# 概述

我们组 UDP 数据包时常考虑字节流顺序、校验和计算等问题，有时候会比较繁琐，那么，有没有一种更简单的方法呢？答案是：借助 libnet 函数库。

libnet 是一个小型的接口函数库，主要用 C 语言写成，提供了低层网络数据包的构造、处理和发送功能。

libnet 的开发目的是：建立一个简单统一的网络编程接口以屏蔽不同操作系统底层网络编程的差别，使得程序员将精力集中在解决关键问题上。

libnet 库提供的接口函数包含 15 种数据包生成器和两种数据包发送器（IP 层和数据链路层）。   
  
提供的接口函数包括：  
1）内存管理（分配和释放）函数  
2）地址解析函数  
3）各种协议类型的数据包构造函数  
4）数据包发送函数（IP层和链路层）  
5）一些辅助函数，如产生随机数、错误报告、端口列表管理等

**libnet 的安装**

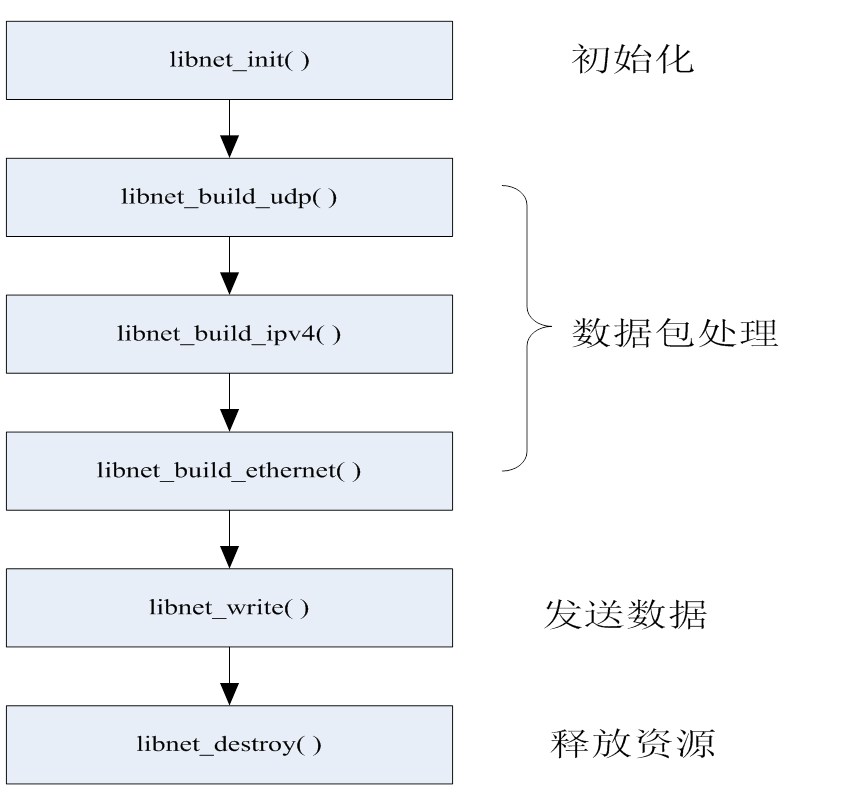


# 流程

**利用libnet函数库开发应用程序的基本步骤：**

1）数据包内存初始化  
2）构造数据包  
3）发送数据  
4）释放资源

以发送 UDP 数据包为例，流程图如下：



**这里需要注意的是组包的顺序**，由上层再到底层，这里为 udp -> ip -> mac，不能反过来。

# 常用函数介绍

以下函数的使用需要包含头文件 libnet.h

libnet\_t \*libnet\_init(int injection\_type, char \*device, char \*err\_buf)

功能：

数据包内存初始化及环境建立

参数：

injection\_type：构造的类型

LIBNET\_LINK，链路层

LIBNET\_RAW4，网络接口层（网络层）

LIBNET\_LINK\_ADV，链路层高级版本

LIBNET\_RAW4\_ADV，   网络层高级版本

device：网络接口，如 "eth0"，或 IP 地址，亦可为 NULL (自动查询搜索)

err\_buf：存放出错的信息

返回值：

成功：一个 libnet \* 类型的指针，后面的操作都得使用这个指针

失败：NULL

void libnet\_destroy(libnet\_t \*l);

功能：

释放资源

参数：

l：libnet\_init() 返回的 libnet \* 指针

返回值：

无

char\* libnet\_addr2name4(u\_int32\_t in, u\_int8\_t use\_name)

功能：

将网络字节序转换成点分十进制数串

参数：

in：网络字节序的 ip 地址

use\_name：

LIBNET\_RESOLVE，  对应主机名

LIBNET\_DONT\_RESOLVE，对应点分十进制 IPv4 地址

返回值：

成功：点分十进制 ip 地址

失败：NULL

u\_int32\_t libnet\_name2addr4(libnet\_t \*l,

char \*host\_name,

u\_int8\_t use\_name)

功能：

将点分十进制数串转换为网络字节序 ip 地址

参数：

l：libnet\_init() 返回的 libnet \* 指针

host\_name：

LIBNET\_RESOLVE，  对应主机名

LIBNET\_DONT\_RESOLVE，对应点分十进制 IPv4 地址

返回值：

成功：网络字节序 ip 地址

失败：-1

u\_int32\_t libnet\_get\_ipaddr4(libnet\_t \*l)

功能：

获取接口设备 ip 地址

参数：

l：libnet\_init() 返回的 libnet \* 指针

返回值：

成功：网络字节序的 ip 地址

失败：-1

struct libnet\_ether\_addr\* libnet\_get\_hwaddr(libnet\_t \*l)

功能：

获取接口设备硬件地址

参数：

l：libnet\_init() 返回的 libnet \* 指针

返回值：

成功：指向 MAC 地址的指针

失败：NULL

libnet\_ptag\_t libnet\_build\_udp(

u\_int16\_t sp, u\_int16\_t dp,

u\_int16\_t len, u\_int16\_t sum,

u\_int8\_t \*payload, u\_int32\_t payload\_s,

libnet\_t \*l, libnet\_ptag\_t ptag)

功能：

构造 udp 数据包

参数：

sp: 源端口号

dp：目的端口号

len：udp 包总长度

sum：校验和，设为 0，libnet 自动填充

payload：负载，为给应用程序发送的文本内容，没有内容时可设置为 NULL

payload\_s：负载长度，给应用程序发送文本内容的长度，或为 0

l：libnet\_init() 返回的 libnet \* 指针

ptag：协议标记，第一次组新的发送包时，这里写 0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。

返回值：

成功：协议标记

失败：-1

libnet\_ptag\_t libnet\_build\_tcp(

u\_int16\_t sp, u\_int16\_t dp,

u\_int32\_t seq, u\_int32\_t ack,

u\_int8\_t control, u\_int16\_t win

u\_int16\_t sum, u\_int16\_t urg,

u\_int16\_t len, u\_int8\_t \*payload,

u\_int32\_t payload\_s, libnet\_t \*l,

libnet\_ptag\_t ptag )

功能：

构造 tcp 数据包

参数：

sp：源端口号

dp：目的端口号

seq：序号

ack：ack 标记

control：控制标记

win：窗口大小

sum：校验和，设为 0，libnet 自动填充

urg：紧急指针

len：tcp包长度

payload：负载，为给应用程序发送的文本内容，可设置为 NULL

payload\_s：负载长度，或为 0

l：libnet\_init() 返回的 libnet \* 指针

ptag：协议标记，第一次组新的发送包时，这里写 0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。

返回值：

成功：协议标记

失败：-1

libnet\_ptag\_t libnet\_build\_tcp\_options(

u\_int8\_t \*options,

u\_int32\_t options\_s,

libnet\_t \*l,

libnet\_ptag\_t ptag )

功能：

构造 tcp 选项数据包

参数：

options：tcp 选项字符串

options\_s：选项长度

l：libnet 句柄，libnet\_init() 返回的 libnet \* 指针

ptag：协议标记，第一次组新的发送包时，这里写 0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。

返回值：

成功：协议标记

失败：-1

libnet\_ptag\_t libnet\_build\_ipv4(

u\_int16\_t ip\_len, u\_int8\_t tos,

u\_int16\_t id, u\_int16\_t flag,

u\_int8\_t ttl, u\_int8\_t prot,

u\_int16 sum, u\_int32\_t src,

u\_int32\_t dst, u\_int8\_t \*payload,

u\_int32\_t payload\_s,libnet\_t \*l,

libnet\_ptag\_t ptag )

功能：

构造一个 IPv4 数据包

参数：

ip\_len：ip 包总长

tos：服务类型

id：ip 标识

flag：片偏移

ttl：生存时间

prot：上层协议

sum：校验和，设为 0，libnet 自动填充

src：源 ip 地址

dst：目的ip地址

payload：负载，可设置为 NULL（这里通常写 NULL）

payload\_s：负载长度，或为 0（这里通常写 0 ）

l：libnet 句柄，libnet\_init() 返回的 libnet \* 指针

ptag：协议标记，第一次组新的发送包时，这里写 0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。

返回值：

成功：协议标记

失败：-1

libnet\_ptag\_t libnet\_build\_ipv4\_options(

u\_int8\_t\*options, u\_int32\_t options,

libnet\_t\*l, libnet\_ptag\_t ptag)

功能：

构造 IPv4 选项数据包

参数：

options：tcp 选项字符串

options\_s：选项长度

 l：libnet 句柄，libnet\_init() 返回的 libnet \* 指针

 ptag：协议标记，若为 0，建立一个新的协议

返回值：

成功：协议标记

失败：-1

libnet\_ptag\_t libnet\_build\_arp(

u\_int16\_t hrd, u\_int16\_t pro,

u\_int8\_t hln, u\_int8\_t pln，

u\_int16\_t op, u\_int8\_t \*sha,

u\_int8\_t \*spa, u\_int8\_t \*tha，

u\_int8\_t \*tpa, u\_int8\_t \*payload，

u\_int32\_t payload\_s, libnet\_t \*l,

libnet\_ptag\_t ptag )

功能：

构造 arp 数据包

参数：

hrd：硬件地址格式，ARPHRD\_ETHER（以太网）

pro：协议地址格式，ETHERTYPE\_IP（ IP协议）

hln：硬件地址长度

pln：协议地址长度

op：ARP协议操作类型（1：ARP请求，2：ARP回应，3：RARP请求，4：RARP回应）

sha：发送者硬件地址

spa：发送者协议地址

tha：目标硬件地址

tpa：目标协议地址

payload：负载，可设置为 NULL（这里通常写 NULL）

payload\_s：负载长度，或为 0（这里通常写 0 ）

l：libnet 句柄，libnet\_init() 返回的 libnet \* 指针

ptag：协议标记，第一次组新的发送包时，这里写 0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。

返回值：

成功：协议标记

失败：-1

libnet\_ptag\_t libnet\_build\_ethernet(

u\_int8\_t\*dst, u\_int8\_t \*src,

u\_int16\_ttype, u\_int8\_t\*payload,

u\_int32\_tpayload\_s, libnet\_t\*l,

libnet\_ptag\_t ptag )

功能：

构造一个以太网数据包

参数：

dst：目的 mac

src：源 mac

type：上层协议类型

payload：负载，即附带的数据，可设置为 NULL（这里通常写 NULL）

payload\_s：负载长度，或为 0（这里通常写 0 ）

l：libnet 句柄，libnet\_init() 返回的 libnet \* 指针

ptag：协议标记，第一次组新的发送包时，这里写 0，同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值。

返回值：

成功：协议标记

失败：-1

int libnet\_write(libnet\_t \* l)

功能：

发送数据包

参数：

l：libnet 句柄，libnet\_init() 返回的 libnet \* 指针

返回值：

成功：发送数据包的长度

失败：返回 -1

# 使用实例

这里是在 ubuntu 下通过原始套接字组一个 udp 数据包，给 PC 机的网络调试助手发送信息（对比：《[原始套接字实例：发送 UDP 数据包](http://blog.csdn.net/tennysonsky/article/details/44925057" \t "https://blog.csdn.net/lianghe_work/article/details/_blank)》）：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <libnet.h>

int main(int argc, char \*argv[])

{

    char send\_msg[1000] = "";

    char err\_buf[100] = "";

    libnet\_t \*lib\_net = NULL;

    int lens = 0;

    libnet\_ptag\_t lib\_t = 0;

    unsigned char src\_mac[6] = {0x00,0x0c,0x29,0x97,0xc7,0xc1};//发送者网卡地址00:0c:29:97:c7:c1

    unsigned char dst\_mac[6] = {0x74,0x27,0xea,0xb5,0xff,0xd8};//接收者网卡地址‎74-27-EA-B5-FF-D8

    char \*src\_ip\_str = "192.168.31.163"; //源主机IP地址

    char \*dst\_ip\_str = "192.168.31.248"; //目的主机IP地址

    unsigned long src\_ip,dst\_ip = 0;

    lens = sprintf(send\_msg, "%s", "this is for the udp test");

    lib\_net = libnet\_init(LIBNET\_LINK\_ADV, "eth0", err\_buf);    //初始化

    if(NULL == lib\_net)

    {

        perror("libnet\_init");

        exit(-1);

    }

    src\_ip = libnet\_name2addr4(lib\_net,src\_ip\_str,LIBNET\_RESOLVE);  //将字符串类型的ip转换为顺序网络字节流

    dst\_ip = libnet\_name2addr4(lib\_net,dst\_ip\_str,LIBNET\_RESOLVE);

    lib\_t = libnet\_build\_udp(   //构造udp数据包

                                8080,

                                8080,

                                8+lens,

                                0,

                                send\_msg,

                                lens,

                                lib\_net,

                                0

                            );

    lib\_t = libnet\_build\_ipv4(  //构造ip数据包

                                20+8+lens,

                                0,

                                500,

                                0,

                                10,

                                17,

                                0,

                                src\_ip,

                                dst\_ip,

                                NULL,

                                0,

                                lib\_net,

                                0

                            );

    lib\_t = libnet\_build\_ethernet(  //构造以太网数据包

                                    (u\_int8\_t \*)dst\_mac,

                                    (u\_int8\_t \*)src\_mac,

                                    0x800, // 或者，ETHERTYPE\_IP

                                    NULL,

                                    0,

                                    lib\_net,

                                    0

                                );

    int res = 0;

    res = libnet\_write(lib\_net);    //发送数据包

    if(-1 == res)

    {

        perror("libnet\_write");

        exit(-1);

    }

    libnet\_destroy(lib\_net);    //销毁资源

    printf("----ok-----\n");

    return 0;

 }

编译代码时，需要加上 -lnet：