TCP/IP 网络编程与 C 代码示例

摘要：本文主要内容包括 ：TCP/IP 网络编程中用到的基本概念 ,在 win32 下使用 winsock 进行编程的指导以及对应的源码。

在 OSI 的各层所使用的协议：

1.应用层： Telnet,FTP,HTTP,DNS,SMTP,POP3

2.传输层： TCP,UDP

TCP: 面向连接的可靠的传输协议， 通信前建立三次握手， 握手成功后才能通信，对数据准确性要求较高的场合使用， 如从网上载的安装文件， 不能缺少任何信息

UDP: 是无连接的，不可靠的传输协议，不需要建立连接，也没有重传和确认的机制，在实时性要求较高， 但对数据准确度要求不是很高的场合使用， 如视频会议，在线观看电影，当中丢失个别数据包并不影响整体的效果。

3. 网络层： IP

因为 OSI 七层结构较为复杂，所以使用较多的是 TCP/IP 模型，现在 TCP/IP 已经成为 Internet 上通用的工业标准

TCP/IP 模型包括 4 个层次：应用层，传输层，网络层，网络接口

端口

1. 为了标识通信实体中进行通信的进程（应用程序）， TCP/IP 协议提出了协议端口的概念

2. 端口是一种抽象的软件结构（包括一些数据结构和 I/O 缓冲区）。应用程序通过系统调用和某端口建立连接（ binding ）后，传输层传给该端口的数据都被相应的进程所接收，相应进程发给传输层的数据都通过该端口输出

3. 端口用一个整数型标识符来表示，即端口号。端口号跟协议相关， TCP/IP 传输层的两个协议 TCP 和 UDP 是完全独立的的两个软件模块， 因此各自的端口号也相互独立

4. 端口使用一个 16 位的数字来表示，它的范围是 0~65535 ，1024 以下的端口号保留给预定义的服务，例如， http 使用 80 端口

套接字（ Socket ）

1.Socket 的出现，使得程序员可以很方便的访问 TCP/IP ，从而开发各种网络应用的程序

2. 套接字存在于通信区域中， 通信区域也叫地址族， 他是一个抽象的概念， 主要用于通过套接
通信的进程的共有特性综合在一起。 套接字通常只与同一个区域的套接字交换数据。

套接字的类型

1.流式套接字（ SOCK_STREAM ）
提供面向连接的，可靠的数据传输服务，数据无差错，无重复的发送，且按发送的顺序接收，基于 TCP 协议

2.数据保式套接字 (SOCK_DGRAM)
提供无连接的服务， 数据包以独立包形式发送， 不提供无错误的保证， 数据可能丢失或重复，且接收顺序混乱，基于 UDP 协议

基于 TCP （面向连接）的 Socket 编程

服务器端顺序：

1. 加载套接字库

2.创建套接字（ socket）

3. 将套接字绑定到一个本地地址和端口上（ bind ）

4. 将套接字设为监听模式，准备接收客户请求（ listen ）

5. 等待客户请求的到来； 当请求带来后， 接受连接请求， 返回一个新的对应于此次连接的套接字（ accept ）

6. 用返回的套接字和客户端进行通信（ send/recv ）

7. 返回，等待另一个客户请求

8.关闭套接字 (closesocket)

客户端程序：

1. 加载套接字库

2.创建套接字 (socket)

3. 向服务器发送连接请求（ connect ）

4.和服务器端进行通信 (send/receive)

5.关闭套接字 (closesocket)

基于 UDP （面向无连接）的 socket 编程

服务器端（接收端）程序：

1. 加载套接字库

2.创建套接字（ socket）

3. 将套接字绑定到一个本地地址和端口上（ bind ）

4.等待接收数据（ recvfrom）

5.关闭套接字 (closesocket)

客户端（发送端）程序

1. 加载套接字库

2.创建套接字（ socket）

3. 向服务器发送数据（ sendto ）

4.关闭套接字 (closesocket)

创建基于 TCP 协议的 CS 程序的 Server 端所涉及的相关函数说明 （按使用的先后顺序排列）：

1.intWSAStartup （WORDwVersionRequested,LPWSADATAlpWSAData ）:作用是加载套接字库和进行套接字库的版本协商

a.参数 wVersionRequested:用于指定准备加载的 Winsock 库的版本，高位字节指定所需要的 Winsock 库的副版本，低位字节则是主版本，可用 MAKEWORD(X,Y)(其中，x 为高位字节，y 为低位字节)方便获得

wVersionRequested的正确值。

b.参数 lpWSAData：指向 WSADATA 结构的指针， WSAStartup 用其加载的库版本有关的信息填在这个结构中

2.SOCKETsocket(intaf,inttype,intprotocol) :

a. 参数 af 指定地址族，对于 TCP/IP 协议的套接字，它只能是 AF_INET(也可写成 PF_INET) 。

b. 参数 type 指定 Socket 类型，对于 1.1 版本的 Socket ，它只支持两种类型的套接字， SOCK_STREAM 指定产生流式套接字， SOCK_DGRAM 产生数据报套接字。

c. 参数 protocol 与特定的地址家族相关的协议，如果指定为 0，那么他就会根据地址格式和套接字类别， 自动为你选择一个合适的协议。 这是推荐使用的一种选择协议的方式。

3. intbind(SOCKETs,conststructsockaddrFAR*name,intnamelen):

a. 第一个参数指定要绑定的套接字，第二个参数指定该套接字的本地地址信息，是指向 sockaddr 结构的指针变量，由于该地址结构是为了所有的地址家族准备使用的，这个结构可能（通常会）随使用的网络协议不同而不同，所以，要用第三个参数指定该地址结构的长度。 sockaddr 机构定义如下：

```
structsockaddr
{
u_shortsa_family;
charsa_data[14];
}
```

b. 上述结构第一个字段指定该地址家族，在这里必须设为 AF_INET 。sa_data 仅仅是表示要求一块内存分配区， 起到占位的作用， 该区域中指定与协议相关的具体地址信息。 由于实际要求的只是内存区， 所以对于不同的协议家族， 用不同的结构来替换 sockaddr 。在 TCP/IP 中，我们可以用 SOCKADDR_IN 结构来代替 sockaddr ，以方便我们填写地址信息。

```
c.structSOCKADDR_IN{
shortsin_family;
unsignedshortsin_port;
```

structin_addrsin_addr;

charsin_zero[8];

};

sin_family 表示地址族，对于 IP 地址，sin_family 成员将一直是 AF_INET ；成员 sin_port 指定的将要分配给套接字的端口；成员 sin_addr 给出的是套接字的主机 IP 地址；sin_zero 只是一个填充数，以使 sockaddr_in 结构和 sockaddr 结构的长度一样，一般不用设置。

除了 sin_family 外，SOCKADDR_IN 其他成员是按网络字节顺序表示的。 所以需要进行转换： htonl(INADDR_ANY) ， htons(6000)，其中 6000 是端口号。

另外结构体的名称大写和小写指的是同一个。

将 IP 地址指定为 INADDR_ANY ，允许套接字向任何分配给本机器的 IP 地址发送或接收数据。一般一台机器一个网卡，但对于多网卡的机器， INADDR_ANY 将简化应用程序的编写。 将地址指定为 INADDR_ANY ，允许一个独立的应用接受发自多个接口的回应。如果我们只想让套接字使用多个 IP 中的一个地址，必须指定实际地址，要做到这一点，可以用 inet_addr() 函数，这个函数需要一个 IP 地址（如 192.168.80.88 ），返回一个适合分配给 S_addr 的 u_long 类型的数值。 Inet_ntoa() 函数完成相反的转换，它接受一个 in_addr 结构体类型的参数并返回一个以点分十进制的 IP 地址字符串。

htonl 把一个 u_long 类型从主机字节序转换为网络字节序。

htons 把一个 u_short 类型从主机字节序转换为网络字节序。

4. intlisten(SOCKETs,intbacklog): 将套接字设置为监听模式，其中第二个参数设置等待请求连接的最大的值， 即如果设置为 n，则前 n 个请求会放置在系统的请求连接队列中，应用程序会依次对这些请求进行服务，但第 n+1 个连接请求则会被拒绝。

5. SOCKETaccept(SOCKETs,conststructsockaddrFAR*addr,intFAR*ad

drlen):

从客户端接收请求， 并创建连接， 如果连接成功， 则会返回一个当前成功建立连接的套接字， 该套接字不是上面创建的监听套接字， 而是仅仅适用于当前的一个请求连接，如果建立连接失败，则返回值是 INVALID_SOCKET ，并且可以适

用 WSAGetLastError() 函数得到相关的失败信息，具体的 errorcode 具体意义见 MSDN 中 accept 函数的最后部分的介绍

6. send 函数：向客户端发送指定信息

7. recv 函数：得到从客户端传递过来的信息

8.closesocket(SOCKETs)将指定的套接字关闭，从而释放资源

9.WSACleanup()：终止对 winsocket 库的使用

10.hello

服务器端的实现过程（ Win32 控制台程序）：

说明：

1.对于 winsock 库的类的使用，必须包含 winsock2.h 头文件

2.在 setting 中的 link 下的 object/librarymodules 中添加“ ws2_32.lib ”，注意和前面的字段之间用空格分隔

3. 本服务器程序先于服务器端启动

```
#include"winsock2.h"
#include"stdio.h"

void main()
{
//加载套接字（ winsock）库，加载这段代码拷贝于 MSDN 中 WSAStartup 的介绍
WORDwVersionRequested;
WSADATAwsaData;
int err;

wVersionRequested=MAKEWORD(1,1); //版本号为 1.1

err=WSAStartup(wVersionRequested,&wsaData);
if (err!=0){
return;
}
```

```
if(LOBYTE(wsaData.wVersion)!=1||
HIBYTE(wsaData.wVersion)!=1){
WSACleanup();
return;
}

//创建套接字
SOCKETsockServer=socket(AF_INET,SOCK_STREAM,0); //SOCK_STREAM 参数
设置为 TCP 连接
SOCKADDR_INaddrServer; //设置服务器端套接字的相关属性
addrServer.sin_addr.S_un.S_addr=htonl(INADDR_ANY);//设置 IP
addrServer.sin_family=AF_INET;
addrServer.sin_port=htons(1234);//设置端口号

//将套接字绑定到本地地址和指定端口上
bind(sockServer,(SOCKADDR*)&addrServer, sizeof(SOCKADDR));

//将套接字设置为监听模式，并将最大请求连接数设置成 5，超过此数的请求全
部作废
listen(sockServer,5);

SOCKADDR_INaddrClient; //用来接收客户端的设置，包括 IP 和端口
int len=sizeof(SOCKADDR);
while(1)//不断监听
{
//得到创建连接后的一个新的套接字，用来和客户端进行沟通，原套接字继续监
听客户的连接请求
SOCKETsockConn=accept(sockServer,(SOCKADDR*)&addrClient,&len);
```

```
if(sockConn!=INVALID_SOCKET)  //创建成功
{
charsendInfo[100];
//inet_ntoa 将结构转换为十进制的   IP 地址字符串
sprintf(sendInfo,"welcome%stothistest!",inet_ntoa(addClient.sin_addr));
//成功建立连接后向客户端发送数据，结果将显示在客户端上
send(sockConn,sendInfo,strlen(sendInfo)+1,0);

//从客户端接收数据，结果显示在服务器上
charrecvInfo[100];
recv(sockConn,recvInfo,100,0);
printf( "%s\n",recvInfo);

//将本次建立连接中得到套接字关闭
closesocket(sockConn);
}
else
{
int errCode=WSAGetLastError();
printf( "theerrcodeis:%d\n",errCode);
}

}

//如果本程序不是死循环，那么在此处还应添加以下代码：
closesocket(sockServer);//对一直处于监听状态的套接字进行关闭
WSACleanup();//终止对 winsocket 库的使用

}
```

创建基于 TCP 协议的 CS 程序的 Client 端所涉及的相关函数说明（按使用的先后顺序排列）：

1.intWSAStartup（WORDwVersionRequested,LPWSADATAlpWSAData）：说明同上

2.SOCKETsocket(intaf,inttype,intprotocol): 说明同上

3.connect 函数：同服务器建立连接

4.send/recv:发送与接收，同上

5.closesocket：关闭套接字，同上

客户端的实现过程（ win32console 程序）：

说明：

1.对于 winsock 库的类的使用，必须包含 winsock2.h 头文件

2.在 setting 中的 link 下的 object/librarymodules 中添加“ ws2_32.lib ”，注意和前面的字段之间用空格分隔

3.启动客户端程序之前必须先启动服务器端的程序

```
#include"winsock2.h"
#include"stdio.h"

void main()
{
//加载套接字库
WORDwVersionRequested;
WSADATAwsaData;
int err;

wVersionRequested=MAKEWORD(1,1);//版本好为 1.1

err=WSAStartup(wVersionRequested,&wsaData);
```

```
if(err!=0){
return;
}

if(LOBYTE(wsaData.wVersion)!=1||
HIBYTE(wsaData.wVersion)!=1){
WSACleanup();
return;
}

SOCKETsockClient=socket(AF_INET,SOCK_STREAM,0); //SOCK_STREAM 参数
设置为 TCP 连接

SOCKADDR_INaddrServer; //服务器地址结构
addrServer.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");//服务器地址
addrServer.sin_port=htons(1234);//服务器端口号
addrServer.sin_family=AF_INET;

//与服务器端建立连接，进行通信
int connReult=connect(sockClient,(SOCKADDR*)&addrServer,sizeof(SOCKADDR)
);

if(connReult!=WSAEADDRNOTAVAIL)    //访问成功
{
//成功建立连接后向服务器端发送数据，结果将显示在服务器端上
send(sockClient,"thisislisi!" ,strlen("thisiszhangsan!")+1,0);

//接收来自服务器端发送来的信息
charrecvInfo[100];
recv(sockClient,recvInfo,100,0);
```

```
printf( "%s\n",recvInfo);
}
else
{
int errCode=WSAGetLastError();
printf( "theerrcodeis:%d\n",errCode);
}
closesocket(sockClient);
WSACleanup();
}
```

创建基于 UPD 的 CS 程序服务器端源代码：

说明:服务器端的代码在 XP 下运行 recvfrom 函数执行不成功，返回一个 10022 的错误代码，参看网上资源，得知 10022 代表某个参数设置错误。

花了大半天的时间研究后终于发现原来是 6000 端口在作怪，可能本机的 UDP 的 6000 端口在防火墙中被屏蔽了，通不过，但在 TCP 的 CS 程序中 6000 端口是通过的，程序调试一切正常，将端口修改为其他端口即可

```
#include "winsock2.h"
#include "stdio.h"

void main()
{
// 加载套接字（ winsock ）库，加载这段代码拷贝于 MSDN 中 WSAStartup 的介绍
WORDwVersionRequested;
WSADATAwsaData;
int err;

wVersionRequested=MAKEWORD(1,1);        // 版本号为 1.1
```

```
err=WSAStartup(wVersionRequested,&wsaData);
if (err!=0){
return   ;
}

if (LOBYTE(wsaData.wVersion)!=1||
HIBYTE(wsaData.wVersion)!=1){
WSACleanup();
return   ;
}

// 创建套接字
// 注意第二个参数和   TCP 设置不同
SOCKETsockServer=socket(AF_INET,SOCK_DGRAM,0);

SOCKADDR_INaddrServer;       // 设置服务器端套接字的地址结构的相关属性
addrServer.sin_addr.S_un.S_addr=htonl(INADDR_ANY);                // 设置 IP
addrServer.sin_family=AF_INET;
addrServer.sin_port=htons(6000);            // 设置端口号

// 将套接字和服务器地址结构绑定
bind(sockServer,(SOCKADDR*)&addrServer,            sizeof (SOCKADDR));

SOCKADDR_INaddrClient;
int len=  sizeof (SOCKADDR);
char buffer[100];
// 从客户端接收数据
intresult=recvfrom(sockServer,buffer,100,0,(SOCKADDR*)&addrClient,&len);
if (result==SOCKET_ERROR)
{
```

```
int errCode=WSAGetLastError();
printf(   "error:%d"     ,errCode);
}
else
{
printf(   "thismessagefromclient:%s\n"           ,buffer);
}
closesocket(sockServer);
WSACleanup();
}
```

创建基于 UDP 的 CS 程序客户端的代码：

```
#include   "winsock2.h"
#include   "stdio.h"

void main()
{
// 加载套接字（ winsock ）库，加载这段代码拷贝于   MSDN 中 WSAStartup   的介绍
WORDwVersionRequested;
WSADATAwsaData;
int err;

wVersionRequested=MAKEWORD(1,1);          // 版本号为 1.1

err=WSAStartup(wVersionRequested,&wsaData);
if (err!=0){
return   ;
}

if (LOBYTE(wsaData.wVersion)!=1||
```

```
HIBYTE(wsaData.wVersion)!=1){
WSACleanup();
return ;
}

// 创建套接字
SOCKETsockClient=socket(AF_INET,SOCK_DGRAM,0);          // 注意第二个参数和 TCP 设置
不同

SOCKADDR_INaddrServer;      // 服务器地址结构
addrServer.sin_addr.S_un.S_addr=inet_addr(          "127.0.0.1"    ); // 服务器地址
addrServer.sin_family=AF_INET;
addrServer.sin_port=htons(6000);          // 服务器端口号

sendto(sockClient,     "thismessagefromclient"       ,strlen(   "thismessagefromclient"       )+1,0,(
SOCKADDR*)&addrServer,       sizeof (SOCKADDR));
closesocket(sockClient);
WSACleanup();
}
```

基于 UDP 的聊天程序服务器端源程序：

说明：

1. 代码基本和上述 UDP 服务器端的程序相类似，只是多了 sendto 功能，即交互的功能

2. 支持退出请求响应

```
#include   "winsock2.h"
#include   "stdio.h"

void main()
```

```
{
// 加载套接字（ winsock ）库，加载这段代码拷贝于  MSDN 中 WSAStartup  的介绍
WORDwVersionRequested;
WSADATAwsaData;
int err;

wVersionRequested=MAKEWORD(1,1);          // 版本号为 1.1

err=WSAStartup(wVersionRequested,&wsaData);
if (err!=0){
return  ;
}

if (LOBYTE(wsaData.wVersion)!=1||
HIBYTE(wsaData.wVersion)!=1){
WSACleanup();
return  ;
}

// 创建套接字
// 注意第二个参数和  TCP 设置不同
SOCKETsockServer=socket(AF_INET,SOCK_DGRAM,0);

SOCKADDR_INaddrServer;       // 设置服务器端套接字的地址结构的相关属性
addrServer.sin_addr.S_un.S_addr=htonl(INADDR_ANY);                // 设置 IP
addrServer.sin_family=AF_INET;
addrServer.sin_port=htons(6000);           // 设置端口号

// 将套接字和服务器地址结构绑定
bind(sockServer,(SOCKADDR*)&addrServer,            sizeof (SOCKADDR));
```

```
SOCKADDR_INaddrClient;

int len= sizeof (SOCKADDR);

char recvBuf[100];

char tempBuf[100];

char sendBuf[100];


while (1)

{

// 从客户端接收数据

int result=recvfrom(sockServer,recvBuf,100,0,(SOCKADDR*)&addrClient,&len);

if (result==SOCKET_ERROR)

{

int errCode=WSAGetLastError();

printf( "error:%d" ,errCode);

break ; // 退出循环

}

else

{

if (recvBuf[0]== 'q' ) // 对方请求退出

{

// 同样将退出请求发送回去

sendto(sockServer, "q" ,strlen( "q" )+1,0,(SOCKADDR*)&addrClient, sizeof (SOCKAD

DR));

printf( "thechatend!" );

break ;

}

else

{

// 将对方发送过来的信息前段加上 IP 地址进行输出
```

```
sprintf(tempBuf,    "%ssaid:%s"    ,inet_ntoa(addrClient.sin_addr),recvBuf);
printf(  "%s\n"   ,tempBuf);

// 将用户的键盘输入发送到对方
printf(  "pleasinputthemessage:\n"        );
gets(sendBuf);    //        得        到        键        盘        输        入        内        容
sendto(sockServer,sendBuf,strlen(sendBuf)+1,0,(SOCKADDR*)&addrClient,                    sizeof
(SOCKADDR));
}
}
}

closesocket(sockServer);
WSACleanup();
}
```

基于 UDP 的聊天程序客户端端源程序：

```
#include   "winsock2.h"
#include   "stdio.h"

void main()
{
// 加载套接字（ winsock ）库，加载这段代码拷贝于   MSDN 中 WSAStartup   的介绍
WORDwVersionRequested;
WSADATAwsaData;
int err;

wVersionRequested=MAKEWORD(1,1);          // 版本号为 1.1

err=WSAStartup(wVersionRequested,&wsaData);
```

```
if (err!=0){
return   ;
}

if (LOBYTE(wsaData.wVersion)!=1||
HIBYTE(wsaData.wVersion)!=1){
WSACleanup();
return   ;
}

// 创建套接字
SOCKETsockClient=socket(AF_INET,SOCK_DGRAM,0);            // 注意第二个参数和    TCP  设置
不同

SOCKADDR_INaddrServer;       // 服务器地址结构
addrServer.sin_addr.S_un.S_addr=inet_addr(              "127.0.0.1"    ); // 服务器地址
addrServer.sin_family=AF_INET;
addrServer.sin_port=htons(6000);            // 服务器端口号

char tempBuf[100];
char recvBuf[100];
char sendBuf[100];

int len=  sizeof  (SOCKADDR);

while  (1)
{
printf(   "pleseinputthemessage:\n"          );
gets(sendBuf);      // 得到用户的键盘输入
sendto(sockClient,sendBuf,strlen(sendBuf)+1,0,(SOCKADDR*)&addrServer,                    sizeof
```

```
(SOCKADDR));

int result=recvfrom(sockClient,recvBuf,100,0,(SOCKADDR*)&addrServer,&len);
if (result==SOCKET_ERROR)
{
int errCode=WSAGetLastError();
printf( "error:%d" ,errCode);
break ; // 退出循环
}
else
{
if (recvBuf[0]== 'q' ) // 对方请求退出
{
// 同样将退出请求发送回去
sendto(sockClient, "q" ,strlen( "q" )+1,0,(SOCKADDR*)&addrServer, sizeof (SOCKAD
DR));
printf( "thechatend!" );
break ;
}
else
{
// 将对方发送过来的信息前段加上 IP 地址进行输出
sprintf(tempBuf, "%ssaid:%s" ,inet_ntoa(addrServer.sin_addr),recvBuf);
printf( "%s\n" ,tempBuf);
}
}

}

closesocket(sockClient);
```

```
WSACleanup();
}
```

小技巧：

1.sendto 和 recvfrom 函数中第四个类型为 structsockaddrFAR* 的参数一般采用 SOCKADDR_IN 结构类型，且都是对方的 SOCKADDR_IN 数据：如在服务器端，因为定义了自身的 SOCKADDR_INaddrServer ，其中的结构参数进行了具体设置，其中 IP 地址一项是设置为自动选择自己的网卡，且将套接字和该结构体进行了绑定， 然后申明了一个空的 SOCKADDR_INaddClient ，即对其中的结构参数不做任何的初始化， 但没有关系，recvfrom 方法调用在前，能得到客户端的详细信息 （这有点坐享其成的感觉， 服务器端就是等着先由客户端发送消息过来）， 将 addClient 填充满，然后调用 sendto 方法时，该结构体已经具有是客户端的信息了， 所以 sendto 的时候是正确的。 服务器端接收和发送调用的地址结构参数都是客户端的。

在客户端，定义了服务器端的地址结构，其中 IP 地址一项，填写的就是具体的目标服务器地址，发送和接收都是有针对性的， sendto 在前先发送消息到服务器，地址参数是服务器端的， 然后 revefrom 接收消息，地址参数也是服务器的。客户端接收和发送调用的地址结构参数都是服务器端的。

2. 本例子中的聊天室程序有一个缺陷，就是必须服务端先启，然后客户端启动，并且得先由客户端发送消息过来后，服务器端才能知道到底是谁发送消息来了，然后将有关信息按先前发送过来的相关地址信息发送回去。 对于客户端， 因为已经在地址结构中设置了服务器的 IP 地址，所以发送是有目的性的。综上就是先有客户端发送消息，再服务器端发送消息，然就是交替进行了。