```
            {
                pSock = (CServiceSocket *)m_connectionList.GetNext
                (posname);
                if(pSock->Name == pMsg->From  )
                {
                    CloseSocket(pSock);
                }
                else
                {
                    SendMsg(pSock, pMsg);
                }
            }
            break;

        case 3:

for( posname=m_connectionList.GetHeadPosition();posname;)
            {
                pSock = (CServiceSocket *)m_connectionList.GetNext
                (posname);
                if(pSock->Name == pMsg->To )
                {
                    SendMsg(pSock, pMsg);
                    break;
                }
            }
            break;

        case 4:
            pMsg->To = "";

for( posname=m_connectionList.GetHeadPosition();posname;)
            {
                pSock = (CServiceSocket *)m_connectionList.GetNext
```

```
                (posname);
                if(pSock->Name != pMsg->From )
                {
                    SendMsg(pSock, pMsg);
                }
            }
            break;
        case 5:
            //pSocket->Name = pMsg->From ;
            pMsg->ShortMessage = "";

for( posname=m_connectionList.GetHeadPosition();posname;)
            {
                pSock = (CServiceSocket *)m_connectionList.GetNext
                (posname);
                pMsg->ShortMessage = pMsg->ShortMessage +
                pSock->Name;
                pMsg->ShortMessage = pMsg->ShortMessage + "##";
            }
            SendMsg(pSocket, pMsg);
            break;

        case 6:

for( posname=m_connectionList.GetHeadPosition();posname;)
            {
                pSock = (CServiceSocket *)m_connectionList.GetNext
                (posname);
                if (pSock->Name == pMsg->To)
                {
//pSock->GetPeerName(m_strAddrTemp,(unsigned long &) &m_nPortTemp);
                    pMsg->ShortMessage = m_strAddrTemp;
                }
            }
```

```
            SendMsg(pSocket, pMsg);
            break;
        }

        DisplayMsg(pMsg->Type ,pMsg->From ,pMsg->To ,
        pMsg->ShortMessage );

    }while (!pSocket->m_pArchiveIn->IsBufferEmpty());

    }
    catch(...)
    {
        CString m_strTemp01;
        m_strTemp01 = "";

for( posname=m_connectionList.GetHeadPosition();posname;)
        {
            pSock = (CServiceSocket *)m_connectionList.GetNext
            (posname);
            m_strTemp01 = m_strTemp01 + pSock->Name;
            m_strTemp01 = m_strTemp01 + "##";
        }


for( posname=m_connectionList.GetHeadPosition();posname;)
        {
            pSock = (CServiceSocket *)m_connectionList.GetNext
            (posname);
            if (pSock->Name == pMsg->To)
            {
                //pSock->GetPeerName(m_strAddrTemp,(unsigned
                //long &) &m_nPortTemp);
                CloseSocket(pSock);
```

```
                }
                else
                {
                    pMsg->ShortMessage = "Logout";
                    pMsg->Type = 2;
                    pMsg->From = pSock->Name ;
                    pMsg->To = pSock->Name;
                    SendMsg(pSock, pMsg);

                    pMsg->ShortMessage = m_strTemp01;
                    pMsg->Type = 5;
                    pMsg->From = "Server";
                    pMsg->To = pSock->Name;
                    SendMsg(pSock, pMsg);
                }
            }

        }

    }

    void CServerDoc::SendMsg(CServiceSocket *pSocket, CMessage
*pMessage)
    {
        pSocket->SendMsg(pMessage);
    }

    CMessage* CServerDoc::ReadMsg(CServiceSocket *pSocket)
    {
        pSocket->ReceiveMsg(m_pShortMessage);

        return m_pShortMessage;
```

```
}

void CServerDoc::DisplayString(LPCTSTR lpszMessage)
{
    ((CServerView*)m_viewList.GetHead())->
    DisplayString(lpszMessage);

}

void CServerDoc::DisplayStringWithCRLF(LPCTSTR lpszMessage)
{
    ((CServerView*)m_viewList.GetHead())->
    DisplayStringWithCRLF(lpszMessage);
}

void CServerDoc::CloseSocket(CServiceSocket *pSocket)
{
    pSocket->Close();

    POSITION pos,temp;
    for(pos = m_connectionList.GetHeadPosition(); pos != NULL;)
    {
        temp = pos;
       CServiceSocket*pSock = (CServiceSocket *)m_connectionList.
       GetNext(pos);
        if (pSock == pSocket){
            m_connectionList.RemoveAt(temp);
            break;
        }
    }

    delete pSocket;
```

```
        pSocket = NULL;
    }

    void CServerDoc::DisplayMsg(int Type, CString From, CString To,
CString ShortMessage)
    {
        DisplayString(From);
        DisplayString(" say towards ");
        DisplayString(To);
        DisplayString(" : ");
        DisplayString(ShortMessage);
        DisplayString("\r\n");

    }

    void CServerDoc::DeleteContents()
    {
        // TODO: Add your specialized code here and/or call the base class
        delete m_pSocket;
        m_pSocket = NULL;

        CString temp;
        temp="Server has been shutdown!";

        CMessage* pMsg = new CMessage;
        while(!m_connectionList.IsEmpty())
        {
            CServiceSocket* pSocket = (CServiceSocket *)
            m_connectionList.RemoveHead();
            if(pSocket==NULL )
                continue;
            if (pMsg == NULL)
                continue;
            pMsg->From = pSocket->Name ;
```

```
            pMsg->ShortMessage  = _TEXT("Server has been shutdown!");
            pMsg->To = _TEXT("All");
            pMsg->Type = 9;

            SendMsg(pSocket, pMsg);

            if(!pSocket->IsAborted())
            {
                pSocket->ShutDown();
                BYTE Buffer[50];
                while (pSocket->Receive(Buffer,50) > 0);
                delete pSocket;
            }
        }
        delete pMsg;

        if (!m_viewList.IsEmpty())
            ((CEditView*)m_viewList.GetHead())->
            SetWindowText(_T(""));

        CDocument::DeleteContents();
    }


    void CServerDoc::OnConfig()
    {
        // TODO: Add your command handler code here
        CParamDlg m_dlgParam;
        m_dlgParam.m_nPort = m_nPort;
        if (m_dlgParam.DoModal() == IDOK)
        {
            m_nPort = m_dlgParam.m_nPort ;

            CString temp;
```

```
temp="Server has been shutdown.";

CMessage* pMsg = new CMessage;
while(!m_connectionList.IsEmpty())
{
    CServiceSocket* pSocket = (CServiceSocket *)
    m_connectionList.RemoveHead();
    if(pSocket==NULL )
        continue;
    if (pMsg == NULL)
        continue;
    pMsg->From = pSocket->Name ;
    pMsg->ShortMessage  = _TEXT("Server has been
    shutdown!");
    pMsg->To = _TEXT("All");
    pMsg->Type = 9;

    SendMsg(pSocket, pMsg);

    if(!pSocket->IsAborted())
    {
        pSocket->ShutDown();
        BYTE Buffer[50];
        while (pSocket->Receive(Buffer,50) > 0);
        delete pSocket;
    }
}
delete pMsg;

delete m_pSocket;

m_pSocket = new CAcceptSocket(this);
if (m_pSocket->Create(m_nPort)){
    if (m_pSocket->Listen())
```

```
                return ;
        }

        }

}
```

6. CServerView 的代码（略）
7. CMainFrame 的代码（略）
8. CParamDlg 的代码(略)

参数设定对话框类，以设置服务器的响应端口号。

15.3.2 客户端代码

客户端应用名为 Client，它是一个对话框程序，生成对话框的步骤在前面章节已详细介绍过，这里就不一一描述了。

另外在系统中还定义了三个类：

（1） CMessage 类：它是从 CObject 类中派生而来，目的是为了使用 CObject 类的序列化函数，以便将数据发送到与服务套接字（CServiceSocket ）相关联的文档中(注意中间还有 CSocketFile 类在起作用)。另外还有列在上面所定义的消息结构的成员变量。

（2）CServiceSocket 类： 是一个服务于用户请求的套接字类，该套接字类处理所有发出和接收数据，在类中定义了三个成员变量。

- CSocketFile* m_pFile
- CArchive* m_pArchiveIn
- CArchive* m_pArchiveOut

第一个变量是与该套接字类相关联的 CSocketFile 类，第二个变量是与 CSocketFile 类相关联的接收数据文档，第三个变量是与 CSocketFile 类相关联的发送数据文档。 该套接字对象与上述变量结合实现发送和接收数据的功能。

（3）CUserList 类：从 CListCtrl 派生而来，主要显示在线用户列表。双击指定的在线人员，可弹出对话框，给该人员发送消息。

下面介绍其具体代码。

1. CClientApp 的代码

```
    // Client.h : main header file for the CLIENT application
    //

    #if !defined(AFX_CLIENT_H__BA90C631_530F_45C6_96E9_C96E75E7EAB1_
_INCLUDED_)
    #define
AFX_CLIENT_H__BA90C631_530F_45C6_96E9_C96E75E7EAB1__ INCLUDED_

    #if _MSC_VER > 1000
    #pragma once
    #endif // _MSC_VER > 1000

    #ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
    #endif

    #include "resource.h"       // main symbols

    /////////////////////////////////////////////////////////////////////
    // CClientApp:
    // See Client.cpp for the implementation of this class
    //

    class CClientApp : public CWinApp
    {
    public:
        CClientApp();

    // Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CClientApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL
```