第 3 章 Ethernet 帧的封装与解析

3.1 设 计 目 的

帧是在数据链路层中进行数据传输的基本单位。熟悉帧结构对于理解网络协议的概念、网络层次结构、协议执行过程，以及网络问题处理的一般方法具有重要意义。本章练习的目的是根据数据链路层的基本原理，通过封装与解析标准格式的 Ethernet 帧，了解 Ethernet 帧结构中各个字段的含义与用途，从而深入理解网络协议的基本概念与工作原理。

3.2 相 关 知 识

本章涉及的相关知识包括数据链路层的概念与 Ethernet 帧结构。

3.2.1 数据链路层的概念

网络中的两台主机之间通信要遵循相同的网络协议。OSI 参考模型是国际标准化组织 ISO 制定的网络互联参考模型，它是一种分层次的网络体系结构。OSI 参考模型要求各个结点具有相同的层次，不同结点的相同层次具有相同功能，每层使用下层提供的服务并向上层提供服务。图 3-1 给出了 OSI 参考模型的结构。OSI 参考模型从下至上依次是物理层、数据链路层、网络层、传输层、表示层、会话层与应用层。其中，应用层包括各种熟悉的网络协议，如 FTP、HTTP、SMTP 与 TELNET 等。

图 3-1 OSI 参考模型的结构

OSI 参考模型的每层都有各自的数据传输单位，在经过不同层时需要组装成符合要求的单位。帧(frame)是在数据链路层中进行数据传输的基本单位。

根据 OSI 参考模型的规定，数据链路层(data link layer)是参考模型的第 2 层。数据链路层的主要功能是：在底层的物理层提供的服务基础上，在作为通信实体的主机之间建立数据链路连接，在这个连接上输出以帧尾单位的数据包，并采取差错控制与流量控制的方法，将有差错的物理连接变成无差错的数据链路。

3.2.2 Ethernet 帧的结构

术语“帧”最早来源于串行线路上的通信。发送结点在发送数据的前后各添加特殊的字符构成帧，这些特殊的字符就是帧头与帧尾。Ethernet 从某种程度上可以被看作网络结点之间的数据链路层连接。IEEE 802.3 标准是在 Ethernet V2.0 规范的基础上制定的，但是 Ethernet V2.0 规范和 IEEE 802.3 标准中的 Ethernet 帧结构有一些差别。本书将按 IEEE 802.3 标准的帧结构进行讨论。图 3-2 给出了 Ethernet 帧结构。帧的基本长度单位是字节(byte)，可以简写为 B。

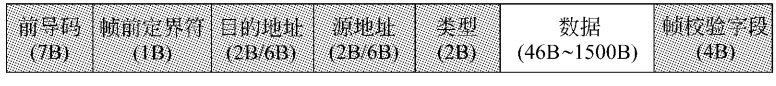

图 3-2　Ethernet 帧结构

IEEE 802.3 标准的 Ethernet 帧结构由 6 部分组成。

1. 前导码与帧前定界符

前导码由 56 位(7 字节)的 10101010……10101010 比特序列组成，换算成十六进制就是 7 个连续的 0xaa。从 Ethernet 物理层的角度来看，从网卡开始接收数据到进入稳定状态需要一定的时间。设置该字段的目的是保证网卡在帧的目的地址到来前达到稳定状态。帧前定界符可以视为前导码的延续。帧前定界符由 8 位(1 字节)的 10101011 比特序列组成，换算成十六进制就是 1 个 0xab。

如果将前导码与帧前定界符一起来看，则在 62 位的 10101010……10 比特序列后出现 11，在这个 11 后面出现的是帧的目的地址字段。也就是说，在数据开始时出现前导码与帧前定界符，才能够代表这是一个合法的帧。前导码与帧前定界符主要起到接收同步的作用，这 8 字节的数据在接收后不需要保留，也不会计入帧头的长度字段值中。

2. 目的地址与源地址

目的地址与源地址分别表示帧的接收结点与发送结点的硬件地址。网络结点之间进行通信需要使用硬件地址，这个地址在不同层次有不同的表示方法，在数据链路层使用的是网卡的 MAC 地址。这个 MAC 地址在出厂时就固化在网卡的 EPROM 中，并且可以保证该地址在全球范围内是唯一的。MAC 地址与 IP 地址是不同的概念，无论网卡被连

接在哪个具体的局域网中，也无论连接网卡的计算机移动到哪个位置，网卡的 MAC 地址都是固定不变的。

在研究 Ethernet 帧结构时，主要讨论以下两个问题。

(1) 目的地址和源地址长度可以是 2 字节或 6 字节。早期的 Ethernet 曾使用过 2 字节的地址。目前，所有 Ethernet 都使用 6 字节(48 位)的地址。MAC 地址由两个部分组成：3 字节的公司唯一标识(OUI)与 3 字节的扩展唯一标识(EUI)。图 3-3 给出了 MAC 地址的表示方法。MAC 地址是由十六进制数用连字符隔开来表示，例如 08-01-00-2A-10-C3。

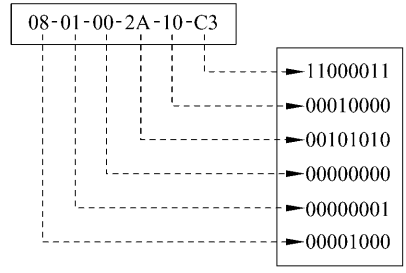

图 3-3　MAC 地址的表示方法

(2) Ethernet 帧的目的地址有 3 种类型：单结点地址(unicast address)、多结点地址(multicast address)和广播地址(broadcast address)。其中，目的地址的第一位为 0 表示单结点地址，第一位为 1 表示多结点地址，目的地址为全 1 则表示广播地址。如果目的地址是单结点地址，表示该帧只能被与目的地址相同的结点接收。

3. 长度字段

长度字段(2 字节)表示数据字段包含的字节数。IEEE 802.3 标准规定 Ethernet 帧的数据字段最小长度为 46 字节，最大长度为 1500 字节。由于帧头部分长度为 18 字节(前导码与帧前定界符不计入长度)，因此帧的最小长度为 64 字节，最大长度为 1518 字节。规定最小长度的目的是使接收结点能有足够时间检测到冲突，并及时通知源结点重新传输位成功接收的帧。

4. 数据字段

数据字段用来保存发送给目的结点的实际数据。由于帧数据字段的最小长度为 46 字节，如果一个帧的数据长度少于 46 字节，则应将数据字段填充至 46 字节。填充字符可以是任意的字符，在实际应用中经常用 0 完成填充。但是，填充部分不会计入长度字段的值中。另外，帧数据字段的最大长度为 1500 字节。

5. 帧校验字段

帧校验字段(4 字节)用来判断帧在传输中是否出错。IEEE 802.3 标准规定帧校验的范围包括：目的地址、源地址、长度、数据等字段。前导码与帧前定界符不需要进行帧校验。帧校验采用 32 位的 CRC 校验，即 CRC-32 校验算法。CRC-32 校验的生成多项

式为：

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

另外，在某些帧结构中还会包括帧类型字段，用来识别该帧所承载的数据类型。当一个帧到达目的结点时，操作系统会根据帧类型决定用哪个协议软件模块进行处理。自识别帧的优点是允许在同一网络中混合使用多种协议。

3.3 例题分析

3.3.1 设计要求

根据给出的 IEEE 802.3 格式的 Ethernet 帧结构，编写程序来解析并显示帧的各个字段值，并将得到的数据字段值组合写入输出文件。Ethernet 帧数据从输入文件中获得，默认的输入文件为二进制数据文件。在本练习中为了简便起见，只读取帧校验字段值而不进行 CRC 校验，每个帧的数据字段按最大长度 100 字节封装。程序设计的具体要求如下。

(1) 要求程序为命令行程序。例如，可执行文件名为 FrameParse.exe，则程序的命令行格式如下。

```
FrameParse input_file output_file
```

其中，input_file 为输入文件，output_file 为输出文件。

(2) 要求将全部字段内容显示在控制台上，具体格式如下。

帧 1 开始解析
前导码：xx xx xx xx xx xx xx
帧前定界符：xx
目的地址：xx-xx-xx-xx-xx-xx
源地址：xx-xx-xx-xx-xx-xx
长度字段：xx xx
数据字段：……
帧校验字段：xx xx xx xx

帧 2 开始解析
⋮

由于帧数据字段封装的是文本信息，因此该字段内容请按字符串格式输出，其他各字段均按十六进制格式输出。

(3) 要求有良好的编程规范与注释。编程所使用的操作系统、语言和编译环境不限，

但在提交的说明文档中需要加以注明。

(4) 要求撰写说明文档,包括程序的开发思路、工作流程、关键问题、解决思路,以及进一步的改进等。

3.3.2 关键问题

1. 文件的读写操作

由于 Ethernet 帧数据需要从输入文件获得,而数据字段内容也需要写入输出文件,因此首先要完成对文件的相关操作。例如,文件的打开、读取、写入与定位等。通过 fstream、ifstream 与 ofstream 可以完成文件的相关操作。其中,fstream 可以对文件进行读写操作;ifstream 可以对文件进行读操作;ofstream 可以对文件进行写操作。下面,给出的是几个文件操作函数。

- file. open(): 按照指定方式打开文件,例如十进制、二进制或十六进制等。
- file. read(): 从文件指针位置读取指定字节的数据。
- file. write(): 向文件指针位置写入指定字节的数据。
- file. get(): 从文件指针位置读取一字节的数据。
- file. put(): 向文件指针位置写入一字节的数据。
- file. seekg(): 将文件指针移到指定位置。
- file. tellg(): 获得文件指针位置的偏移量。

下面,给出的是获得文件数据长度的代码。

```
fstream file;                              //创建输入文件流
file.open(argv[1],ios::binary);            //打开指定输入文件
file.seekg(0,ios::end);                    //将文件指针移到文件结尾
int length= file.tellg();                  //获得到文件结尾的偏移量
⋮
```

2. 解析帧头部字段

在完成 Ethernet 帧解析的过程中,首先要进行的是帧头部的解析,这个处理过程是比较简单的。这时,只需将前导码(7 字节)、帧前定界符(1 字节)、目的地址(6 字节)、源地址(6 字节)、长度字段(2 字节)的值,根据每个字段的规定长度依次读取,然后按照题目要求的格式输出即可。在对每个字段的值进行读取的过程中,需要注意同一字段的值按哪种次序(顺序或逆序)读取,这就涉及到网络字节序的问题。

前导码与帧前定界符共同起接收同步的作用,前导码的值为 7 个 10101010(十六进制为 0xaa),帧前定界符的值为 10101011(十六进制为 0xab),因此“aaaaaaaaaaaaaaab”代

表着一个帧的开始。由于在输入文件中有可能封装着多个帧，因此需要根据前导码与帧前定界符找到每个帧的开始位置。在练习中为了简便起见，不考虑在数据字段中出现与前导码、帧前定界符相同数据的情况。

下面，给出的是确定帧开始位置的代码。

```
file.open(argv[1],ios::binary);                    //打开指定输入文件
file.seekg(0,ios::beg);                            //将文件指针移到开始处
while(文件未结束)
{
    for(i=0;i<7;i++)                               //查找前导码的位置
     if(file.get()!=0xaa)
          ⋮
    if(file.get()!=0xab)                           //查找帧前定界符的位置
          ⋮
}
```

3. 解析数据字段

在进行帧数据字段的解析过程中，需要注意的问题是数据字段的长度。IEEE 802.3 标准规定帧数据字段的最小长度为 46 字节，最大长度为 1500 字节。如果数据长度小于 46 字节，需要通过填充“0”来补足 46 字节，但是这些“0”的个数不计入长度字段。需要根据长度字段值处理数据字段，如果得到长度字段的值小于 46，则需要将填充的“0”从数据字段去掉；如果得到长度字段的值等于 1500，则需要判断紧随其后是否仍有一个帧。

下面，给出的是输出数据字段值的代码。

```
char * data=new char[length];                      //创建数据缓冲区
file.read(data,length);                            //将数据从输入文件读到缓冲区
outfile.write(data,length);                        //将数据从缓冲区写入输出文件
if(length<46)                                      //数据长度小于 46,去掉填充的 0
    for(i=0;i<46-length;i++)
        file.get();
delete[] data;                                     //释放数据缓冲区
```

4. 程序流程图

图 3-4 给出了主程序流程图。要求输入的命令行参数必须正确，除了程序本身的名称以外，还需要有一个输入文件名与一个输出文件名。如果命令行参数的个数不是两个，或输入文件、输出文件无法正确打开，则程序在输出错误信息后退出。

3.3.3 程序源代码

```
#include <fstream.h>
#include <iostream.h>
#include <string.h>

void main(int argc,char * argv[])
{
    if(argc!=3)
        //检查命令行参数
    {
        cout<<endl<<"请按以下格式输入命
    令行: FrameParse input_file
                    output_file"<<endl;

        return;
    }

    fstream outfile;
        //创建输出文件流
    outfile.open(argv[2],ios::in|ios::out
    |ios::binary|ios::trunc);
        //打开指定输出文件,二进制读写方式

    fstream infile;
        //创建输入文件流
    infile.open (argv [1], ios :: in | ios ::
    binary|ios::nocreate);
        //打开指定输入文件,二进制读方式
    if(!infile.is_open())
        //检查输入文件能否打开
    {
        cout<<endl<<"无法打开输入文件"
    <<endl;
        return;
    }

    bool bframe=true;
    int nframes=0;
```

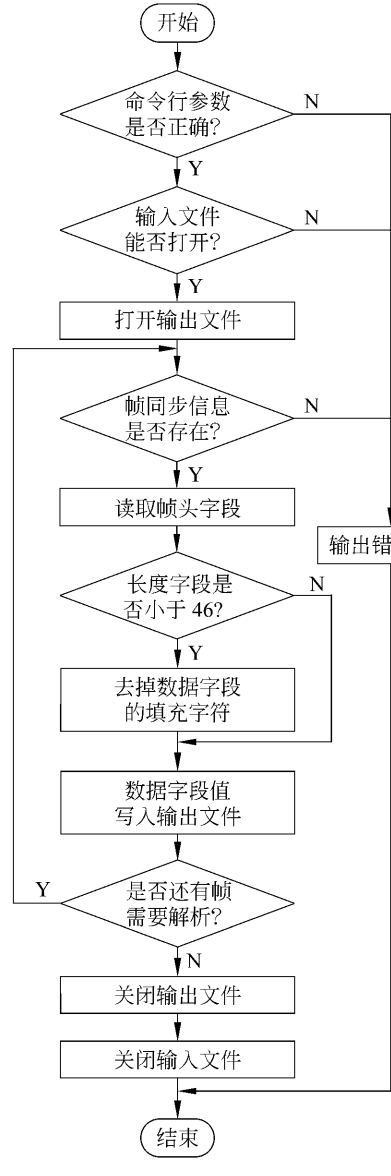

图 3-4 主程序流程图

```
int nframenum=0;
int nframelen=0;
while(bframe)                                    //判断帧是否需要拆分
{
    nframenum++;
    cout<<endl<<"帧"<<nframenum<<"开始解析"<<endl;

    nframes=infile.tellg();
    for(int i=0;i<7;i++)                         //查找 7字节前导码
        if(infile.get()!=0xaa)
        {
            cout<<"没找到合法的帧"<<endl;
            infile.close();
            return;
        }
    if(infile.get()!=0xab)                       //查找 1字节帧前定界符
    {
        cout<<"没找到合法的帧"<<endl;
        infile.close();
        return;
    }

    infile.seekg(nframes,ios::beg);
    cout<<endl<<"前导码: ";
    for(i=0;i<7;i++)                             //写入 7字节前导码
        cout<<hex<<infile.get()<<dec<<" ";
    cout<<endl<<"帧前定界符: ";
    cout<<hex<<infile.get();                     //写入 1字节帧前定界符

    cout<<endl<<"目的地址: ";
    for(i=0;i<6;i++)
    {
        cout<<hex<<infile.get()<<dec;            //读取 6字节目的地址
        if(i!=5)                                 //格式:xx-xx-xx-xx-xx-xx
            cout<<"- ";
    }

    cout<<endl<<"源地址: ";
    for(i=0;i<6;i++)
```

```
        {
            cout<<hex<<infile.get()<<dec;   //读取 6 字节源地址
            if(i!=5)                        //格式:xx-xx-xx-xx-xx-xx
                cout<<"- ";
        }

        cout<<endl<<"长度字段: ";
        cout<<hex<<infile.get()<<" ";       //读取长度字段的高 8 位
        nframelen=infile.get();             //读取长度字段的低 8 位
        cout<<hex<<nframelen;

        char * data=new char[nframelen];
        infile.read(data,nframelen);        //将数据从输入文件读到 data
        outfile.write(data,nframelen);      //将数据从 data 写入输出文件
        cout<<endl<<"数据字段: ";
        for(i=0;i<nframelen;i++)
            cout<<data[i];
        delete data;

        if(nframelen<100)                   //数据长度大于 100,分成多个帧
            bframe=false;
        if(nframelen<46)                    //数据长度小于 46,处理补 0 问题
            for(i=0;i<46-nframelen;i++)
                infile.get();

        cout<<endl<<"帧校验字段: ";
        for(i=0;i<4;i++)                    //写入 4 字节帧校验字段
            cout<<hex<<infile.get()<<dec<<" ";
        cout<<endl;
    }

    cout<<endl<<"帧全部解析完成"<<endl;
    outfile.close();                        //关闭指定输出文件
    infile.close();                         //关闭指定输入文件
}
```

图 3-5 给出了 Ethernet 帧的解析过程。程序命令行输入为 FrameParse input output,从 input 中读取数据并根据长度进行分帧或补 0,依次将前导码、帧前定界符、目的地址、源地址、长度、数据与帧校验字段的值写入 output。

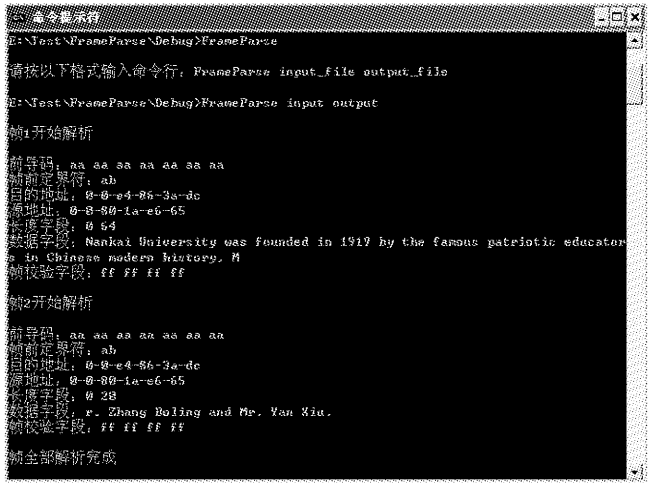

图 3-5　Ethernet 帧的解析过程

3.4　练　习　题

根据 IEEE 802.3 格式的 Ethernet 帧结构，编写程序来将原始数据封装成一个或多个帧，并将这些帧的各个字段值写入输出文件。原始数据从输入文件中获得，默认的输入文件为二进制数据文件。在本练习中，只填写帧校验字段而不进行 CRC 校验，每个帧的数据字段按最大长度 100 字节封装。程序设计的具体要求如下。

(1) 要求程序为命令行程序。例如，可执行文件名为 FrameEncap.exe，则程序的命令行格式如下。

```
FrameEncap input_file output_file
```

其中，input_file 为输入文件，output_file 为输出文件。

(2) 要求将部分字段内容显示在控制台上，具体格式如下。

帧 1 开始封装

长度字段：xx xx

数据字段：……

帧 2 开始封装

⋮

由于帧数据字段封装的是文本信息，因此该字段内容请按字符串格式输出，其他各字