第7章 网络编程技术

- 网络基础知识
- 网络编程基本概念
- Winsock主要函数
- 聊天室程序实例

计算机网络的定义

- 计算机网络是以能相互共享资源的方式互联的自治计算机系统的集合
  - 网络用于实现计算机资源共享
  - 互联的计算机是分布在不同地理位置的多台“自治计算机”
  - 连网计算机在通信中遵循相同的网络协议

网络体系结构

- 网络协议
  - 主机之间为交换数据而事先约定的规则
- 网络体系结构
  - 层次(Layer)与接口(Interface)
- 网络参考模型
  - OSI参考模型与TCP/IP参考模型

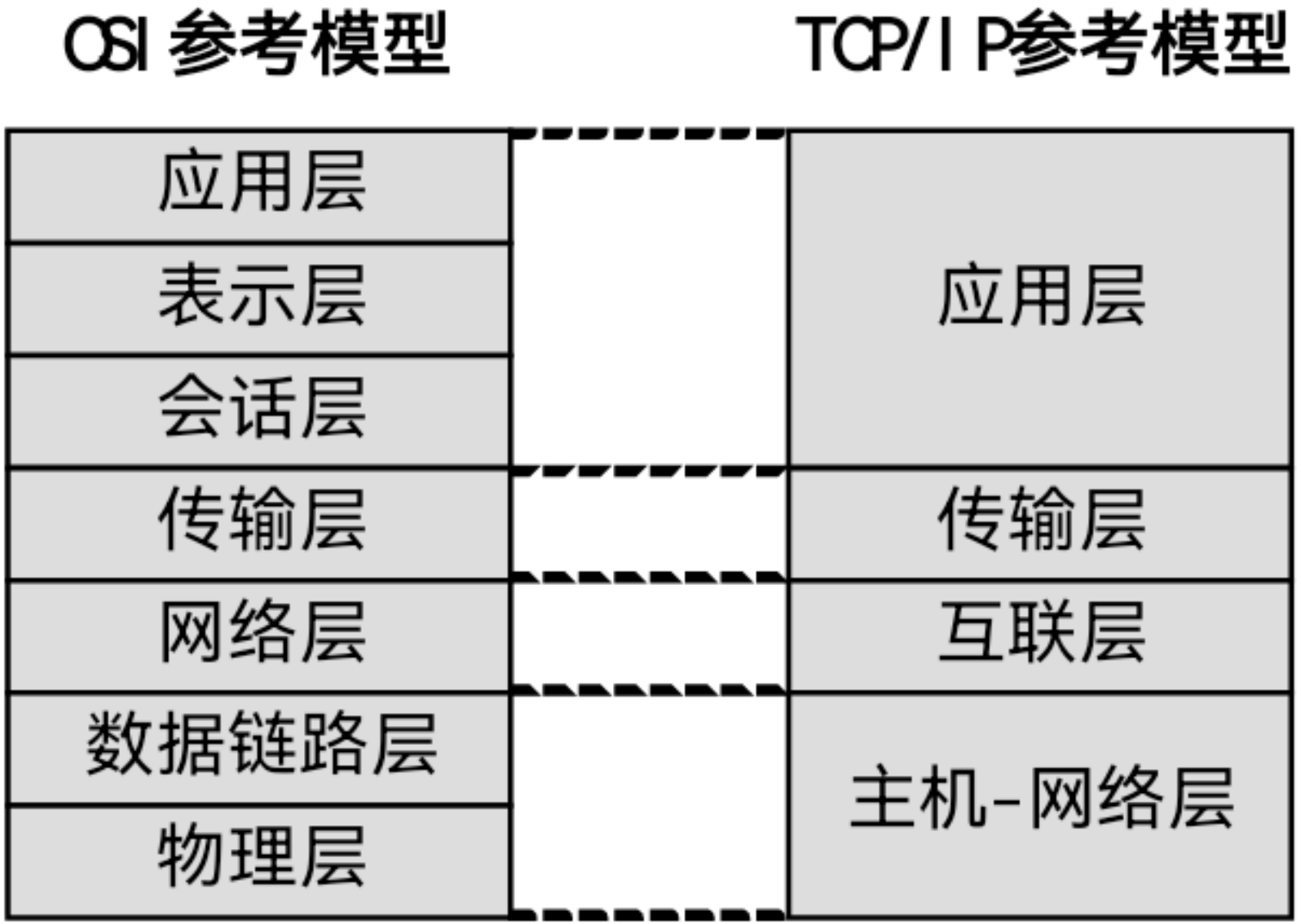
TCP/IP协议

- Windows网络通信建立在TCP/IP协议基础上
- TCP/IP协议族包含一系列构成互联网基础结构的网络协议
- TCP/IP代表两个重要协议
  - TCP：传输控制协议
  - IP：互联网协议

TCP/IP参考模型

- TCP/IP参考模型
  - 应用层(Application Layer)
  - 传输层(Transport Layer)
  - 互联层(Internet Layer)
  - 主机-网络层(Host-to-network Layer)

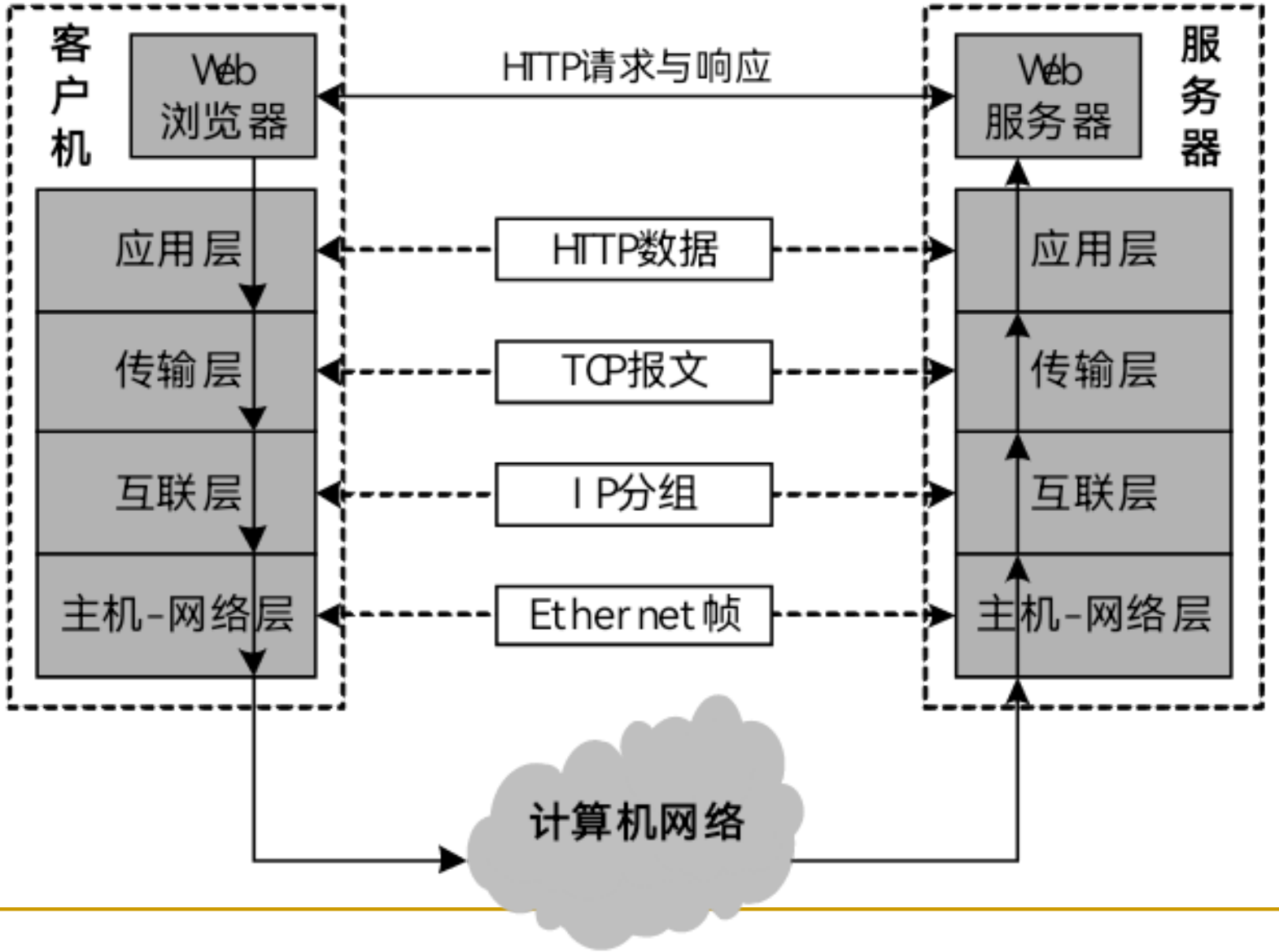
OSI模型与TCP/IP模型



典型的TCP/IP协议

- 互联层协议
  - IP(数据包的寻址、分片和重组等)
- 传输层协议
  - TCP(有连接服务)与UDP(无连接服务)
- 应用层协议
  - HTTP(网页浏览)、FTP(文件传输)、SMTP与POP(电子邮件)、Telnet(远程登录)

数据包的封装过程


客户机
Web浏览器
应用层
传输层
互联层
主机-网络层
HTTP请求与响应
HTTP数据
TCP报文
IP分组
Ethernet帧
计算机网络
服务器
Web服务器
应用层
传输层
互联层
主机-网络层

应用层协议的依赖性

- 依赖于TCP的应用层协议
  - HTTP、SMTP、POP、IMAP、FTP、TELNET
- 依赖于UDP的应用层协议
  - SNMP、TFTP
- 依赖于TCP/UDP的应用层协议
  - DNS、DHCP

网络编程基本概念

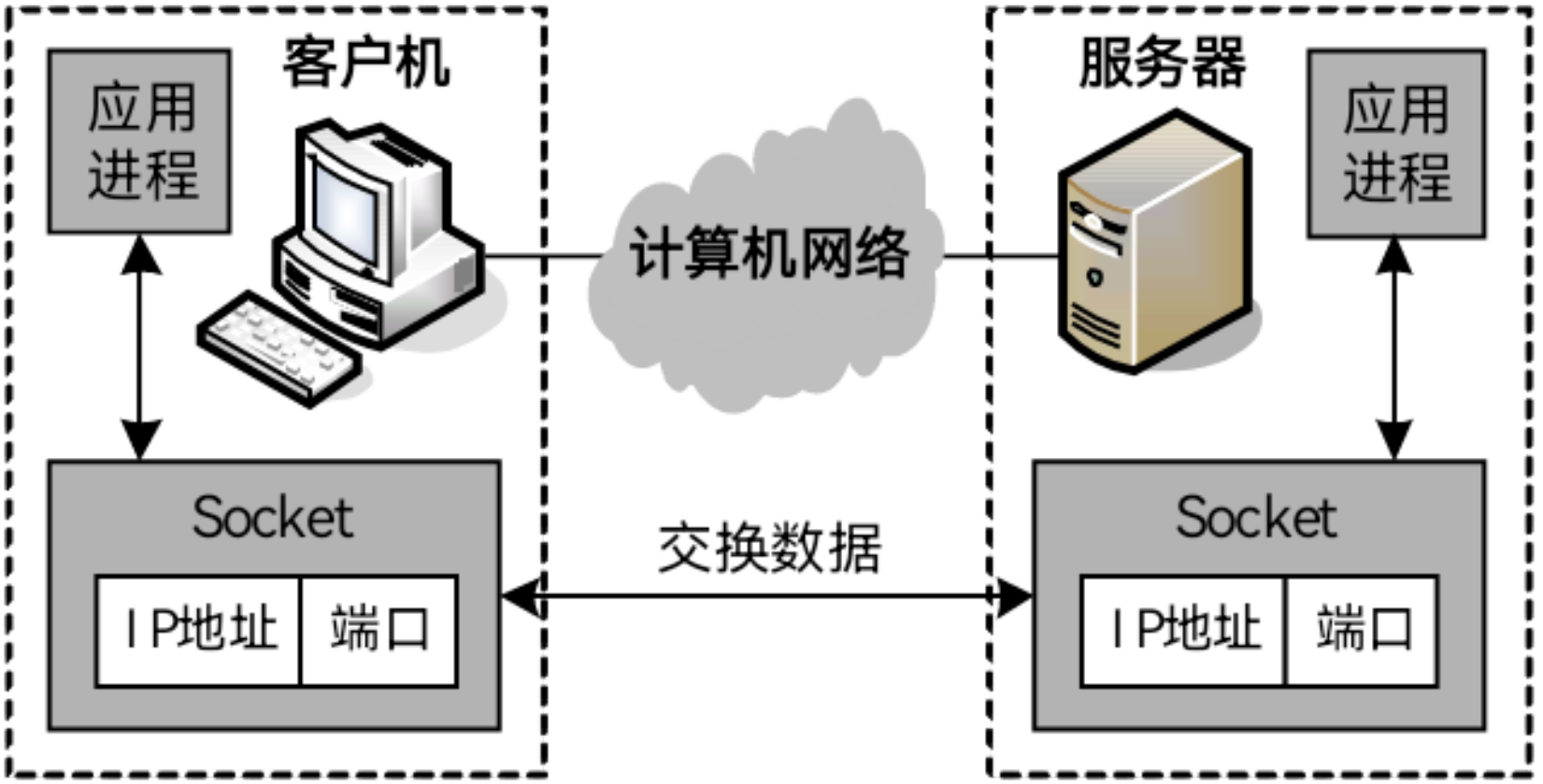
- 网络协议的概念
- 套接字的概念
- 客户机/服务器的概念

套接字的概念

- 套接字是一种网络编程接口
  - 英文：Socket
  - 字面意思：插座、插口
- 应用程序与网络协议之间的编程接口
- 套接字在TCP/IP模型中位于传输层
  - 对TCP、UDP协议进行抽象
  - 不涉及应用层协议

客户机/服务器模型

- 套接字是网络通信的端点



Windows套接字

- 套接字概念最初是由BSD Unix实现
- Microsoft移植Unix套接字大部分函数，形成Windows套接字
- Windows套接字针对消息驱动机制，对Unix套接字进行扩展，定义一部分新函数

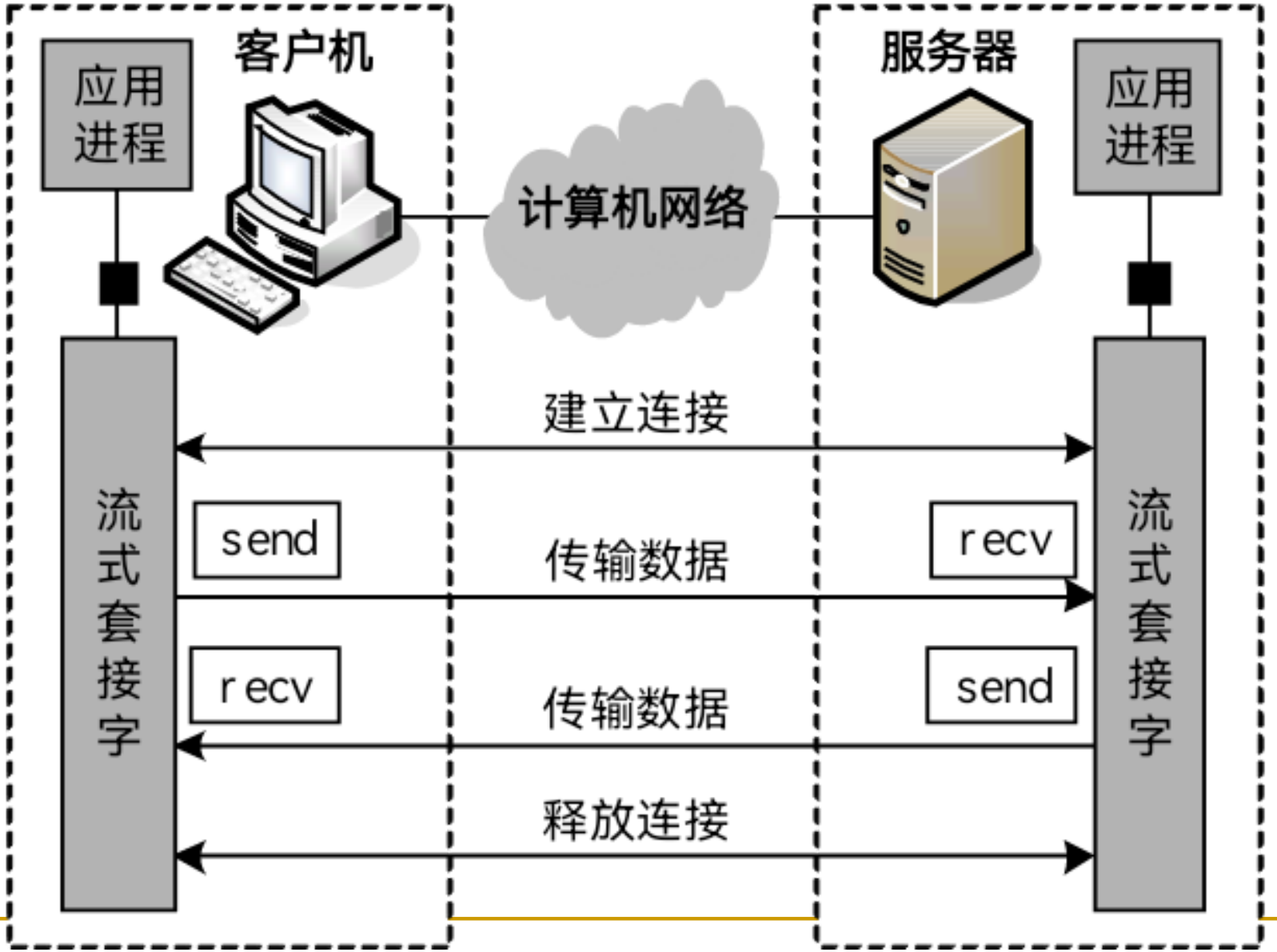
MFC提供的Windows套接字

- MFC提供的Windows套接字
  - 利用面向对象概念进行编程
  - 比直接调用Win32 API方便
- 与套接字相关的MFC类
  - CSocket类：套接字API的同步封装
  - CAsyncSocket类：套接字API的异步封装

套接字的种类

- 流式套接字(stream socket)
  - 使用TCP协议进行通信
  - 面向连接、可靠的数据流传输服务
  - 数据包不会出现丢失、重复、乱序
- 数据报套接字(datagram socket)
  - 使用UDP协议进行通信
  - 无连接、不可靠的数据报传输服务
  - 数据包可能出现丢失、重复、乱序

流式套接字(1)


客户机
服务器
应用进程
应用进程
计算机网络
流式套接字
流式套接字
建立连接
send
传输数据
recv
recv
传输数据
send
释放连接

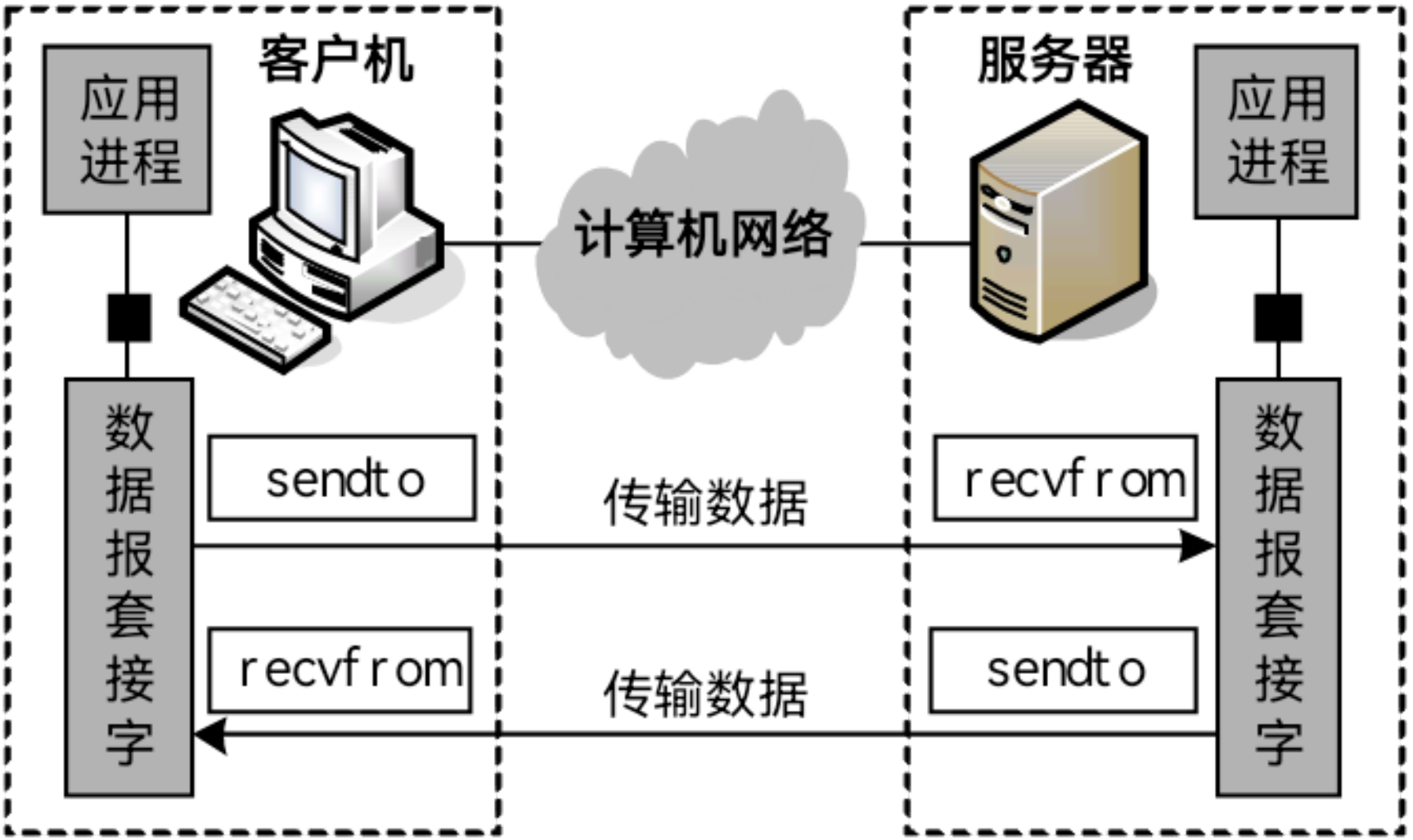
流式套接字(2)

- 服务器端
  - 创建套接字(socket)
  - 将套接字与IP地址、端口绑定(bind)
  - 将套接字设为监听模式(listen)
  - 当接收到连接请求后，返回用于此次连接的套接字(accept)
  - 用返回的套接字与客户端通信(send/recv)
  - 关闭套接字(closesocket)

流式套接字(3)

- 客户端
  - 创建套接字(socket)
  - 向服务器发出连接请求(connect)
  - 与服务器端通信(send/recv)
  - 关闭套接字(closesocket)

数据报套接字(1)


客户机
服务器
应用进程
应用进程
计算机网络
数据报套接字
数据报套接字
sendto
recvfrom
传输数据
recvfrom
sendto
传输数据

数据报套接字(2)

- 服务器端
  - 创建套接字(socket)
  - 将套接字与IP地址、端口绑定(bind)
  - 从客户端接收数据(recvfrom)
  - 向客户端发送数据(sendto)
  - 关闭套接字(closesocket)

数据报套接字(3)

- 客户端
  - 创建套接字(socket)
  - 向服务器端发送数据(sendto)
  - 从服务器端接收数据(recvfrom)
  - 关闭套接字(closesocket)

套接字工作模式

- 阻塞模式（发请求后等待响应）
  - 同步模式
  - CSocket类
- 非阻塞模式（发出请求后不等待，去干别的）
  - 异步模式
  - CAsyncSocket类

阻塞模式(1)

- 在阻塞模式下
  - 套接字函数在被调用后，一直等到全部操作完成后才返回
- 在建立连接时
  - 函数必须等到连接建立好
  - 调用函数的线程在此期间被挂起
  - 程序看起来像停止响应

阻塞模式(2)

- 以阻塞模式执行套接字函数，可能出现在执行某个函数需要长时间等待
- 建立多个线程执行每个套接字函数，程序编写起来比较繁琐

Winsock相关函数(1)

- WSAStartup()函数：加载Winsock库
  - int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData)
  - wVersionRequested指定Winsock版本，例如MAKEWORD(2,2)
  - lpWSAData是指向WSADATA结构的指针

Winsock相关函数(2)

- socket()函数：创建套接字
  - SOCKET socket(int af, int type, int protocol)
  - af指定地址族，TCP/IP协议为AF_INET，UNIX的协议为AF_UNIX
  - type指定套接字类型，流式套接字为SOCK_STREAM，数据报套接字为SOCK_DGRAM
  - protocol指定协议，IP协议为IPPROTO_IP

Winsock相关函数(3)

- bind()函数：将套接字与IP地址、端口绑定
  - int bind(SOCKET s, const struct sockaddr FAR *name, int namelen)
  - s指定绑定的套接字
  - name指定绑定的本地地址，是指向sockaddr结构的指针，当地址族为TCP/IP协议时，使用sockaddr_in结构
  - namelen指定本地地址的长度

Winsock相关函数(4)

- sockaddr_in结构

```
struct sockaddr_in {
  short sin_family;          //协议族
  u_short sin_port;          //端口号
  struct in_addr sin_addr;   //IP地址
  char sin_zero[8];
};
```

Winsock相关函数(5)

- inet_addr()函数：地址转换函数
  - 将IP地址从点分十进制转换为长整形，以便填入in_addr结构
  - unsigned long inet_addr(const char FAR *cp)
- inet_ntoa()函数：地址转换函数
  - 将IP地址从长整形转换为点分十进制
  - char FAR *inet_ntoa(struct in_addr in)

Winsock相关函数(6)

- listen()函数：将套接字设为监听模式
  - int listen(SOCKET s, int backlog)
  - s指定监听的套接字
  - backlog指定套接字维护的连接请求队列长度，SOMAXCONN表示最大值

Winsock相关函数(7)

- connect()函数：请求与服务器建立连接
  - int connect(SOCKET s, const struct sockaddr FAR *name, int namelen)
  - s指定客户端发送请求的套接字
  - name指定服务器端的套接字地址，通常使用的是sockaddr_in结构
  - namelen指定套接字地址结构的长度

Winsock相关函数(8)

- accept()函数：接受客户端的连接建立请求
  - SOCKET accept(SOCKET s, struct sockaddr FAR *addr, int FAR *addrlen)
  - s指定服务器端接收请求的套接字
  - name指定服务器端的套接字地址，通常使用的是sockaddr_in结构
  - namelen指定套接字地址结构的长度

Winsock相关函数(9)

- send()函数：发送数据
  - int send(SOCKET s, const char FAR *buf, int len, int flags)
  - s指定发送端使用的套接字
  - buf指定发送端保存等待发送数据的缓冲区
  - len指定发送数据的字节数
  - flags是附加标志，通常设置为0

Winsock相关函数(10)

- sendto()函数：发送数据
  - int sendto(SOCKET s, const char FAR *buf, int len, int flags, const char FAR *to, int tolen)
  - to指定保存接收数据的缓冲区指针
  - tolen指定接收数据的字节数

Winsock相关函数(11)

- recv()函数： 接收数据
  - int recv(SOCKET s, char FAR *buf, int len, int flags)
  - s指定接收端使用的套接字
  - buf指定保存接收数据的缓冲区指针
  - len指定接收数据的字节数
  - flags是附加标志，通常设置为0

Winsock相关函数(12)

- recvfrom()函数：接收数据
  - int recvfrom(SOCKET s, const char FAR *buf, int len, int flags, const char FAR *from, int fromlen)
  - from指定保存接收数据的缓冲区指针
  - fromlen指定接收数据的字节数

Winsock相关函数(13)

- 字节序转换函数
  - htonl()：无符号长整型数从主机字节序转换为网络字节序
  - ntohl()：无符号长整型数从网络字节序转换为主机字节序
  - htons()：无符号短整型数从主机字节序转换为网络字节序
  - ntohs()：无符号短整型数从网络字节序转换为主机字节序

Winsock相关函数(14)

- 主机信息获取函数
  - int gethostname(char FAR *name, int namelen)：获取主机名
  - struct hostent FAR *gethostbyname (const char FAR *name)：获取主机地址

Winsock相关函数(15)

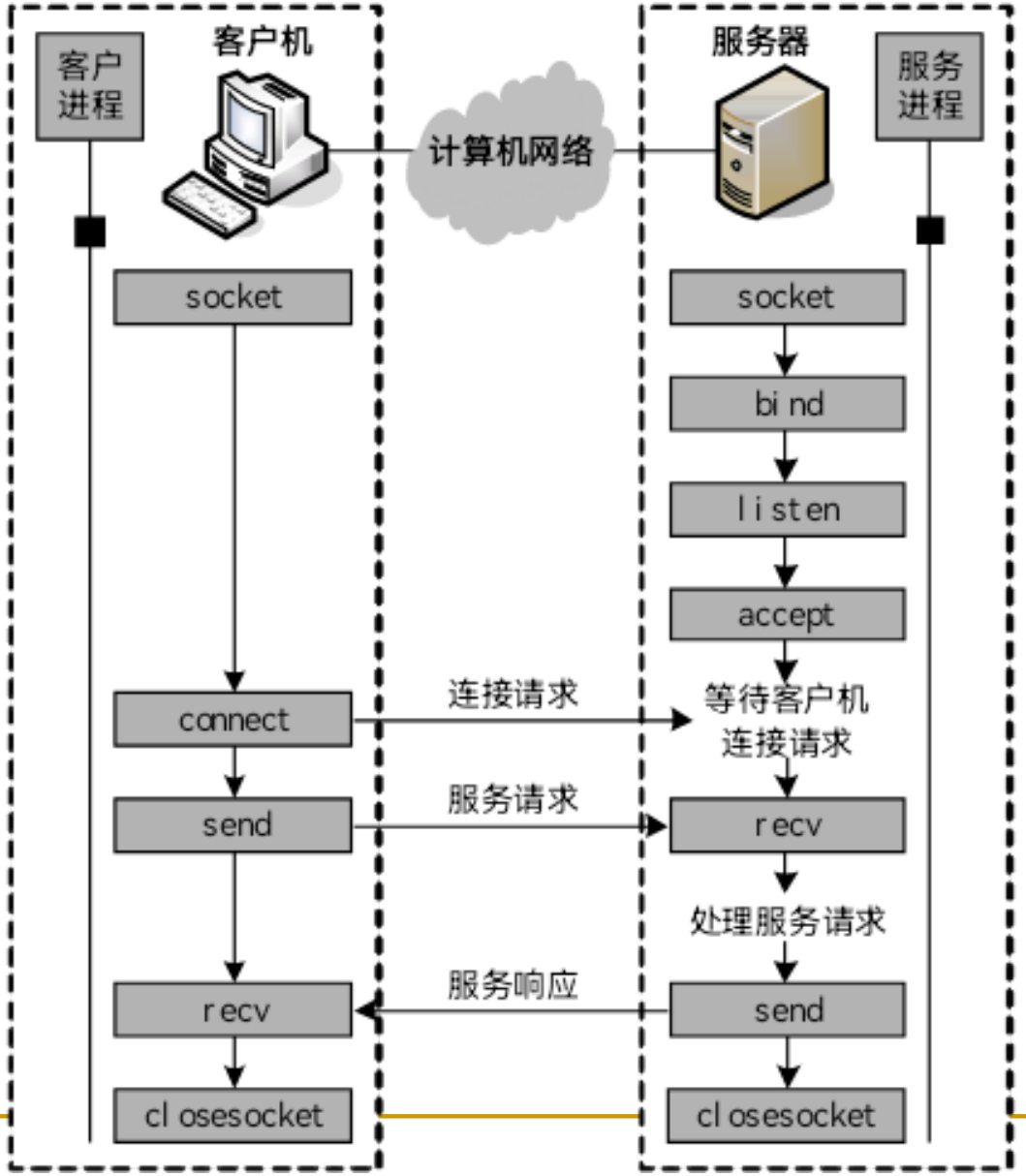
- 获得错误类型函数
  - 如果某个函数返回SOCKET_ERROR，说明该函数在处理过程中出现错误
  - 错误可能由不同原因引起，可以通过调用GetLastError()函数获得错误码
  - 错误码格式为100XX。例如，10055表示缓冲区不足而无法执行操作

Winsock相关函数(16)

- closesocket()函数：关闭套接字
  - int closesocket(SOCKET s)
- WSACleanup()函数：卸载Winsock库
  - int WSACleanup()

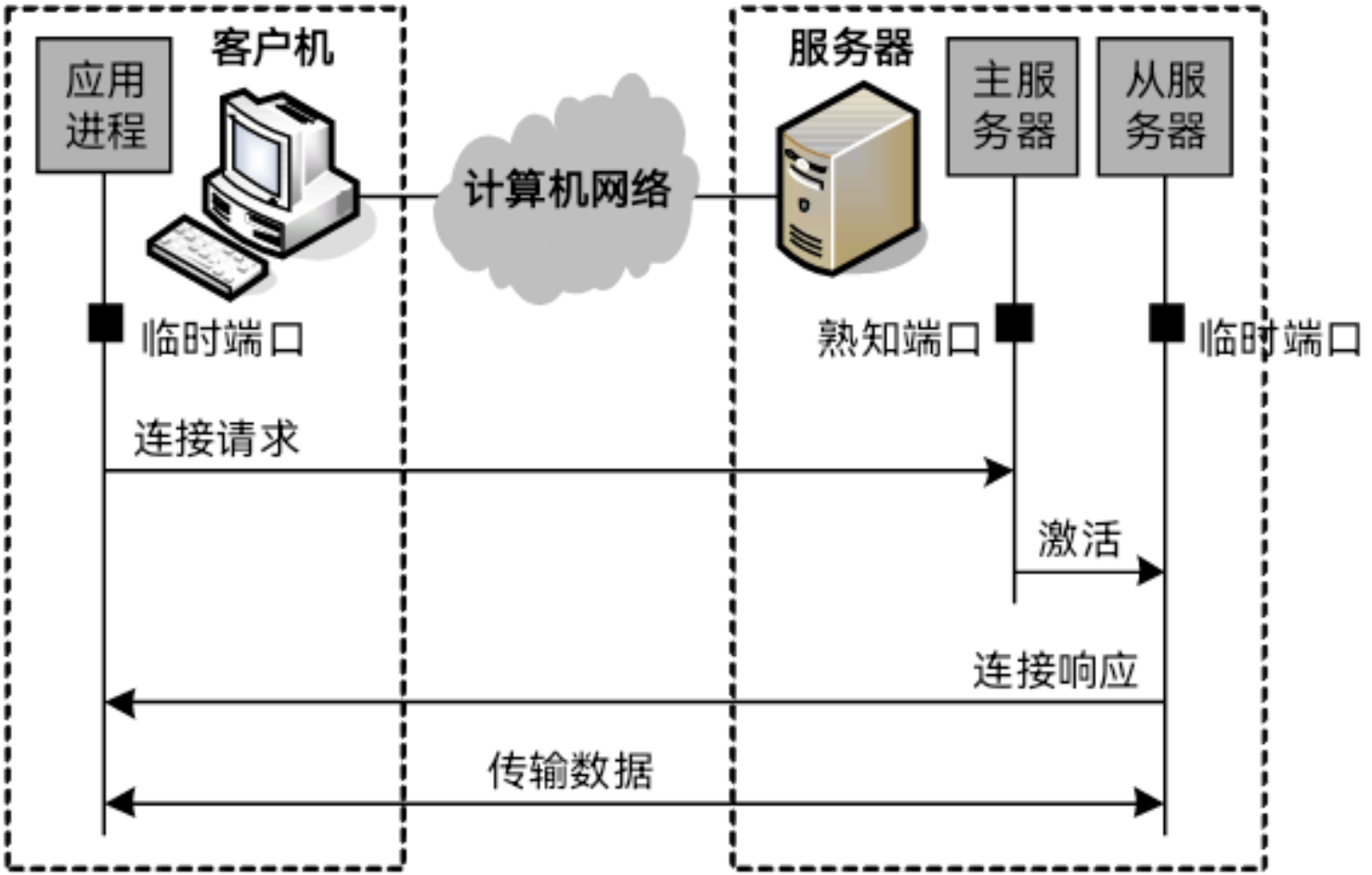
基于TCP的网络编程(1)

- 客户端
  - 请求服务
  - 临时端口
- 服务器端
  - 提供服务
  - 熟知端口



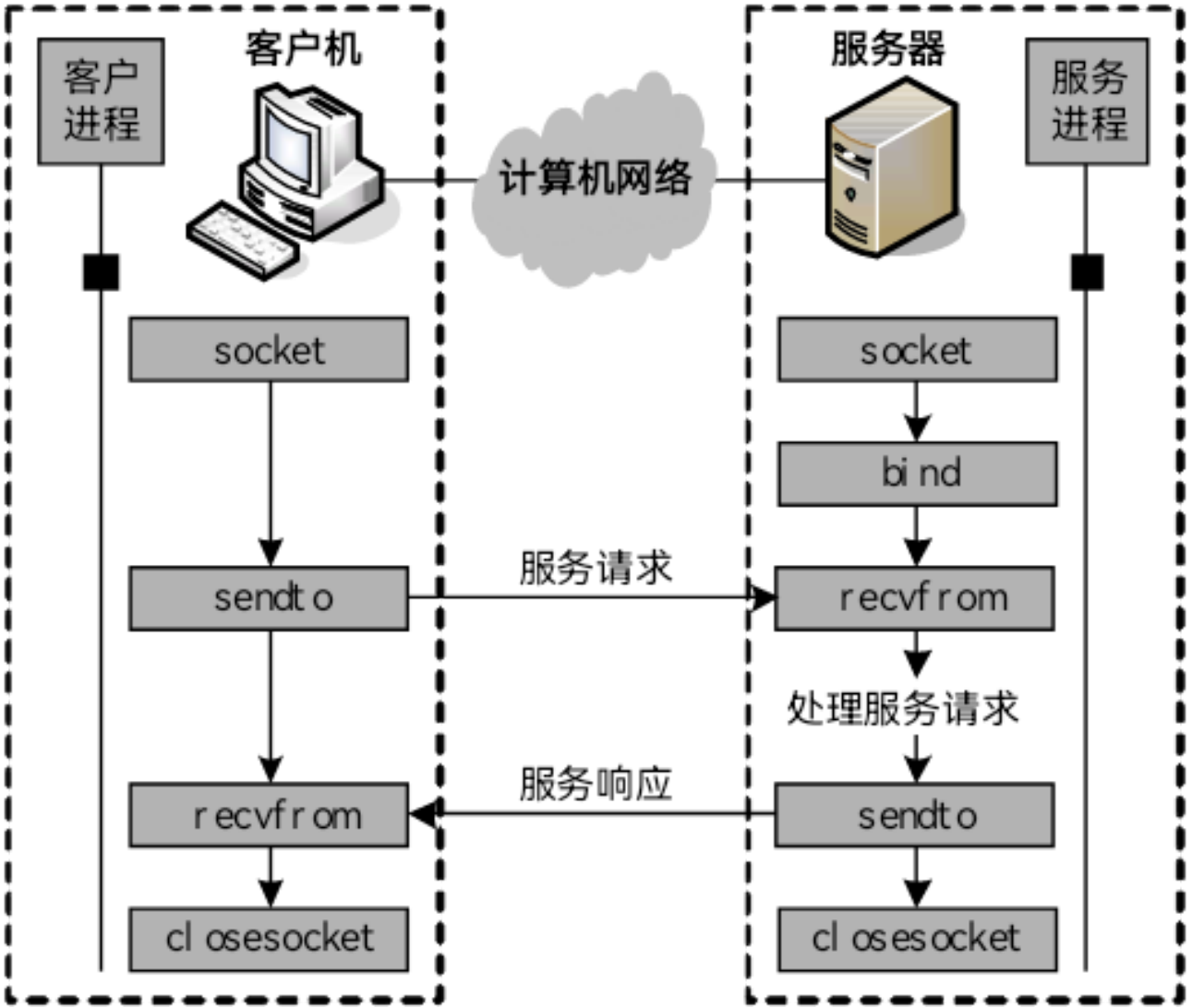
基于TCP的网络编程(2)

- 并发服务器模式



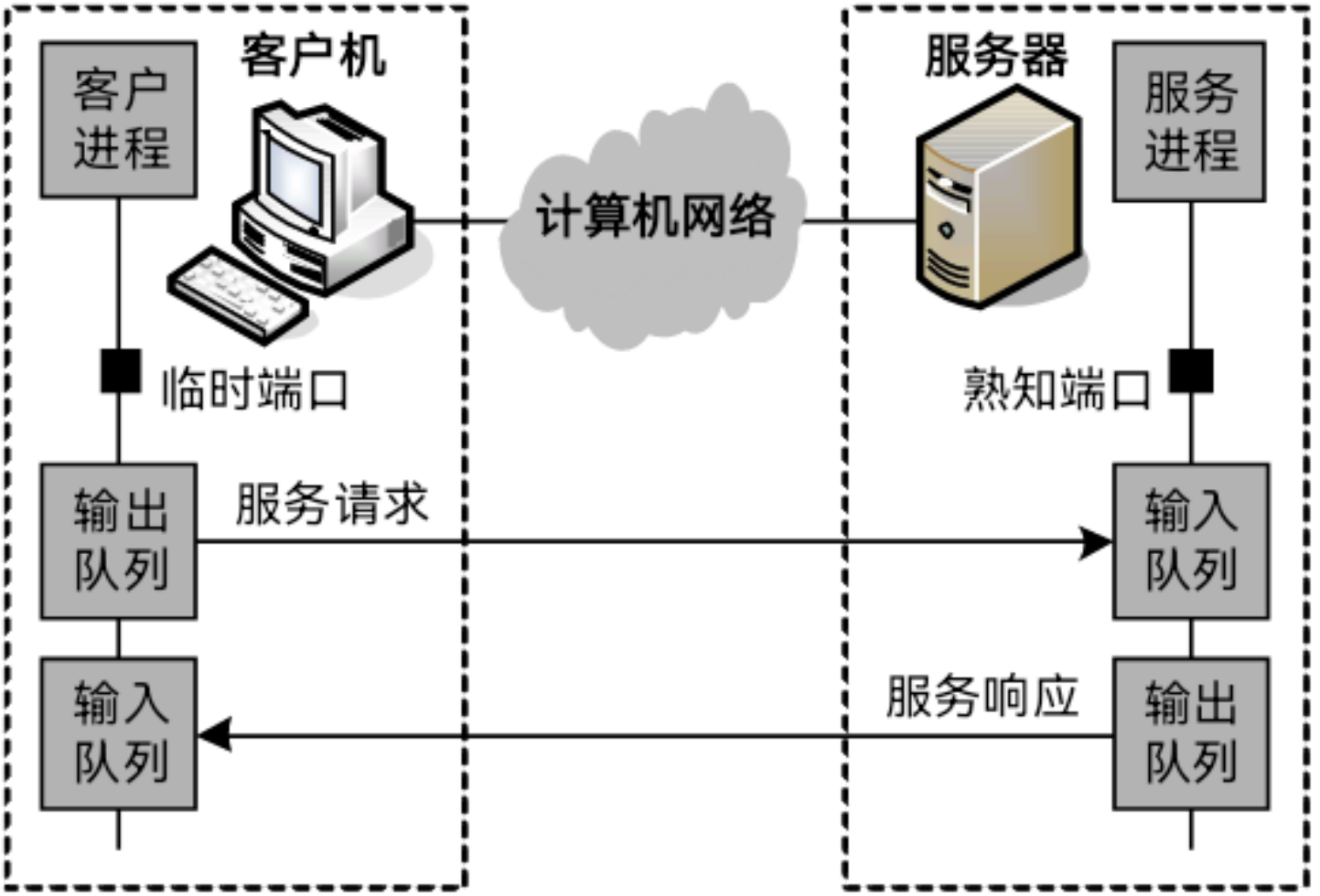
基于UDP的网络编程(1)

- 客户端
  - 请求服务
  - 临时端口
- 服务器端
  - 提供服务
  - 熟知端口



基于UDP的网络编程(2)

- 重复服务器模式



非阻塞模式(1)

- BSD Unix是命令行方式的系统
  - 套接字适合以阻塞模式工作
- Windows是消息驱动的系统
  - 套接字适合以非阻塞模式工作
  - 所有套接字函数都有非阻塞版本
  - CAsyncSocket对象默认工作在非阻塞模式

非阻塞模式(2)

- 在非阻塞模式下
  - 套接字函数被调用后会立即返回
  - 即使它执行的操作没有全部完成
- 当函数最终完成时执行的操作
  - Windows通过发送消息方式通知程序
  - 该模式适合Windows消息驱动体系

CAsyncSocket相关函数(1)

成员函数	说明
Create	创建套接字
AsynSelect	设置套接字通知事件
Accept	接受套接字上的连接
Bind	将本地地址与套接字绑定
Listen	监听进入的连接请求
IOCtl	控制套接字的方式
Close	关闭套接字

CAsyncSocket相关函数(2)

成员函数	说明
Receive	从套接字接收数据
ReceiveFrom	从指定的源地址接收数据
Send	向套接字发送数据
SendTo	向指定目的地址发送数据
Attach	将句柄附属于套接字对象
Detach	从套接字对象中分离句柄
GetLastError	获得最后一个错误操作码

CAsyncSocket相关函数(3)

成员函数	说明
GetPeerName	获得连接的对方套接字地址
GetSockName	获得套接字本地名称
OnAccept	通知侦听套接字可接受连接
OnClose	通知套接字对方已经关闭
OnConnect	通知套接字连接尝试已完成
OnReceive	通知套接字可接收数据
OnSend	通知套接字可发送数据

CAsyncSocket相关函数(4)

- 套接字创建过程
  - 客户端与服务器调用Create创建套接字
- Windows是消息驱动的系统
  - 客户端调用Connect连接服务器
  - 服务器调用Listen监听客户端连接请求
  - 服务器调用Accept与客户端建立连接
- 数据传输过程
  - 调用Send或SendTo发送数据，调用Receive或ReceiveFrom接收数据

CAsyncSocket相关函数(5)

- 调用WSAAsyncSelect执行异步I/O模式
- 处理感兴趣的网络事件
  - FD_CONNECT：连接请求的通知
  - FD_ACCEPT：接受连接请求的通知
  - FD_READ：读出数据的通知
  - FD_WRITE：写入数据的通知
  - FD_CLOSE：套接字关闭的通知

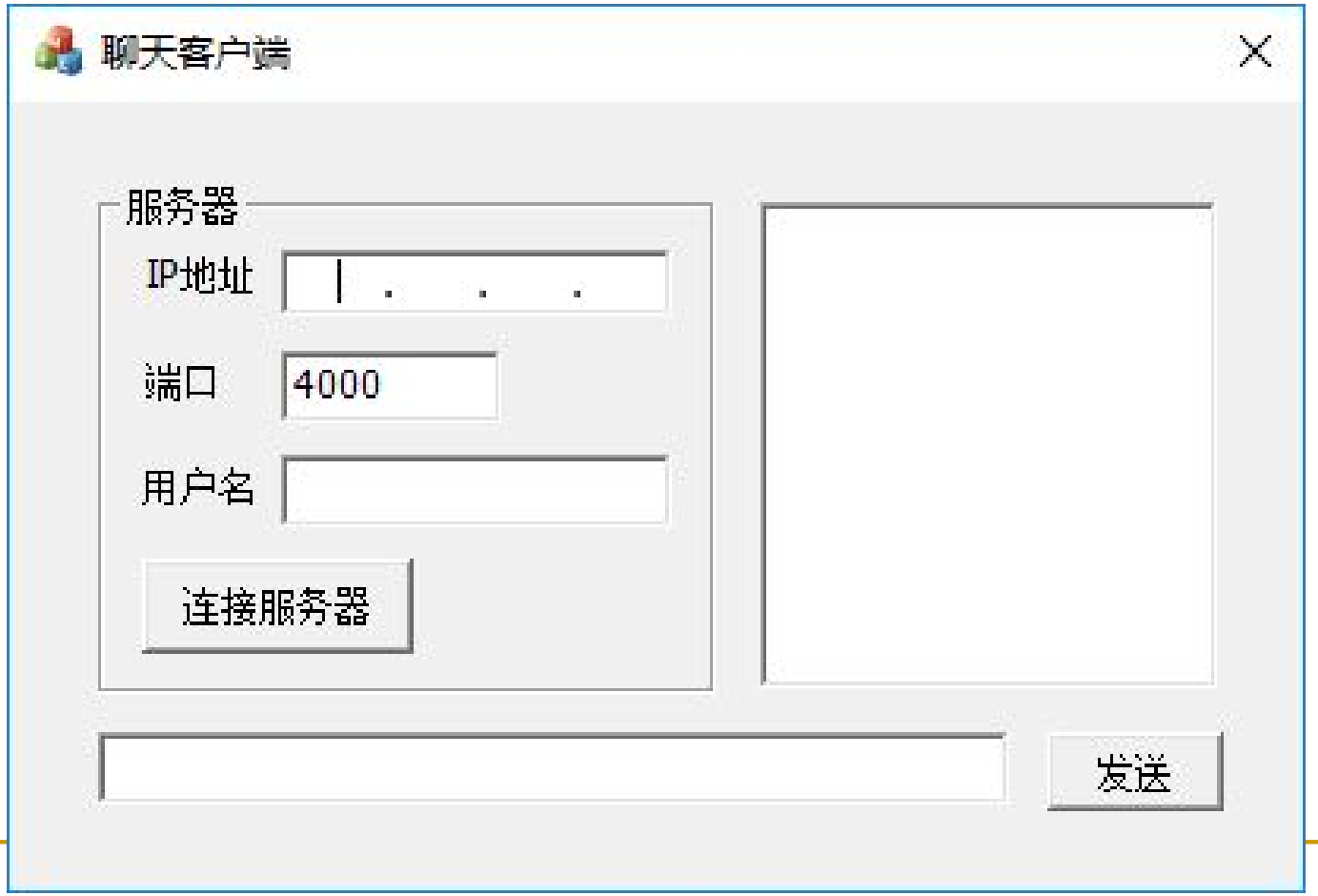
简单的聊天室程序

- 编写简单的聊天室程序
  - 使用流式套接字进行通信
  - 基于有连接的TCP协议
- 包括客户端和服务器端
  - 验证客户机/服务器模型
  - 服务器支持多个客户端连接

聊天室客户端实例(1)

例7-1

- 创建对话框程序ChatClient，选中Windows Sockets



聊天室客户端实例(2)

- 在CChatClientDlg类中

```
CClient Client;
CIPAddressCtrl m_ip;
CString m_name;
int m_port;
CString m_recv;
CString m_send;
```

- 在CChatClientDlg构造函数中

```
m_port=4000;
```

聊天室客户端实例(3)

- 在CChatClientDlg::OnConnectserver()中

```
char temp[25];
UpdateData(true);
strcat(temp,m_name);
BYTE a1,a2,a3,a4;
m_ip.GetAddress(a1,a2,a3,a4);
CString a;
a.Format("%d.%d.%d.%d",a1,a2,a3,a4);
Client.InitClient(m_hWnd,m_port,a);
send(Client.m_clientsocket,temp,strlen
(temp)+1,0);
```

聊天室客户端实例(4)

- 在CChatClientDlg::OnSendmsg()中

```
UpdateData(true);
CString a;
a=m_name;
a+=" 说：  \r\n  ";
a+=m_send;
send(Client.m_clientsocket,a,strlen(a)+1,0);
```

- 在CChatClientDlg::OnClose()中

```
closesocket(Client.m_clientsocket);
```

聊天室客户端实例(5)

- 在自定义消息OnClientMessage()中

```
CEdit* output=NULL;
char q[1024];
switch(lParam)  {
case FD_CONNECT:
  if(GetLastError()==0)
    MessageBox("连接服务器成功！ ");
  break;
case FD_READ:
  recv(Client.m_clientsocket,q,1024,0);
```

聊天室客户端实例(6)

- 在自定义消息OnClientMessage()中

```
 m_recv+=q;  m_recv+="\r\n";
  output=(CEdit *)GetDlgItem(IDC_RECV);
  output->SetWindowText(m_recv);
  output->LineScroll(output->GetLineCount());
  break;
case FD_CLOSE:
  break;
default:
  MessageBox("连接服务器失败！ " ); }
```

聊天室客户端实例(7)

- 在CClient类中

```
SOCKET m_clientsocket;
HWND m_hwnd;
UINT m_port;
sockaddr_in m_serveraddr;
```

- 在CClient::InitClient()中

```
WSADATA Data;
int status;
m_hwnd=hwnd;
m_port=port;
status=WSAStartup(MAKEWORD(2,0),&Data);
```

聊天室客户端实例(8)

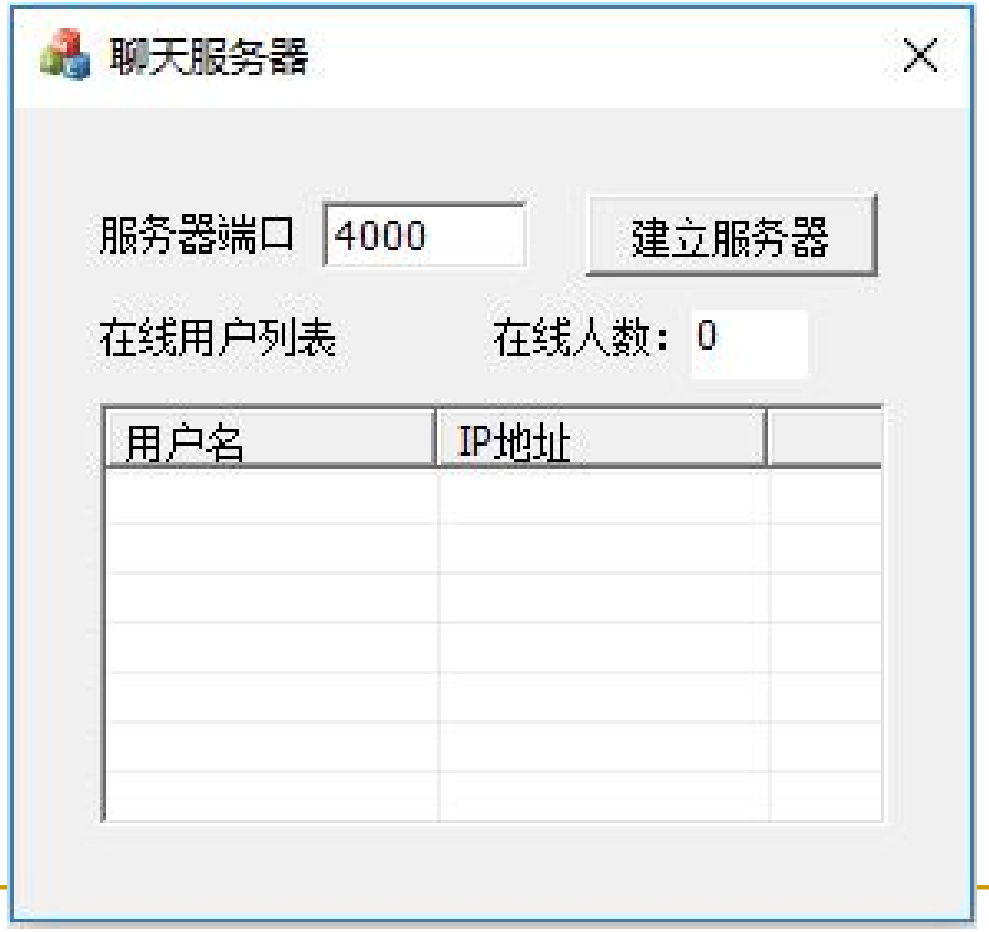
- 在CClient::InitClient()中

```
m_clientsocket=socket(AF_INET,SOCK_STREAM,0)
;
status=WSAAsyncSelect(m_clientsocket,m_hwnd,
CLI_MESSAGE,
FD_CONNECT|FD_READ|FD_CLOSE);
m_serveraddr.sin_port=htons(m_port);
m_serveraddr.sin_family=AF_INET;
m_serveraddr.sin_addr.S_un.S_addr=inet_addr (ip);
status=connect(m_clientsocket,(LPSOCKADDR)
&m_serveraddr,sizeof(m_serveraddr));
```

聊天室服务器实例(1)

例7-2

- 创建对话框程序ChatServer，选中Windows Sockets



聊天室服务器实例(2)

- 在CChatServerDlg类中

```
CServer server;
SOCKET m_socket[1024];
SOCKADDR_IN m_addr[1024];
int m_num;
int m_port;
CListCtrl m_user;
```

- 在CChatServerDlg构造函数中

```
m_port=4000;
```

聊天室服务器实例(3)

- 在CChatServerDlg::OnInitDialog()中

```
m_user.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_GRIDLINES);
m_user.InsertColumn(0,"用户名");
m_user.SetColumnWidth(0,100);
m_user.InsertColumn(1,"IP地址");
m_user.SetColumnWidth(1,100);
```

- 在CChatServerDlg::OnClickedSetserver()中

```
if(m_port>=1024&&m_port<=65535)
  if(true==server.SetServer(m_hWnd,m_port))
    MessageBox("服务器建立成功！ ");
```

聊天室服务器实例(4)

- 在自定义消息OnServerMessage()中

```
char q[1024];  CString temp;
switch(lParam)  {
case FD_ACCEPT:
  m_socket[m_num]=accept(server.m_server
socket,(SOCKADDR*)&m_addr[m_num],&len);
  m_num++;  UpdateData(false);  break;
case FD_READ:
  for(i=0;i<m_num;i++)
    if(wParam==m_socket[i])
      recv(m_socket[i],q,1024,0);  break;
```

聊天室服务器实例(5)

- 在自定义消息OnServerMessage()中

```
if(q[0]==1)
{ for(j=0;j<strlen(q);j++)
    q[j]=q[j+1];
    m_user.InsertItem(m_num-1,q);
    temp=inet_ntoa(m_addr[i].sin_addr);
    m_user.SetItemText(m_num-1,1,temp);  }
else
{ for(i=0;i<m_num;i++)
    send(m_socket[i],q,strlen(q)+1,0);  }
break;
```

聊天室服务器实例(6)

- 在自定义消息OnServerMessage()中

```
case FD_CLOSE:
  for(i=0;i<m_num;i++)
  { if(wParam==m_socket[i])
    { temp=inet_ntoa(m_addr[i].sin_addr);
      for(j=0;j<m_user.GetItemCount();j++)
      { if(m_user.GetItemText(j,1)==temp)
        { m_user.DeleteItem(j); break; }}
      for(j=i;j<m_num-1;j++)
        m_socket[j]=m_socket[j+1]; break; }}
  m_num--;  UpdateData(false);  break; }
```

聊天室服务器实例(7)

- 在CServer类中

```
SOCKET m_serversocket;
HWND m_hwnd;
UINT m_port;
sockaddr_in m_serveraddr;
```

- 在CServer::SetServer()中

```
WSADATA Data;
int status;
m_hwnd=hwnd;
m_port=port;
status=WSAStartup(MAKEWORD(2,0),&Data);
```

聊天室服务器实例(8)

- 在CServer::SetServer()中

```
m_serversocket=socket(AF_INET,
SOCK_STREAM,0);
status=WSAAsyncSelect(m_serversocket,m_hwnd
, SER_MESSAGE,
FD_ACCEPT|FD_READ|FD_CLOSE);
m_serveraddr.sin_port=htons(m_port);
m_serveraddr.sin_family=AF_INET;
m_serveraddr.sin_addr.s_addr=htonl(INADDR_
ANY);
status=bind(m_serversocket,(LPSOCKADDR)&m_
serveraddr,sizeof(m_serveraddr));
status=listen(m_serversocket,5);
```

第7次作业

- 设计一个简单的聊天室程序，要求至少具有以下功能：
  - 实现聊天室的服务器程序，能够识别发送方的IP地址，并转发接收到的每条消息
  - 实现聊天室的客户端程序，能够发送与接收每条消息
  - 使用数据报套接字进行通信

谢谢大家