

# .NET中的Drag and Drop操作（一）

转载

jumtre



于 2013-12-11 11:14:06 发布



阅读量2.2k



收藏 1



点赞数

分类专栏:

C#

WPF

WinForm



C# 同时被 3 个专栏收录 ▾

1 订阅

65 篇文章

订阅专栏

## 一 C#中Drag and Drop的用法

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        listView1.View = View.List;
        listView2.View = View.List;
    }

    private void listView1_DragEnter(object sender, DragEventArgs e)
    {
        if (e.Data.GetDataPresent(DataFormats.FileDrop))
            e.Effect = DragDropEffects.Copy;
    }

    private void listView1_DragLeave(object sender, EventArgs e)
    {
    }
}
```

```

    }

private void listView1_DragOver(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
        e.Effect = DragDropEffects.Copy;
}

private void listView1_DragDrop(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        String[] files = (String[])e.Data.GetData(DataFormats.FileDrop);
        foreach (String s in files)
        {
            ListViewItem item = new ListViewItem(s);
            listView1.Items.Add(item);
        }
    }
}

private void listView1_ItemDrag(object sender, ItemDragEventArgs e)
{
    ListViewItem[] itemTo = new ListViewItem[((ListView)sender).SelectedItems.Count];
    for (int i = 0; i < itemTo.Length; i++)
    {
        itemTo[i] = ((ListView)sender).SelectedItems[i];
    }

    //System.Runtime.InteropServices.ComTypes.IDataObject obj;
    //System.Runtime.InteropServices.ComTypes.FORMATETC formatEtc;
    //System.Runtime.InteropServices.ComTypes.STGMEDIUM stgMedium;

```

```

//formatEtc = new System.Runtime.InteropServices.ComTypes.FORMATETC()
//{
//    cfFormat = 15,
//    dwAspect = System.Runtime.InteropServices.ComTypes.DVASPECT.DVASPECT_CONTENT,
//    lindex = 1,
//    ptd = IntPtr.Zero,
//    tymed = System.Runtime.InteropServices.ComTypes.TYMED.TYMED_HGLOBAL
//};

//stgMedium = new System.Runtime.InteropServices.ComTypes.STGMEDIUM()
//{
//    pUnkForRelease = null,
//    tymed = System.Runtime.InteropServices.ComTypes.TYMED.TYMED_HGLOBAL,
//    unionmember = IntPtr.Zero
//};
//obj.SetData(formatEtc, stgMedium, true);
((ListView)(sender)).DoDragDrop(itemTo, DragDropEffects.Copy);
}

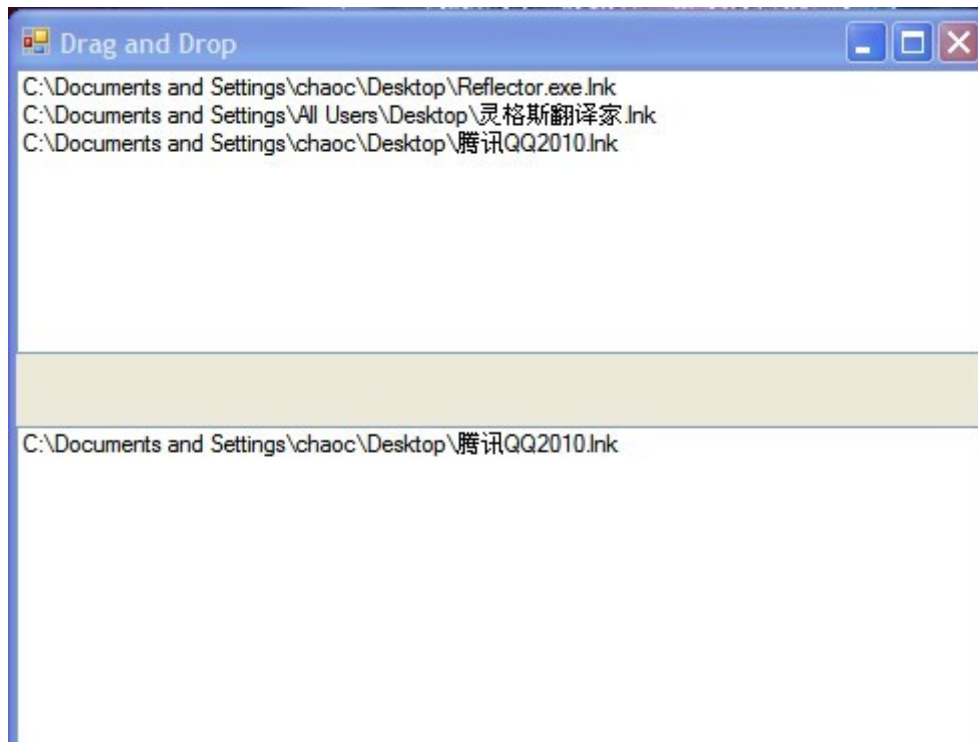
private void listView2_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(ListViewItem)))
        e.Effect = DragDropEffects.Copy;
}

private void listView2_DragDrop(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(ListViewItem)))
    {
        ListViewItem[] files = (ListViewItem[])e.Data.GetData(typeof(ListViewItem));
        foreach (ListViewItem s in files)
        {

```

<pre>         ListViewItem item = s.Clone() as ListViewItem;     } } </pre>	<pre> listView2.Items.Add(item); </pre>
---	---

上面是一段使用两个ListView显示信息的代码，其中ListView1接受我们拖动一个文件到他的窗体上，并显示文件路径；而ListView2是接受我们从ListView1中拖动文件路径，显示在自己的view中。程序的运行结果入下。



整个程序很简单，实现起来也没什么难度。下面就简单介绍下实现的步骤，如何使用不是本文的重点，但是必须知道和DragDrop相关的方法和属性。

下面是控件处理拖拽的步骤方法：

- 1：将能接受拖放的控件的AllowDrop属性设置为True，可以在设计和运行时设置；
- 2：为能接受拖放的控件添加4个事件分别是DragEnter，DragOver，DragLeave，DragDrop，ItemDrag；
- 3：实现DragEnter，DragOver，DragLeave事件对应的方法，当用鼠标拖拽一个对象到控件的窗口时，首先触发DragEnter，然后是DragOver，当离开窗体时触发DragLeave。主要操作是设置判断对象是否是要接受的类型，以及鼠标的样式；
- 4：实现DragDrop事件对应的方法；当用户拖拽对象到控件上，并释放时触发，这时主要是接受对象并操作显示；
- 5：实现ItemDrag的事件对应的方法，调用DoDragDrop方法，传递要拖拽的数据对象；

按上面的步骤，我们就能简单的实现一个控件接受拖拽进来的对象，并且拖拽一个对象到其他地方。具体可以参见上面的代码。当然在接受拖拽的几个事件中，我们还可以进行更多的一些控制，比如按下Ctrl或Alt时进行不同的操作。而对TreeView这用，根据拖拽的位置，放入不同节点下的操作，可以通过DragEventArgs的坐标位置来判断，这里就不深入介绍了，因为比较简单。

## 二 分析实现方法前的疑问

拖拽本身并不神秘，它的本质实际是一个数据交换的过程。控件接受从其他地方来的数据，并进行处理。数据交换有多种方法，Windows中剪贴板可能就是用的最多，但最不被注意的一种方法。











平时我们复制一段文字，一张图片，一个目录，甚至是Excel中的一段表格，然后粘贴到其他分区或者是程序中。有没有发现剪贴版是多么的神奇，什么格式的数据他都能复制。我们常用的资源管理器是最好的一个列子。你可以把任何格式的文件拖拽到另一个目录。不要理所当然的认为这只是Move或Copy，我们讨论的是Drag和Drop，为什么任何格式文件她都能正确，甚至是我们自己创建的格式的文件？

## 三 C#中Drag和Drop的实现浅析

这里只能说是浅析，因为很多东西都是涉及到底层的操作，而这里只介绍.NET中能看的到的地方。

### 1：关于AllowDrop属性

在对一个控件进行拖拽编程时，我们必须把AllowDrop属性设置为True，这样才能在拖拽时触发上面介绍过的4个事件。我们先来看看这个属性。

Search		
AllowDrop		
Member	Declaring Type	Assembly
 AllowDrop	System.Windows.Forms.Control	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.ToolStrip	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.ToolStripItem	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.GroupBox	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.PictureBox	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.ProgressBar	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.RichTextBox	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.Splitter	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.ToolStripPanel	System.Windows.Forms, Version=4.0.0.0, C...
 AllowDrop	System.Windows.Forms.WebBrowserBase	System.Windows.Forms, Version=4.0.0.0, C...

在Reflector中搜索AllowDrop属性，发现了10个类实现了这个属性。其中Control是所有控件的基类，而其他9个都是继承与Control的，所以AllowDrop应该是一个虚方法。我们具体看看Control对AllowDrop属性的实现，因为其他大部分控件都没有重写此方法。

```
[DefaultValue(false), SRCategory("CatBehavior"), SRDescription("ControlAllowDropDescr")]
public virtual bool AllowDrop
{
    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]
    get
    {
        return this.GetState(0x40);
    }
    set
    {

```



get方法是简单的返回状态，而set中有些不太清楚的代码不用管，看看重要的代码，IsHandleCreated是检查窗体句柄，然后就是 **SetAcceptDrops** 方法，来设置属性。

```

internal void SetAcceptDrops(bool accept) | {
    if ((accept != this.GetState(0x80)) && this.IsHandleCreated)
    {
        try
        {
            if (Application.OleRequired() != ApartmentState.STA)
            {
                throw new ThreadStateException(SR.GetString("ThreadMustBeSTA"));
            }
        }
    }
}

```

);

上面的代码中可以看到2个Native方法，**RegisterDragDrop**和**RevokeDragDrop**，从名字就能看出，我们设置AllowDrop属性时，就是在注册或移除注册。一下是两个方法的原型，以及MSDN对方法的解释

```

//Registers the specified window as one that can be the target of an OLE drag-and-drop operation and specifies the IDropTarget

[DllImport("ole32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern int RegisterDragDrop(HandleRef hwnd, IOleDropTarget target);

//Revokes the registration of the specified application window as a potential target for OLE drag-and-drop operations.

[DllImport("ole32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern int RevokeDragDrop(HandleRef hwnd);

```

查看调用这2个方法时传递的参数 **【HandleRef(this, this.Handle), DropTarget(this)】**

```
[StructLayout(LayoutKind.Sequential), ComVisible(true)]
public struct HandleRef
{
    internal object m_wrapper;
    internal IntPtr m_handle;
    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]
    public HandleRef(object wrapper, IntPtr handle);
    public object Wrapper { [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image bound
    public IntPtr Handle { [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image bounda
    public static explicit operator IntPtr(HandleRef value);
    public static IntPtr ToIntPtr(HandleRef value);
}
```

第一个参数传递的是一个HandleRef，它包装一个托管对象，该对象保存使用平台invoke（调用）传递给非托管代码的资源句柄。如果使用平台调用来调用托管对象，而且该对象在该调用之后不在其他地方引用，则垃圾回收器有可能终结该托管对象。此操作将释放资源并使句柄无效，从而导致平台 invoke 调用失败。使用 **HandleRef** 包装句柄，可保证该托管对象在平台 invoke 调用完成前不被垃圾回收。

第二个参数是实现了Native接口的DropTarget对象，这个参数后面具体介绍。

传递了这2个对象后，变完成了可接受拖拽窗口的注册，而RevokeDragDrop方法拖过接受一个注册的句柄引用参数，取消注册。可见一个小小的AllowDrop属性已经做了这么多的事情。但是为什么要注册呢？

## 2 控件实现IDropTarget接口

既然上面注册Drop时，传递了控件的句柄和控件对象，那么拖拽一定和控件有密不可分的关系（废话）。我们还是从Control类下手。观察Control控件，我们发现它定义了4个事件，并且实现了IDropTarget接口。

```
public interface IDropTarget
```



```

{
    // Methods
    void OnDragDrop(DragEventArgs e);
    void OnDragEnter(DragEventArgs e);
    void OnDragLeave(EventArgs e);
    void OnDragOver(DragEventArgs e);
}

public class Control : Component, IDropTarget
{
    //event
    public event DragEventHandler DragDrop;
    [SRDescription("ControlOnDragEnterDescr"), SRCategory("CatDragDrop")]
    public event DragEventHandler DragEnter;
    [SRCategory("CatDragDrop"), SRDescription("ControlOnDragLeaveDescr")]
    public event EventHandler DragLeave;
    [SRCategory("CatDragDrop"), SRDescription("ControlOnDragOverDescr")]
    public event DragEventHandler DragOver;
    [SRCategory("CatPropertyChanged"), SRDescription("ControlOnEnabledChangedDescr")]

    //method
    [EditorBrowsable(EditorBrowsableState.Advanced)]
    protected virtual void OnDragDrop(DragEventArgs drgevent);
    [EditorBrowsable(EditorBrowsableState.Advanced)]
    protected virtual void OnDragEnter(DragEventArgs drgevent);
    [EditorBrowsable(EditorBrowsableState.Advanced)]
    protected virtual void OnDragLeave(EventArgs e);
    [EditorBrowsable(EditorBrowsableState.Advanced)]
    protected virtual void OnDragOver(DragEventArgs drgevent);

    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]
    void IDropTarget.OnDragDrop(DragEventArgs drgEven
    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]

```

```

    void IDropTarget.OnDragEnter(DragEventArgs drgEvent);

    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]
    void IDropTarget.OnDragLeave(EventArgs e);

    [TargetedPatchingOptOut("Performance critical to inline this type of method across NGen image boundaries")]
    void IDropTarget.OnDragOver(DragEventArgs drgEvent);
}

```

看到上面的代码，并没有什么特殊的地方，显示的实现了IDropTarget的接口，而内部实现也相当简单，就是调用了类中相对用的方法。我们给控件注册了Drop，这样当这些事件发生时，就会调用IDropTarget接口实现的方法，进而调用我们编写的事件处理方法，实现拖拽。但是这个过程又是如何连接起来的呢？

### 3 控件注册和调用的实现

首先我们还是回到前面所提到的注册，注册的时候我们是调用了RegisterDragDrop方法。RegisterDragDrop的作用是告诉系统:某个窗口（hwnd参数指定）可以接受拖放，接管拖放的接口是target;指向一个IOleDropTarget接口。

```

[ComImport, Guid("00000122-0000-0000-C000-000000000046"), InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
public interface IOleDropTarget
{
    [PreserveSig]
    int OleDragEnter([In, MarshalAs(UnmanagedType.Interface)] object pDataObj, [In, MarshalAs(UnmanagedType.U4)] int grfKeyStat
    [PreserveSig]
    int OleDragOver([In, MarshalAs(UnmanagedType.U4)] int grfKeyState, [In, MarshalAs(UnmanagedType.U8)] long pt, [In, Out] ref
    [PreserveSig]
    int OleDragLeave();
    [PreserveSig]
    int OleDrop([In, MarshalAs(UnmanagedType.Interface)] object pDataObj, [In, MarshalAs(UnmanagedType.U4)] int grfKeyState, [I
}

```

在MSDN中我们发现搜索不到IOleDropTarget接口，搜索到的是IDropTarget（COM），查看接口的GUID其实是一样的，这里应该就是为了区分.NET中的IDropTarget接口，它是一个COM接口。MSDN对此接口的描述：[http://msdn.microsoft.com/en-us/library/ms679679\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms679679(VS.85).aspx)

我们的控件要实现拖拽，必须实现 IOleDropTarget中的方法；我们在要使用时，不需要去调用DropEnter等方法，而是调用DoDragDrop，这个方法会在相应操作时调用对应的方法。

我们的Control实现了IDropTarget（.NET）接口，在注册时，我们第2个参数传递的是IDropTarget（COM）接口的DropTarget对象；这2个接口是如何关联起来的。构造函数中使用的this，也就是控件自身的对象。在DropTarger对象中，我们看到实现的接口方法

```
internal class DropTarget : UnsafeNativeMethods.IOleDropTarget
{
    // Fields
    private IDataObject lastDataObject;
    private DragDropEffects lastEffect;
    private IDropTarget owner;

    // Methods
    public DropTarget(IDropTarget owner)
    {
```

)

我们发现，这个DropTarget对象内部在实现接口方法时，使用了owner.Method()的方式来调用，而这个owner就是构造函数中传入的控件对象。也就是说，注册是，我们提供了一个IDropTarget（COM）接口的对象，这个对象内部实际是调用我们控件实现的IDropTarget（.NET）接口的方法。

系统在调用时是这样的：当拖拽的对象落一个窗体中时，检查是窗体的句柄是否注册，如果没有注册不能进行拖拽操作。如果已经注册，则根据注册是传入的第二个参数，根据鼠标的操作，调用IDropTarget (COM) 接口的方法。在.NET中，控件没有直接实现此接口，而是调用了DropTarget对象中实现的接口方法，而此对象保存了注册时控件的一个引用，通过控件对象引用，调用了IDropTarget (.NET) 接口的方法。这样，我们的控件就间接的实现了IDropTarget (COM) 接口，具备了拖拽的功能。

#### 4 控件拖动功能的实现

前面比较详细的分析了一个控件如何接受拖拽对象的实现，接下来就是控件中数据拖拽的实现了。对应于IDropTarget接口，还存在一个IDropSource接口。同样，我们发现.NET中有一个IOleDropSource的接口来表示此接口。但是却没有自己的IDropSource接口。下面是MSDN关于此接口的介绍：[http://msdn.microsoft.com/en-us/library/ms690071\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms690071(VS.85).aspx)

我们仍旧通过Control来分析。我们注意到，Control控件不仅仅实现了IDropTarget接口，还是实现了一个ISupportOleDropSource的接口。通过观察发现，她应该就是那个IDropSource接口。因为接口方法是一样的。

```
internal interface ISupportOleDropSource
{
    // Methods
    // Gives visual feedback to an end user during a drag-and-drop operation.
    void OnGiveFeedback(GiveFeedbackEventArgs gfbevent);

    // Determines whether a drag-and-drop operation should continue.
    void OnQueryContinueDrag(QueryContinueDragEventArgs qcdevent);
}
```

这是一个没有公开的接口，所以MSDN上无法搜索到这个接口的信息。同样也存在一个联系这2个接口的对象DropSource。其中包含了一个ISupportOleDropSource peer 的对象，这里运行的方式和前面DropTarget完全一样。

```
internal class DropSource : UnsafeNativeMethods.IOleDropSource
{
    // Fields
```

```

private const int DragDropSCancel = 0x40101;
private const int DragDropSDrop = 0x40100;

private const int DragDropSUseDefaultCursors = 0x40102;
private ISupportOleDropSource peer;

// Methods
public DropSource(ISupportOleDropSource peer);
public int OleGiveFeedback(int dwEffect);
public int OleQueryContinueDrag(int fEscapePressed, int grfKeyState);
}

```

在最开始，我们实现拖拽时，并不需要实现这2个方法，这2个方法只是用来显示拖拽效果和指示能否进行拖拽的，我们并没有显示的用到。在我们最开始的列子，我们是通过一个ItemDrag事件，调用 **DoDragDrop**方法来进行Drag操作。我们还是先来看看这个方法吧。

## 5 DoDragDrop的实现分析

国际惯例，还是来看Control吧@\_@!

```

[UIPermission(SecurityAction.Demand, Clipboard=UIPermissionClipboard.OwnClipboard)]
public DragDropEffects DoDragDrop(object data, DragDropEffects allowedEffects)
{
    int[] finalEffect = new int[1];
    UnsafeNativeMethods.IOleDropSource dropSource = new DropSource(this);
    IDataObject dataObject = null;
    if (data is IDataObject)
    {
        dataObject = (IDataObject) data;
    }
}

```



此方法接受2个参数，第一个就是我们要传送的data，第二个参数是拖拽时的鼠标效果。

方法一开头就出现了我们熟悉的东西，新建了一个DropSource对象，并且把控件自身传递给了他，保存在peer字段中。接下来是对参数data的操作。这里涉及到了IDataObject这个接口，这里不详细介绍。主要总用就是把我们的数据包装成IDataObject接口对象进行传送，这样接送双反都能接受和发送任何对象。最后调用了一个Native方法DoDragDrop。

```
SafeNativeMethods.DoDragDrop(dataObject, dropSource, (int) allowedEffects, finalEffect);  
  
[DllImport("ole32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]  
public static extern int DoDragDrop(IDataObject dataObject, UnsafeNativeMethods.IOleDropSource dropSource, int allowedEffects,
```

简单的说说这4个参数：

dataObject -- 是要传递的数据，实现了IDataObject接口，就是我们的data参数；

dropSource -- 这是实现了IOleDropSource接口对象，就是我们.NET的DropSource对象，包装了control对象；

allowedEffects -- 是我在从控件拖拽时，要实现的效果。

finalEffect -- 是在拖拽中，对source操作的效果。

下面是MSDN对整个过程的完整描述，通过前面接受，我想不需要翻译也能大概明白了

If you are developing an application that can act as a data source for an OLE drag-and-drop operation, you must call **DoDragDrop** when you detect that the user has started an OLE drag-and-drop operation.

The **DoDragDrop** function enters a loop in which it calls various methods in the **IDropSource** and **IDropTarget** interfaces. (For a successful drag-and-drop operation, the application acting as the data source must also implement **IDropSource**, while the target application must implement **IDropTarget**.)

1. The **DoDragDrop** function determines the window under the current cursor location. It then checks to see if this window is a valid drop target.
2. If the window is a valid drop target, **DoDragDrop** calls **IDropTarget::DragEnter**. This method supplies an effect code indicating what would happen if the drop actually occurred. For a list of valid drop effects, see the **DROPEFFECT** enumeration.
3. **DoDragDrop** calls **IDropSource::GiveFeedback** with the effect code so that the drop source interface can provide appropriate visual feedback to the user. The *pDropSource* pointer passed into **DoDragDrop** specifies the appropriate **IDropSource** interface.
4. **DoDragDrop** tracks mouse cursor movements and changes in the keyboard or mouse button state.
  1. If the user moves out of a window, **DoDragDrop** calls **IDropTarget::DragLeave**
  2. If the mouse enters another window, **DoDragDrop** determines if that window is a valid drop target and then calls **IDropTarget::DragEnter** for that window.
  3. If the mouse moves but stays within the same window, **DoDragDrop** calls **IDropTarget::DragOver**.
5. If there is a change in the keyboard or mouse button state, **DoDragDrop** calls **IDropSource::QueryContinueDrag** and determines whether to continue the drag, to drop the data, or to cancel the operation based on the return value.
  1. If the return value is **S\_OK**, **DoDragDrop** first calls **IDropTarget::DragOver** to continue the operation. This method returns a new effect value and **DoDragDrop** then calls **IDropSource::GiveFeedback** with the new effect so appropriate visual feedback can be set. For a list of valid drop effects, see the **DROPEFFECT** enumeration. **IDropTarget::DragOver** and **IDropSource::GiveFeedback** are paired so that as the mouse moves across the drop target, the user is given the most up-to-date feedback on the mouse's position.
  2. If the return value is **DRAGDROP\_S\_DROP**, **DoDragDrop** calls **IDropTarget::Drop**. The **DoDragDrop** function returns the last effect code to the source, so the source application can perform the appropriate operation on the source data, for example, cut

the data if the operation was a move.

3. If the return value is DRAGDROP\_S\_CANCEL, the **DoDragDrop** function calls **IDropTarget::DragLeave**.

总的来说当用户调用DoDragDrop方法以后，就进入到一个循环中。此方法会一直跟中鼠标，检查鼠标所在的窗体是否实现IDropTarget，如果实现了则调用DropEnter，并通过调用GiveFeedBack来显示效果。而在鼠标在控件上时，调用DropOver，并也是通过GiveFeedBack来显示效果。而当进入到另一个窗体时重复上面的过程。如果在拖拽过程中，键盘或鼠标按键发生变化，可以通过QueryContinueDrag来检查是否能继续操作，根据不同的返回结果，调用DropOver或DropLeave。

通过GiveFeedBack我们能通过判断当前的Effects来使用自己的Cursor，但是我们只能设置是否显示默认的光标，而无法指定光标的样式，在XP上没什么区别，但是在WIN7上，我们拖动的时候会在光标上显示【复制】或【移动】等文字，但我们程序却无法显示ss；而通过QueryContinueDrag可以在鼠标，ESC,SHIFT,CTRL能按键状态变化时决定是放弃拖拽，还是继续，还是放下。但是有一个地方那个不太明白，我们并没有显示实现这个方法，系统是如何在我们点击ESC时取消掉的呢？另一个问题，我们能否实现WINDOWS那种，拖拽一个文件时，显示的是文件的图标呢？

## 四 总结

通过以上的分析，基本了解了在.NET平台上是如何实现Drag和Drop操作的。关键就是实现IDropTarget和IDropSouce两个COM接口。然后使用系统提供的API函数DoDragDrop来跟踪整个拖拽的过程，并调用这2个接口来完成拖拽操作。只是在.NET总对这2个接口进行了包装，并且使用了一个AllowDrop属性来注册窗体。

上面只是介绍了实现的原理，而没有深入到传输的数据格式。而且我们程序大多数时候是从外部拖放文件进来，或者是内部相互拖拽。而当我们程序要把数据拖放到WINDOWS资源管理器中又应该怎么做呢。下一篇文章将主要介绍数据是如何传递的。将涉及到OLE和COM相关的一些内容。

转载自：[http://blog.csdn.net/cc\\_net/article/details/5793342](http://blog.csdn.net/cc_net/article/details/5793342)