和下载，它们之间通过消息接口进行通信，如下图所示：



应用程序可以单独下载，并在 TinyWSN 中控制是否启动，如下图所示

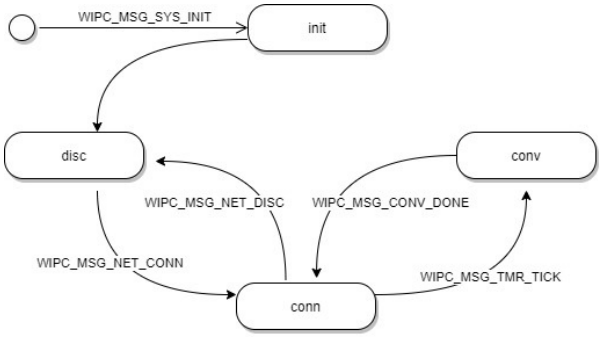
应用程序的开发语言是 C 语言，这也是嵌入开发的常见语言，可以非常方便的操作各种外设，期间也评估过使用脚本语言开发应用，即使最小巧的 lua 也会增加不少代码空间，而且操作和控制不够直接，有点像隔靴搔痒，最终还是回归到 C 语言。

✓ 编程模型

由于应用程序和系统程序共用一个处理器，需要避免相互影响，特殊是一些无线协议的关键的时序，所以应用程序的采用的是消息触发的状态机模型，这种模型在嵌入式开发中是很常见的，也很容易理解，它的好处就是系统响应速度快，避免轮询等待的耗时操作，它的宗旨就是：“不用等，好了我会通知你”，也就是说启动一个操作后，要避免轮询等待操作完成，直接退出交还控制权给系统，等待操作完成的通知消息，所有需要等待的同步操作变成异步通知，应用程序需要处理下面三个来源的消息，完成和外部系统的交互：

- 系统程序产生的消息
- 中断服务产生的消息
- 用户程序直接的消息

如下图所示，就是一个典型的状态机的例图：



可以把状态机当做是一个有记忆的生命体（例如是一条狗），它会对外部的触发做出应急反应，当伸手去抚摸它的头，它就兴奋地会摇头晃尾做出反应，但是如果之前先踢了它一脚，它已经处于愤怒状态中，再伸手去抚摸它的头，它可能就会咬你一口，所以对触发的反应取决于它的当前的状态，而当前的状态又是对以前触发的累积的结果。

✓ 内存模型

由于应用程序是独立编译和连接的，它下载到 flash 中预定的区域，而上电后入口是系统程序，它负责初始化的整个系统，但它只加载了系统程序的.data 段和初始化.bss 段，这也给应用程序带来了一点限制，无法使用全局变量，只能使用栈变量，这包括以下两种情况所以需要避免使用全局变量和静态的局部变量，可以看下面的一个例子

```
int g_state;            // 全局变量，不能
void foo(void){
   static int last;     // 静态变量，不能
   int i,j,k;           // 堆栈变量，可以
}
```

为了解决全家变量使用的限制，在应用程序的消息处理入口处，每次都会传递一块内存的地址，作为应用程序的全局内存，由应用程序全权进行管理，如下所示就是应用程序的消息处理入口，

```
wipc_result_t wipc_msg_proc (wipc_msg_type_t msg, void *in, void *out, void *ctx)
```

其中参数说明如下:

名称	说明
msg	消息类型
in	消息内容
out	返回数据
ctx	运行环境

其中 ctx 就是它的运行环境，其实是系统提供的一块内存，由应用程序进行管理，当接受到初始上电消息，应用程序就初始化这块内存，以后系统每次发送消息时，都会回传这块内存，这样就能解决全局变量的使用限制的问题。

✓ 中断模型

应用程序可以使用未被占用的中断，在实战教程的演示应用程序中，它支持按键用于唤醒系统和翻转 LED 状态，可以参见它的具体实现

drivers\wku.c

按常规初始化中断就可以了，包括使能中断，初始触发电平等等，所有未被系统程序的占用的中断，都会转到应用程序的中断入口，这样就可以对中断进行处理。

✓ 低功耗处理

应用程序可以独占使用自己的外设（GPIO, UART , I2C, SPI, TIMER 等等），当它初始化这些设备后，就需要负责处理它的功耗，下面是和功耗处理有关的几条消息

名称	方向	说明
WIPC_MSG_LPM_REQ	输入	每次进入低功耗前发起的询问? 如果拒绝了，就不进入低功耗
WIPC_MSG_LPM_INIT	输入	进入低功耗，可以关闭设备
WIPC_MSG_LPM_EXIT	输入	退出低功耗，重新启动设备
WIPC_MSG_LPM_HALT	输出	主动暂停系统低功耗处理
WIPC_MSG_LPM_CONT	输出	主动恢复系统低功耗处理

通过上面这几条消息，就可以和系统程序协调低功耗处理流程，使得整体功耗达到产品要求。

总之，通过对模块二次开发的几个问题的讨论，可以看出应用程序的开发流程和常规的嵌入式开发流程是一致，只是需要注意一下提到的几个问题，就可以灵活自由地开发自己的应用。在实战教程中使用的 LED 演示程序就是通过这种方式开发的，下面源码库的链接

https://gitee.com/tinywsn/fw-stm32l1-wbed-usr

同时它还包含其他市面上常见的传感器的演示例程，可以参见它的具体实现

名称	说明
am2320	温湿度传感器
bh1750	光照传感器
dht1x	温湿度传感器
ds18b20	温度传感器
sht2x	温湿度传感器