第 5 章　FTP 客户端实现之二

在第 4 章实现了一个 FTP 客户端程序，那么与本章有什么区别呢？第一，FTP 客户端所基于的应用程序框架不同，第 4 章基于对话框，本章将基于 SDI 开发；第二，开发时的精力分配不同，第 4 章的精力主要集中在与 FTP 服务器的“交流”上，本章将把这种底层的工作交给 MFC 封装的类来实现，主要精力会集中在界面的美化上。

5.1　FTP 客户端简介

本节将会带领大家快速了解本章将要实现的 FTP 客户端的各种功能。包括以树形视图浏览本地文件夹资源、以列表方式显示 FTP 服务器上的文件资源、用拖动文件的方式实现文件的上传和下载。

5.1.1　树形结构的应用

在主窗体的左侧视图中显示选定本地文件夹内的所有文件资源，结构为树形，可以动态地改变本地文件夹的选择，如图 5.1 所示。前方有加号说明路径中还有子路径，单击加号打开此路径，加号变减号，子文件将显示在子树中。鼠标移过此视图时树子项会加亮显示。图标 H 表示文件夹，图标 F 表示文件。

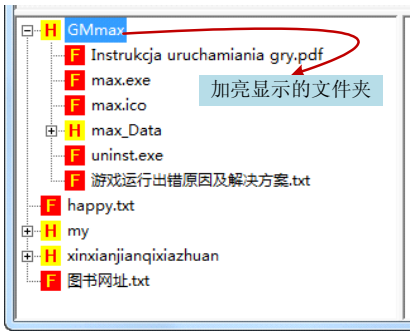

图 5.1　本地文件夹资源显示

5.1.2　列表结构的应用

在主窗体的右侧视图中，将以列表图标的形式显示 FTP 服务器下的所有文件资源，如图 5.2 所示。

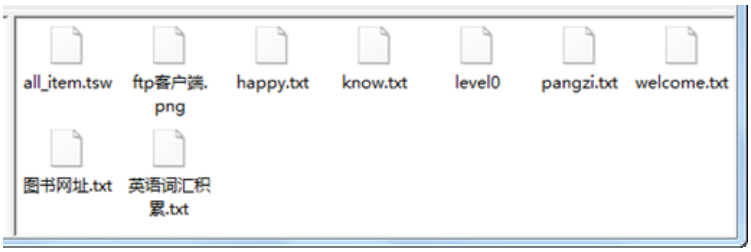

图 5.2　FTP 服务器上的文件资源

5.1.3　信息框的应用

在主窗体的正中央有个信息的显示框，是用于描述用户的一些操作，如图 5.3 所示。

图 5.3　信息显示框

5.1.4　浮动对话框的应用

主程序的最顶端是用来填写本地文件夹路径和连接 FTP 服务器的浮动对话框，如图 5.4 所示。

图 5.4　浮动对话框

那么，最后来看一下本章 FTP 客户端的全貌吧，如图 5.5 所示。

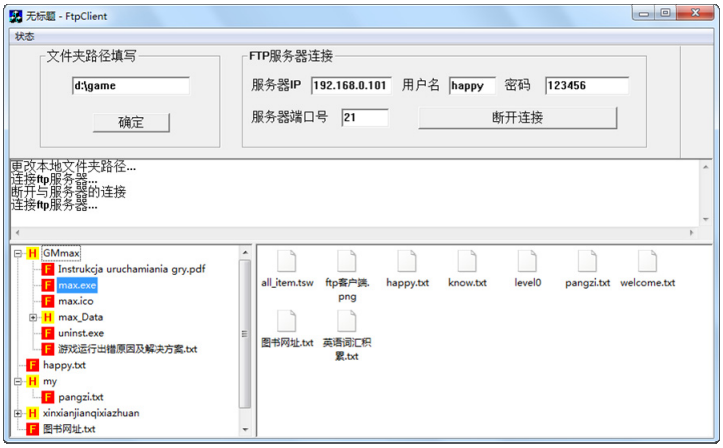

图 5.5　FTP 客户端全貌

5.2　关键技术讲解

本节主要介绍本章要用到的所有关键技术，包括如何制作浮动对话框，然后像工具栏

一样加到菜单之下；如何将客户区分栏；如何实现对树形和列表视图项目的拖动。

我们创建的工程是基于 SDI 的，命名为 FtpClient。在向导的第 6 步：选择 CFtpClientView 基于 CLiStView 类，如图 5.6 所示。

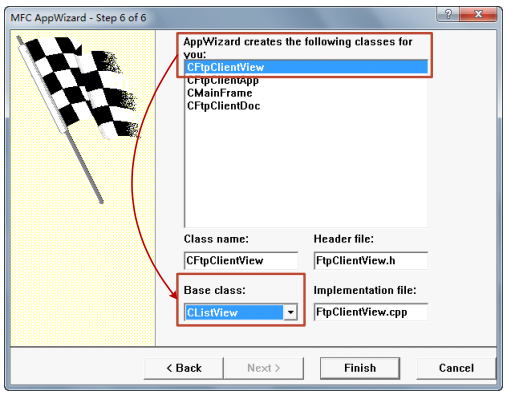

图 5.6　修改 CFtpClientView 的基类

5.2.1　制作、使用浮动对话框

浮动对话框，顾名思义就是可以浮动在主窗体之上。制作方法如下所述。

（1）在资源视图中插入对话框资源，修改 ID 为 IDD_FLOAT_DLG；然后修改属性：去掉对话框的边缘，将 Style 改为 Child。如图 5.7 所示。

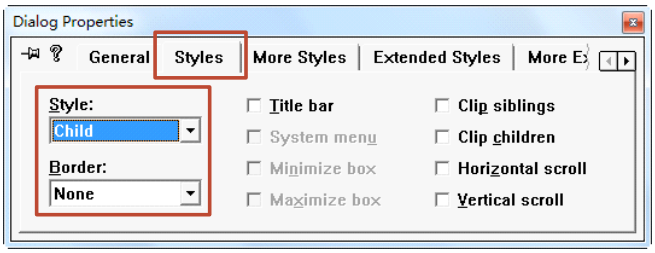

图 5.7　对话框属性设置

（2）为对话框拖放控件，然后进行设计，用户可以根据自己的喜好摆放，笔者的设计如图 5.8 所示。

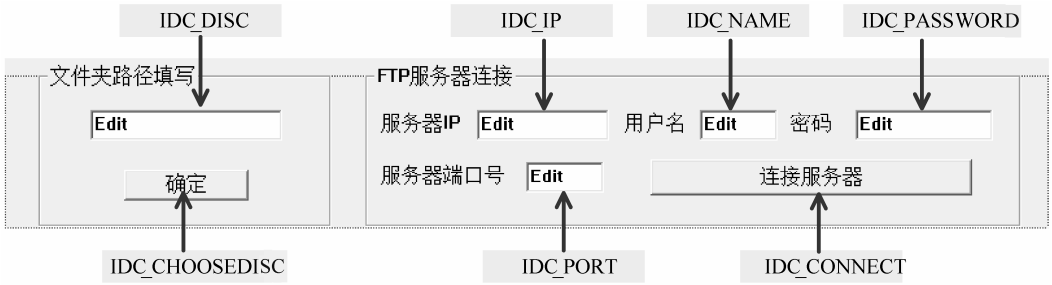

图 5.8　浮动对话框界面设计

（3）在 CMainFrame 中添加一个浮动对话框的变量 m_myDlg。

```
class CMainFrame : public CFrameWnd
{
...
// Attributes
public:
    CDialogBar      m_myDlg;            //浮动对话框
...
};
```

在 CMainFrame 的 OnCreate()成员函数中完成两个任务。第一，去掉由向导为我们添加的工具栏和状态栏，它们影响到了程序的美观；第二，添加显示刚才设计的对话框的代码。

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    //注释掉由向导创建的工具栏和状态栏
/*  if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
         WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
         CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }   */          //注释结束的位置

    // TODO: Delete these three lines if you don't want the
    // toolbar to be dockable
    //  m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    //  EnableDocking(CBRS_ALIGN_ANY);
    //  DockControlBar(&m_wndToolBar);

    //添加我们自己设计的浮动对话框
    if( !m_myDlg.Create(this,IDD_FLOAT_DLG,
            CBRS_TOP | CBRS_HIDE_INPLACE,IDD_FLOAT_DLG) )
    {
        TRACE0("Failed to create dialog bar m_myDlg\n");
        return -1;
    }
    //设置浮动对话框停靠在框架的顶部
    m_myDlg.EnableDocking(CBRS_ALIGN_TOP);
    //设置框架窗口的顶部运行停靠
    EnableDocking(CBRS_ALIGN_TOP);
    //框架指定浮动对话框停靠
    DockControlBar(&m_myDlg);

```

```
45      return 0;
46  }
```

程序中去掉了工具栏和状态栏的功能，所以可以将代表两个工具的对象 m_wndStatusBar、m_wndToolBar 也注释掉。它们定义在类 CMainFrame 的头文件中，如下：

```
01  class CMainFrame : public CFrameWnd
02  {
03  ...
04  protected:  // control bar embedded members
05  //  CStatusBar  m_wndStatusBar;
06  //  CToolBar    m_wndToolBar;
07  ...
08  // Generated message map functions
09  protected:
10  };
```

不注释掉也不会影响程序的编译执行，用户可自由处理。通过调用类 CDialogBar 的成员函数 Create()，装载我们设计的对话框资源模版、创建对话框窗口、设置它的样式，最后关联窗口到 CDialogBar 对象 m_myDlg 上。函数原型如下：

```
virtual BOOL Create(
   CWnd* pParentWnd,
   UINT nIDTemplate,
   UINT nStyle,
   UINT nID
);
```

参数及其含义介绍如下。

- pParentWnd：指向装载浮动对话框的父窗口的指针，我们直接使用了 this。
- pIDTemplate：对话框资源的 ID 号。
- nStyle：对话框在框架窗口的位置，可以是 CBRS_TOP、CBRS_BOTTOM 等。
- nID：对话框控件的 ID 号。同参数 pIDTemplate。

其他成员函数如 EnableDocking()的使用很简单，代码中已经加入了注释，此处不再详细讲解。

那么，编译运行程序后就会发现，工具栏和状态栏消失了，取而代之的是我们自己设计的浮动对话框，用鼠标尝试拖动它，会有如图 5.9 所示效果。

图 5.9　拖动浮动对话框

用户可以任意移动它，如果单击对话框上面的关闭按钮关闭它后，我们需要重新启动程序让它再次显示。因为本程序没有实现再次显示浮动窗口功能，用户可以自己实现。

5.2.2　客户区的分割

将整个客户区分割为 3 个部分：用于显示用户操作的信息窗口、用于显示本地文件夹资源的树形视图窗口和用于显示 FTP 服务器上文件资源的列表视图窗口。效果如图 5.10 所示。

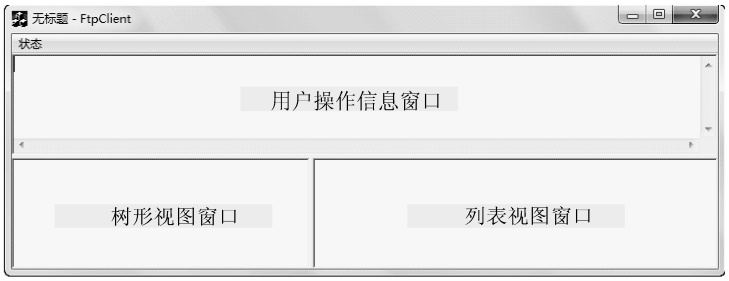

图 5.10　窗口分割效果图

为类 CMainFrame 添加两个成员变量，如下：

```
01  class CMainFrame : public CFrameWnd
02  {
03  ...
04  // Attributes
05  public:
06      CSplitterWnd    m_splitter1;
07      CSplitterWnd    m_splitter2;
08  ...
09  };
```

类 CSplitterWnd 提供了分割窗口的功能，就是一个窗口包含多个窗格。

为类 CMainFrame 添加函数 OnCreateClient()的实现，程序编写如下：

```
01  BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
02                                  CCreateContext* pContext)
03  {
04      // TODO: Add your specialized code
05      // here and/or call the base class
06
07      //窗口的第一次分割
08      if( !m_splitter1.CreateStatic(this,2,1) )
09      {
10          return false;
11      }
12      if( !m_splitter1.CreateView(0,0,RUNTIME_CLASS(CMsgShow),
13                                      CSize(0,0),pContext) )
14      {
15          return false;
16      }
17      m_splitter1.SetRowInfo(0,100,50);       //设置行高
18
19      //窗口的第二次分割
20      if( !m_splitter2.CreateStatic(&m_splitter1,1,2,
21          WS_CHILD |WS_VISIBLE,m_splitter1.IdFromRowCol(1,0)) )
22      {
```

```
23          return false;
24      }
25      if( !m_splitter2.CreateView(0,0,RUNTIME_CLASS(CFileTree),
26                                      CSize(0,0),pContext) )
27      {
28          return false;
29      }
30      if( !m_splitter2.CreateView(0,1,RUNTIME_CLASS(CFtpClientView),
31                                      CSize(0,0),pContext) )
32      {
33          return false;
34      }
35      m_splitter2.SetColumnInfo(0,300,50);   //设置列宽
36
37      return true;
38  }
```

调用类 CSplitterWnd 的成员函数 CreateStatic()创建静态的分割窗口，函数原型如下：

```
virtual BOOL CreateStatic(
   CWnd* pParentWnd,
   int nRows,
   int nCols,
   DWORD dwStyle = WS_CHILD | WS_VISIBLE,
   UINT nID = AFX_IDW_PANE_FIRST
);
```

参数及其含义如下所述。

- pParentWnd：分割窗口的父框架窗口。
- nRows：分割的行数，要求不大于 16。
- nCols：分割的列数，要求不大于 16。
- dwStyle：指定的窗口样式。有默认参数。
- nID：子窗口的 ID。ID 可以默认为 AFX_IDW_PANE_FIRST，但是当分割窗口是嵌套在其他分割窗口中时，必须是嵌套窗口的 ID。

第一次分割窗口时，父窗口是框架 CMainFrame，分割成 2 行 1 列。第二次分割窗口时，是嵌入在第一次分割的窗口中的，所以父窗口为 m_splitter1，分割为 1 行 2 列，嵌套的窗口 ID 通过类 CSplitterWnd 的成员函数 IdFromRowCol()获得。

类 CSplitterWnd 的成员函数 CreateView()为静态分割窗口创建窗格，原型如下：

```
virtual BOOL CreateView(
   int row,
   int col,
   CRuntimeClass* pViewClass,
   SIZE sizeInit,
   CCreateContext* pContext
);
```

参数及其含义如下所述。

- row：指定放置新视图的窗口行。
- col：指定放置新视图的窗口列。
- pViewClass：指定一个 CRuntimeClass 作为新视图。
- sizeInit：指定新视图的初始大小，即长和宽。
- pContext：为用来创建上下文的指针创建视图。

调用类 CSplitterWnd 的成员函数 SetRowInfo()和 SetColumnInfo()分别设置分割窗口的行高取值范围和列宽取值范围。函数原型如下：

```
void SetRowInfo(
   int row,
   int cyIdeal,
   int cyMin
);
void SetColumnInfo(
   int col,
   int cxIdeal,
   int cxMin
);
```

参数及其含义如下所述。

- row、col：指定分割窗口的行、列，用于定位。
- cyIdeal、cxIdeal：以像素为单位，为分割窗口指定理想的行高、列宽。
- cyMin、cxMin：以像素为单位，为分割窗口指定最小的行高、列宽。

在函数 OnCreateClient()中，我们将 3 个视图 CMsgShow、CFileTree 和 CFtpClientView 指定到相应的分割窗格中。前两个是我们利用类向导添加的新类，分别基于类 CEditView 和 CTreeView，最后一个是我们创建工程时由向导为我们创建的视图类，基于类 CListView。至此客户区分割的操作代码添加完毕。

5.2.3　树形视图项目拖动效果

我们可以通过捕获 3 个事件来添加拖动效果的代码：鼠标左键选中项目并且开始拖动、鼠标移动和鼠标左键抬起。

1．选中视图项

我们需要用类向导添加一个新类 CFileTree，基于 CTreeView，如图 5.11 所示。

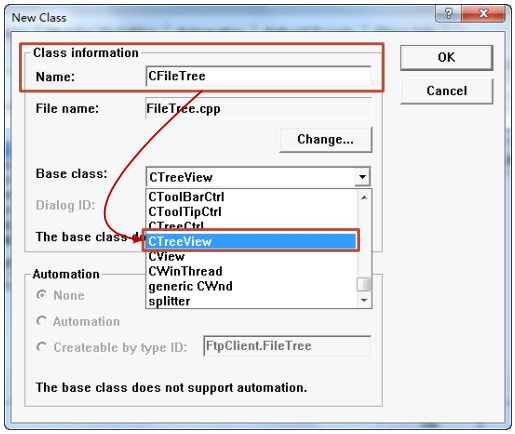

图 5.11　添加新类 CFileTree

在类 CFileTree 的实现文件中，添加文件包含指令如下：

```
#include "MainFrm.h"
#include "FtpClientView.h"
```

再利用类向导为它添加函数 OnBegindrag()，如图 5.12 所示。

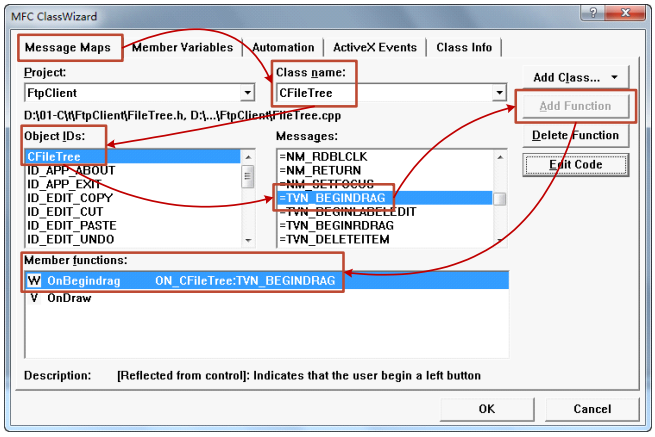

图 5.12　为类 CFileTree 添加消息响应

为函数 OnBegindrag()添加代码，如下：

```
01  void CFileTree::OnBegindrag(NMHDR* pNMHDR, LRESULT* pResult)
02  {
03      NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
04
05      // TODO: Add your control notification handler code here
06
07      //树的叶子节点
08      m_hItemDragS = pNMTreeView->itemNew.hItem;
09
10      ...                              //略去部分程序其他功能的代码
11
12      //得到用于拖动时显示的图像列表
13      m_pDragImage = tree->CreateDragImage( m_hItemDragS );
14      if( !m_pDragImage )
15          return;
16
17      //开始拖动
18      m_bDragging = true;
19      //拖动时的图像索引、鼠标指针相对于图像的左上角
20      m_pDragImage->BeginDrag( 0,CPoint(16,8) );
21
22      CPoint  pt = pNMTreeView->ptDrag;
23      ClientToScreen( &pt );
24      m_pDragImage->DragEnter(NULL,pt );
25
26      //捕获鼠标的所有事件
27      SetCapture();
28
29      *pResult = 0;
30  }
```

函数 OnBegindrag()中的变量是类 CFileTree 的公有成员变量，定义如下：

```
01  class CFileTree : public CTreeView
```

```
02  {
03  ...
04  // Attributes
05  public:
06      CTreeCtrl       *tree;              //指向 treeview 本身
07      CImageList*     m_pDragImage;       //拖动时显示的图像列表
08      HTREEITEM       m_hItemDragS;       //被拖动的标签项
09      BOOL            m_bDragging;        //鼠标是否处于拖动状态
10  ...
11  };
```

成员变量在类 CFreeTree 的构造函数初始化如下：

```
01  CFileTree::CFileTree()
02  {
03      tree = &GetTreeCtrl();
04      m_bDragging = false;
05  }
```

类 CTreeCtrl 的成员函数 GetTreeCtrl()返回树视图控件的引用。函数 OnBegindrag()中结构 NM_TREEVIEW 定义如下：

```
typedef struct _NM_TREEVIEW {
     NMHDR hdr;
     UINT action;
     TV_ITEM itemOld;
     TV_ITEM itemNew;
     POINT ptDrag;
} NM_TREEVIEW;
```

参数及其含义如下所述。

- hdr：另一个包含通知消息信息的结构 NMHDR。
- action：通知具体操作的标志。
- itemOld：一个包含旧项目状态信息的结构 TV_ITEM。当通知消息没有用到它时会被置 0。
- itemNew：一个包含新项目状态信息的结构 TV_ITEM。当通知消息没有用到它时会被置 0。
- ptDrag：引起通知消息被发送，客户区事件发生时鼠标的坐标位置。

我们要从这个结构中获取两个信息：itemNew.hItem 和 ptDrag。前者是结构 TV_ITEM，用来指定或返回树视图项的属性。结构 TV_ITEM 的字段 hItem 放的是这个结构指向树视图项的句柄 HTREEITEM，被保存在 m_hItemDragS 变量中。

然后用类 CTreeCtrl 的一个成员函数和类 CImageList 的两个成员函数完成图像拖动的准备工作，它们是：函数 CreateDragImage()用来为指定的树视图项创建拖动时的位图、函数 BeginDrag()表示拖动位图操作的开始、函数 DragEnter()用来在拖动操作期间在指定的位置显示位图和锁定更新。函数 BeginDrag()的原型如下：

```
BOOL BeginDrag(
   int nImage,
   CPoint ptHotSpot
);
```

参数及其含义如下所述。

- nImage：索引号从 0 开始的位图号，用来指定位图。
- ptHotSpot：起始拖动时鼠标的坐标位置，坐标是相对于位图的左上角而言的。

函数 DragEnter()的原型如下：

```
static BOOL PASCAL DragEnter(
   CWnd* pWndLock,
   CPoint point
);
```

参数及其含义如下所述。

- pWndLock：指向拥有拖动图像的窗口指针。若参数赋值为 NULL，这个函数拖动图像的坐标是相对于桌面窗口的，即屏幕坐标的左上角。
- point：显示拖动图像的位置。坐标是相对于窗口或者屏幕坐标，不是客户区坐标。

所以我们在使用函数 DragEnter()的时候，用类 CWnd 的成员函数 ClientToScreen()，将给定的客户区点坐标转换为屏幕点坐标。最后调用类 CWnd 的成员函数 SetCapture()，以后不管鼠标的位置在哪里，所有的鼠标后续输入都会被送到当前的窗口处理。

至此，鼠标左键选中项目并且开始拖动事件的捕捉和处理代码的编写和解释完毕。

2．图像随鼠标移动

利用类向导为类 CFileTree 添加鼠标移动事件。如图 5.13 所示。

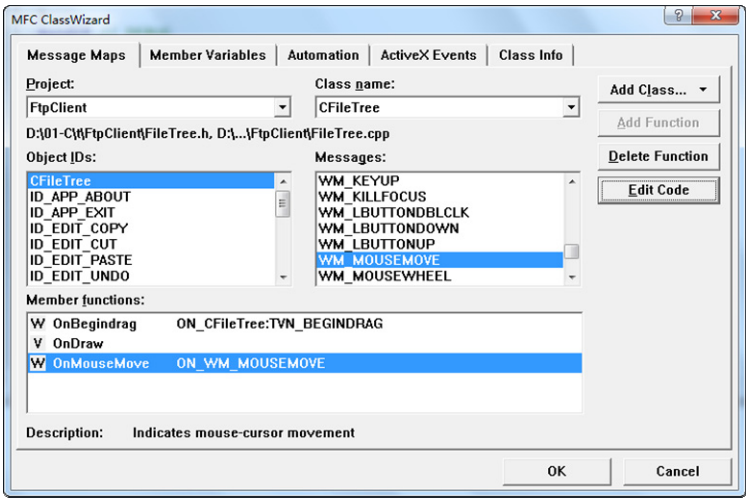

图 5.13　添加鼠标移动事件

为函数 OnMouseMove()添加代码，如下：

```
01  void CFileTree::OnMouseMove(UINT nFlags, CPoint point)
02  {
03      // TODO: Add your message handler code here and/or call default
04
05      HTREEITEM  hItem;
06
07      CMainFrame* mFrm = (CMainFrame*)AfxGetMainWnd();
08      CFtpClientView *pEView =
09           (CFtpClientView *)(mFrm->m_splitter2.GetPane(0,1));
10
```

```
    //矩形位置
    CRect listRt,treeRt;
    pEView->GetCtrlRect(&listRt);  //自定义的函数
    GetCtrlRect(&treeRt);          //同上

    //区域
    CRgn listRgn,treeRgn;
    listRgn.CreateRectRgn(listRt.left,listRt.top,
                          listRt.right,listRt.bottom);
    treeRgn.CreateRectRgn(treeRt.left,treeRt.top,
                          treeRt.right,treeRt.bottom);

    CPoint  pt = point;
    ClientToScreen(&pt);

    //PtInRegion()
    //Determines whether a specified point is in the region.
    if( m_bDragging &&
     ( listRgn.PtInRegion(pt) || treeRgn.PtInRegion(pt) ) )
    {
        CImageList::DragMove( pt );
    }
    else
    {
        //HitTest() -- Returns the current position of the cursor
        //related to the CTreeCtrl object.
        if( (hItem = tree->HitTest(point)) != NULL )
        {
            //鼠标经过时高亮显示
            tree->SelectDropTarget( hItem );
        }
    }

    CTreeView::OnMouseMove(nFlags, point);
}
```

函数 OnMouseMove()首先调用 AfxGetMainWnd()，获取指向当前程序主框架 CMainFrame 的指针，保存在变量 mFrm 中，通过此变量调用其成员变量 m_splitter2 的成员函数，即类 CSplitterWnd 的成员函数 GetPane()，得到指定行列窗格的指针。这里获取的是列表视图窗格的指针，保存在变量 pEView 中。

我们要在类 CFileTree 中添加一个自定义的成员函数 GetCtrlRect()，如图 5.14 所示。

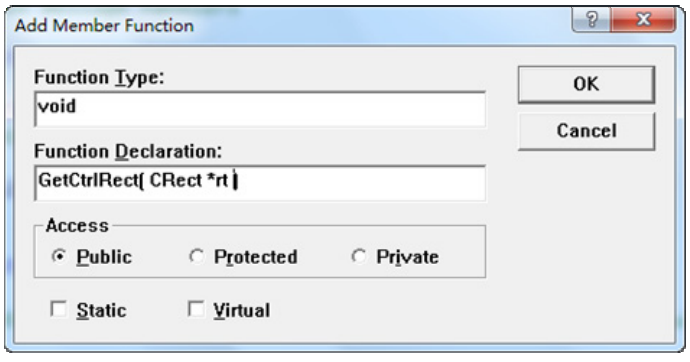

图 5.14　添加自定义的成员函数

添加如下代码即可。

```
01  void CFileTree::GetCtrlRect(CRect *rt)
02  {
03      tree->GetClientRect(rt);
04      ClientToScreen(*rt);
05  }
```

简单地封装了两个函数，功能是获取树视图窗口的矩形大小，即窗口大小；然后将坐标转换为相对屏幕的坐标值。同样，需要在类 CFtpClientView 中添加这样一个自定义的函数 GetCtrlRect()，如下：

```
01  void CFtpClientView::GetCtrlRect(CRect *rt)
02  {
03      filelist->GetClientRect(rt);
04      ClientToScreen(*rt);
05  }
```

函数 OnMouseMove()完成的功能是创建两个“区域”。实例化两个类 CRgn 的对象 listRgn 和 treeRgn，调用类 CRgn 的成员函数 CreateRectRgn()创建两个矩形区域，分别覆盖了树形结构视图和列表结构视图。函数 CreateRectRgn()的原型如下：

```
BOOL CreateRectRgn(
   int x1,
   int y1,
   int x2,
   int y2
);
```

参数及其含义如下所述。

- x1、y1：指定矩形区域左上角点的坐标位置。
- x2、y2：指定矩形区域右下角点的坐标位置。

函数 OnMouseMove()最后会判断鼠标是否处于移动的状态，是在树形结构视图区域还是在列表结构视图区域。通过类 CRgn 的成员函数 PtInRegion()判断指定的点是否在指定的区域范围内。

在指定的区域范围内，并且当前正处在移动的状态时，就该调用拖动操作的第 4 个函数了，它是类 CImageList 的成员函数 DragMove()，原型如下：

```
static BOOL PASCAL DragMove(
   CPoint pt
);
```

pt 是拖动操作时，鼠标新的位置点。这个函数将图像移动到指定的新的坐标点，也就是图像会随着鼠标移动效果展现。

若是不能满足刚才的判断条件，还有另一个有趣的效果需要实现。我们通过类 CTreeCtrl 的成员函数 HitTest()判断鼠标点相对树视图控件的位置，若是在控件内部的话，会返回指定位置树视图项的句柄；当指定位置不在任何一个树视图项上时，则返回 NULL。即鼠标在树视图窗口上“划过”，相应的树视图项就会有“被选中”的效果。通过调用类 CTreeCtrl 的成员函数 SelectDropTarget()，以一种表明树视图项被选中的样式，重绘树视图相应项，参数是树视图项的句柄。

3．鼠标图像释放

利用类向导为类 CFileTree 添加最后一个事件：鼠标左键弹起。如图 5.15 所示。

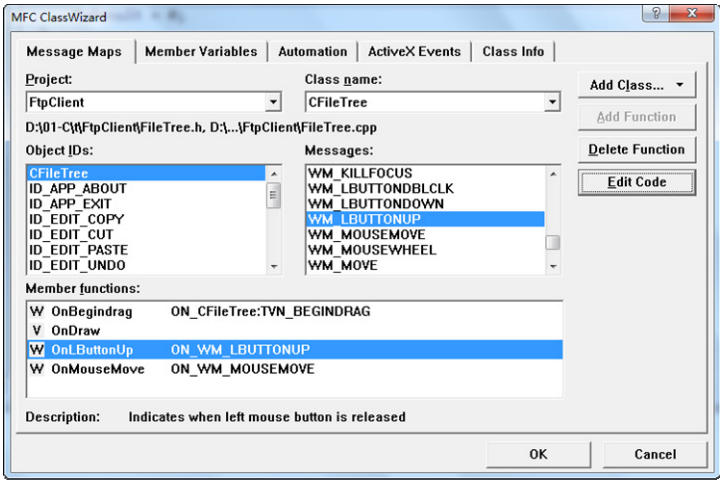

图 5.15　添加鼠标左键弹起事件

为函数 OnLButtonUp()添加代码，如下：

```
01  void CFileTree::OnLButtonUp(UINT nFlags, CPoint point)
02  {
03      // TODO: Add your message handler code here and/or call default
04
05      CMainFrame* mFrm = (CMainFrame*)AfxGetMainWnd();
06      CFtpClientView *pEView =
07           (CFtpClientView *)(mFrm->m_splitter2.GetPane(0,1));
08
09      if(m_bDragging)
10      {
11          m_bDragging = false;
12
13          CImageList::DragLeave(this);    //解锁窗口
14          CImageList::EndDrag();          //结束拖放
15          ReleaseCapture();               //释放鼠标
16
17          tree->SelectDropTarget(NULL);
18
19          CPoint  pt = point;
20          ClientToScreen(&pt);
21
22          CRect listRt;
23          pEView->GetCtrlRect(&listRt);  //自定义函数
24
25          CRgn listRgn;
26          listRgn.CreateRectRgn(listRt.left,listRt.top,
27          listRt.right,listRt.bottom);
28
29          //若文件拖动到了服务器视图的区域内
30          if( listRgn.PtInRegion(pt) )
31          {
32              ...                         //省略用于其他目的的代码
```

```
33          }
34      }
35
36      CTreeView::OnLButtonUp(nFlags, point);
37  }
```

函数 OnLButtonUp()会调用函数 AfxGetMainWnd()获取本程序主框架的指针，并保存在变量 mFrm 中，用 mFrm 通过其数据成员 m_splitter2 获取列表视图窗格的指针。

当确定鼠标是在拖动图像时左键才弹起，即 m_bDragging 为 true 时，我们用最后 2 个函数来完成拖动工作。它们是类 CImageList 的成员函数：DragLeave()用来解锁参数指定的窗口、隐藏图像、允许窗口更新；EndDrag()用来结束拖动操作。函数原型如下：

```
static BOOL PASCAL DragLeave(
   CWnd* pWndLock
);
static void PASCAL EndDrag( );
```

通过函数 ReleaseCapture()释放鼠标的捕获。然后获取列表视图矩形大小、创建覆盖列表视图的区域、判断鼠标点移动到了创建的区域范围之内时所添加的任意的操作。

5.2.4　列表视图项目拖动效果

与树形视图项目拖动效果类似，我们可以通过捕获 3 个事件来添加拖动效果的代码：鼠标左键选中项目并且开始拖动、鼠标移动和鼠标左键弹起。

首先，在类 CFtpClientView 的实现文件中头部添加文件包含指令，如下：

```
#include "MainFrm.h"
#include "FileTree.h"
```

在类 CFtpClientView 的头文件中添加类的声明，代码如下：

```
class CFtpClientDoc;
```

1．选中视图项

利用类向导为类 CFtpClientView 添加函数 OnBegindrag()，如图 5.16 所示。

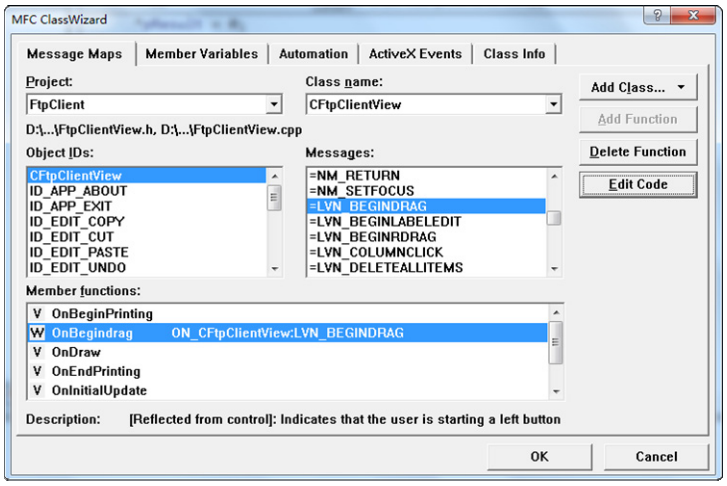

图 5.16　为类 CFtpClienView 添加消息响应

为函数 OnBegindrag()添加代码，如下：

```
void CFtpClientView::OnBegindrag(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    // TODO: Add your control notification handler code here

    int nSelected = filelist->GetNextItem(-1, LVNI_SELECTED);

    //得到用于拖动时显示的图像列表
    CPoint pt;
    filelist->ClientToScreen( &pt );
    m_pImageList = filelist->CreateDragImage(nSelected, &pt);
    if( !m_pImageList )
        return;

    //开始拖动
    m_isDragging  =  true;
    m_pImageList->BeginDrag( 0, CPoint(8,8) );
    ClientToScreen( &pt );
    m_pImageList->DragEnter( NULL,pt );
    SetCapture();

    ...                                 //省略其他功能代码

    *pResult = 0;
}
```

函数 OnBegindrag()中的变量是类 CFtpClientView 的公有成员变量，定义如下：

```
class CFtpClientView : public CListView
{
...
// Attributes
public:
    CListCtrl   *filelist;      //自身窗体的指针
    CImageList  *m_pImageList;  //图像列表
    bool        m_isDragging;   //判断拖动状态
...
};
```

成员变量在类 CFtpClientView 的构造函数初始化如下：

```
CFtpClientView::CFtpClientView()
{
    // TODO: add construction code here
    filelist=&GetListCtrl();
    m_isDragging = false;
}
```

函数 OnBegindrag()调用类 CListCtrl 的两个成员函数：GetNextItem()和 CreateDragImage()。函数 GetNextItem()用来检索满足指定条件的列表项，函数原型如下：

```
int GetNextItem(
   int nItem,
   int nFlags
) const;
```

参数及其含义如下所述。

- nItem：指定检索的起始位置，位置即列表项的索引。
- nFlags：被请求列表项与指定索引列表项的几何关系，如 LVNI_ABOVE 等；或者是被请求列表项的状态，如 LVNI_SELECTED。本例使用这一标识，表示被选中的状态。

函数 CreateDragImage()用来为指定的列表项创建拖动图像，函数原型如下：

```
CImageList* CreateDragImage(
   int nItem,
   LPPOINT lpPoint
);
```

参数及其含义如下所述。

- nItem：要创建拖动图像的列表项的索引。
- lpPoint：结构 POINT 的指针。图像左上角相对于列表视图坐标的起始位置。

函数 OnBegindrag()剩下的操作，我们应该很熟悉了，包括调用类 CImageList 的成员函数 BeginDrag()，表示开始拖动图像；调用类 CImageList 的成员函数 DragEnter()，锁定视图的更新；调用类 CWnd 的成员函数 SetCapture()捕获鼠标后续的所有事件。

2．图像随鼠标移动

利用类向导为类 CFtpClientView 添加下一个事件：鼠标移动。如图 5.17 所示。

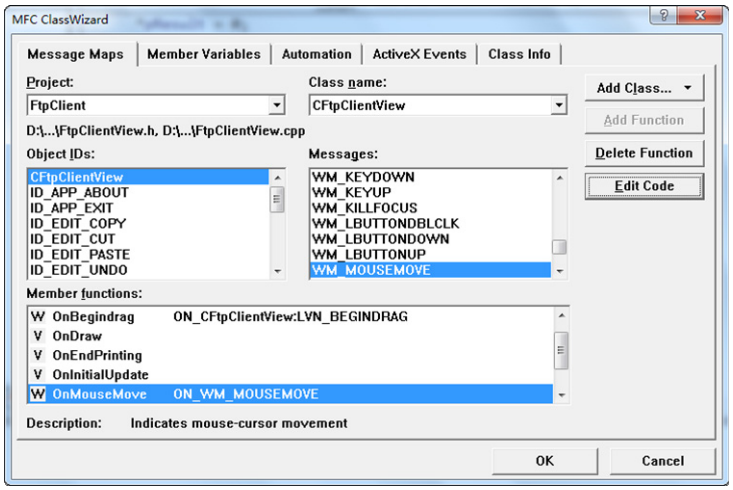

图 5.17　添加鼠标移动事件

为函数 OnMouseMove()添加代码，如下：

```
01  void CFtpClientView::OnMouseMove(UINT nFlags, CPoint point)
02  {
03      // TODO: Add your message handler code here and/or call default
04
05      CMainFrame* mFrm = (CMainFrame*)AfxGetMainWnd();
06      CFileTree *pEView =
07              (CFileTree *)(mFrm->m_splitter2.GetPane(0,0));
08
09      //获取窗口矩形大小
10      CRect listRt,treeRt;
```

```
11      GetCtrlRect(&listRt);           //自定义函数
12      pEView->GetCtrlRect(&treeRt);   //同上
13
14      //创建窗口矩形区域
15      CRgn listRgn,treeRgn;
16      listRgn.CreateRectRgn(listRt.left,listRt.top,
17                            listRt.right,listRt.bottom);
18      treeRgn.CreateRectRgn(treeRt.left,treeRt.top,
19                            treeRt.right,treeRt.bottom);
20
21      CPoint  pt = point;
22      ClientToScreen(&pt);
23      if( m_isDragging &&
24          (listRgn.PtInRegion(pt) || treeRgn.PtInRegion(pt) ) )
25      {
26          CImageList::DragMove(pt);
27      }
28
29      CListView::OnMouseMove(nFlags, point);
30  }
```

函数 OnMouseMove()的实现过程：获取树形视图、列表视图窗口矩形大小，创建覆盖树形视图、列表视图窗口的区域，检测鼠标是否处于拖动图像状态，是否在树形视图或列表视图的区域范围内，调用类 CImageList 的成员函数 DragMove()拖动图像。

3．鼠标图像释放

利用类向导为类 CFtpClientView 添加最后一个事件：鼠标左键弹起。如图 5.18 所示。

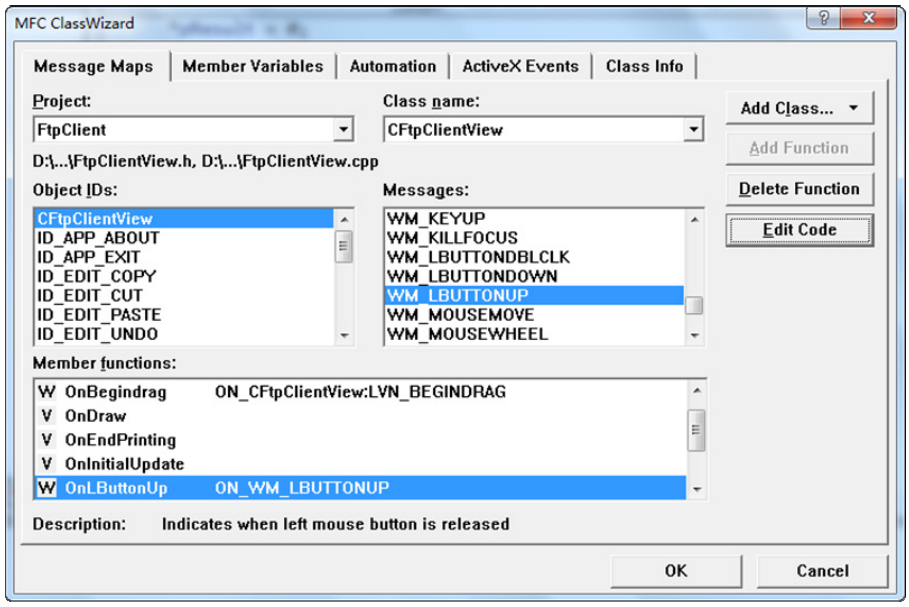

图 5.18　添加鼠标左键弹起事件

为函数 OnLButtonUp()添加代码，如下：

```
01  void CFtpClientView::OnLButtonUp(UINT nFlags, CPoint point)
02  {
03      // TODO: Add your message handler code here and/or call default
```

```
04
05      CMainFrame* mFrm = (CMainFrame*)AfxGetMainWnd();
06      CFileTree *pEView = (CFileTree *)
07                          (mFrm->m_splitter2.GetPane(0,0));
08
09      if( m_isDragging )
10      {
11          m_isDragging  =  false;
12          CImageList::DragLeave(this);
13          CImageList::EndDrag();
14          ReleaseCapture();
15
16          CRect treeRt;
17          pEView->GetCtrlRect(&treeRt);
18
19          CRgn treeRgn;
20          treeRgn.CreateRectRgn(treeRt.left,treeRt.top,
21                                treeRt.right,treeRt.bottom);
22
23          ClientToScreen( &point);
24          CPoint  pt = point;
25          if( treeRgn.PtInRegion(pt))     //释放点的位置
26          {
27              ...                         //添加处理事件的代码
28          }
29      }
30
31      CListView::OnLButtonUp(nFlags, point);
32  }
```

函数 OnLButtonUp()实现的功能是解锁拖动窗口、结束拖动操作、释放鼠标的捕获、获取树视图矩形大小、创建覆盖树视图的区域、判定鼠标点在树视图区域之内。

5.3　实现 FTP 客户端

这个实例是通过 WinInet API 来实现 FTP 客户端的，无需考虑底层的通信协议和数据传输工作，所以我们把近一半的精力用在了华丽的程序界面设计上。

5.3.1　WinInet 类介绍

MFC 提供的 WinInet 类是对 WinInet API 的封装，为我们提供了更加方便的编程接口。主要用到两个类：CInternetSession 和 CFtpConnection。

1．CInternetSession类

用来创建或者初始化一个或多个同步的网络会话。它的构造函数原型如下：

```
CInternetSession(
   LPCTSTR      pstrAgent = NULL,
   DWORD_PTR    dwContext = 1,
   DWORD        dwAccessType = PRE_CONFIG_INTERNET_ACCESS,
   LPCTSTR      pstrProxyName = NULL,
   LPCTSTR      pstrProxyBypass = NULL,
```

```
   DWORD        dwFlags = 0
);
```

我们的实例程序直接使用了所有的默认参数值。

当我们要在服务器上执行指定的服务，如 FTP 服务，必须要先建立连接，用到的成员函数是 GetFtpConnection()。函数原型如下：

```
CFtpConnection* GetFtpConnection(
   LPCTSTR          pstrServer,
   LPCTSTR          pstrUserName = NULL,
   LPCTSTR          pstrPassword = NULL,
   INTERNET_PORT    nPort = INTERNET_INVALID_PORT_NUMBER,
   BOOL             bPassive = FALSE
);
```

参数及其含义如下所述。

- pstrServer：包含 FTP 服务器 IP 地址的字符串。
- pstrUserName：包含用户名的字符串，若为 NULL，默认匿名登录。
- pstrPassword：包含登录密码的字符串。
- nPort：服务器的端口号，对于 FTP 服务默认为 21。
- bPassive：为这个会话指定被动或主动的模式，默认为主动模式。

返回一个指向类 CFtpConnection 的指针。

2．CFtpConnection类

此类主要用于管理 FTP 服务连接，并允许用户直接操作服务器目录和文件。我们主要用到了此类的两个成员函数：PutFile()用来上传文件；GetFile()用来下载文件。函数原型如下：

```
BOOL PutFile(
   LPCTSTR      pstrLocalFile,
   LPCTSTR      pstrRemoteFile,
   DWORD        dwFlags = FTP_TRANSFER_TYPE_BINARY,
   DWORD_PTR    dwContext = 1
);
```

参数及其含义如下所述。

- pstrLocalFile：包含要上传文件路径的字符串。
- pstrRemoteFile：包含要保存在 FTP 服务器上文件路径的字符串。
- dwFlags：指定文件的传输方式，以二进制或者 ASCII 码形式传输。默认为二进制形式。
- dwContext：文件的标识。

依据返回值判定上传操作是否成功。函数 GetFile()的原型如下：

```
BOOL GetFile(
   LPCTSTR          pstrRemoteFile,
   LPCTSTR          pstrLocalFile,
   BOOL             bFailIfExists = TRUE,
   DWORD            dwAttributes = FILE_ATTRIBUTE_NORMAL,
   DWORD            dwFlags = FTP_TRANSFER_TYPE_BINARY,
   DWORD_PTR        dwContext = 1
);
```

参数及其含义如下所述。

- pstrRemoteFile：包含要接收 FTP 服务器上文件路径的字符串。
- pstrLocalFile：包含要在本地上创建的文件路径的字符串。
- bFailIfExists：当下载的路径上有同名的文件时，是否会导致下载失败。默认为 True，即会导致下载失败。
- dwAttributes：表示文件的属性。
- dwFlags：指定调用发生时的条件。
- dwContext：文件检索的上下文标识。

用户可以依据函数 GetFile()的返回值来判定下载操作是否成功。

5.3.2　FTP 服务器操作

通过浮动对话框获取用户输入的信息，调用类的成员连接、登录 FTP 服务器，最后将 FTP 服务器根目录下的所有文件显示在列表视图中。浮动对话框如图 5.19 所示。

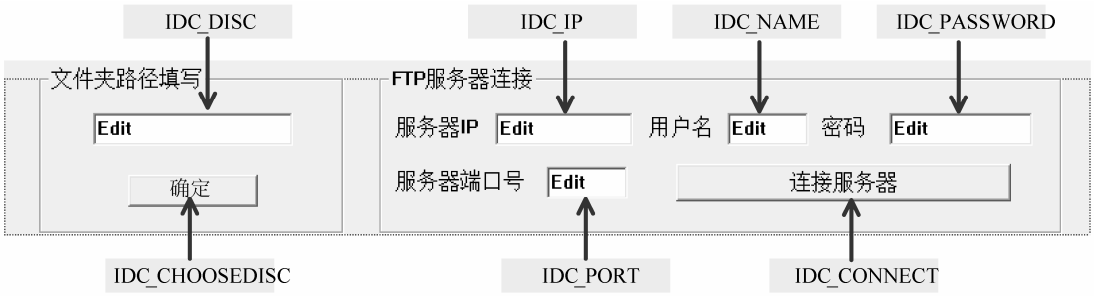

图 5.19　浮动对话框

在类 CMainFrame 中手动添加“连接服务器”按钮和“确定”按钮的消息响应函数 OnConnect()和 OnChooseDisc()，步骤如下所述。

（1）在类 CMainFrame 头文件中添加文件包含指令，用来支持 WinInet 类，如下：

```
#include < afxinet.h >          //为了使用 CFtpConnect CInternetSession
```

在类中添加成员变量和成员函数，如下：

```
01  class CMainFrame : public CFrameWnd
02  {
03  ...
04  // Attributes
05  public:
06      CFtpConnection      *m_pFtpConnection;
07      CInternetSession    *m_pInetSession;
08      CString             m_curPath;          //服务器根目录
09      CString             strDisc;            //本地文件路径--由用户输入
10
11  ...
12
13  // Generated message map functions
14  protected:
15      //{{AFX_MSG(CMainFrame)
```

```
16      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
17      ...
18      //}}AFX_MSG
19
20      //add new
21      afx_msg void OnConnect();
22      afx_msg void OnChooseDisc();
23      DECLARE_MESSAGE_MAP()
24  };
```

（2）在类 CMainFrame 的实现文件中添加文件包含指令，用来支持 3 个分割窗口视图，如下：

```
#include "MsgShow.h"        //3 个窗体的头文件
#include "FileTree.h"
#include "FtpClientView.h"
```

添加消息映射，即控件 ID 与处理事件函数建立联系，如下：

```
01  BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
02      //{{AFX_MSG_MAP(CMainFrame)
03      ...
04      ON_WM_CREATE()
05      //}}AFX_MSG_MAP
06
07      //add new
08      ON_BN_CLICKED(IDC_CONNECT, OnConnect)
09      ON_BN_CLICKED(IDC_CHOOSEDISC,OnChooseDisc)
10  END_MESSAGE_MAP()
```

添加了两个按钮单击事件的消息映射。类 CMainFrame 的构造函数如下：

```
01  CMainFrame::CMainFrame()
02  {
03      // TODO: add member initialization code here
04
05      m_bConnect = false;
06      m_pInetSession = new CInternetSession;
07      m_curPath = "\\";
08  }
```

可以看到，构造函数只是初始化了一些成员变量。

1．连接FTP服务器

编写 “连接服务器”按钮的消息响应函数 OnConnect()，如下：

```
01  void CMainFrame::OnConnect()
02  {
03      //获取视图窗口的指针
04      CFtpClientView *pView =
05                  (CFtpClientView *)(m_splitter2.GetPane(0,1));
06      CMsgShow *pEdit = (CMsgShow *)(m_splitter1.GetPane(0,0));
07
08      //判断此时与服务器的连接状态
09      if(!m_bConnect)
10      {
11          //获取浮动对话框上填写的信息
12          CString strHost,strName,strPass;
13
```

```
14          m_myDlg.GetDlgItemText(IDC_IP,strHost);
15          m_myDlg.GetDlgItemText(IDC_NAME,strName);
16          m_myDlg.GetDlgItemText(IDC_PASSWORD,strPass);
17
18          //处理异常的操作
19          try
20          {
21              //连接 FTP 服务器
22              m_pFtpConnection=
23                  m_pInetSession->GetFtpConnection(strHost,
24                                              strName,strPass);
25          }
26          catch(CInternetException *pEx)
27          {
28              TCHAR szError[256];
29              if(pEx->GetErrorMessage(szError,256))
30              {
31                  AfxMessageBox(szError);
32                  return;
33              }
34              else
35              {
36                  AfxMessageBox("There was an exception.");
37                  return;
38              }
39
40              pEx->Delete();
41              m_pFtpConnection=NULL;
42          }
43
44          //遍历服务器上的文件，调用自定义的函数 BrowseDir()
45          pView->BrowseDir(m_curPath,m_pFtpConnection);
46
47          //消息框显示信息，调用自定义函数 ShowMsg()
48          pEdit->ShowMsg("连接 ftp 服务器...");
49
50          //改变按钮的文字
51          m_myDlg.GetDlgItem(IDC_CONNECT)->SetWindowText("断开连接");
52          m_bConnect = true;
53
54          //禁用控件
55          m_myDlg.GetDlgItem(IDC_IP)->EnableWindow(false);
56          m_myDlg.GetDlgItem(IDC_NAME)->EnableWindow(false);
57          m_myDlg.GetDlgItem(IDC_PASSWORD)->EnableWindow(false);
58          m_myDlg.GetDlgItem(IDC_PORT)->EnableWindow(false);
59      }
60      else    //断开连接
61      {
62          ...
63      }
64  }
```

响应函数 OnConnect()功能的实现步骤：获取浮动对话框上由用户填写的登录信息，保存在 3 个字符串变量中，它们是 strHost、strName 和 strPass，调用类 CInternetSession 的成员函数 GetFtpConnection()连接 FTP 服务器，代码中对异常的情况做了一些处理，代码包含在 try 和 catch 的语句块中。

为类 CFtpClientView 添加成员函数 BrowseDir()，用来遍历服务器根目录下的所有文

件，代码编写如下：

```
01  void CFtpClientView::BrowseDir(CString strDir,CFtpConnection *ftpCon)
02  {
03      CFtpFileFind Ffind(ftpCon);
04      int col = 0;
05
06      BOOL IsTrue = Ffind.FindFile(strDir + "*");
07
08      while(IsTrue)
09      {
10          IsTrue = Ffind.FindNextFile();
11
12          if( !Ffind.IsDots() && !Ffind.IsDirectory() )
13          {
14              filelist->InsertItem(col,
15                  (LPCTSTR)Ffind.GetFileName(),0);
16              col++;
17          }
18      }
19  }
```

函数 BrowseDir()使用了类 CFtpFileFind，用来辅助 FTP 服务器上网络文件的检索。主要调用了此类的 5 个成员函数：

- FindFile()函数用来查找 FTP 服务器上指定的文件。
- FindNextFile()函数继续对指定条件的文件进行检索，需要在 FindFile()函数之后使用。
- IsDots()函数用来确定找到的文件的文件名是否包含“.”或“..”，它们其实就是目录。
- IsDirectory()函数用来确定找到的文件是否为目录。
- GetFileName()函数用来获取找到的文件的文件名。

通过 while 循环将满足条件的文件全部插入到列表视图中。用类 CListCtrl 的成员函数 InsertItem()实现，原型如下：

```
int InsertItem(
  int      nItem,
  LPCTSTR  lpszItem,
  int      nImage
);
```

参数及其含义如下所述。

- nItem：要将列表项插入索引视图的位置。
- lpszItem：列表项标签文本的指针。
- nImage：列表项对应的图像索引。

类 CMsgShow 是我们自己新建的基于 CEditView 的类。为此类添加成员函数 ShowMsg()，用来将指定的字符串显示在信息显示窗格中，代码如下：

```
01  void CMsgShow::ShowMsg(CString strMsg)
02  {
03      CString strTemp;
04
05      m_editView->GetWindowText(strTemp);
06      if(strTemp.GetLength() != 0)
```

```
    {
        strTemp += "\r\n";
        strTemp += strMsg;
        m_editView->SetWindowText(strTemp);
        return;
    }
    else
    {
        m_editView->SetWindowText(strMsg);
    }
}
```

函数 ShowMsg()中的 m_editView 是类 CMsgShow 的成员变量，在类 CMsgShow 的构造函数中被初始化，代码如下：

```
CMsgShow::CMsgShow()
{
    m_editView = &GetEditCtrl();
}
```

构造函数调用了类 CEditView 的成员函数 GetEditCtrl()，用来获取指向编辑视图的指针，然后保存在成员变量 m_editView 中。

函数 ShowMsg()的功能实现过程是获取当前编辑视图窗格的文本内容，依据之前的内容再添加新的文本信息。函数 ShowMsg()还使用到类 CEditView 继承自类 CWnd 的两个成员函数：

- GetWindowText()函数用来获取窗口的文本，并保存在传入的参数字符串中。
- SetWindowText()函数用来将参数字符串显示在窗口中。

用于连接服务器的函数 OnConnect()，在连接到服务器以后会改变自身按钮的文本为“断开连接”，并将浮动对话框上的文本框设置为禁用。

2．断开连接

当程序与 FTP 服务器处于连接状态，再次单击“断开连接”按钮时，将关闭与服务器的连接，如下：

```
void CMainFrame::OnConnect()
{
    ...

    //判断此时与服务器的连接状态
    if(!m_bConnect)
    {
        ...
    }
    else    //断开连接
    {
        //删除窗口中的所有项，调用自定义函数 deleteItem()
        pView->deleteItem();

        //关闭连接
        if(m_pFtpConnection!=NULL)
        {
            m_pFtpConnection->Close();
            delete m_pFtpConnection;
            m_pFtpConnection = NULL;
```

```
21          }
22          m_bConnect = false;
23
24          //用信息框记录用户操作的信息
25          pEdit->ShowMsg("断开与服务器的连接");
26
27          //改变按钮的文字
28          m_myDlg.GetDlgItem(IDC_CONNECT)->SetWindowText("连接服务器");
29
30          //设置控件可用
31          m_myDlg.GetDlgItem(IDC_IP)->EnableWindow(true);
32          m_myDlg.GetDlgItem(IDC_NAME)->EnableWindow(true);
33          m_myDlg.GetDlgItem(IDC_PASSWORD)->EnableWindow(true);
34          m_myDlg.GetDlgItem(IDC_PORT)->EnableWindow(true);
35      }
36  }
```

函数 OnConnect()调用了列表视图的成员函数 deleteItem()，用来清空列表视图的所有列表项，函数 deleteItem()的实现如下：

```
01  void CFtpClientView::deleteItem()
02  {
03      filelist->DeleteAllItems();
04  }
```

这是一个很简单的函数封装，只调用了类 CListCtrl 的成员函数 DeleteAllItems()，甚至连参数都不需要。

函数 OnConnect()的后续操作是调用类 CFtpConnection 的成员函数 Close()关闭与 FTP 的连接，在信息显示框中显示文本信息“断开与服务器的连接”，改变“断开连接”按钮的文本为“连接服务器”，最后将浮动对话框中被禁用的文本框设置为可用。

5.3.3　遍历本地文件夹资源

用户需要手动填写“文件夹路径”文本框，在单击“确定”按钮时，树结构视图中将会显示出该文件夹下的所有文件资源。

为浮动对话框的 “确定”按钮添加消息响应函数 OnChooseDisc()，代码编写如下：

```
01  void CMainFrame::OnChooseDisc()
02  {
03      //获取文本框中的信息
04      m_myDlg.GetDlgItemText(IDC_DISC,strDisc);
05
06      //调用 CFileTree 的函数显示文件列表
07      CFileTree *pView = (CFileTree *)(m_splitter2.GetPane(0,0));
08
09      //先清除树视图的所有项
10      pView->deleteItem();
11
12      //遍历文件
13      pView->BrowseDir(strDisc,NULL);
14
15      //消息框显示信息
16      CMsgShow *pEdit = (CMsgShow *)(m_splitter1.GetPane(0,0));
17      pEdit->ShowMsg("更改本地文件夹路径...");
```

```
18  }
```

函数 OnChooseDisc()中，调用到类 CFileTree 的成员函数 deleteItem()，用来清除树结构视图中的所有项，实现如下：

```
01  void CFileTree::deleteItem()
02  {
03      tree->DeleteAllItems();
04  }
```

这也是个简单的函数封装，只调用了类 CTreeCtrl 的成员函数 DeleteAllItems()，甚至连参数都不需要。

类 CFileTree 的成员函数 BrowseDir()，用来实现遍历文件夹中的文件资源，并按结构插入到树中，实现代码如下：

```
01  void CFileTree::BrowseDir(CString strDir,HTREEITEM hParent)
02  {
03      CFileFind   fFind;
04      CString     strFileName;
05
06      if(strDir.Right(2) != "\\")     //important!
07      {
08          strDir += "\\*.*";
09      }
10
11      BOOL IsTrue = fFind.FindFile(strDir);
12
13      while(IsTrue)
14      {
15          IsTrue = fFind.FindNextFile();
16
17          if( fFind.IsDirectory() && !fFind.IsDots() )
18          {
19              CString     strPath = fFind.GetFilePath();
20              strFileName = fFind.GetFileName();
21
22              HTREEITEM   hChild = tree->InsertItem(strFileName,
23                                                    0,0,hParent);
24
25              BrowseDir(strPath,hChild);
26          }
27          else  if( !fFind.IsDirectory() )
28          {
29              strFileName = fFind.GetFileName();
30              tree->InsertItem(strFileName,1,1,hParent);
31          }
32      }
33
34      fFind.Close();
35  }
```

函数 BrowseDir()的功能实现过程是构造遍历文件的字符串，调用类 CFileFind 的 6 个成员函数：

- FindFile()函数用来查找本地指定路径下的文件资源。
- FindNextFile()函数继续对本地指定路径下的文件进行检索，需要在 FindFile()函数之后使用。
- IsDots()函数用来确定找到的文件的文件名是否包含“.”或“..”，它们其实就是

目录。

- IsDirectory()函数用来确定找到的文件是否是目录。
- GetFileName()函数用来获取找到的文件的文件名。
- GetFilePath()函数用来获取指定文件的完整路径，或者叫做绝对路径。

类 CTreeCtrl 的成员函数 InsertItem()用来向树结构视图中添加项，函数原型如下：

```
HTREEITEM InsertItem(
   LPCTSTR      lpszItem,
   int          nImage,
   int          nSelectedImage,
   HTREEITEM    hParent = TVI_ROOT,
   HTREEITEM    hInsertAfter = TVI_LAST
);
```

参数及其含义如下所述。

- lpszItem：插入项的文本指针。
- nImage：插入项未被选中时的图标索引。
- nSelectedImage：插入项被选中时的图标索引。
- nParent：父节点项的句柄，默认为根节点的句柄。
- nInsertAfter：新插入项的插入位置，默认插入到最后。

树结构视图 CFileTree 关联的图像列表是在函数 OnInitialUpdate()中确定的，代码如下：

```
01  void CFileTree::OnInitialUpdate()
02  {
03      CTreeView::OnInitialUpdate();
04      // TODO: Add your specialized code here
05      // and/or call the base class
06
07      //获取图标句柄
08      HICON hicon1 = AfxGetApp()->LoadIcon(IDI_ICON1);
09      HICON hicon2 = AfxGetApp()->LoadIcon(IDI_ICON2);
10
11      //创建图标列表
12      m_lpImagelist.Create(16,16,ILC_COLOR16,2,2);
13      m_lpImagelist.Add(hicon1);
14      m_lpImagelist.Add(hicon2);
15
16      //关联图像列表
17      tree->SetImageList(&m_lpImagelist,TVSIL_NORMAL);
18  }
```

我们当然得先在资源编辑器中插入两个图标资源，一个用来表示文件，另一个用来表示文件，如图 5.20 所示。

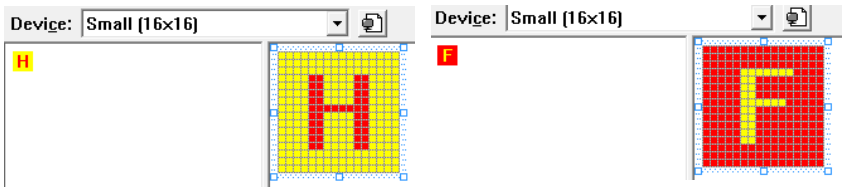

图 5.20　自己制作的小图标

类 CWinApp 的成员函数 LoadIcon()，将加载指定 ID 的图标资源，返回图标的句柄。创建图标列表调用到类 CImageList 的成员函数 Create()，函数原型如下：

```
BOOL Create(
   int   cx,
   int   cy,
   UINT nFlags,
   int   nInitial,
   int   nGrow
);
```

参数及其含义如下所述。

- cx、cy：图像的长、宽值，以像素为单位。
- nFlags：指定创建的图像列表的类型。
- nInitial：图像列表起始包含的图像数。
- nGrow：当系统需要改变列表，为新图像准备空间时，图像列表可生成的图像数。此参数替代改变的图像列表所能包含的新图像数。

调用类 CImageList 的成员函数 Add()，将指定的图标句柄加入到图像列表中，调用类 CTreeCtrl 的成员函数 SetImageList()，用来关联树结构视图与图像列表，函数原型如下：

```
CImageList* SetImageList(
   CImageList   *pImageList,
   int          nImageListType
);
```

参数及其含义如下所述

- pImageList：指向图像列表的指针。若为 NULL，树视图的所有图标将会被移除。
- nImageListType：被设置的图标列表的类型。TVSIL_NORMAL 表示设置的图像列表为树结构项，包含了选中和未被选中的图像。

另外，成功操作的图结构视图的显示效果，如图 5.21 所示。

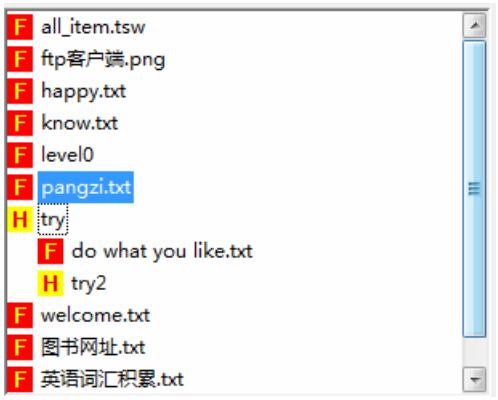

图 5.21　树结构视图的效果

要想改变显示效果，可以在类 CFileTree 中重载函数 PreCreateWindow()，用来改变树结构视图的样式，代码如下：

```
01  BOOL CFileTree::PreCreateWindow(CREATESTRUCT& cs)
02  {
```

```
03      // TODO: Add your specialized code here
04      // and/or call the base class
05
06      cs.style |= TVS_HASBUTTONS|TVS_HASLINES|
07                      TVS_LINESATROOT|TVS_EDITLABELS;
08
09      return CTreeView::PreCreateWindow(cs);
10  }
```

改变了样式以后显示的效果，如图 5.22 所示。

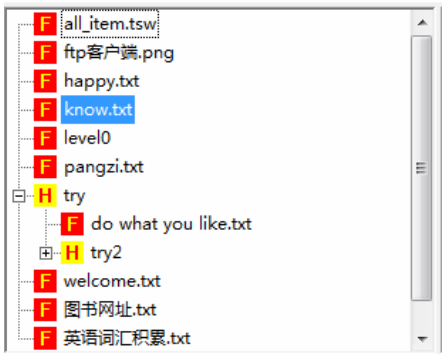

图 5.22　改变了样式后树结构视图的效果

5.3.4　拖动文件实现上传

文件上传到 FTP 服务器，只需要调用类 CFtpConnetion 的成员函数 PutFile()就可以了，但是需要准备此函数的参数：要上传的本地文件的路径在开始拖动文件时获得，函数 PutFile()的调用是在鼠标左键弹起时。

注意：搭建自己的 FTP 服务器，并且创建了登录的用户时，默认这个用户是没有上传文件的权限的，需要设置权限，如图 5.23 所示 Serv-U 的权限设置。

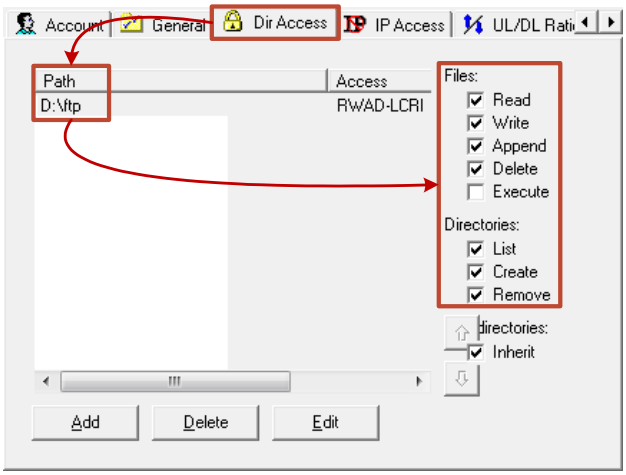

图 5.23　Serv-U 用户目录权限设置

在类 CFileTree 的成员函数 OnBegindrag()中添加如下代码：

```
void CFileTree::OnBegindrag(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
    // TODO: Add your control notification handler code here

    //树的叶子节点
    m_hItemDragS = pNMTreeView->itemNew.hItem;

    //获取选择的文件名
    m_filename = tree->GetItemText(m_hItemDragS);

    //有父节点的话，需要获取父节点的文本
    HTREEITEM   m_pParentCode = tree->GetParentItem(m_hItemDragS);
    m_fileLname = m_filename;
    while(m_pParentCode)
    {
        CString strTemp = m_fileLname;
        m_fileLname = tree->GetItemText(m_pParentCode);
        m_fileLname += "\\";
        m_fileLname += strTemp;
        m_pParentCode = tree->GetParentItem(m_pParentCode);
    }

    ...                     //省略

    *pResult = 0;
}
```

函数 OnBegindrag()添加的功能是获取拖动文件的文件名，若它有父节点的话还需要获取父节点的文件名。m_filename 和 m_fileLname 是定义在类 CFileTree 中的成员变量，用来保存文件名和文件的部分路径，完整路径会在鼠标左键弹起时构建。

添加鼠标左键弹起事件的响应函数 OnLButtonUp()，代码如下：

```
void CFileTree::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    CMainFrame* mFrm = (CMainFrame*)AfxGetMainWnd();
    CFtpClientView *pEView =
            (CFtpClientView *)(mFrm->m_splitter2.GetPane(0,1));
    CMsgShow    *pShowMsg =
            (CMsgShow   *)(mFrm->m_splitter1.GetPane(0,0));

    if(m_bDragging)
    {
        m_bDragging = false;

        ...

        //若文件拖动到了服务器视图的区域内
        if( listRgn.PtInRegion(pt) )
        {
            //构建路径
            CString sourFile = mFrm->strDisc + "\\" + m_fileLname;
            CString destFile = "//" + m_filename;

```

```
24              BOOL IsTrue =
25                  mFrm->m_pFtpConnection->PutFile(sourFile, destFile);
26
27              //有错时的提示
28              if(!IsTrue)
29              {
30                  DWORD errorNum = GetLastError();
31                  CString strNum;
32                  strNum.Format("发生错误，错误号：%d",errorNum);
33                  AfxMessageBox(strNum);
34                  return;
35              }
36
37              //填写消息提示窗口
38              CString strMsg = "成功添加文件：";
39              strMsg += m_filename;
40              pShowMsg->ShowMsg(strMsg);
41          }
42      }
43
44      CTreeView::OnLButtonUp(nFlags, point);
45  }
```

函数 OnLButtonUp()添加的功能是构建本地文件的路径，构建上传到 FTP 服务器上的路径，调用函数 PutFile()上传文件，发送错误时会有错误提示信息，成功上传文件的话会在信息显示窗口显示操作信息。

5.3.5　拖动文件实现下载

将 FTP 服务器上的文件下载到本地，只需要调用类 CFtpConnetion 的成员函数 GetFile()就可以了，但是需要准备此函数的参数：要下载到本地的文件路径，函数 GetFile()的调用是在鼠标左键弹起时。

在类 CFtpClientView 的成员函数 OnBegindrag()中添加的代码如下：

```
01  void CFtpClientView::OnBegindrag(NMHDR* pNMHDR, LRESULT* pResult)
02  {
03      NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
04      // TODO: Add your control notification handler code here
05
06      ...
07
08      //获取选择项的文件路径
09      POSITION pos = filelist->GetFirstSelectedItemPosition();
10      int nItem = filelist->GetNextSelectedItem(pos);
11      m_filename = filelist->GetItemText(nItem,0);
12      m_fileCopy = m_curPath + m_filename;
13
14      *pResult = 0;
15  }
```

函数 OnBegindrag()添加的功能是获取拖动文件的文件名，构建 FTP 服务器上该文件的路径文本。

在类 CFtpClientView 的成员函数 OnLButtonUp()中添加的代码如下：

```
01  void CFtpClientView::OnLButtonUp(UINT nFlags, CPoint point)
```

```
{
    // TODO: Add your message handler code here and/or call default

    CMainFrame* mFrm = (CMainFrame*)AfxGetMainWnd();
    CFileTree *pEView = (CFileTree *)(mFrm->m_splitter2.GetPane(0,0));

    if( m_isDragging )
    {
        m_isDragging  =  false;

        ...

        if( treeRgn.PtInRegion(pt))     //释放点的位置
        {
            CString strTemp = "\\";
            //下载文件
            mFrm->m_pFtpConnection->GetFile(m_fileCopy,
                mFrm->strDisc + strTemp + m_filename);
        }
    }

    CListView::OnLButtonUp(nFlags, point);
}
```

函数 OnLButtonUp()添加的功能是构建 FTP 服务器上文件的路径，构建本地文件存放位置的路径，调用函数 GetFile()下载 FTP 服务器上的文件。

5.3.6　多次修改的头文件

经过了多次成员函数和成员变量的添加，我们来看一下最终类的头文件中我们添加了些什么。首先是列表结构视图，头文件部分的代码如下：

```
// FtpClientView.h : interface of the CFtpClientView class
//
/////////////////////////////////////////////////////////////////////////
...

class CFtpClientDoc;
class CFtpConnection;

class CFtpClientView : public CListView
{
    ...

// Attributes
public:
    HIMAGELIST  sys_large_icon;
    CImageList  m_IconList;     //记录的是与列表控件关联的图像列表
    CListCtrl   *filelist;      //自身窗体的指针
    CImageList  *m_pImageList;  //图像列表
    bool        m_isDragging;   //判断拖动状态
    CString     m_fileCopy;     //FTP 服务器上的文件路径
    CString     m_filename;     //选中项的文件名
    CString     m_curPath;      //服务器的根目录

// Operations
public:
```

```
    //检索服务器根目录下所有的文件（不包括文件夹）
    VOID BrowseDir(CString strDir,CFtpConnection *ftpCon);
    //获取列表视图矩形的大小
    void GetCtrlRect(CRect *rt);
    //删除列表视图的所有项
    VOID deleteItem();

...
};
```

树结构视图，部分头文件如下：

```
// FileTree.h : header file
//
/////////////////////////////////////////////////////////////
 CFileTree view

class CFileTree : public CTreeView
{
...
// Attributes
public:
    CTreeCtrl       *tree;              //指向 treeview 本身
    CImageList      m_lpImagelist;      //存放自定义的文件和文件夹图标
    CImageList*     m_pDragImage;       //拖动时显示的图像列表
    HTREEITEM       m_hItemDragS;       //被拖动的标签项
    BOOL            m_bDragging;        //鼠标是否处于拖动状态
    CString         m_filename;         //被拖动的文件名
    CString         m_fileLname;        //保存父节点+子节点文件名，
                                        //用来构造路径
...
// Operations
public:
    //遍历文件夹--由用户输入路径
    void BrowseDir(CString strDir,HTREEITEM hParent);
    void deleteItem();                      //删除树视图中的所有项
    void GetCtrlRect(CRect *rt);            //获取树视图窗口矩形大小
...
};
```

主框架 CMainFrame，部分头文件如下：

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////
...
#include < afxinet.h >     //为了使用 CFtpConnect CInternetSession

class CMainFrame : public CFrameWnd
{
...
// Attributes
public:
    CSplitterWnd    m_splitter1;
    CSplitterWnd    m_splitter2;
    BOOL            m_bConnect;         //与 FTP 服务器的连接状态
    CFtpConnection  *m_pFtpConnection;
    CInternetSession *m_pInetSession;
```

```
17      CString         m_curPath;          //服务器根目录
18      CString         strDisc;            //本地文件路径--由用户输入
19      CDialogBar      m_myDlg;            //浮动对话框
20  ...
21  protected:  // control bar embedded members
22  //  CStatusBar  m_wndStatusBar;         //注释掉状态栏
23  //  CToolBar    m_wndToolBar;           //注释掉工具栏
24
25  // Generated message map functions
26  protected:
27      //{{AFX_MSG(CMainFrame)
28          ...
29      //}}AFX_MSG
30
31      afx_msg void OnConnect();           //按钮“连接服务器”响应函数
32      afx_msg void OnChooseDisc();        //按钮“确定”响应函数
33      DECLARE_MESSAGE_MAP()
34  };
```

5.4　小　　结

本章实现了一个较第 4 章更为华丽的 FTP 客户端，也是网上最普遍的 FTP 客户端。但是本程序中尚有许多功能缺失，例如，本客户端只会遍历指定的本地文件夹下的所有文件资源，而非所有本地计算机上的资源；两个视图显示的图标比较单一等等。当然，有兴趣完善本章实例的读者可以自由修改。