第 5 章　基于 CSocket 类的聊天程序

MFC CSocket 类是从 CAsyncSocket 类派生出来的，是对 Winsock API 的高级封装。CSocket 类继承了 CAsyncSocket 类的许多成员函数，这些函数封装了 Windows 套接字应用程序编程接口。在 CSocket 和 CAsyncSocket 两个套接字类中，这些成员函数的用法是一致的。与 CAsyncSocket 类相比，CSocket 类提供了一个更高级别的 Winsock 编程接口，简化了编程工作，表现在以下 3 个方面。

(1) CSocket 类结合 CSocketFile 类和 CArchive 类来处理数据的发送和接收，就像使用 MFC 的序列化协议一样。

(2) CSocket 类管理了通信的许多方面，如字节顺序问题和字符串转换问题。而使用原始 API 或者 CAsyncSocket 类时，都必须由用户自己来做，这就使 CSocket 类比 CAsyncSocket 类更容易使用。

(3) 最重要的是，CSocket 类为 Windows 消息的后台处理提供了阻塞的工作模式，一些成员函数，如 Receive()、Send()、ReceiveFrom()、SendTO()和 Accept()等，在不能立即发送或接收数据时，不会立即返回一个 WSAEWOULDBLOCK 错误，它们会等待，直到操作结束。这对于使用 CArchive 类来进行同步数据传输是必需的。

5.1　CSocket 类

5.1.1　基本编程模型

CSocket 类、CSocketFile 类和 CArchive 类相结合使用套接字，就像使用 MFC 的序列化协议一样，简单方便。使用 CSocket 类来编程，可以充分利用应用程序框架的优势，使用可视化编程语言的控件方便地构造图形化的用户界面。同时，可以充分利用应用程序框架的消息驱动的处理机制，方便地对各种不同的网络事件做出响应，从而简化了程序的编写。

在服务器和客户端中，针对流式套接字使用 CSocket 类编程的一般步骤如表 5-1 所示。

表 5-1　使用 CSocket 类编程的一般步骤

序号	服　务　器	客　户　端
1	CSocket sockServ; // 创建空的服务器监听套接字对象	CSocket sockClient; // 创建空的客户端的套接字对象
2	sockServ. Create(nPort); //用众所周知的端口，创建监听套接字对象的底层套接字句柄	sockClient. Create(); //创建套接字对象的底层套接字
3	sockServ. Listen(); //启动对客户端连接请求的监听	

续表

序号	服　务　器	客　户　端
4		sockClient. Connect(strAddr, nPort); //请求连接到服务器

5	CSocket sockRecv;
	// 创建空的服务器连接套接字对象
	sockServ. Accept(sockRecv);
	//接受客户的连接请求,并将其他任务转交给连接套接字对象	

6	CSockFile * file;
	file = new CSockFile(&sockRecv);
	//创建文件对象并关联到连接套接字对象
		CSockFile * file;
		file = new CSockFile(&sockClent);
		//创建文件对象,并关联到套接字对象

7	CArchive * arIn, arOut;
	//归档对象必须关联到文件对象
	arIn = CArchive(&file, CArchive::load);
	//创建用于输入的归档对象
	arOut = CArchive(&file, CArchive::store);
	//创建用于输出的归档对象
		CArchive * arIn, arOut;
		//归档对象必须关联到文件对象
		arIn = CArchive(&file, CArchive::load);
		//创建用于输入的归档对象
		arOut = CArchive(&file, CArchive::store);
		//创建用于输出的归档对象

8	arIn >> dwValue;
	//数据输入
	arOut << dwValue;
	//数据输出,输入或输出可以反复进行
		arIn >> dwValue;
		//数据输入
		arOut << dwValue;
		//数据输出

9	sockRecv. Close();
	sockServ. Close();
	//传输完毕,关闭套接字对象
		sockClient. Close();
		//传输完毕,关闭套接字对象

还要强调以下几点。

(1) 服务器程序在创建专用于监听的套接字对象时,要指定为这种服务分配的众所周知的保留端口号,这样才能使客户程序正确地连接到服务器。

(2) 服务器接收到连接请求后,必须创建一个专门用于连接的套接字对象,并将以后的连接和数据传输工作移交给它。

(3) 通常,一个服务器应用程序应能处理多个客户端的连接请求,对于每个客户端的套接字对象,服务器应用程序都应该创建一个与它们进行连接和数据交换的相应的套接字对象。因此,服务器应该采用动态的方式来创建这些连接套接字对象。

(4) 使用 CSocket 类的最大优点在于,应用程序可以在连接的两端通过 CArchive 对象来进行数据传输。通过 CArchive 对象来进行数据传输时,应用程序不需要直接调用 CSocket 类的数据传输成员函数,而是利用由 CArchive 对象、CSocketFile 对象和 CSocket 对象级联而形成的数据传输管道,如图 5-1 所示。

在发送端,将需要传输的数据插入用于发送的 CArchive 对象,CArchive 对象会将数据传输到 CSocketFile 对象中,再交给 CSocket 对象,由套接字来发送数据;在接收端,按照相反的顺序传递数据,最终应用程序从用于接收数据的 CArchive 对象中获取传输过来的数据。向 CArchive 对象插入数据的操作符是“<<”,从 CArchive 对象获取数据的操作符是“>>”。

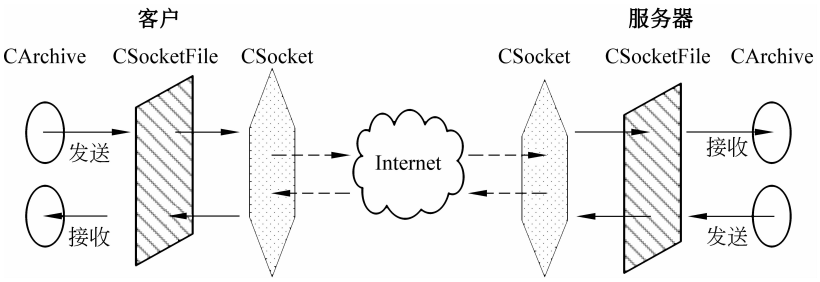

图 5-1　CSocket 类、CArchive 类和 CSocketFile 类在传输数据时的作用

(5) 一个特定的 CArchive 对象只能进行单方向的数据传递，要么用于输入，要么用于输出。因此，如果应用程序既要发送数据又要接收数据，那么就必须在每一端创建两个独立的 CArchive 对象：一个用 CArchive::load 属性创建，用来接收数据；另一个用 CArchive::store 属性创建，用来发送数据。这两个 CArchive 对象可以共享同一个 CSocketFile 对象和 CSocket 套接字对象。

(6) 当把需要发送的数据都写入 CArchive 对象以后，还必须调用 CArchive::Flush() 函数刷新 CArchive 对象的缓冲区，这时数据才真正从网络上发送出去，否则接收端就收不到数据。

5.1.2　创建 CSocket 类对象

创建 CSocket 类对象可分为两个步骤。

(1) 调用 CSocket 类的构造函数，创建一个空的 CSocket 类对象。

(2) 调用此 CSocket 类对象的 Create() 成员函数，创建对象的底层套接字。

函数原型如下：

```
BOOL Create(UINT nSocketPort=0, int nSocketType=SOCK_STREAM,
LPCTSTR lpszSocketAddress=NULL);
```

如果打算使用 CArchive 对象和套接字一起进行数据传输工作，则必须使用流式套接字。

5.1.3　连接的建立

CSocket 类使用基类 CAsyncSocket 的同名成员函数 Connect()、Listen() 和 Accept() 来建立服务器和客户端套接字之间的连接，使用方法相同，不同的是 CSocket 类的 Connect() 函数和 Accept() 函数支持阻塞调用。例如，在调用 Connect() 函数时会发生阻塞，直到成功地建立了连接或者有错误发生才返回。在多线程的应用程序中，一个线程发生阻塞，其他线程仍能处理 Windows 事件。

CSocket 对象从不调用 OnConnect() 事件处理函数。

5.1.4　数据的收发

创建 CSocket 类对象后，对于数据报套接字，直接使用 CSocket 类的成员函数 SendTo()

和 ReceiveFrom()来发送和接收数据。对于流式套接字,首先在服务器和客户端之间建立连接,然后使用 CSocket 类的成员函数 Send()和 Receive()来发送和接收数据,它们的调用方式与 CAsyncSocket 类相同,不同的是 CSocket 类的成员函数工作在阻塞的模式。例如,一旦调用了 Send()函数,在所有数据发送之前,程序或线程将处于阻塞的状态。一般将 CSocket 类与 CArchive 类和 CSocketFile 类相结合来发送和接收数据,这将使编程更为简单。

CSocket 对象从不调用 OnSend()事件处理函数。

5.1.5 关闭套接字和清除相关对象

使用完 CSocket 对象以后,应用程序应当调用它的 Close()成员函数来释放套接字占用的系统资源,也可以调用它的 ShutDown()成员函数来禁止套接字的读写操作。而对于相应的 CArchive 对象、CSocketFile 对象和 CSocket 对象,可以将它们销毁,也可以不做处理,因为当应用程序终止时,会自动调用这些对象的析构函数,从而释放这些对象占用的资源。

5.2 基于 CSocket 类的聊天程序实例

5.2.1 程序的功能

基于 CSocket 类的聊天程序实例采用客户-服务器模式,服务器可以同时与多个客户端建立连接,为多个客户端提供服务。服务器接收客户端发来的信息,然后将它转发给聊天室的其他客户端,从而实现多个客户端之间的信息交换。服务器动态地统计进入聊天室的客户端数目并显示出来;及时显示新的客户端进入聊天室和客户端退出聊天室的信息,也转发给其他的客户端。进入服务器程序后,用户应首先输入监听端口号,单击“启动”按钮启动监听,等待客户端的连接请求,当客户端的连接请求到来时,服务器接收它,然后进入与客户机的会话期。服务器程序动态地为新的客户端创建相应的套接字对象,并采用链表来管理客户端的套接字对象,从而实现一个服务器为多个客户端服务的目标。

用户可以同时启动多个客户程序。进入客户程序后,用户首先输入要连接的服务器名称、服务器的监听端口和客户机的名称,单击“连接”按钮,就能与服务器建立连接。然后输入信息,单击“发送”按钮向服务器发送聊天信息。在客户程序的列表框中,能实时显示聊天室的所有客户端发送的信息,以及客户端进出聊天室的信息。

基于 CSocket 类的聊天程序实例的技术要点如下:

(1) 如何从 CSocket 类派生出自己所需的 Winsock 类;

(2) 如何利用 CSocketFile 类、CArchive 类和 CSocket 类的合作来实现网络进程之间的数据传输;

(3) 如何用链表管理多个动态客户端的套接字,实现服务器和所有参与聊天的客户端所显示信息的同步更新。

聊天室服务器程序和客户程序的用户界面分别如图 5-2 和图 5-3 所示。

5.2.2 创建服务器程序

利用可视化语言的集成开发环境创建服务器应用程序的步骤如下。

图 5-2　聊天室服务器程序的用户界面

图 5-3　聊天室客户程序的用户界面

1. 利用 MFC AppWizard 创建服务器应用程序框架

工程名为 ChatRServer，应用程序类型选择 Dialog based，语言支持选择中文，选择 Windows Sockets 支持，其他为系统的默认值。所创建的程序将自动创建两个类：一个是应用程序类 CChatRServerApp，对应的文件是 ChatRServer. h 和 ChatRServer. cpp；另一个是对话框类 CChatRServerDlg，对应的文件是 ChatRServerDlg. h 和 ChatRServerDlg. cpp。

2. 为对话框添加控件对象

设置完成的 ChatRServer 程序主对话框如图 5-2 所示，然后按照表 5-2 修改对话框中各个控件的属性。

表 5-2　ChatRServer 程序主对话框中的控件属性

控 件 类 型	控 件 ID	Caption
静态文本 1(Static Text 1)	IDC_STATIC_SERVERNAME	名称
静态文本 2(Static Text 2)	IDC_STATIC_SERVERPORT	端口

续表

控 件 类 型	控 件 ID	Caption
静态文本 3(Static Text 3)	IDC_STATIC_NUM	当前在线：0 人
组合框 1(Group Box 1)	IDC_STATIC_MSG	聊天信息
组合框 2(Group Box 2)	IDC_STATIC_SERVER	服务器
编辑框 1(Edit Box 1)	IDC_EDIT_SERVERNAME	
编辑框 2(Edit Box 2)	IDC_EDIT_SERVERPORT	
命令按钮 1(Button 1)	IDC_BUTTON_START	启动
命令按钮 2(Button 2)	IDC_BUTTON_STOP	终止
命令按钮 3(Button 3)	IDC_BUTTON_EXIT	退出
列表框(List Box)	IDC_LIST_MSG	

3. 为对话框中的控件对象定义相应的成员变量

用类向导为对话框中的控件对象定义相应的成员变量，按照表 5-3 输入即可。

表 5-3　服务器程序对话框中控件对象对应的成员变量

Control IDs （控件 ID）	Member variable name （变量名称）	Category （变量类别）	Variable type （变量类型）
IDC_EDIT_SERVERNAME	m_ServerName	Value	Cstring
IDC_EDIT_SERVERPORT	m_ServerPort	Value	int
IDC_STATIC_NUM	m_Num	Control	CStatic
IDC_LIST_MSG	m_Msg	Control	CListBox

4. 创建从 CSocket 类继承的派生类

从 CSocket 类派生两个套接字类：一个类名为 CChatLSocket，专用于监听客户端的连接请求，为它添加 OnAccept 事件处理函数；另一个类名为 CChatCSocket，专用于与客户端建立连接并交换数据，为它添加 OnReceive 事件处理函数。这两个类都要添加一个指向对话框类的指针变量：

```
CChatRServerDlg * m_pDlg;
```

为 CChatCSocket 添加以下成员变量：

```
CSocketFile * m_pFile;          // 定义 CSocketFile 对象指针变量
CArchive * m_pArchiveIn;        // 定义输入型 CArchive 对象指针变量
CArchive * m_pArchiveOut;       // 定义输出型 CArchive 对象指针变量
```

为 CChatCSocket 添加以下成员函数：

```
void Initialize();                          // 初始化
void SendMsg(CMsg * pMsg);                  // 发送消息
void ReceiveMsg(CMsg * pMsg);               // 接收消息
```

5. 为对话框类添加控件对象事件的响应函数

按照表 5-4,用类向导为服务器程序对话框中的控件对象添加事件响应函数,主要是对于“启动”按钮和“终止”按钮的单击事件的处理函数。

表 5-4　服务器程序控件对象对应的事件响应函数

控 件 类 型	Object IDs (对象标识)	Messages (消息)	Member functions (成员函数)
命令按钮	IDC_BUTTON_START	BN_CLICKED	OnButtonStart
命令按钮	IDC_BUTTON_STOP	BN_CLICKED	OnButtonStop
命令按钮	IDC_BUTTON_EXIT	BN_CLICKED	OnButtonExit

6. 为 CChatRServerDlg 对话框类添加其他成员变量和成员函数

成员变量:

```
CChatLSocket * m_sListenSocket;         // 定义侦听套接字指针变量
CPtrList m_cList;                       // 定义连接列表变量
```

成员函数:

```
void OnAccept();                            // 接受连接请求
void OnReceive(CCSocket * pSocket);         // 获取客户端的发送消息
void ForwardMsg(CChatCSocket * pSocket, CMsg * pMsg);
                                            //向聊天室的所有客户端转发消息
```

7. 创建专用于数据传输序列化处理的类 CMsg

为了利用 CSocket 类及其派生类,可以和 CSocketFile 对象、CArchive 对象合作进行数据发送和接收。这里构造一个专用于消息传输的类,该类必须从 CObject 类派生。

选中“插入”|“新建类”菜单选项,弹出 New Class 对话框,在 Class type 下拉列表框中选择 Generic Class,在 Name 文本框中输入类名 CMsg,在基类的 Derived From 文本框中输入 CObject,单击 OK 按钮即可。

为 CMsg 类添加以下成员变量和成员函数:

```
CString m_strText;                      // 字符串成员
BOOL m_bClose;                          // 是否关闭状态
virtual void Serialize(CArchive& ar);   // 序列化函数
```

8. 添加事件函数和成员函数的代码

主要在 CChatRServerDlg 对话框类的 ChatRServerDlg.cpp 文件中和两个套接字类的实现文件中,添加用户自己的事件函数和成员函数的代码。

5.2.3 服务器程序主要功能的代码和分析

1. CChatLSocket 类对应的文件

(1) ChatLSocket.h 头文件如下：

```
……
class CChatRServerDlg;                          // 关于对话框类的声明
/////////////////////////////////////////////////////////////////////////////
// CChatLSocket command target
// 定义侦听套接字类
class CChatLSocket : public CSocket
{
    // 动态类声明
DECLARE_DYNAMIC(CChatLSocket);

// Attributes
public:

// Operations
public:
    // 构造函数,并且加入了入口参数
    CChatLSocket(CChatRServerDlg* pDlg);
    // 析构函数
    virtual ~CChatLSocket();

// Overrides
public:
    CChatRServerDlg * m_pDlg;                   // 定义对话框类指针变量
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChatLSocket)
    public:
    // 从类向导中添加的事件响应函数
    virtual void OnAccept(int nErrorCode);    // 响应 FD_ACCEPT 事件
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(CChatLSocket)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

// Implementation
protected:

};
……
```

(2) ChatLSocket.cpp 文件如下：

```
……
CChatLSocket::CChatLSocket(CChatRServerDlg* pDlg)
{
    // 初始化对话框类指针成员变量
    m_pDlg=pDlg;
}

CChatLSocket::~ CChatLSocket()
{
    // 程序结束时释放对话框类指针成员变量
    m_pDlg=NULL;
}

// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CChatLSocket, CSocket)
    //{{AFX_MSG_MAP(CChatLSocket)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

//////////////////////////////////////////////////////////////////////////
// CChatLSocket member functions
// FD_ACCEPT 事件处理函数
void CChatLSocket::OnAccept(int nErrorCode)
{
    // TODO: Add your specialized code here and/or call the base class
    // 先执行基类的 OnAccept 函数
    CSocket::OnAccept(nErrorCode);
    // 再调用对话框类的 OnAccept 函数
    m_pDlg->OnAccept();
}

IMPLEMENT_DYNAMIC(CChatLSocket,CSocket)
```

2. CChatCSocket 类对应的文件

(1) ChatCSocket.h 头文件如下：

```
……
class CChatCSocket : public CSocket
{
// 动态类声明
    DECLARE_DYNAMIC(CChatCSocket);

// Attributes
public:

// Operations
```

```
public:
    // 构造函数,并且加入了入口参数
    CChatCSocket(CChatRServerDlg* pDlg);
    // 析构函数
    virtual ~CChatCSocket();

// Overrides
public:
    void ReceiveMsg(CMsg* pMsg);
    void SendMsg(CMsg* pMsg);
    void Initialize();
    CArchive * m_pArchiveOut;                  // 定义输出型 CArchive 对象指针变量
    CArchive * m_pArchiveIn;                   // 定义输入型 CArchive 对象指针变量
    CSocketFile * m_pFile;                     // 定义 CSocketFile 对象指针变量
    CChatRServerDlg * m_pDlg;                  // 定义对话框类指针变量
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChatCSocket)
    public:
    // 从类向导中添加的事件响应函数
    virtual void OnReceive(int nErrorCode);  // 响应 FD_READ 事件
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(CChatCSocket)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

// Implementation
protected:

};
……
```

(2) ChatCSocket.cpp 文件如下：

```
……
// 初始化各成员变量
CChatCSocket::CChatCSocket(CChatRServerDlg* pDlg)
{
    m_pDlg=pDlg;
    m_pFile=NULL;
    m_pArchiveIn=NULL;
    m_pArchiveOut=NULL;
}
// 程序结束时释放各成员变量
CChatCSocket::~ CChatCSocket()
{
    m_pDlg=NULL;
    if (m_pArchiveOut !=NULL) delete m_pArchiveOut;
```