论 CSplitterWnd 分割窗口之间的通信

张永良，黄欢，田会丽，邓迎宾
昆明理工大学信息工程与自动化学院，昆明 （650051）
E-mail：zyl_km@163.com

摘　要：本文讨论了 CSplitterWnd 分割窗口之间的通信，以及在多线程系统中的通信，解决了在各种应用环境中的分割窗口之间的通信问题。

关键词：CSplitterWnd，VC++，分割窗口，通信
中图分类号：TP311.52

0 引言

目前，基于窗口分割的应用开发十分流行，如我们在广泛使用的讯雷或者网际快车等工具，他们都拥有复杂的窗口界面，在这些界面中窗口被分割为若干的区域，每一区域都为用户显示不同的内容，这样的开发技术是很值得大家借鉴的。它对我们开发友好的人性化的用户界面是必须而且很重要的，它甚至影响到我们整个软件开发项目成功与否。窗口分割技术是在同一个框架窗口下同时显示多个窗口的一项技术。运用分割窗口，可以在较短时间内展示给用户更多的信息量，方便了用户的操作性，从而使得用户界面[1]更加的友好，增强了软件的可使用性和可操作性。MFC[2]提供的 CSplitterWnd[2]类，它的功能可将窗口分割成多个可滚动的帧，帧之间的边界称为分割条，可用分割条来调整每个帧的相对大小。对 Window 来说，CSplitterWnd 对象是一个真正的窗口，它完全占据了框架窗口的客户区域，而视窗口则占据了切分窗口的窗片区域。切分窗口并不参与命令传递机制，（窗片中）活动的视窗从逻辑上来看直接被连到了它的框架窗口中。经 CSplitterWnd 分割的每个窗口都被相同的或者不同的视图所填充，且在实际应用中我们不仅要分割出所需要的窗口，而且往往还需要能在各种复杂环境下实现各窗口之间的自由通信，然而这种窗口之间的自由通信往往不能迎刃而解，是我们的程序开发陷入困境，因此，本文针对这种情况，讨论了使用 CSplitterWnd 类来分割的窗口与窗口之间怎样实现其畅通无阻的自由通信方法。

1 关于 CSplitterWnd 类

在 Microsoft VC++6.0 中的 CSplitterWnd 类是一种特殊的框架窗口，它分割的每个窗口都被相同的或者不同的视图所填充。当窗口被切分后用户可以使用鼠标移动切分条来调整窗口的相对尺寸。

CSplitterWnd 的构造函数主要包括下面三个：

BOOL Create(CWnd* pParentWnd,int nMaxRows,int nMaxCols,SIZE sizeMin,CCreateContext* pContext,DWORD dwStyle,UINT nID);

功能描述：该函数用来创建动态切分窗口。 参数含义：pParentWnd 切分窗口的父框架窗口； nMaxRows,nMaxCols 是创建的最大的列数和行数； sizeMin 是窗格的现实大小；pContext 大多数情况下传给父窗口。nID 是字窗口的 ID 号。

BOOL CreateStatic(CWnd* pParentWnd,int nRows,int nCols,DWORD dwStyle,UINT nID)

功能描述：用来创建切分窗口。 参数含义同上。

BOOL CreateView (int row,int col,CruntimeClass* pViewClass,SIZE sizeinit,CcreateContext* pContext);

功能描述：为静态切分的窗口的网格填充视图。在将视图于切分窗口联系在一起的时候必 须先将切分窗口创建好。参数含义同上。

2 创建分割窗口

2.1 创建动态分割窗口

切分窗口的创建方法可以分为动态和静态两种创建方法。动态分割窗口使用 Create 方法来创建。下面的代码将创建 2x2 的窗格。

```
CSplitterWnd m_wndSplitter;
m_wndSplitter.Create(this,2,2,CSize(100,100),NULL)
;
```

但是动态创建的分割窗口的窗格数目不能超过 2x2，而且对于所有的窗格，都必须共享同一个视图，所受的限制也比较多，因此本文不将动态创建作为讨论重点，主要精力放在静态分割窗口的创建上。

2.2 创建静态分割窗口

与动态创建窗口方法相比，静态创建方法可以最多创建 16x16 的窗格。不同的窗格可以使用 CreateView 填充不同的视图。

下面的代码将创建静态分割窗口：

```
//创建一个静态分割窗口，分为2行1列
if(m_wndSplitter.CreateStatic(this,2,1)==NULL)
        return -1;
//将第1行0列再分开1行2列
if(m_wndSplitter1.CreateStatic(&m_wndSplitter,1,2,WS_CHIL
D|WS_VISIBLE,
                m_wndSplitter.IdFromRowCol(1, 0))==NULL)
        return -1;
//将第0行1列再分开2行1列
if(m_wndSplitter2.CreateStatic(&m_wndSplitter1,2,1,WS_CHIL
D|WS_VISIBLE,
                m_wndSplitter1.IdFromRowCol(0, 1))==NULL)
        return -1;
//将CTopView连接到0行0列
m_wndSplitter.CreateView(0,0,RUNTIME_CLASS(CTopView)
,CSize(0,115),NULL);
//将CUserListView连接到0行0列
m_wndSplitter1.CreateView(0,0,RUNTIME_CLASS(CUserList
View),CSize(200,0),NULL);
//将CDisplayMsgView连接到0行0列
m_wndSplitter2.CreateView(0,0,RUNTIME_CLASS(CDisplay
MsgView),CSize(0,420),NULL);
//将CControlView连接到1行0列
m_wndSplitter2.CreateView(1,0,RUNTIME_CLASS(CControlV
iew),CSize(0,0),NULL);
```

创建分割窗口效果如下图1所示：

图 1　用 CSplitterWnd 创建分割窗口

3 分割窗口的通信

3.1 有文档相连的视图之间的通信

由于跟文档类[3]相连的视图类[3]不能完全的与（除文档类之外的）其余的视图类通信。因此只能让它们都与相应得文档类通信。在文档的 OnOpenDocument()函数中设置相应的指针以获得各个视图。代码如下：

```
CTopView* pTopView;
CUserListView* pUserListView;
CDisplayMsgView* pDisplayMsgView;
CControlView* pControlView;
CView* pView;
POSITION pos;
while(pos!=NULL){
        pView=GetNextView(pos);
        if(pView->IsKindOf(RUNTIME_CLASS(pTopView))!=
NULL)        pTopView=(CTopView*)pView;
        else
if(pView->IsKindOf(RUNTIME_CLASS(CUserListView))!=N
ULL)  pUserListView=(CUserListView*)pView;
        else
if(pView->IsKindOf(RUNTIME_CLASS(CDisplayMsgView))!
=NULL)
pDisplayMsgView=(CDisplayMsgView*)pView;
        else
if(pView->IsKindOf(RUNTIME_CLASS(CControlView))!=NU
LL)
                pControlView=(CControlView*)pView;
}
```

这样在文档类中就获得了跟它相连的所有的视图的指针，通过所获得的视图指针就能方便地使各分割窗口所填充的视图之间自由通信了。

3.2 无文档视图与文档关联视图之间的通信

如图 1 中所示，现在假设 CTopView 不与文档相关联的，CUserListView 与文档相

关联，那么此时又怎样来实现 CTopView 与 CUserListView 之间的通信呢？我们知道由于 CUserListView 只能安全的与关联的文档 CUserListDoc 通信，因此，CTopView 如果需要跟 CUserListView 之间通信，也必须借助于文档类。为此需要在 CTopView 中获得文档 CUserListDoc 的指针，在此可以通过主窗口类 MainFrame 来获得程序的任意窗口类的指针。代码实现如下：

```
//在CTopView获得主窗口指针
CMainFrame*
MainFrame=(CMainFrame*)this->GetParent()->GetParent();
CUserListDoc*
pDoc=(CUserListDoc*)MainFrame->GetActiveDocument();
if(pDoc!=NULL) pDoc->Test();
CUserListDoc中的相应的处理函数Test()代码如下：
CUserListView* pUserListView;
POSITION pos;
CView* pView;
while(pos!=NULL) {
        pView=GetNextView(pos);
        if(pView->IsKindOf(RUNTIME_CLASS(CUserListView
))!=NULL)
        pUserListView=(CUserListView*)pView;
}
pUserListView->Test();
```

3.3 无文档关联视图之间的通信

如图 1 中所示，现在假设 CTopView 和 CUserListView 都不与文档相关联的，那么此时又怎样来实现 CTopView 与 CUserListView 之间的通信呢？此时我们可以通过主窗口类 MainFrame 来获得程序的任意窗口类的指针，这样就很方便的实现它们之间的自由通信。下面的代码给出在 CTopView 中访问 CUserListView 中的方法 Test()。 代码如下：

```
CMainFrame*
MainFrame=(CMainFrame*)this->GetParent()->GetParent();
CUserListView*
pUserListView=(CUserListView*)MainFrame->m_wndSplitter1
.GetPane(0,0);
pUserListView->Test();
```

3.4 基于多线程的通信

根据上面的分析结果，只要能够获得应用程序主窗口类 MainFrame，那么就可以通过主窗口类 MainFrame 来获得应用程序中任意窗口类的指针，以实现其窗口之间的自由通信。但是在多线程[3]应用程序中，由于线程的安全机制，线程只能访问它自己创建的 MFC 对象，不能访问主线程中的对象，因此只能依靠发送窗口消息来解决这一问题。其实现方法如下：

1.定义窗口消息：

```
#define WM_USER_TEST WM_USER+1
```

2.在主窗口类 MainFrame 实现消息映射：ON_MESSAGE(WM_USER_TEST,Test)

3.在线程中发送窗口消息：

在线程中首先获得主窗口指针 pMainFrame，通过 pMainFrame 来发送主窗口消息，

```
pMainFrame->SendMessage(WM_USER_TEST,0
,0);
```

这样就使得我们能够在多线程应用环境中，自由地实现各分割窗口之间的通信。

4. 结论

通过本文对 Microsoft VC++ 6.0 中 MFC 提供的 CSplitterWnd 类来怎样创建分割窗口，以及怎样实现各窗口之间在复杂应用环境下自由通信方法的详细讨论，解决了我们运用 CSplitterWnd 来开发应用程序界面的一些困惑问题，使开发的应用程序界面更加丰富，更加自由，更加灵活，从而极大了提高了我们软件开发项目的生命力，也为我们软件开发提高了工作效率，节省了开发成本。

参考文献

[1] 冯峰。Visual C++开发高级界面实例[M].北京：人民邮电出版社，2000.
[2]〔美〕 Microsoft公司著。Microsoft Visual C++ 6.0 MFC类库参考手册[M].北京：北京希望电脑公司出版，1999.
[3] 孙鑫，余安萍。VC++深入详解[M].北京：电子工业出版社，2006.

On the Communication Between Windows Splitted By CSplitterWnd

Zhang Yongliang，Huang Huan，Tian Huili，Deng Yingbin
Kunming University of Science and Technology，Kunming（650051）

Abstract

This paper discusses the communication among windows split by CSplitterWnd, and the communication in the multithread application system, in order to resolve the communication among split windows in every kind of application system.

Keywords：CSplitterWnd，VC++，Split Windows，Communication

作者简介：
张永良（1976-），男，云南陆良人，硕士研究生，主要研究方向为图像处理及网络通信；
黄欢（1966-），女，副教授，主要研究方向为图像信号处理及模式识别。