数据接收层处理说明

本文档用于说明工程代码的编码细节，但更详细的信息应该关注代码文件中的注释。

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 更新日期 | 更新内容 | 操作者 |
| V1 | 2016/8/11 | 创建本文档，添加网络层有关编码流程解析 | 李培锋 |
| V2 | 2016/8/12 | 添加蓝牙本地下单层有关编码流程解析 | 李培锋 |

目录

[1 网络层 1](#_Toc458799636)

[1.1 初始化LWIP 1](#_Toc458799637)

[1.2 连接服务器 1](#_Toc458799638)

[1.3 从网络接收报文并解析 2](#_Toc458799639)

[1.4 发送报文至网络 5](#_Toc458799640)

[1.5 存储数据至本地缓冲区 6](#_Toc458799641)

[2 蓝牙本地下单层 7](#_Toc458799642)

[2.1 初始化串口 7](#_Toc458799643)

[2.2 本地接收数据 7](#_Toc458799644)

[2.3 接收订单 8](#_Toc458799645)

[2.4 接收应答 10](#_Toc458799646)

[2.5 发送状态报文 11](#_Toc458799647)

[2.6 从缓冲区读取一定数据 13](#_Toc458799648)

[2.7 删除缓冲区 14](#_Toc458799649)

[2.8 校验和 15](#_Toc458799650)

# 网络层

## 初始化LWIP

### 概要

该部分用于初始化网络环境，包括设置本地MAC地址、IP或选择开启DHCP服务。

* 函数原型: void LwIP\_Init(void)

### 代码文件位置

代码位于[USER/netconf.c](../../USER/netconf.c)。

## 连接服务器

### 概要

该部分用于向服务器发起链接请求，故在该部分需要设置服务器IP，并向服务器发送请求建立报文，以让服务器得知是哪台主控板连接了。

* 函数原型: void con\_to\_server(void)

### 代码文件位置

代码位于[USER/netconf.c](../../USER/netconf.c)。

## 从网络接收报文并解析

### 概要

从网络缓冲区读取数据并解析其是否是一个正确的批次，是则根据是否为加急存入相应的本地缓冲区，否则丢弃该报文。

调用该函数的线程优先级应高于进入打印队列线程的优先级，目的是为了保证缓冲区尽可能满。

* 函数原型:void receive\_connection(struct netconn \*conn)
* 参数: conn :全局TCP链接

### 流程图



流程图 1 网络解析报文主干



流程图 2 解析批次头



流程图 3 读取解析批次后剩下的有效数据

### 代码文件位置

代码位于[USER/netconf.c](../../USER/netconf.c)，去除调试宏的代码位于[DOC\数据接收处理\去除调试信息的代码文件\network\receive\_connection.c](去除调试信息的代码文件/network/receive_connection.c)。

## 发送报文至网络

### 概要

封装好报文后，从该接口发送数据至网络端。

封装报文的工作由API “Pack\_Req\_Or\_Status\_Message”完成，其根据报文协议来封装数据，详情见[USER/pack\_data.c](../USER/pack_data.c)/Pack\_Req\_Or\_Status\_Message，而该发送接口仅是调用LWIP发送接口。

* 函数原型：void write\_connection(struct netconn \*conn, req\_type type, u8\_t symbol, u32\_t preservation)
* 参数: conn :链接

type :报文类型

symbol :标志位

preservation:保留字段，批次中，高16位为批次序号，订单中时且低16位为批次内序号

### 代码文件位置

代码位于[USER/netconf.c](../../USER/netconf.c)。

## 存储数据至本地缓冲区

### 概要

该API为本地及从网络接收到订单数据后欲将其放置于本地缓冲区而设。但在将数据放置于缓冲区前，先判断缓冲区是否有充足的容量，否则等待其有足够容量再继续填充。

该API实质调用Write\_Buf以完成填充工作，详情见[USER/queue\_buf.c](../USER/queue_buf.c)/Write\_Buf。

* 函数原型: void put\_in\_buf(char \*data, u16\_t len, u16\_t urg)
* 参数: data :存储数据的指针

len :数据长度

urg :加急标志

### 流程图



流程图 4 存储数据至缓冲区

### 代码文件位置

代码位于[USER/netconf.c](../../USER/netconf.c)。

# 蓝牙本地下单层

## 初始化串口

### 概要

蓝牙本地下单所用的串口为串口4，TX引脚为PC10，RX引脚为PC11。

由于手机蓝牙使用的开发包指定的波特率为9600，故使用9600的波特率。另外，数据字长度为8位，停止位1位，无奇偶校验。

* 函数原型: void Usart4\_Init(void)

### 代码文件位置

代码位于[USER/stm32\_init.c](../../USER/stm32_init.c)。

## 本地接收数据

### 概要

在本地接收数据这一模块中，主要是向从串口缓冲区读取到的数据进行解析，判断其是否是一个正确的订单或应答，并通过信号量将接收到订单或应答的信息发送给指定的线程，如打印队列（有订单数据时需要处理）和发送状态报文（超时重传需要应答）。

* 函数原型: void local\_receive(void)

### 流程图



流程图 5 本地接收数据解析流程

### 代码文件位置

代码位于[USER/local\_conf.c](../../USER/local_conf.c)。

## 接收订单

### 概要

在接收订单模块中，主要是先获取订单头部，再分析整个订单的长度，之后读取一份完整的订单，再进行校验和计算。

如果校验成功，则获取订单号，并判断是否为重传订单，无论是否为重传订单都反馈应答给蓝牙端。

如果不是重传订单，则将其填充进缓冲区，并通知打印队列有新的订单进入缓冲区了。

* 函数原型: void recOrder(char \*data, u16\_t len)
* 参数: data: 从缓冲区获取的数据

len: 数据长度

### 流程图



流程图 6 接收订单处理流程

### 代码文件位置

代码位于[USER/local\_conf.c](../../USER/local_conf.c)。

## 接收应答

### 概要

在接收应答模块中，先根据应答报文的长度读取一定的数据，再校验这份数据是否正确。

如果校验正确则判断此份应答是打印机还是订单的状态反馈应答，然后根据不同的应答反馈给处理发送状态报文的地方，因其超时重传需要应答响应。

最后删除被读取过后无效的缓冲区。

根据校验结果返回应答接收是否成功。

* 函数原型: Status recAck(u16\_t offset)
* 参数: offset: 由于缓冲区可能会有错误数据，故先要判断状态报文头，在读取到状态报文头之后，之后的数据才会是属于应答的数据。不先把这个偏移的无效数据清除掉的原因是，状态报文头可能与订单数据重复，而每次接收数据后都是先判断是否为应答的，故可能把属于订单的数据给清除掉，从而用传入偏移量的方法。

### 流程图



流程图 7 接收应答处理流程图

### 代码文件位置

代码位于[USER/local\_conf.c](../../USER/local_conf.c)。

## 发送状态报文

### 概要

本发送状态报文部分是隶属于本地发送报文的，与网络发送状态无关。

此部分利用串口发送报文至蓝牙端，由于蓝牙不可靠传输，所以需要采用超时重传机制。

对于不同的状态反馈，所采用的超时重传时间是不一样的，如打印机状态反馈超时重传时间为160ms，而订单状态反馈为800ms，但具体重传时间可以另行商榷。

此外，超时重传3次则不再重传了，以免浪费时间和资源。

* 函数原型: void SendStatusToLocal(req\_type type, u8\_t symbol, u32\_t preservation)
* 参数: type: 报文类型，打印机或订单状态

symbol: 标志位，详情见数据交互表

preservation: 保留字段，打印机状态时为打印单元序号，订单状态时为订单序号

### 流程图



流程图 8 发送状态报文流程图

### 代码文件位置

代码位于[USER/local\_conf.c](../../USER/local_conf.c)。

## 从缓冲区读取一定数据

### 概要

本模块用于从串口缓冲区读取一定的数据。

之所以要读取一定的数据，其实是为了方便处理。当已知了报文和订单长度后，便可以直接读取该长度的数据，再对此份数据进行处理，从而提高了解决问题的效率。

可以看到，在网络缓冲区使用读取的方法是比较死板的，那是因为无法直接从网络缓冲区要求读取一定量的数据，或者说有，只不过需要一个更大的缓冲区来缓冲这些数据，再来被将这些数据按需分配给需要的线程。然而这是不太合理的，因为内存有限，硬件资源导致无法达到这个美好的设想。

* 函数原型: s8\_t ReadAssignBytes(char \*data, u16\_t len)
* 参数: data: 存储数据的数组

len: 指定读取的长度

### 流程图



流程图 9 接收指定长度数据流程

### 代码文件位置

代码位于[USER/local\_conf.c](../../USER/local_conf.c)。

## 删除缓冲区

### 概要

在读取了一定量的有效缓冲区数据之后，需要清除掉这些部分，这通常使用改变串口的读指针来实现。

但是，如果读取了这些数据后判断不为有效数据时，同样也要对其清除，以保证接下来接收的数据有位置可以存放，而下一次从串口缓冲区读取数据时也可以读取到正确的数据。

* 函数原型: s8\_t DeleteBuf(u16\_t len)
* 参数: len: 指定删除的缓冲区长度
* 返回值: BUF\_ARG\_ERR: 长度错误，欲清除的缓冲区长度大于缓冲区现有数据长度

BUF\_OK: 删除成功

### 流程图



流程图 10 清除缓冲区流程

### 代码文件位置

代码位于[USER/local\_conf.c](../../USER/local_conf.c)。

## 校验和

### 概要

校验和部分用于校验数据在传输过程中是否出错。

此部分主要用于蓝牙数据传输校验部分，而在网络端由TCP来负责可靠传输。

此部分的校验和方式采用的是UDP校验和算法，使用这个算法的原因是因为，对于打印机来说，没有必要采取太复杂的算法，因为这会大大加大系统负荷。故不使用CRC算法。

* 函数原型: u16\_t Check\_Sum(u16\_t \*data, int len)
* 参数: data: 待校验的数据

len: 数据的长度

### 流程图



流程图 11 校验和算法

### 代码文件位置

代码位于:[USER/more\_infomation.c](../../USER/more_infomation.c)。