鼠标与键盘控制篇

第 1 章

鼠标操作控制

- 获取鼠标信息
- 鼠标基本设置
- 鼠标操作在实际中的应用

1.1　获取鼠标信息

实例 001	获取鼠标双击时间间隔 光盘位置：光盘\MR\01\001	初级 趣味指数：★★★

实例说明

鼠标是计算机的一个重要组成部分，它有很多默认的设置，如双击时间间隔、闪烁频率、移动速度等。本实例使用 C#实现了获取鼠标双击时间间隔的功能，实例运行效果如图 1.1 所示。

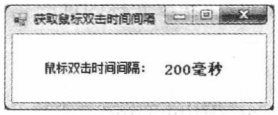

图 1.1　获取鼠标双击时间间隔

关键技术

本实例实现时主要用到了 API 函数 GetDoubleClickTime，下面对其进行详细介绍。

GetDoubleClickTime 函数主要用来判断连续两次鼠标单击之间会被处理成双击事件的间隔时间，其声明语法如下：

```
[DllImport("user32.dll", EntryPoint = "GetDoubleClickTime")]
public extern static int GetDoubleClickTime();                    //重写 API 函数
```

参数说明

返回值：int 类型，表示以毫秒表示的双击时间。

说明：程序中使用系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间，下面遇到类似的情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetMouseTimeSpan。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示鼠标双击的时间间隔。

（3）程序主要代码如下：

```
[DllImport("user32.dll", EntryPoint = "GetDoubleClickTime")]
public extern static int GetDoubleClickTime();                    //重写 API 函数
private void Frm_Main_Load(object sender, EventArgs e)
{
    label2.Text = GetDoubleClickTime()+"毫秒";                    //显示双击鼠标的时间间隔
}
```

秘笈心法

心法领悟 001：使用 OpenWrite 方法打开现有文件并进行写入。

使用 OpenWrite 方法打开现有文件并进行写入时，首先需要生成 FileStream 类的一个对象，用来记录要打开的文件路径及名称，然后定义一个 Byte 类型的数组，存储以 Default 方式编码的要写入文件的内容，最后调用 FileStream 对象的 Write 方法向指定文件中写入内容。使用 OpenWrite 方法打开现有文件并进行写入的代码如下：

```
FileStream FStream = File.OpenWrite(textBox1.Text);
Byte[] info =Encoding.Default.GetBytes(textBox2.Text);
FStream.Write(info, 0, info.Length);
```

实例 002 获取光标闪烁的频率

光盘位置：光盘\MR\01\002

初级

趣味指数：★★★

实例说明

本实例使用 C#实现了获取光标闪烁频率的功能，实例运行效果如图 1.2 所示。

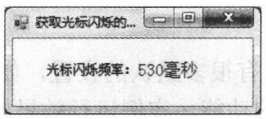

图 1.2 获取光标闪烁的频率

关键技术

本实例实现时主要用到了 API 函数 GetCaretBlinkTime，下面对其进行详细介绍。

GetCaretBlinkTime 函数主要用来判断插入符光标的闪烁频率，其声明语法如下：

```
[DllImport("user32", EntryPoint = "GetCaretBlinkTime")]
public extern static int GetCaretBlinkTime();                    //重写 API 函数
```

参数说明

返回值：int 类型，表示插入符连续两次闪烁间隔的时间，以毫秒为单位，零表示函数调用失败。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetMouseFrequency。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示光标闪烁的频率。

（3）程序主要代码如下：

```
[DllImport("user32", EntryPoint = "GetCaretBlinkTime")]
public extern static int GetCaretBlinkTime();                    //重写 API 函数
private void Frm_Main_Load(object sender, EventArgs e)
{
    label2.Text = GetCaretBlinkTime() + "毫秒";                  //显示光标闪烁频率
}
```

秘笈心法

心法领悟 002：如何读取文件中的第一行数据？

读取文件中的第一行数据时，首先需要生成 StreamReader 类的一个对象，以指定要读取的文件，然后调用 StreamReader 对象的 ReadLine 方法将指定文件中的第一行数据读取出来。读取文件中第一行数据的代码如下：

```
openFileDialog1.Filter = "文本文件(*.txt)|*.txt";
openFileDialog1.ShowDialog();
textBox1.Text = openFileDialog1.FileName;
StreamReader SReader=new StreamReader(textBox1.Text,Encoding.Default);
textBox2.Text = SReader.ReadLine();
```

实例 003 获取鼠标键数

光盘位置：光盘\MR\01\003

初级

趣味指数：★★★

实例说明

鼠标有多种样式，如有的只有左右两个键，有的带滑轮等，不同样式的鼠标，按键数目是不一样的。本实

例使用 C#实现了获取鼠标键数的功能，实例运行效果如图 1.3 所示。

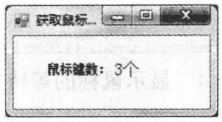

图 1.3　获取鼠标键数

关键技术

本实例实现时主要用到了 API 函数 GetSystemMetrics，下面对其进行详细介绍。

GetSystemMetrics 函数主要用来返回与 Windows 环境有关的信息，其声明语法如下：

```
[DllImport("user32",EntryPoint="GetSystemMetrics")]
public static extern int GetSystemMetrics(int intcount);                    //重写 API 函数
```

参数说明

❶ intcount：指定欲获取的信息。

❷ 返回值：取决于具体的常数索引。

说明：本实例在使用 GetSystemMetrics 函数时，为其 intcount 参数定义了 SM_CMOUSEBUTTONS 常量值，该值的常量值为 43，表示鼠标按钮（按键）的数量，如果没有鼠标，则为 0。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetMouseNumbers。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示鼠标的按键数目。

（3）程序主要代码如下：

```
public const int SM_CMOUSEBUTTONS=43;                                       //定义一个常量值
[DllImport("user32",EntryPoint="GetSystemMetrics")]
public static extern int GetSystemMetrics(int intcoutn);                    //重写 API 函数
private void Form1_Load(object sender, EventArgs e)
{
    int intCon = GetSystemMetrics(SM_CMOUSEBUTTONS);                        //获取鼠标键数
    label2.Text = intCon + "个";                                            //显示在 Label 控件中
}
```

秘笈心法

心法领悟 003：如何向文件中写入数据？

向文件中写入数据时，首先要生成 StreamWriter 类的一个对象，以指定要写入数据的文件，然后调用该对象的 Write 方法向指定文件中写入数据。向文件中写入数据的代码如下：

```
StreamWriter SWriter = new StreamWriter(textBox1.Text);
SWriter.Write(textBox2.Text);
SWriter.Close();
```

实例 004　显示鼠标的等待光标

光盘位置：光盘\MR\01\004　　初级　　趣味指数：★★★

实例说明

在不同的操作状态，鼠标会显示不同的外观，如人们熟知的鼠标的正常选择状态、精确选择状态和帮助选择状态等，那么如何手动设置鼠标处于不同的状态呢？本实例使用 C#实现了使鼠标处于等待状态的功能，实例运行效果如图 1.4 所示。

图 1.4　显示鼠标的等待光标

关键技术

本实例实现时主要用到了 Form 类的 Cursor 属性，下面对其进行详细介介绍。

Form 类的 Cursor 属性用来获取或设置当鼠标指针位于窗体上时显示的光标，其语法格式如下：

```
public virtual Cursor Cursor { get; set; }
```

参数说明

属性值：一个 Cursor，表示当鼠标指针位于窗体上时显示的光标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShowWaitMouse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.Cursor = Cursors.WaitCursor;                                    //将当前窗体的光标样式修改为等待光标
}
```

秘笈心法

心法领悟 004：如何读取文件中的所有数据？

读取文件中的所有数据时，首先需要生成 StreamReader 类的一个对象，以指定要读取的文件，然后调用 StreamReader 对象的 ReadToEnd 方法将指定文件中的所有数据读取出来。读取文件中所有数据的代码如下：

```
StreamReader SReader = new StreamReader(textBox1.Text, Encoding.Default);
textBox2.Text = SReader.ReadToEnd();
```

实例 005　获得鼠标在窗体上的位置

光盘位置：光盘\MR\01\005　　初级　　趣味指数：★★★

实例说明

在开发精确度比较高的程序（如电子地图程序、屏幕抓图程序等）时，通常都需要准确定位鼠标的位置，那么如何实现该功能呢？本实例解决了这一问题。运行本实例，在窗体的任意位置单击鼠标，程序会适时将鼠标的当前位置显示在窗体上。实例运行效果如图 1.5 所示。

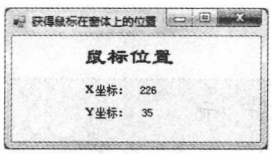

图 1.5　获得鼠标在窗体上的位置

关键技术

本实例实现时主要用到了 MouseEventArgs 类的 X 属性和 Y 属性，下面分别对它们进行详细介绍。

（1）MouseEventArgs 类的 X 属性

MouseEventArgs 类用来为 MouseUp、MouseDown 和 MouseMove 事件提供数据，其 X 属性用来获取鼠标在产生鼠标事件时的 X 坐标，其语法格式如下：

```
public int X { get; }
```

参数说明

属性值：鼠标的 X 坐标（以像素为单位）。

（2）MouseEventArgs 类的 Y 属性

该属性用来获取鼠标在产生鼠标事件时的 Y 坐标，其语法格式如下：

```
public int Y { get; }
```

参数说明

属性值：鼠标的 Y 坐标（以像素为单位）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetMousePosition。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Label 控件，分别用来显示鼠标当前位置的 X 坐标和 Y 坐标。

（3）程序主要代码如下：

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    this.labX.Text = e.X.ToString();                    //显示 X 坐标
    this.labY.Text = e.Y.ToString();                    //显示 Y 坐标
}
```

秘笈心法

心法领悟 005：如何创建临时文件？

在 C#中创建一个临时文件的代码如下：

```
textBox1.Text = Path.GetTempFileName();
FileInfo fin = new FileInfo(textBox1.Text);
StreamWriter sw = fin.AppendText();
sw.Write(textBox2.Text);
sw.Close();
```

实例 006　记录鼠标行为

光盘位置：光盘\MR\01\006　　初级　　趣味指数：★★★☆

实例说明

在 Windows 操作系统中，按下鼠标右键时，会弹出快捷菜单，而按下鼠标左键时，可以拖动窗口，那么操作系统是如何区分用户按下的是鼠标右键还是左键呢？本实例将通过 C#程序解决该问题。运行本实例，当用户对鼠标进行操作时，程序会自动记录并显示鼠标的操作行为。实例运行效果如图 1.6 所示。

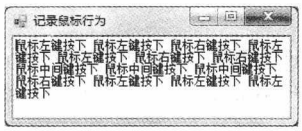

图 1.6　记录鼠标行为

关键技术

本实例实现时主要用到了 MouseEventArgs 类的 Button 属性，下面对其进行详细介绍。

MouseEventArgs 类的 Button 属性用来获取曾按下的是哪个鼠标按键，其语法格式如下：

```
public MouseButtons Button { get; }
```

参数说明

属性值：MouseButtons 枚举值之一。MouseButtons 枚举值及说明如表 1.1 所示。

表 1.1　MouseButtons 枚举值及说明

枚　举　值	说　　明	枚　举　值	说　　明
Left	鼠标左键曾按下	Middle	鼠标中间键曾按下
None	未曾按下鼠标按键	XButton1	第一个 XButton 曾按下
Right	鼠标右键曾按下	XButton2	第二个 XButton 曾按下

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RecordMouse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示鼠标按键的行为。

（3）程序主要代码如下：

```
private void textBox1_MouseDown(object sender, MouseEventArgs e)
{
    string str = textBox1.Text;                          //记录文本框中的内容
    if (e.Button == MouseButtons.Right)                  //如果按下鼠标右键
    {
        str += "鼠标右键按下 ";                           //记录鼠标右键按下
    }
    if (e.Button == MouseButtons.Left)                   //如果按下鼠标左键
    {
        str += "鼠标左键按下 ";                           //记录鼠标左键按下
    }
    if (e.Button == MouseButtons.Middle)                 //如果按下鼠标中间键
    {
        str += "鼠标中间键按下 ";                         //记录鼠标中间键按下
    }
    textBox1.Text = str;                                 //显示鼠标按键行为
}
```

秘笈心法

心法领悟 006：如何实现文件替换？

用 File.Create 方法创建文件时，如果在用户的机器上已经存在该文件，则新文件会自动替换掉原来的文件。实现文件替换功能的代码如下：

```
File.Create(textBox1.Text);
```

实例 007　通过截取系统消息判断鼠标的单击键

光盘位置：光盘\MR\01\007　　中级　　趣味指数：★★★★

实例说明

程序开发中，有效地截取系统消息有时会起到事半功倍的效果，如开发人员可以通过截取系统消息获取鼠标操作或者键盘操作等。本实例通过截取系统消息实现了获取鼠标单击键的功能，实例运行效果如图 1.7 所示。

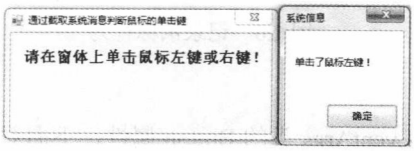

图 1.7　通过截取系统消息判断鼠标的单击键

关键技术

本实例实现的关键是如何截取系统消息，下面对如何在 C#中截取系统消息进行详细讲解。

在 C#中截取系统消息时，主要有以下两种方案：

- 通过在 Form 中重写 Control 类的虚方法 WndProc 来截取 Windows 消息。
- 通过实现 IMessageFilter 接口来创建消息筛选器，从而截取 Windows 消息。

下面对这两种方案中用到的关键技术进行详细讲解。

（1）WndProc 方法

WndProc 方法主要用来处理 Windows 消息，该方法为虚方法，其语法格式如下：

```
protected virtual void WndProc(ref Message m)
```

参数说明

m：Message 结构，表示要处理的 Windows 消息。Message 结构的属性及说明如表 1.2 所示。

表 1.2　Message 结构的属性及说明

属　性	说　明
HWnd	获取或设置消息的窗口句柄
LParam	指定消息的 LParam 字段
Msg	获取或设置消息的 ID 号
Result	指定为响应消息处理而向 Windows 返回的值
WParam	获取或设置消息的 WParam 字段

（2）IMessageFilter 接口

IMessageFilter 接口用来定义消息筛选器接口，该接口的 PreFilterMessage 方法用来在调度消息之前将其筛选出来，其语法格式如下：

```
bool PreFilterMessage(ref Message m)
```

参数说明

❶ m：要调度的消息，无法修改此消息。

❷ 返回值：如果筛选消息并禁止消息被调度，则为 true；如果允许消息继续到达下一个筛选器或控件，则为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SysInfo。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示操作程序的提示信息。

（3）程序主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)                    //窗体加载消息筛选器
{
    Application.AddMessageFilter(mf);                                     //添加消息筛选器，以便在向目标传送 Windows 消息时监视这些消息
}
private void Frm_Main_FormClosing(object sender, FormClosingEventArgs e)//从窗体中移除一个消息筛选器
{
    Application.RemoveMessageFilter(mf);                                  //从应用程序的消息泵移除一个消息筛选器
}
protected override void WndProc(ref Message m)                            //方法一，重写 WndProc 虚方法，与方法二不可同时存在
{
    switch (m.Msg)                                                        //判断系统消息的 ID 号
    {
        case 513:
            MessageBox.Show("单击了鼠标左键！", "系统信息");
            m.Result = (IntPtr)0;                                         //为了响应消息处理而向 Windows 返回的值
            break;
        case 516:
            MessageBox.Show("单击了鼠标右键！", "系统信息");
            m.Result = (IntPtr)0;                                         //为了响应消息处理而向 Windows 返回的值
            break;
        default:
```

```
            base.WndProc(ref m);
            break;
    }
}
//方法二，实现 IMessageFilter 接口，从而获得 Windows 消息，与方法一不可同时存在
public class MessageFilter : IMessageFilter
{
    public bool PreFilterMessage(ref Message message)                    //实现 PreFilterMessage 方法
    {
        switch (message.Msg)                                             //判断系统消息的 ID 号
        {
            case 513:
                MessageBox.Show("单击了鼠标左键！","系统信息");
                return true;
            case 516:
                MessageBox.Show("单击了鼠标右键！","系统信息");
                return true;
            default:
                return false;
        }
    }
}
```

秘笈心法

心法领悟 007：如何使用递归法删除文件夹中的所有文件？

使用递归法删除文件夹中所有文件时，首先需要生成 DirectoryInfo 类的一个对象，以指定文件夹，然后生成一个 FileSystemInfo 类型的数组，用来记录指定文件夹中的所有目录及子目录，最后，循环访问 FileSystemInfo 类型数组中的文件，并调用 FileInfo 对象的 Delete 方法将其一一删除。

1.2 鼠标基本设置

实例 008	定义鼠标指针形状 光盘位置：光盘\MR\01\008	初级 趣味指数：★★★

实例说明

鼠标的指针形状有很多种，如人们经常看到的“小箭头”形状、单击超链接时的“小手”形状等，那么如何在程序中自定义鼠标的指针形状呢？本实例实现了该功能。运行本实例，将鼠标移动到“鼠标放在这”文本框上方，鼠标指针会呈现“小手”形状。实例运行效果如图 1.8 所示。

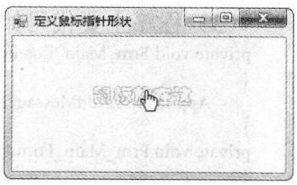

图 1.8 定义鼠标指针形状

关键技术

本实例实现时主要用到了 Label 控件的 Cursor 属性，下面对其进行详细介绍。

Label 控件的 Cursor 属性用来获取或设置当鼠标指针位于控件上时显示的光标，其语法格式如下：

```
public virtual Cursor Cursor { get; set; }
```

参数说明

属性值：一个 Cursor，表示当鼠标指针位于控件上时显示的光标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetMouseShape。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来在其上方改

变鼠标的指针形状。

（3）程序主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    this.labCur .Cursor =Cursors.Hand;                    //定义鼠标形状
}
```

秘笈心法

心法领悟 008：如何更改文件扩展名？

更改文件扩展名时，首先需要生成 FileInfo 类的一个对象，以指定要更改的文件，然后调用该对象的 MoveTo 方法实现文件扩展名的更改操作。更改文件扩展名的代码如下：

```
FileInfo FInfo = new FileInfo(textBox1.Text);
string strPath = textBox1.Text.Substring(0,textBox1.Text.LastIndexOf(".")+1)+textBox3.Text.Trim();
FInfo.MoveTo(strPath);
```

实例 009　自定义鼠标的图片

光盘位置：光盘\MR\01\009　　初级　　趣味指数：★★★★☆

实例说明

用户在使用鼠标时，为了使鼠标更加美观，可以自定义鼠标的图片，本实例将介绍如何使用 C#实现自定义鼠标图片的功能。运行本实例，单击“Click Me 改变鼠标图片”按钮，即可将鼠标图片设置为指定的图片。实例运行效果如图 1.9 所示。

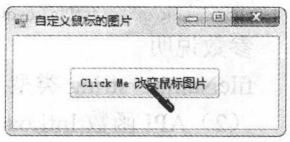

图 1.9　自定义鼠标的图片

关键技术

本实例实现时主要用到了 Form 类的 Cursor 属性，下面对其进行详细介绍。

Form 类的 Cursor 属性用来获取或设置当鼠标指针位于窗体上时显示的光标，其语法格式如下：

```
public virtual Cursor Cursor { get; set; }
```

参数说明

属性值：一个 Cursor，表示当鼠标指针位于窗体上时显示的光标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetMouseImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行自定义鼠标图片操作。

（3）程序主要代码如下：

```
private void btnClickMe_Click(object sender, EventArgs e)
{
    this.Cursor = new Cursor("pen_il.cur");                    //改变鼠标图片
}
```

秘笈心法

心法领悟 009：如何实现复制文件？

复制文件时，首先需要生成 FileInfo 类的一个对象，以指定要复制的文件，然后调用该对象的 MoveTo 方法将指定文件复制到指定路径下。实现复制文件的代码如下：

```
FileInfo FInfo = new FileInfo(textBox1.Text);
if (FInfo.Exists)
{
    FInfo.MoveTo(textBox2.Text + textBox1.Text.Substring(textBox1.Text.LastIndexOf("\\") + 1, (textBox1.Text.Length - textBox1.Text.LastIndexOf
```

```
        ("\\") - 1));
}
```

实例 010 自定义动画鼠标

光盘位置：光盘\MR\01\010

中级

趣味指数：★★★★☆

实例说明

人们在电脑中使用的鼠标样式，一般都是系统默认的，为了使自己的电脑更具有特色，用户可以自己设置鼠标样式。本实例将对改变 C#窗体和系统鼠标样式进行详细讲解，实例运行效果如图 1.10 所示。

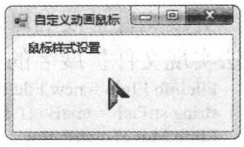

图 1.10 自定义动画鼠标

关键技术

本实例实现时，主要使用 API 函数 LoadCursorFromFile、IntLoadCursorFromFile 和 SetSystemCursor 来改变系统的鼠标样式。下面对本实例中用到的关键技术进行详细讲解。

（1）API 函数 LoadCursorFromFile

该函数是在一个指针文件或一个动画指针文件（扩展名分别是.cur 和.ani）的基础上创建一个指针，其声明语法如下：

```
[DllImport("user32.dll")]
public static extern IntPtr LoadCursorFromFile(string fileName);
```

参数说明

fileName：string 类型，包含指针的文件的名字。

（2）API 函数 IntLoadCursorFromFile

该函数将要修改的鼠标图片存入到指定目录下，其声明语法如下：

```
[DllImport("user32", EntryPoint = "LoadCursorFromFile")]
public static extern int IntLoadCursorFromFile(string lpFileName);
```

参数说明

lpFileName：文件路径。

（3）API 函数 SetSystemCursor

该函数用于改变任何一个标准系统指针，其声明语法如下：

```
[DllImport("user32", EntryPoint = "SetSystemCursor")]
public static extern void SetSystemCursor(int hcur, int id);
```

参数说明

❶ hcur：光标的句柄，该函数用 hcur 标识的光标的内容代替 id 定义的系统光标内容。系统通过调用 DestroyCursor 函数销毁 hcur，因此 hcur 不能是由 LoadCursor 函数载入的光标。要指定一个从资源载入的光标，先用 CopyCursor 函数复制该光标，然后把该副本传送给 SetSystemCursor 函数。

❷ id：指定由 hcur 的内容替换系统光标，常用的系统光标标识符及说明如表 1.3 所示。

表 1.3 常用的系统光标标识符及说明

标 识 符	值	说 明
OCR_APPSTARTING	32650	标准箭头和小的沙漏
OCR_NORAAC	32512	标准箭头
OCR_CROSS	32515	交叉十字线光标
OCR_HAND	32649	手的形状（Windows NT 5.0 和以后版本）
OCR_IBEAM	32513	I 形
OCR_NO	32648	斜的圆
OCR_SIZEALL	32646	4 个方位的箭头分别指向北、南、东、西

续表

标 识 符	值	说　明
OCR_SIZENS	32645	双箭头，分别指向北和南
OCR_SIZENWSE	32642	双箭头，分别指向西北和东南
OCR_SIZEWE	32644	双箭头，分别指向西和东
OCR_UP	32516	垂直箭头
OCR_WAIT	32514	沙漏

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetAnimateMouse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏。

（3）程序主要代码如下。

在 ToolS_From 项的 Click 事件中，设置当前鼠标指针位于窗体时显示自定义的鼠标图标，代码如下：

```
private void ToolS_From_Click(object sender, EventArgs e)
{
    Cursor myCursor = new Cursor(Cursor.Current.Handle);                                    //创建 Cursor 类
    IntPtr colorCursorHandle = LoadCursorFromFile("0081.ani");                              //鼠标图标路径
    myCursor.GetType().InvokeMember("handle", BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.SetField, null,
        myCursor, new object[] { colorCursorHandle });                                      //获取鼠标
    this.Cursor = myCursor;                                                                 //设置当前鼠标指针位于窗体时显示的光标
}
```

在 ToolS_FromRevert 项的 Click 事件中，恢复当前鼠标指针位于窗体时的鼠标图标，代码如下：

```
private void ToolS_FromRevert_Click(object sender, EventArgs e)
{
    this.Cursor = Cursors.Default;
}
```

在 ToolS_System 项的 Click 事件中，用自定义鼠标图标设置系统鼠标的正常选择鼠标、移动鼠标、不可用鼠标和超链接鼠标，代码如下：

```
private void ToolS_System_Click(object sender, EventArgs e)
{
    int cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\01.cur");                          //设置正常选择鼠标
    SetSystemCursor(cur, OCR_NORAAC);
    cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\03.cur");                              //设置移动鼠标
    SetSystemCursor(cur, OCR_SIZEALL);
    cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\04.cur");                              //设置不可用鼠标
    SetSystemCursor(cur, OCR_NO);
    cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\06.cur");                              //设置超链接鼠标
    SetSystemCursor(cur, OCR_HAND);
}
```

注意：当用自定义鼠标图标设置鼠标时，应将其图标放置在系统盘的 WINDOWS\Cursors 目录下，否则 API 函数 IntLoadCursorFromFile 无法在指定文件的基础上创建一个指针。

在 ToolS_SystemRevert 项的 Click 事件中，恢复系统鼠标的正常选择鼠标、移动鼠标、不可用鼠标和超链接鼠标，代码如下：

```
private void ToolS_SystemRevert_Click(object sender, EventArgs e)
{
    int cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\arrow_m.cur");                     //恢复正常选择鼠标
    SetSystemCursor(cur, OCR_NORAAC);
    cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\move_r.cur");                          //恢复移动鼠标
    SetSystemCursor(cur, OCR_SIZEALL);
    cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\no_r.cur");                            //恢复不可用鼠标
    SetSystemCursor(cur, OCR_NO);
    cur = IntLoadCursorFromFile(@"C:\WINDOWS\Cursors\hand.cur");                            //恢复超链接鼠标
    SetSystemCursor(cur, OCR_HAND);
}
```

秘笈心法

心法领悟 010：如何修改文件名？

修改文件名称时，需要调用 FileInfo 类的 MoveTo 方法实现，代码如下：

```
FileInfo FInfo = new FileInfo(textBox1.Text);
string strPath = textBox1.Text.Substring(0, textBox1.Text.LastIndexOf("\\") + 1) + textBox2.Text + "." + textBox3.Text;
FInfo.MoveTo(strPath);
```

实例 011 交换鼠标左右键功能

光盘位置：光盘\MR\01\011　　初级　　趣味指数：★★★★

实例说明

当用户双击“控制面板”/“鼠标”时，会弹出“鼠标属性”对话框，在该对话框中可以对鼠标进行相关设置。本实例将根据“鼠标属性”对话框开发一个程序，用来控制鼠标左右键的切换，实例运行效果如图 1.11 所示。

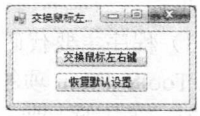

图 1.11　交换鼠标左右键功能

关键技术

本实例实现时主要用到了 API 函数 SwapMouseButton，下面对其进行详细介绍。

SwapMouseButton 函数主要用来决定是否互换鼠标左右键的功能，其声明语法如下：

```
[System.Runtime.InteropServices.DllImport("user32.dll", EntryPoint = "SwapMouseButton")]
public extern static int SwapMouseButton(int bSwap);                    //重写 API 函数
```

参数说明

❶ bSwap：如果为非零，则互换两个鼠标按键的功能，否则恢复正常状态。

❷ 返回值：非零表示鼠标按键的功能在调用该函数之前已经互换，否则返回零。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChangeRLKey。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行交换鼠标左右键和恢复鼠标默认设置操作。

（3）程序主要代码如下：

```
[System.Runtime.InteropServices.DllImport("user32.dll", EntryPoint = "SwapMouseButton")]
public extern static int SwapMouseButton(int bSwap);                    //重写 API 函数
public void DefaultRightButton()
{
    SwapMouseButton(1);                                                 //鼠标右键
}
public void DefaultLeftButton()
{
    SwapMouseButton(0);                                                 //鼠标左键
}
private void btnChange_Click(object sender, EventArgs e)
{
    this.DefaultRightButton();                                          //交换鼠标左右键
}
private void btnReset_Click(object sender, EventArgs e)
{
    this.DefaultLeftButton();                                           //恢复默认设置
}
```

秘笈心法

心法领悟 011：如何删除文件？

删除文件时，需要用到 FileInfo 类的 Delete 方法，该方法用来永久删除文件。删除文件的代码如下：

```
FileInfo FInfo = new FileInfo(textBox1.Text);
FInfo.Delete();
```

实例 012　限制鼠标在某一区域工作

光盘位置：光盘\MR\01\012　　中级　　趣味指数：★★★★☆

实例说明

鼠标是一个用于操作计算机的外部输入设备，在屏幕中，用一个具有特殊形状的图标来表示当前鼠标。鼠标的移动范围是整个屏幕，用它可以向任意窗口发送指令，但是一些恶意软件故意改变鼠标移动的区域，将它固定在某个范围内，而这项技术也可以应用在普通的应用软件中。本实例使用 C#实现了一个限制鼠标活动区域的功能，实例运行效果如图 1.12 所示。

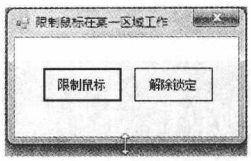

图 1.12　限制鼠标在某一区域工作

关键技术

本实例实现限制鼠标在某一区域工作的功能时，主要用到了 Cursor 类中的相关属性。下面对 Cursor 类进行详细介绍。

Cursor 类表示用于绘制鼠标指针的图像，该类的常用属性及说明如表 1.4 所示。

表 1.4　Cursor 类的常用属性及说明

属　性	说　明
Clip	获取或设置表示光标的剪辑矩形的边界
Current	获取或设置代表鼠标光标的光标对象
Handle	获取光标句柄
Position	获取或设置光标位置
Size	获取光标对象的大小

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ControlMouseRange。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行限制和解除鼠标活动区域的操作。

（3）程序主要代码如下：

```
private void button1_Click(object sender, EventArgs e)                    //限制鼠标活动区域
{
    this.Cursor = new Cursor(Cursor.Current.Handle);                       //创建 Cursor 对象
    Cursor.Position = new Point(Cursor.Position.X, Cursor.Position.Y);      //设置鼠标位置
    Cursor.Clip = new Rectangle(this.Location, this.Size);                 //设置鼠标的活动区域
}
private void button2_Click(object sender, EventArgs e)                    //解除对鼠标活动区域的限制
{
    Screen[] screens = Screen.AllScreens;                                  //获取显示的数组
    this.Cursor = new Cursor(Cursor.Current.Handle);                       //创建 Cursor 对象
    Cursor.Clip = screens[0].Bounds;                                       //解除对鼠标活动区域的限制
}
```

秘笈心法

心法领悟 012：如何拖放文件？

将所选的文件拖放到窗体中的实现代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
```

```
    this.AllowDrop = true;                                          //设置窗体为允许拖放
}
private void Form1_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
        for (int i = 0; i < files.Length; i++)                      //将拖放入窗体的文件的文件名加入 ListBox
        {
            listBox1.Items.Add(files[i]);
        }
    }
}
```

实例 013　屏蔽鼠标按键

光盘位置：光盘\MR\01\013　　中级　　趣味指数：★★★★☆

实例说明

当平时上网遇到好的帖子并对其进行复制操作时，往往会出现单击鼠标右键无效的情况，这样将直接导致无法进行复制，这种做法虽然限制浏览者的权限，但它保护了文章作者的合法权益，因此被广泛采纳。本实例将详细讲解如何屏蔽鼠标按键，实例运行效果如图 1.13 所示。

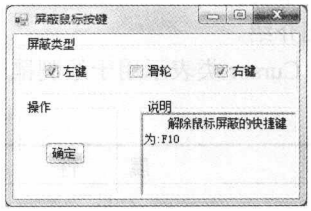

图 1.13　屏蔽鼠标按键

关键技术

本实例实现时主要用到了钩子的安装及释放技术、系统热键的解除屏蔽技术。下面对本实例中用到的关键技术进行详细讲解。

（1）钩子的安装及释放技术

Hook（钩子）函数是 Windows 提供的一种消息处理机制，是指在程序正常运行过程中接收信息之前预先启动的函数，它主要用来检查和修改传递给本程序的 Windows 消息，钩子实际上是一个处理消息的程序段。通过系统调用，将其挂入系统，每当发出特定消息时，在没有到达窗口前，钩子程序就先捕获该消息，对截获的消息进行处理，还可以强制结束消息的传递。

钩子的安装用到 API 函数 SetWindowsHookEx，其语法格式如下：

```
[DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern int SetWindowsHookEx(int HookType, HOOKPROCEDURE methodAddress, IntPtr handler, int dwThreadId);
```

SetWindowsHookEx 函数中的参数及说明如表 1.5 所示。

表 1.5　SetWindowsHookEx 函数中的参数及说明

参　数	说　明
HookType	表示钩子类型，钩子与消息类型相对应
methodAddress	表示钩子函数入口地址
handler	表示钩子所在实例的句柄
dwThreadId	表示钩子监视的线程号

钩子的释放用到 API 函数 UnhookWindowsHookEx，其语法格式如下：

```
[DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall, SetLastError = true)]
public static extern bool UnhookWindowsHookEx(IntPtr idHook);
```

参数说明

idHook：表示将要卸载的句柄。

本实例中为了实现对鼠标按键的屏蔽，需要进行钩子的安装，代码如下：

```
public bool StartHook()                                             //自定义安装钩子方法
{
```

```
    IntPtr InstallHook = (IntPtr)(4194304);                                    //定义安装钩子的句柄
    if (this.mouseHandler == IntPtr.Zero)                                      //判断 mouseHandler 的值是否为 0
    {
        this.m_MouseHookProcedure = new HOOKPROCEDURE(MouseHookProcedure);     //创建钩子函数地址
        //安装钩子函数
        this.mouseHandler = (IntPtr)SetWindowsHookEx(WH_MOUSE_LL, m_MouseHookProcedure, InstallHook, 0);
        if (this.mouseHandler == IntPtr.Zero)                                  //判断 mouseHandler 的值是否为 0
        {
            this.UnInstallHook();                                              //释放钩子
            return false;                                                      //创建失败
        }
    }
    return true;                                                               //创建成功
}
```

钩子使用完毕后，需要对它进行释放，否则，将造成不必要的麻烦，代码如下：

```
public bool UnInstallHook()                                                    //自定义卸载钩子函数方法
{
    bool Result = true;                                                        //定义一个 bool 型的标识变量
    if (this.mouseHandler != IntPtr.Zero)                                      //当 mouseHandler 的值不为 0 时
    {
        Result = (UnhookWindowsHookEx(this.mouseHandler) && Result);           //获取是否进行卸载
        this.mouseHandler = IntPtr.Zero;                                       //重新定义 mouseHandler
    }
    return Result;                                                             //返回该标识变量
}
```

（2）系统热键的解除屏蔽技术

与钩子类似，系统热键的使用也需要安装和卸载。安装时用到 API 函数 RegisterHotKey，其语法格式如下：

```
[DllImport("user32.dll", SetLastError = true)]
public static extern bool RegisterHotKey(IntPtr Hwnd, int Id, KeyModifiers keyModifiers, Keys key);
```

RegisterHotKey 函数中的参数及说明如表 1.6 所示。

表 1.6　RegisterHotKey 函数中的参数及说明

参　数	说　明
Hwnd	表示要定义热键窗口的句柄
Id	表示所定义热键的 ID
keyModifiers	表示热键是否在按下所定义的按键之后才会生效
key	表示定义热键的内容

卸载时用到 API 函数 UnregisterHotKey，其语法格式如下：

```
[DllImport("user32.dll", SetLastError = true)]
public static extern bool UnregisterHotKey(IntPtr Hwnd, int Id);
```

参数说明

❶ Hwnd：表示将要释放的热键的句柄。

❷ Id：表示将要释放的热键的 ID。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NullificationMouse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示解除屏蔽操作快捷键的提示；添加 3 个 CheckBox 控件，分别用来标识是否屏蔽左键、右键和滑轮；添加一个 Button 控件，用来执行屏蔽鼠标按键操作。

（3）程序主要代码如下。

本实例用到自定义类 MouseHook，在自定义类内部首先定义程序中要使用的全局对象及变量，代码如下：

```
public const int WH_MOUSE_LL = 14;                                  //定义鼠标左键的键值
public const int WM_MOUSEMOVE = 0x200;                              //定义鼠标移动对应的键值
public const int WM_LBUTTONDOWN = 0x201;                            //定义按下鼠标左键对应的键值
public const int WM_RBUTTONDOWN = 0x204;                            //定义按下鼠标右键对应的键值
```