

第5章

文件操作

本章导读

文件操作是应用程序必不可少的功能。与文件相关的典型操作包括对文件的创建、复制、移动、删除以及对文件内容的读写；目录的创建、移动和枚举目录内容等，这些典型的操作可以使用 System.IO 命名空间的 File 和 Directory 静态类来实现，而非静态类 FileInfo 和 DirectoryInfo 用于取得特定文件和目录的相关属性。此外，在 C# 中主要使用 FileStream 类对各种类型的文件的内容以字节为单位进行随机读写，而 StreamReader 和 StreamWriter 类用于简化对文本文件的读写。

5.1 File 类

File 类用于完成对文件典型的操作，如复制、移动、创建、打开、删除以及对文件内容的读、写和追加操作。也可使用 File 类获取和设置文件的属性或有关文件创建、访问及写入操作的日期等信息。

File 是静态类，包含主要的静态方法，如表 5.1 所示。

表 5.1 File 类主要的静态方法

方 法 名	说 明
Copy	将源文件复制到目标位置
Move	将源文件移动到目标位置
Delete	删除指定的文件
Exist	判断给定的文件是否存在
Create	创建文件，返回 FileStream 对象
OpenRead	以只读方式打开文件，返回 FileStream 对象
OpenWrite	以只写方式打开文件，返回 FileStream 对象
ReadAllText	读出指定文本文件内容，返回字符串
WriteAllText	以覆盖方式将文本写入文件
AppendAllText	以追加方式将文本写入文件

注：所有的 File 方法都要求指定操作所需要的文件路径。

一个典型的文件路径包含若干部分，如文件名、扩展名、根目录、子目录和父目录等。如有文件路径信息为：

C:\Windows\System32\config\news.png

则该路径信息包含以下几个概念。

- (1) 文件绝对路径: C:\Windows\System32\config\news.png。
- (2) 文件目录: C:\Windows\System32\config。
- (3) 文件名: news.png。
- (4) 文件扩展名: png。
- (5) 根目录: C:\。
- (6) config 是 C:\Windows\System32 的子目录,而 C:\Windows\System32 是 config 的父目录。

目录有时又称路径、文件或文件夹的位置等,如上面文件的绝对路径,又叫文件的全路径、文件及其位置或文件及其所在的文件夹等,为描述方便,在不会产生歧义的情况下,本章对这些概念的多种叫法不做严格的区分。

注意,本章所有与文件操作相关的类都包含在 System.IO 命名空间中。因此,直接使用这些类时,需要声明对该命名空间的引用。

File 类是静态类,常用的静态方法如下。

- (1) Exists 方法。
 - ① 方法签名: bool Exists(string path)。
 - ② 功能: 判断指定的文件是否存在。
 - ③ 参数: path 代表指定的文件。
 - ④ 返回值: 布尔类型,true 表示文件存在,false 表示文件不存在。
- (2) Copy 方法。
 - ① 方法签名: void Copy (string sourceFileName, string destFileName [, bool overwrite])。
 - ② 功能: 将源文件复制为目标文件。
 - ③ 参数: 第一个参数 sourceFileName 代表源文件,第二个参数 destFileName 代表目标文件,第三个参数 Overwrite 参数表示是否覆盖目标文件,可选,如果取值为 true,则覆盖存在的目标文件;如果取值为 false,在目标文件已存在时将抛出异常,默认值为 false。
- 返回值: 无。
- (3) Delete 方法。
 - ① 方法签名: void Delete(string path)。
 - ② 功能: 删除指定的文件。
 - ③ 参数: 参数 path 代表要删除的文件。如果要删除的文件不存在,不会抛出异常。
 - ④ 返回值: 无。
- (4) Move 方法。
 - ① 方法签名: void Move(string sourceFileName, string destFileName)。
 - ② 功能: 将源文件移动到目标位置。
 - ③ 参数: 第一个参数 sourceFileName 代表要复制的源文件,第二个参数 destFileName 代表目标文件。如果源文件不存在,则抛出异常。
 - ④ 返回值: 无。

(5) ReadAllText 方法。

① 方法签名: string ReadAllText(string path[, Encoding encoding])。

② 功能: 读取文本文件的内容,然后关闭文件。

③ 参数: 第一个参数 path 表示要读取的文本文件,第二个参数 encoding 表示文本文件的编码方式,可选。如果指定了第二个参数,则按指定的编码读取文本文件内容,否则,按默认的编码读取文本文件内容。Encoding 类在命名空间 System. Text 中。

④ 返回值: 字符串类型数据,代表读取的文本文件的内容。

(6) WriteAllText 方法。

① 方法签名: void WriteAllText (string path, string contents[, Encoding encoding])。

② 功能: 将字符串数据写入到指定的文件中。

③ 参数: 第一个参数代表文件名及其位置,第二个参数表示要写入文本文件的字符串数据,第三个参数表示保存文件时的编码方式,可选。如果指定了第三个参数,则按指定编码保存,否则按系统默认编码保存。

④ 返回值: 无。

(7) ReadAllLines 方法。

① 方法签名: string[] ReadAllLines(string path[, Encoding encoding])。

② 功能: 按默认或指定的编码,从指定的文本文件中读取所有的行数据。

③ 参数: 含义同 ReadAllText 方法中的描述。

④ 返回值: 字符串数组。

(8) AppendAllText 方法。

① 方法签名: void AppendAllText (string path, string contents [, Encoding encoding])。

② 功能: 将字符串数据添加到文本文件末尾。可以指定编码或默认编码。如果指定的文件不存在,则自动创建该文件。

③ 参数: 含义同 ReadAllText 方法中的描述。

④ 返回值: 无。

应用实例 5.1 将 c:\a.txt 复制到 d:\b.txt,如果目标文件存在则覆盖。

新建项目,保存为 MyFile。在 Form1 窗体中添加一个名称为 button1 的按钮,并在其事件处理过程中添加以下的代码。

代码片段

```
if (File.Exists("c:\\a.txt"))
{
    File.Copy("C:\\a.txt", "D:\\b.txt", true);
}
else
{
    MessageBox.Show("源文件文件不存在!", "提示");
}
```

代码说明

- (1) 首先判断源文件 a.txt 是否存在,如果存在,则以覆盖方式复制到 D 盘根目录。
- (2) Copy 方法的功能是将源文件复制到目标位置。其第一个参数表示源文件,第二个参数表示目标位置(可以与源文件同名,或重新命名保存),第三个参数为 true,表示如果目标位置已经存在同名文件,则覆盖; false 表示如果存在同名文件,则抛出异常。

注意:

- (1) 复制文件之前必须判断源文件是否存在,否则会出现“未找到文件”的错误。
- (2) 移动文件前必须判断目标文件是否存在,否则会出现“文件已经存在”的错误。
- (3) 字符串中的“\\”是转义字符,如果要使用原义字符串,则在字符串前加@。
- (4) 如果要重命名文件,可以使用 Move 方法将文件移动到同一位置。

应用实例 5.2 使用文件保存对话框控件将在文本框中输入的内容保存为文本文件; 使用打开文件对话框控件将选择的文本文件的内容显示在文本框中。

新建项目,保存为 MyText。

界面布局

在 Form1 窗体上添加两个 Button 和两个 TextBox 控件,添加一个 SaveFileDialog 控件和一个 OpenFileDialog 控件,保留各个控件默认的名称。分别设置两个文本框的 Multiline 属性为 True; ScrollBars 属性为 Both; WordWrap 属性为 False,让其成为带滚动条的多行文本框。界面布局如图 5.1 所示。

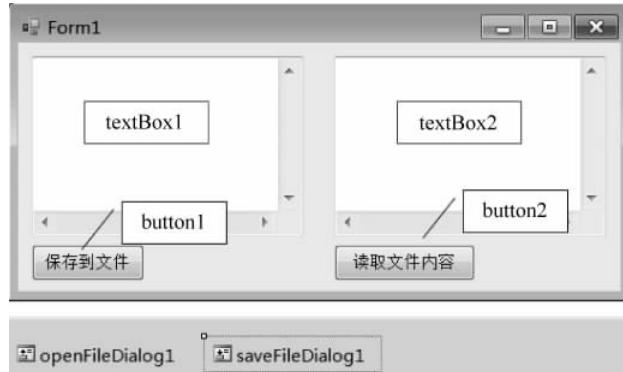


图 5.1 界面布局

程序代码

```
//省略自动生成的 using 部分
using System.IO; //声明对命名空间的引用
namespace MyText
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }
        //“保存到文件”按钮 button1 的事件处理过程
        private void button1_Click(object sender, EventArgs e)
```

```

    {
        saveFileDialog1.Filter = "文本文件|*.txt";           //仅选择文本文件
        saveFileDialog1.Title = "保存文件";                  //对话框标题
        saveFileDialog1.InitialDirectory = @"c:\";          //初始保存目录
        if (saveFileDialog1.ShowDialog() == DialogResult.OK) //选择了文件
        {
            //保存文本框输入的内容到选择的文件
            File.WriteAllText(saveFileDialog1.FileName, textBox1.Text);
        }
    }

    //读取文件内容"按钮 button2 的事件处理过程
    private void button2_Click(object sender, EventArgs e)
    {
        openFileDialog1.Filter = "文本文件|*.txt";           //仅选择文本文件
        openFileDialog1.Title = "打开文件";                  //对话框标题
        openFileDialog1.InitialDirectory = @"c:\";          //初始目录
        if (openFileDialog1.ShowDialog() == DialogResult.OK) //选择了文件
        {
            //读取文件的内容
            textBox2.Text = File.ReadAllText(openFileDialog1.FileName);
        }
    }
}

}//Form1 类结束
}//命名空间结束

```

代码说明

(1) 有关保存文件对话框和打开文件对话框控件的使用已在第 4 章介绍,这里设置了其初始目录属性,并得到选择的文件绝对位置,以便进行读写文件操作。button1 的事件响应代码实现将在文本框中输入的内容保存到选择的文件中,button2 的事件响应代码实现打开选择的文件并将内容显示在文本框中。

(2) WriteAllText 方法的功能是将字符串数据保存到指定的文件中。

(3) ReadAllText 方法的功能是读出指定文件的内容,并返回字符串数据。

另外,所有的控件本质上都是一个类,除了可以通过拖放方式添加到窗体,也可以通过代码方法来创建。上面例子中的保存文件对话框控件,可以通过以下代码来创建而不需要直接拖放到窗体,例如:

```

SaveFileDialog sfd = new SaveFileDialog();           //创建保存文件对话框对象 sfd
sfd.Filter = "文本文件|*.txt";                     //设置属性: 仅选择文本文件
sfd.Title = "保存文件";                            //设置属性: 对话框标题
sfd.InitialDirectory = @"c:\";                      //设置属性: 初始保存的位置
sfd.ShowDialog();                                    //显示对话框

```

运行结果

按 F5 键运行程序,单击界面中的不同按钮将弹出不同对话框,运行结果如图 5.2 所示。

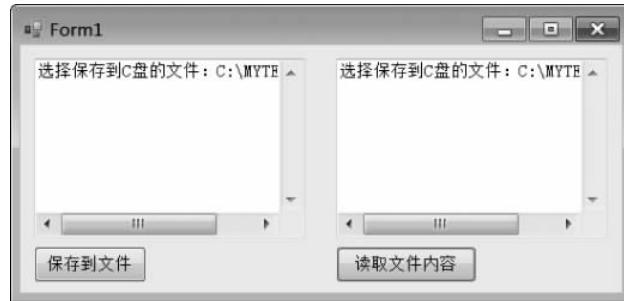


图 5.2 运行结果

典型案例 模仿并实现 Windows 记事本基本功能

本例需要读者先熟练使用 Windows 记事本中的“文件”菜单包含的功能，注意观察在当前编辑的内容修改前后，分别执行记事本应用程序菜单中的“新建”“打开”“保存”“另存为”和“退出”操作时出现的不同提示。

1. 案例分析

1) 系统框架图

本例模仿 Windows 的记事本应用程序的功能，实现了对文本文件内容的基本操作，包括新建文件、打开文本文件并将文件的内容显示在界面的多行文本框中、保存修改后的文件内容，并可以将当前编辑的文件另存到指定的位置。其实现的功能框架如图 5.3 所示。

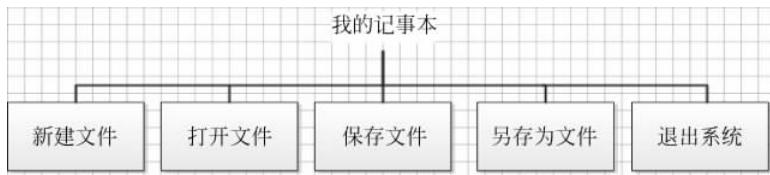


图 5.3 系统框架图

2) 操作逻辑

根据对 Windows 的记事本应用程序的操作过程分析，保存文件和另存文件的操作逻辑如图 5.4 所示。

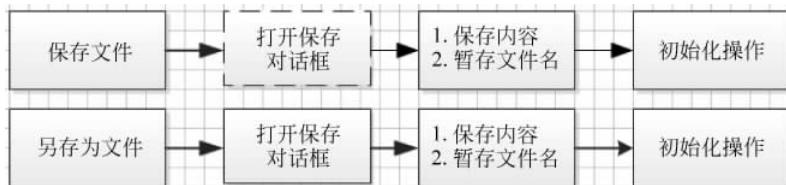


图 5.4 保存和另存功能的操作逻辑示意图

新建文件、打开文件和退出系统操作逻辑类似，如图 5.5 所示。

从图 5.4 和图 5.5 可知，除了“另存为文件”功能之外，执行其他所有操作之前，都必须

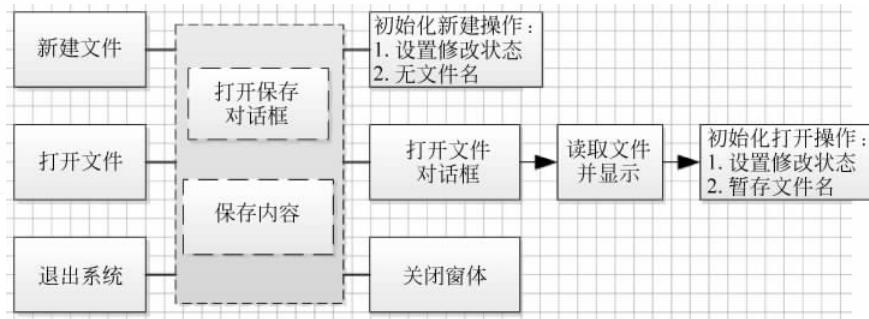


图 5.5 新建文件、打开文件和退出系统功能操作逻辑示意图

注：图中的虚线表示根据实际情况来决定是否需要执行。

判断文本框中已存在的内容是否被修改，如果被修改，则需要提示用户保存。为了代码重用，将每个步骤都可能用到的“保存内容”功能单独设计为 SaveFile 方法。另外，只有两种情况下才需要弹出保存对话框，一种是使用“保存文件”功能时文件位置未指定，一种是使用了“另存为文件”功能，根据这两个条件来决定是否弹出保存文件对话框。

而判断内容是否被修改，可以定义一个 bool 类型的状态变量 isChange，并在文本框中的 TextChanged 事件中进行检测，该事件在文本框的内容发生改变时触发，在该事件中，将状态变量 isChange 设置为 true，调用 SaveFile 方法后将该变量恢复为 false。

此外，执行图 5.5 中“新建文件”“打开文件”和“退出系统”操作前，都要先判断当前编辑的内容是否已保存，然后再执行各自的具体操作。

2. 功能模块分析

(1) 新建文件：如果当前编辑的内容被修改，提示用户保存，然后执行“新建文件”功能的初始化操作，包括：①清空文本框；②设置当前状态为非修改状态；③初始化保存位置。

(2) 打开文件：如果当前编辑的内容被修改，提示用户保存，然后再打开在文件对话框中选择的文件，并将文件的内容显示到文本框中，最后执行“打开文件”功能的初始化操作，包括：①保存选择位置，便于下次直接执行保存功能；②设置当前状态为非修改状态。

(3) 退出系统：如果当前编辑的内容被修改，提示用户保存，然后关闭窗体。

“新建文件”“打开文件”和“退出系统”功能的执行流程都是相同的，如图 5.6 所示。

(4) 保存文件：判断当前文件位置是否已指定，如果已经指定，将当前编辑的内容写入到指定的文件中，也就是直接保存；如果当前无文件位置，打开保存文件对话框，让用户选择保存文件的位置并向选择的文件写入内容。

(5) 另存为文件：直接打开保存文件对话框，让用户重新选择保存位置并执行写入文件的操作，该操作无须判断内容是否修改。

“保存文件”与“另存为文件”功能的执行流程如图 5.7 所示。

3. 具体实现

新建项目，保存为 MyFile。

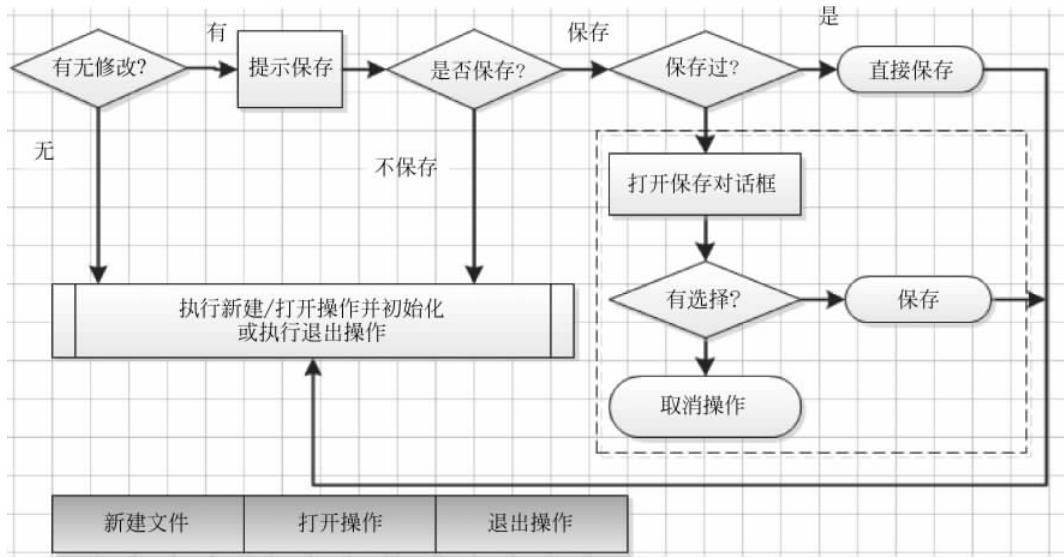


图 5.6 新建文件、打开文件和退出系统的流程图

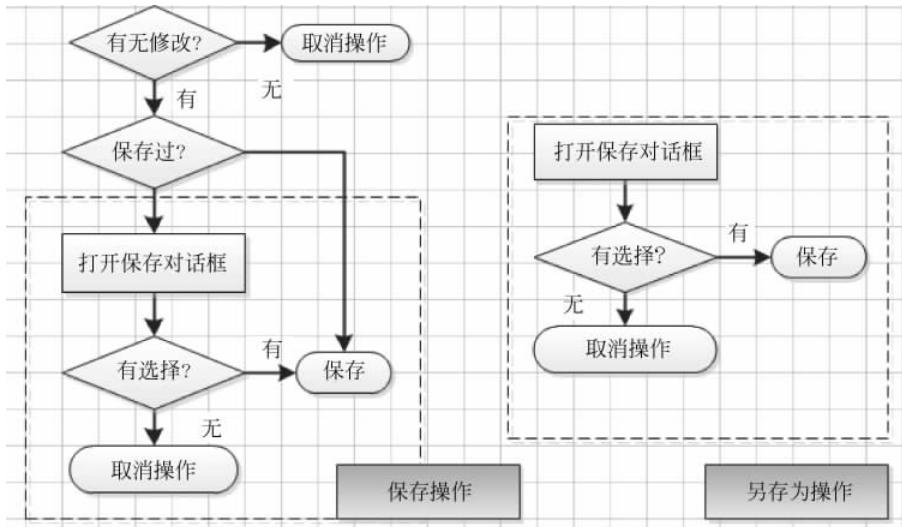


图 5.7 保存与另存为功能的流程图

■ 界面布局

在 Form1 窗体中添加一个 Panel 控件，并将 5 个按钮控件作为子控件添加到其中，修改各个按钮的名称和文本如图 5.8 所示，最后添加一个带滚动条的多行文本框，填充窗体剩下的空间。理解本例功能实现后，请读者自行设计菜单来代替各个按钮的功能。

为了对代码有个整体了解，下面先给出如图 5.9 所示的代码清单。

代码清单说明如下。

(1) 窗体类中定义的 isChange 和 fileName 两个字段，分别代表当前文本框内容是否发生改变，以及文件当前的保存位置。



图 5.8 界面布局

```

using System.IO;
namespace MyFile
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            public bool isChange = false; //是否修改了内容
            public string fileName = ""; //当前文件保存位置
            //新建、打开和退出操作之前，都要判断是否修改了上一次编辑的内容
            void SaveChange()...
            //执行实际的保存操作
            void SaveFile(bool isSaveAs=false)...
            //新建文件按钮 btNewFile 事件处理过程
            private void btNewFile_Click(object sender, EventArgs e)...
            //打开文件按钮 btOpenFile 事件处理过程
            private void btOpenFile_Click(object sender, EventArgs e)...
            //退出按钮 btExist 事件处理过程
            private void btExist_Click(object sender, EventArgs e)...
            //保存按钮 btSave 事件处理过程
            private void btSave_Click(object sender, EventArgs e)...
            //另存为按钮 btSaveAs 事件处理过程
            private void btSaveAs_Click(object sender, EventArgs e)...
            //文本框内容有修改时触发的事件
            private void textBox1_TextChanged(object sender, EventArgs e)...
        }
    }
}

```

图 5.9 代码清单

(2) SaveChange 是自定义方法，在新建、打开和退出操作之前，都需要执行的过程，即如果内容有修改，提示用户是否保存，只有需要保存时才调用保存的方法。

(3) SaveFile 为自定义方法，真正实现将当前编辑的内容保存到指定的文件中。

其余为事件处理过程，其中：

(1) textBox1_TextChanged 事件处理过程，用于检测当前文本框内容是否发生了改变。

(2) btNewFile_Click 为“新建文件”按钮的事件处理过程。

(3) btOpenFile_Click 为“打开文件”按钮的事件处理过程。

(4) btSaveAs_Click 为“另存为...”按钮的事件处理过程。

(5) btSave_Click 为“保存”按钮的事件处理过程。

(6) btExist_Click 为“退出”按钮的事件处理过程。

程序代码

```
//省略自动生成的 using 部分
using System.IO; //File 所在命名空间
namespace MyFile
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); } //构造方法
        public bool isChange = false; //1. 是否修改了内容
        public string fileName = ""; //2. 当前文件保存位置
        //3. 自定义方法
        //新建、打开和退出操作之前，都要判断是否修改了上一次编辑的内容
        void SaveChange()
        {
            if (!isChange) return; //没有修改则无须保存
            if (MessageBox.Show("文件已经修改，是否保存?", "确认",
                MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                SaveFile(); //4. 保存操作
            }
        }
        //5. 自定义方法：保存内容到文件
        void SaveFile(bool isSaveAs = false)
        {
            if (fileName == "" || isSaveAs) //6. 两种情况下需要打开保存对话框
            {
                SaveFileDialog sfd = new SaveFileDialog(); //创建保存对话框对象
                sfd.FileName = ""; //初始化
                if (sfd.ShowDialog() == DialogResult.OK) //选择了文件
                    fileName = sfd.FileName;
                else //取消了选择文件
                    return;
            }
            File.WriteAllText(fileName, textBox1.Text); //7. 保存文件
            isChange = false; //初始化为未修改状态
        }
        //8. "新建文件"按钮 btNewFile 事件处理过程
        private void btNewFile_Click(object sender, EventArgs e)
        {
            SaveChange(); //初始化
            textBox1.Text = ""; //9. 清空文件名，新建的文件没有文件名
            fileName = ""; //10. 处于非修改状态
        }
        //11. "打开文件"按钮 btOpenFile 的事件处理过程
        private void btOpenFile_Click(object sender, EventArgs e)
        {
            SaveChange();
        }
    }
}
```

```

//12. 选择文件并将读取内容显示到文本框
 OpenFileDialog ofd = new OpenFileDialog();
 ofd.FileName = "";
 if (ofd.ShowDialog() == DialogResult.OK)
 {
     fileName = ofd.FileName; //13. 保存当前编辑的文件名
     //14. 读取文件内容并显示到文本框
     textBox1.Text = File.ReadAllText(fileName);
 }
 isChange = false; //处于无修改状态
 }

//15. "退出"按钮 btExist 的事件处理过程
private void btExist_Click(object sender, EventArgs e)
{
    SaveChange();
    Close();
}

//16. "保存"按钮 btSave 的事件处理过程
private void btSave_Click(object sender, EventArgs e)
{
    if (isChange) SaveFile(); //如有修改,则保存
}

//17. "另存为..."按钮 btSaveAs 的事件处理过程
private void btSaveAs_Click(object sender, EventArgs e)
{
    SaveFile(true); //这里将直接保存,并初始化状态
}

//18. 文本框内容改变时触发的事件
private void textBox1_TextChanged(object sender, EventArgs e)
{
    isChange = true;
}

}//Form1 类文件结束
}//命名空间结束

```

代码说明

(1) 注释 1、注释 2：两个自定义字段 isChange 和 fileName。isChange 用于指示文本框内容是否有改变。isChange 的值在文本框的 TextChanged 事件中改变，触发了该事件，则代表文本框的内容已改变；fileName 代表当前保存的文件名及位置，初始值为空，表示当前文件未保存过。

(2) 注释 3：自定义方法 SaveChange。实现在文本框的内容改变时，让用户确认是否保存修改的内容。如果确认修改，则调用 SaveFile 方法实现文件的保存。执行“新建文件”“打开文件”和“退出”操作之前都会调用这段代码，因此设计为方法。

(3) 注释 5：自定义方法 SaveFile。isSaveAs 为可选参数，只有在“另存为”操作时才需要设置为 true，表示必须打开保存文件对话框来选择保存文件的位置，其他操作可以省略该参数，表示仅根据文件名来决定是否打开保存文件对话框。

(4) 注释 6：表示在执行“保存”操作时，如果当前文件位置没有指定，或者在执行“另存为”操作时(isSaveAs 为 true)，需要打开保存文件对话框的。其中：

① 语句“SaveFileDialog sfd = new SaveFileDialog()”表示动态创建保存文件对话框对象 sfd，当然也可以直接使用保存文件对话框控件来实现。

② 语句“if(dlg.ShowDialog() == DialogResult.OK)”表示如果在保存文件对话框中选择了文件，则必定返回 DialogResult.OK 枚举值；否则表示没有选择文件，此时应当中断代码执行，直接退出该方法。

③ 语句“fileName = dlg.FileName;”表示将选择的文件名保存到变量 fileName 中，更新文件保存的位置。如果用户在保存文件对话框中没有选择文件，则通过 return 语句取消操作，不再执行 if 结构中的语句。注释 7 中的 WriteAllText 方法实现将文本框的内容写入到新文件，同时恢复 isSave 的值为 false。

(5) 注释 8：“新建文件”按钮 btNewFile 的事件处理过程。通过 SaveChange 方法实现在该操作之前，判断内容有无修改，提示用户保存文件并实现文件的保存，然后为“新建文件”功能设置初始状态。

(6) 注释 11：“打开文件”按钮 btOpenFile 的事件处理过程，通过 SaveChange 方法实现在该操作之前，判断内容有无修改，提示用户保存文件并实现文件的保存，然后动态创建打开文件对话框对象，让用户选择要文件，并通过 ReadAllText 方法实现将文件内容显示在文本框中。

(7) 注释 15：“退出”按钮 btExit 的事件处理过程。通过 SaveChange 方法实现在退出操作之前，判断内容有无修改，提示用户保存文件并实现文件的保存，然后关闭窗体，退出程序。

(8) 注释 16：“保存”按钮 btSave 的事件处理过程。在文本框内容有修改的情况下，调用 SaveFile 方法来实现保存功能。由于方法参数是可选参数，使用其默认值 false，在该方法中，仅需通过文件位置来决定是否打开保存文件对话框。

(9) 注释 17：“另存为...”按钮 btSaveAs 的事件处理过程。调用 SaveFile 方法来实现保存功能，必须将参数 isSaveAs 设置为 true，强制打开保存文件对话框，并用新的文件位置代替当前的文件位置。

(10) 注释 18：文本框的 TextChanged 事件处理过程。将文件当前状态 isChange 设置为 true，表示文件当前编辑的内容已被修改。

▶ 运行结果

按 F5 键运行程序，在文本框中输入内容，如果单击“新建文件”按钮，将会看到如图 5.10 所示的运行结果。

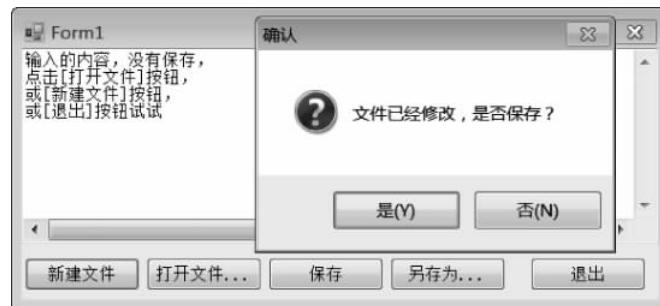


图 5.10 运行结果

5.2 FileInfo 类

FileInfo 类具有与静态类 File 部分相同的功能,它也提供了对指定文件创建、复制、删除、移动和协助创建文件流对象的功能,但 FileInfo 是非静态类,必须创建实例使用其方法和属性。如果需要检测文件的某些信息,例如文件名、扩展名、文件目录名、文件大小等,对于这些 File 类没有的属性,可以使用 FileInfo 类来取得。在使用上也可以把 FileInfo 类当作 File 类的一个补充,FileInfo 类常用的属性如表 5.2 所示。

表 5.2 FileInfo 常用属性

属性名称	说 明
FullName	获取目录或文件的完整目录
DirectoryName	获取表示目录的完整路径的字符串
Name	获取文件名
Extension	获取表示文件扩展名部分的字符串
Length	获取当前文件的大小(字节)
IsReadOnly	获取或设置确定当前文件是否为只读的值
LastAccessTime	获取或设置上次访问当前文件或目录的时间
LastWriteTime	获取或设置上次写入当前文件或目录的时间

应用实例 5.3 取得指定文件的相关信息。

新建项目,保存为 MyFileInfo。

界面布局

在 Form1 窗体上添加三个标签,一个文本框和一个按钮,保留默认的名称,修改各个控件文本如图 5.11 所示。



图 5.11 界面布局

```
//为 button1 按钮添加以下事件处理代码
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog(); //创建对话框对象
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        textBox1.Text = dlg.FileName; //在文本框中显示文件名
        FileInfo fi = new FileInfo(dlg.FileName); //创建 FileInfo 对象
        string s = "文件绝对位置: " + fi.FullName + "\r\n"; //将属性连接成字符串
        s += "路径名: " + fi.DirectoryName + "\r\n";
        s += "文件名: " + fi.Name + "\r\n";
        s += "扩展名: " + fi.Extension + "\r\n";
        s += "文件大小: " + fi.Length + "字节";
        label3.Text = s;
    }
}
```

代码说明

首先动态创建了打开文件对话框对象 dlg, 将在对话框中选择的文件名显示在文本框中, 并以文件名(全路径)作为参数, 创建 FileInfo 对象 fi, 将所需的 fi 的各种属性连接成字符串后显示在标签 label3 中。由于 FileInfo 类是非静态类, 使用时需要创建类的实例(对象), 并以文件的路径作为构造方法的参数。

▶ 运行结果

按 F5 键运行程序, 单击“[...]”按钮选择任意文件, 将看到类似如图 5.12 所示的运行结果。



图 5.12 运行结果

5.3 Directory 类

Directory 类提供了用于创建、移动和枚举目录和子目录的静态方法。主要方法见表 5.3。

表 5.3 Directory 主要方法

方法名称	说 明
CreateDirectory	创建指定路径中的所有目录
Delete	删除指定的目录
Exists	确定给定路径是否引用磁盘上的现有目录
Move	将文件或目录及其内容移到新位置
GetCreationTime	获取目录的创建日期和时间
GetCurrentDirectory	获取应用程序的当前工作目录
GetDirectories	获取指定目录中子目录的名称
GetFiles	返回指定目录中的文件的名称
GetDirectoryRoot	取得根目录
GetFileSystemEntries	返回指定目录中所有文件和子目录的名称
GetLastAccessTime	返回上次访问指定文件或目录的日期和时间
GetLastWriteTime	返回上次写入指定文件或目录的日期和时间
GetLogicalDrives	检索此计算机上格式为“<驱动器号>:\”的逻辑驱动器的名称
SetCreationTime	为指定的文件或目录设置创建日期和时间
SetCurrentDirectory	将应用程序的当前工作目录设置为指定的目录
SetLastAccessTime	设置上次访问指定文件或目录的日期和时间
SetLastWriteTime	设置上次写入目录的日期和时间

以下是 Directory 类常用的方法。

(1) CreateDirectory 方法。

- ① 方法签名: DirectoryInfo CreateDirectory(string path)。
- ② 功能: 创建指定的目录。如果目录已存在, 则不执行任何操作。
- ③ 参数: 参数 path 表示要创建的目录。可以包含多级子目录。

④ 返回值： DirectoryInfo 类型结果，该类型将在 5.4 节介绍，一般可忽略该返回值。

(2) Delete 方法。

① 方法签名： void Delete(string path[, bool recursive])。

② 功能：删除指定的目录。

③ 参数：第一个参数表示要删除的目录。第二个参数表示是否删除子目录或目录下所有文件，可选，如果值为 true，所有的子目录和文件都将被删除，否则只能删除空目录，默认值为 false。

④ 返回值：无。

(3) Exist 方法。

① 方法签名： bool Exists(string path)。

② 功能：判断指定的目录是否存在。

③ 参数：需要判断的目录。

④ 返回值： bool 类型，如果目录存在，返回 true，否则返回 false。

(4) GetCurrentDirectory 方法。

① 方法签名： string GetCurrentDirectory()。

② 功能：获取应用程序当前所在的目录。

③ 参数：无。

④ 返回值： string 类型，当前应用程序所在的目录。

(5) GetDirectories 方法。

① 方法签名： string[] GetDirectories(string path)。

② 功能：获取指定目录中的子目录(全路径)。

③ 参数：指定的目录。

④ 返回值： string 类型数组，包含指定目录下当前子目录的字符串数组。

(6) GetFiles 方法。

① 方法签名： string[] GetFiles(string path)。

② 功能：返回指定目录中所有文件(全路径)。

③ 参数：指定的目录。

④ 返回值： string 类型数组，包含指定目录下当前所有文件的字符串数组。

(7) GetDirectoryRoot 方法。

① 方法签名： string GetDirectoryRoot(string path)。

② 功能：取得指定目录中的根目录。

③ 参数：指定的目录。

④ 返回值： string 类型，表示根目录的字符串。

应用实例 5.4 创建多级文件夹。

```
Directory.CreateDirectory ("c:\\AAA\\BBB\\CCC");
```

说明：CreateDirectory 方法功能是建立指定的文件夹。参数 1 必须指定文件夹的路径。如果文件夹已经存在，则忽略，不会重复创建；否则将自动建立多级文件夹。

应用实例 5.5 取得应用程序当前运行位置。

```
string s = Directory.GetCurrentDirectory();
```

说明：GetCurrentDirectory 方法的功能是取得应用程序当前的运行位置，其结果与 Application.StartupPath 属性值一致。

应用实例 5.6 列出指定文件夹中所有的子文件夹。

```
string[] paths = Directory.GetDirectories("c:\\\\MYOAA", "A *",
SearchOption.AllDirectories);
```

说明：这是 GetDirectories 方法重载形式之一，使用该重载形式可以取得指定文件夹中所有的子文件夹。

- (1) 参数 1：指定文件夹的位置，该文件夹必须存在，否则将出现异常。
- (2) 参数 2：指定查找文件夹的通配符，这里是指取得名称以“A”为首字母的所有文件夹，类似于我们在 Windows 中查找文件时使用的通配符，如“* A *”表示取得名称包含 A 的文件夹。“*”通配符代表任意个字符，而“?”通配符代表一个字符。
- (3) 参数 3：枚举类型 SearchOption 的选项，有两个取值。取值为 AllDirectories 时，表示取得指定文件夹下所有子文件夹，包括子文件夹的下一级文件夹；另外一个取值为 TopDirectoryOnly，表示仅取得指定文件夹中当前的子文件夹，不包含下一级子文件夹。
- (4) 注意：参数 2、参数 3 可以省略。同时省略代表只获取指定目录中的当前所有子文件夹，不包含子文件夹的下一级文件夹；只省略参数 3，表示只获取指定目录中符合搜索条件的文件夹，也不包含子文件夹的下一级文件夹。

GetFiles 方法也有不同的三种重载方式，使用方法与 GetDirectories 方法相似。不同的是 GetFiles 方法是根据搜索条件取得目录中包含的文件。两者的返回值都是字符串数组，包含文件的绝对位置。另外，移动目录方法 Move 只能在同一根目录下进行。

应用实例 5.7 设计一个 CopyTo 方法，实现文件夹的复制功能。

代码片段

```
//实现将源文件夹所有内容，复制到目标文件夹。
//1. 将源文件夹 source 的内容，复制到目标位置 dest
void CopyTo(string source, string dest)
{
    string[] paths                         //2. 保存源文件夹所有子文件夹
    = Directory.GetDirectories(source, " * ", SearchOption.AllDirectories);
    string[] curFiles = Directory.GetFiles(source); //3. 取得文件列表

    Directory.CreateDirectory(dest);           //4. 建立 dest 目录

    //先复制当前目录下的文件
    foreach (string file in curFiles)
    {
        FileInfo fi = new FileInfo(file);      //5. 仅为取得文件名创建 FileInfo 对象
        File.Copy(file, dest + "\\\" + fi.Name, true); //6. 以覆盖方式复制到目标位置
    }
}
```

```

        }
        //依次在目标位置创建同名子文件夹，并复制文件到创建的目标位置
        foreach (string path in paths)
        {
            //7.组合路径
            string pathName = dest + "\\\" + path.Replace(source + "\\\", "");
            Directory.CreateDirectory(pathName);           //8.创建目标位置文件夹

            string[] files = Directory.GetFiles(path);    //9.取得当前文件夹中的文件
            foreach (string file in files)
            {
                FileInfo fi = new FileInfo(file);          //10.取得文件名
                //以覆盖方式复制到目标位置
                File.Copy(file, pathName + "\\\" + fi.Name, true); //11.复制文件
            }
        }
    }
}

```

代码说明

(1) 注释 1：在 CopyTo 方法的签名中，参数 source 代表来源文件夹，参数 dest 代表目标文件夹。该方法实现将 source 位置的所有内容，包括文件和子文件夹，复制到位置 dest 中。结果两者包含的内容完全一致。

(2) 注释 2：GetDirectories 方法实现取得源文件夹中所有的子文件夹并保存到数组 paths 中。“*”代表任意名称的文件夹，SearchOption.AllDirectories 参数表示要取得所有子文件夹。

(3) 注释 3：为了将源文件夹中的当前文件复制到目标位置中，使用仅一个参数的 GetFiles 方法取得源文件中当前包含的文件，保存到字符串数组 curFiles。

(4) 注释 4：CreateDirectory 方法可以重复调用，会自动判断文件夹是否存在，存在则不创建新文件夹。

(5) 注释 5：创建 FileInfo 对象的目的，是为了方便通过其 Name 属性取得给定路径中的文件名部分，以同样的名称将其复制到目标位置。

(6) 注释 6：Copy 方法中的参数 3 设置为 true，这样当目标位置文件存在时，则覆盖；否则将出现“文件已存在”的异常。

(7) 注释 7：Replace 方法将源文件夹位置部分 source 去掉，仅取得其下一级文件夹的相对位置，目的是在目标位置建立与其同名的子文件夹。如源文件夹 source 为 c:\test，当前路径 path 的值为 c:\test\A\B，执行 path.Replace(source + "\\\", "") 语句后，path 的结果为 A\B。这样，可以在目标位置也建立同样相对位置的子文件夹。

(8) 本例在测试时，需确保源文件夹 source 存在。

5.4 DirectoryInfo 类与 Path 类

DirectoryInfo 类提供了与 Directory 类相似的功能，主要针对指定的路径，提供用于创建、移动和枚举目录和子目录的实例方法。Directory 是静态类，而 DirectoryInfo 是非静态

类,需要创建实例来使用其属性和方法。如果要取得与目录相关的信息,可以用 DirectoryInfo 来补充,如取得文件夹的名称、根目录、父目录等。DirectoryInfo 常用的属性和方法分别见表 5.4 和表 5.5。

表 5.4 DirectoryInfo 常用的属性

属性名称	说 明
FullName	获取目录或文件的完整目录
Name	获取此 DirectoryInfo 实例的名称
Parent	获取指定子目录的父目录
Root	获取路径的根部分
Exists	获取指示目录是否存在值
LastAccessTime	获取或设置上次访问当前文件或目录的时间
LastWriteTime	获取或设置上次写入当前文件或目录的时间

表 5.5 DirectoryInfo 常用的方法

方法名称	说 明
Create	创建目录
CreateSubdirectory	在指定路径中创建一个或多个子目录
Delete	指定是否删除子目录和文件
GetDirectories	返回当前目录的子目录
GetFiles	返回当前目录的文件列表
MoveTo	将当前文件夹内容移动到新位置

应用实例 5.8 使用 DirectoryInfo 类创建指定目录和其多级子目录,并将新创建的目录的各个部分属性显示到标签控件中,最后删除创建的目录(包括子目录和文件)。

新建项目,保存为 MyDirectoryInfo。

■ 界面布局

在 Form1 窗体添加一个标签和一个按钮,保留默认名称,界面布局如图 5.13 所示。

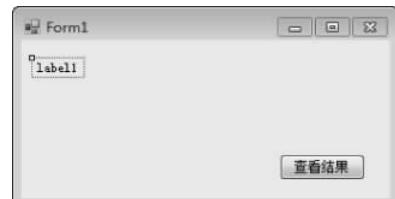


图 5.13 界面布局

■ 代码片段

```
//为"查看按钮"button1 添加事件处理过程,并添加如下代码
private void button1_Click(object sender, EventArgs e)
{
    string s = "";
    string path = "c:\\\\abc\\\\12345";
    DirectoryInfo di = new DirectoryInfo(path);           //1. 创建对象
    di.Create();                                         //2. 如果目录不存在,创建该目录;否则,自动忽略创建操作
    s = "文件夹是否存在:" + di.Exists + "," + Directory.Exists(path) + "\\r\\n"; //3.
    di.CreateSubdirectory("A1\\\\A2\\\\A3");                //4. 使用相对位置,创建子目录
    s += "文件夹是否存在:" + di.Exists + "," + Directory.Exists(path) + "\\r\\n";
    s += "完整路径:" + di.FullName + "\\r\\n";
    s += "根目录:" + di.Root + "\\r\\n";
}
```

```

    s += "父目录: " + di.Parent + "\r\n";
    s += "文件夹名: " + di.Name + "\r\n";
    di.Delete(true); //6. 删除目录,包括子目录和文件
    s += "文件夹是否存在:" + di.Exists + "," + Directory.Exists(path);
    label1.Text = s;
}

```

代码说明

- (1) 注释 1: 创建 DirectoryInfo 对象时指定要创建的目录 path。
- (2) 注释 2 和注释 4: Create 方法用于创建在构造方法参数中指定的目录,而如果在该目录中创建子目录,则使用 CreateSubdirectory 方法。
- (3) 注释 3: DirectoryInfo 对象通过属性 Exists 来判断目录是否存在,而 Directory 对象是通过静态方法 Exists 来判断目录是否存在,两者结果相同。一般对目录的操作使用 Directory 静态方法,而要取得目录相关信息,则使用 DirectoryInfo 对象的属性来获取。
- (4) 注释 5: 父目录属性 Parent 的值是 DirectoryInfo 类型,可以取得目录的相关信息。
- (5) 注释 6: 对象 di 对应的目录为 c:\abc\12345,调用 Delete 方法后,文件夹 12345 及其子文件夹将被删除。

运行结果

按 F5 键启动程序,单击“查看结果”按钮,将看到如图 5.14 所示的运行结果。

Path 类是静态类,其作用是对满足路径格式的字符串进行操作,比如取得给定路径的路径名、文件名、文件扩展名、根目录和父目录,以及可以合并多个字符串为路径等常见的操作。Path 类的大多数成员不与文件系统进行交互,且不验证路径字符串所指定的目录是否存在。由于使用方法比较简单,这里仅列出常用的方法(见表 5.6)供读者参考和自行上机实践,读者也可以借助 MSDN 了解更多的细节。

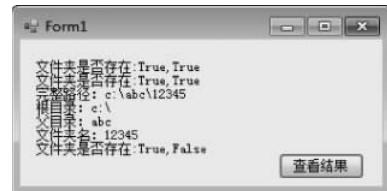


图 5.14 运行结果

表 5.6 Path 常用静态方法

方法名称	说 明
Combine	将多个字符串组合成一个路径
GetDirectoryName(path)	返回指定路径字符串 path 的目录信息,如果 path 表示根目录或为 null,则该目录信息为 null
GetExtension	返回指定的路径字符串的扩展名
GetFileName	返回指定路径字符串的文件名和扩展名
GetFullPath	返回指定路径字符串的绝对路径
GetPathRoot	获取指定路径的根目录信息
HasExtension	确定路径是否包括文件扩展名

典型案例 一个文件选择对话框的设计

设计一个简单的文件选择对话框,类似 OpenFileDialog 控件的功能。当选择根目录时,在一个列表框显示当前根目录下的子目录,并在另一个列表框列出该目录下的文件;当

双击子目录时,显示其下一级的子目录,同时将该子目录的文件显示到另外的列表框。此外,还可以从当前目录返回到上一级目录,运行界面如图 5.15 所示。

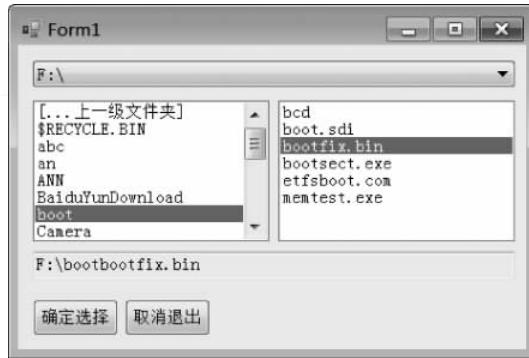


图 5.15 自定义选择文件对话框

1. 案例分析

根目录是指计算机中存在的逻辑驱动器盘符,包括硬盘、移动磁盘、光驱等盘符,例如 C:\、D:\ 等。要取得逻辑驱动器盘符信息,可以使用 Directory 类的静态方法 GetLogicalDrives 获取,它返回的是一个代表盘符的字符串数组。

获取盘符后,使用 Directory 类的静态方法 GetDirectories,得到给定目录的子目录; 使用 Directory 类的静态方法 GetFiles 得到给定目录下的文件,将这些得到的字符串数组信息显示到对应的列表框中。

只要保存了当前目录,在列表框双击该目录时,将选中的目录名和当前目录通过 Path 的静态方法 Combine 连接起来,作为新的当前目录,这样就实现了列举新当前目录的子目录和文件的功能。

当需要返回上一级目录时,取得当前目录的父目录,将其作为当前目录,这样,通过实现列举新当前目录的方法,从而更新列表框的显示。

2. 设计思路和步骤

参照图 5.15 和图 5.16,本案例设计思路和步骤如下。

- (1) 设计一个 GetDriver 方法,用于取得逻辑驱动器信息,显示在组合框中。
- (2) 设计一个 ListDirectory 方法,根据给定目录,取得该目录下的当前子目录,显示在列表框 listBox1 中。
- (3) 设计一个 ListFile 方法,根据给定目录,在列表框 listBox2 中显示该目录下的文件。
- (4) 在列表框 listBox1 触发选择下标改变事件时,取得选中项目录下的所有文件,显示在列表框 listBox2 中。
- (5) 在列表框 listBox1 触发双击事件时,取得选中项目录的子目录,更新列表框 listBox1 的显示。

- (6) 在列表框项 listBox1 中双击“[…上一级文件夹]”项时,显示上一级目录的子目录。
- (7) 在列表框 listBox2 触发选择下标改变事件时,取得选中的文件,显示在文本框中。
- (8) 运行时,取得逻辑驱动器信息填充组合框,并默认选择第一项。

3. 具体实现

新建项目,保存为 MyOpenFileDialog。

■ 界面布局

在 Form1 窗体中添加一个组合框,用于显示根目录,样式设置为 DropDownList; 添加两个列表框,一个用于显示目录名,一个用于显示文件名; 添加一个文本框,用于显示在列表框中选中的文件,最后添加两个操作按钮。保留控件默认的名称,界面布局如图 5.16 所示。

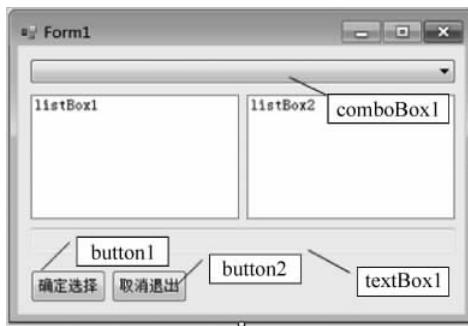


图 5.16 界面布局

■ 代码片段

```
//在窗体类中添加以下两个字段:
string CurDirectory = ""; //记录当前文件夹位置
string CurFile = ""; //当前选择文件的路径
```

■ 代码说明

由于对选择的文件,或者文件夹,只显示名称,而不显示路径,因此需要定义变量来暂存其路径。

- (1) GetDriver 方法设计。

```
void GetDriver()
{
    comboBox1.Items.Clear();
    string[] driverName = Directory.GetLogicalDrives();
    foreach (string s in driverName)
    {
        comboBox1.Items.Add(s);
    }
    comboBox1.SelectedIndex = 0;
}
```

代码说明

先初始化组合框,通过 GetLogicalDrives 方法取得代表驱动器盘符信息的字符串数组,包括硬盘、光盘和移动磁盘的盘符,在列举循环中将数组元素添加到组合框。最后设置当前选中项的下标为 0,此时将触发组合框的 SelectedIndexChanged 事件,在该事件中实现列举当前根目录下的子目录。

(2) ListDirectory 方法设计。

```
void ListDirectory(string path)
{
    //每次列出目录,初始化列表框
    listBox1.Items.Clear();
    listBox1.Items.Add("[…上一级文件夹]");
    textBox1.Text = "";
    string[] subPath = Directory.GetDirectories(path);
    foreach (string s in subPath)                      //只显示文件夹名,不显示路径
    {
        DirectoryInfo di = new DirectoryInfo(s);
        listBox1.Items.Add(di.Name);                  //取得文件夹名,不包含路径名
    }
}
```

代码说明

① ListDirectory 方法的功能是以给定的目录(包括根目录)作为参数,取得其子目录并显示在列表框中。

② 在 ListDirectory 方法中首先初始化列表框,用于重复调用该方法时刷新列表项。添加“[…上一级文件夹]”项作为列表框的第一项,用于执行返回到上一级目录的操作项。

③ 在列举子目录数组的代码中,创建 DirectoryInfo 对象的目的是为了使用其 Name 属性来取得目录名。完整目录在调用该方法的代码中,已保存到 CurDictionary 变量。

(3) ListFile 方法设计。

```
void ListFile(string path)
{
    listBox2.Items.Clear();
    string[] files = Directory.GetFiles(path);
    foreach (string s in files)
    {
        FileInfo fi = new FileInfo(s);
        listBox2.Items.Add(fi.Name);                //只显示文件名
    }
}
```

代码说明

① ListFile 方法实现根据给定的目录,列出该目录下的文件,不包含子目录的文件。

② 同样,为了刷新显示列表框的项,在 ListFile 方法中,先初始化列表框,再通过 GetFiles 方法取得给定目录下包含文件路径的字符串数组。为了在列表框中仅显示文件

名,创建 FileInfo 对象 fi,使用其 Name 属性来取得路径中的文件名。

(4) 窗体 Load 事件处理过程。

创建窗体 Load 事件处理过程,并添加以下代码。

```
private void SelectFileDialog_Load(object sender, EventArgs e)
{
    GetDriver();
}
```

代码说明

在这里仅包含自定义方法 GetDriver,实现程序启动时,取得逻辑驱动器信息添加到组合框中。

(5) 组合框的 SelectedIndexChanged 事件处理过程。

创建组合框 comboBox1 的 SelectedIndexChanged 事件处理过程,并添加以下代码。

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    //选择了驱动器,得到根目录,如 C:\ 
    CurDictionary = comboBox1.SelectedItem.ToString();
    ListDirectory(CurDictionary);
}
```

代码说明

在组合框的选择项发生改变时将触发 SelectedIndexChanged 事件。在该事件中取得代表根目录的选择项,作为 ListDirectory 方法的参数,更新列表框的显示。

(6) listBox1 的 SelectedIndexChanged 的事件处理过程。

双击界面中的列表框 listBox1,在创建的事件处理过程中添加如下代码。

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox1.SelectedIndex < 0) return;
    CurFile = CurDictionary; //默认第一项
    if (listBox1.SelectedIndex != 0) //如果不是第一项
    {
        CurFile = Path.Combine(CurDictionary, listBox1.SelectedItem.ToString());
    }
    ListFile(CurFile);
}
```

代码说明

① 在列表框的选择项发生改变时触发该事件,该过程实现在列表框 listBox1 选中某项时,将该目录下的所有文件显示到列表框 listBox2 中。

② 由于第一项代表的是返回上一级目录,而 CurDictionary 变量保存的是当前显示目录的父目录,因此,如果选择的不是第一项,就将 CurDictionary 与选中项组合成新的目录 CurFile,列举该目录下的所有文件;否则就显示 CurDictionary 下的文件。CurFile 表示要显示文件的当前目录。

③ Path 的静态方法 Combine 实现将两个字符串组合成满足路径格式的新字符串。

(7) ListBox1 的 DoubleClick 事件处理过程。

添加 listBox1 列表框的 DoubleClick 事件,在该事件处理过程中添加如下代码。

```
private void listBox1_DoubleClick(object sender, EventArgs e)
{
    if (listBox1.SelectedIndex < 0) return;

    if (listBox1.SelectedIndex != 0) //双击代表准备进入下一级目录的项
    {
        CurDirectory = Path.Combine(CurDirectory,
            listBox1.SelectedItem.ToString());
        //保存当前选择的文件夹,准备列出其子文件夹
    }
    else //如果双击了"…上一级文件夹"表示退回到上一级文件夹
    {
        if (!IsRoot(CurDirectory)) //如果不是根目录,取得父路径
            CurDirectory = Path.GetDirectoryName(CurDirectory);
        //取得父路径,这个文件夹的路径,不包含这个文件夹
    }
    //刷新显示当前选择的文件夹下的子文件夹
    ListDirectory(CurDirectory);
}
```

代码说明

① 该事件过程实现在列表框中双击列表项时,显示选中项的子目录,并刷新该列表框显示。

② “listBox1.SelectedIndex != 0;”语句表示如果选中的不是第一项,则表示选中项是准备用来作为当前目录的项。Combine 方法将当前保存在 CurDirectory 变量中的目录,与选中项组合成新的目录,并更新 CurDirectory 的值,此时,CurDirectory 就是双击后准备显示其子目录的当前路径,通过 ListDirectory 方法列出其子目录,更新列表框的显示。

③ 如果选中的项是列表栏的第一项,该项表示返回上一级目录,上一级目录也就是保存在变量 CurDictionary 当前值中的父目录,可以通过 Path 类的 GetDirectoryName 方法取得其父目录。如果一个目录是根目录,那它没有父目录,这时当前目录就是根目录。

④ 使用 Directory 对象的 GetDirectoryRoot 方法取得根目录,如果一个目录的根目录是它本身,那么这个目录就是根目录。判断一个目录是否是根目录的功能可用自定义方法 IsRoot 来实现,IsRoot 方法代码如下。

```
bool IsRoot(string path)
{
    string root = Directory.GetDirectoryRoot(path);
    if (root.ToUpper() == path.ToUpper()) return true; //满足条件是根目录
    else return false; //否则不是根目录
}
```

(8) ListBox2 的 SelectedIndexChanged 事件处理过程。

双击界面中的列表框 listBox2，创建事件处理过程并添加如下代码。

```
private void listBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox2.SelectedIndex < 0) return;
    textBox1.Text = Path.Combine(CurFile, listBox2.SelectedItem.ToString());
}
```

代码说明

本过程实现将在列表框 listBox2 中选择的文件名，与当前文件路径组合成绝对路径，显示在文本框中。

运行结果

依次按上述步骤在窗体类文件中添加代码后，按 F5 键运行程序，并在列表框执行相应操作后，将看到类似如图 5.15 所示的运行结果。

如何让打开该窗体的其他窗体取得选中的文件呢？即在图 5.15 中单击“确定选择”按钮时，实现让窗体返回 DialogResult.OK 枚举值并自动关闭窗体，其他窗体根据该返回值，读取选择的文件；单击“取消选择”按钮时，让窗体返回 DialogResult.Cancel 枚举值并自动关闭窗体。这部分内容前面第 4 章已有介绍，这里留给读者自己实现。

5.5 FileStream 类

FileStream 类用于以字节为单位读写任何类型文件。FileStream 类支持对本地文件进行打开、读取、写入和关闭操作，并对其他与文件相关的操作系统句柄进行操作，如管道、标准输入和标准输出。读写操作可以指定为同步或异步操作。

FileStream 对象支持使用 Seek 方法对文件进行随机访问。Seek 允许将读取/写入位置移动到文件中的任意位置，这是通过字节偏移量来完成的，而字节偏移量是相对于查找参考点而言的，该参考点可以是基础文件的开始、当前位置或结尾，分别由 SeekOrigin 类的三个属性表示。

FileStream 只支持最基本的操作，把数据写入字节数组或者从字节数组写入文件中。FileStream 常用的属性和常用的方法分别见表 5.7 和表 5.8。

表 5.7 FileStream 类的常用属性

名称	含义
CanSeek	获取一个值，该值指示当前流是否支持查找
CanRead	获取一个值，该值指示当前流是否支持读取
CanWrite	获取一个值，该值指示当前流是否支持写入
Length	获取用字节表示的流长度
Position	获取或设置此流的当前位置

表 5.8 FileStream 类的常用方法

名 称	含 义
Read	从流中读取字节块并将该数据写入给定缓冲区中
ReadByte	从文件中读取一个字节，并将读取位置提升一个字节
Write	将一个字节写入文件流的当前位置
WriteByte	使用从缓冲区读取的数据将字节块写入该流
Close	关闭当前流并释放与之关联的所有资源(如套接字和文件句柄)
Seek	将该流的当前位置设置为给定值

1. 创建 FileStream 对象常用的三种方法

(1) 使用其构造方法。

```
FileStream fs = new FileStream(Path, FileMode, FileAccess)
```

其中,构造方法的参数说明如下。

- ① Path: 文件路径。
- ② FileMode: 枚举类型 FileMode 的成员之一,见表 5.9。

表 5.9 FileMode 枚举值

Append	打开现有文件并移动指针到文件尾,如果文件不存在则创建新文件
Create	指定操作系统应创建新文件。如果文件已存在,它将被改写
CreateNew	指定操作系统应创建新文件。文件存在则出现异常
Open	指定操作系统应打开现有文件。文件必须存在
OpenOrCreate	指定操作系统应打开文件(如果文件存在);否则,应创建新文件
Truncate	指定操作系统应打开现有文件。文件一旦打开,就将被截断为零字节大小

- ③ FileAccess: 枚举类型 FileAccess 的成员之一,见表 5.10。

表 5.10 FileAccess 枚举值

Read	对文件的只读访问。可从文件中读取数据
Write	文件的只写访问。可将数据写入文件
ReadWrite	对文件的读访问和写访问。可从文件读取数据和将数据写入文件

(2) 使用 File 静态类的方法。

```
FileStream fs = File.OpenRead(string Path);
```

以只读方式打开文件并返回 FileStream 对象。Path 为文件路径,文件必须存在。

```
FileStream fs = File.OpenWrite(string Path);
```

以只写方式创建文件并返回 FileStream 对象。Path 为文件路径,若文件已存在则覆盖。

(3) 使用文件对话框。

另外一种方式是使用 OpenFileDialog 和 SaveFileDialog 控件的 OpenFile 方法。不需要指定任何参数。

① OpenFileDialog 的 OpenFile 方法,以只读方式打开文件并返回 FileStream 对象。

② SaveFileDialog 的 OpenFile 方法,以读写方式打开文件并返回 FileStream 对象。

由于 OpenFile 方法返回的是 Stream 对象,因此需要强制转换为 FileStream 类型使用。使用 OpenFile 方法获取 FileStream 对象的方式比较少用。

2. FileStream 对象的常用方法

以写方式创建 FileStream 对象后,可以使用 WriteByte 方法写一个字节数据到文件中,也可以使用 Write 方法将一个字节数组数据写入文件中。Write 方法的常用用法如下。

```
fs.WriteByte(byte[] buffer, int offset, int count);
```

① buffer: 写入文件的字节数组名。

② offset: 写入文件的偏移量。一般为 0,表示从数组 buffer 的第一个元素开始读取。

③ count: 写入文件的字节数。一般为字节数组 buffer 的长度,表示写入数组的全部内容。

与之相似,以读方式创建 FileStream 对象后,可以使用 ReadByte 方法从文件中读出一个字节数据,也可以使用 Read 方法读出字节数据保存到字节数组,并返回实际读取的字节数。Read 方法的常用用法如下:

```
int num = fs.ReadByte(byte[] buffer, int offset, int count);
```

表示从文件中读取 count 长度字节,保存到字节数组 buffer 中,从 buffer 数组下标为 offset 的位置开始存放。返回值 num 表示实际读取的字节数,如果返回值为 0,表示文件数据读取完毕,或者是空文件。一般 offset 取值为 0,表示读取的字节从数组的第一个元素位置开始存放,而 count 取值为 buffer 数组的长度。

应用实例 5.9 使用 FileStream 对象进行文件读写。使用字节方式读取 C:\A.bmp 图片文件内容,保存为 C:\B.bmp。

代码片段

```
FileStream fr = new FileStream("c:\\\\A.BMP", FileMode.Open, FileAccess.Read);
FileStream fw = new FileStream("c:\\\\B.BMP", FileMode.Create, FileAccess.Write);

long len = fr.Length;
for (int i = 0; i < len; i++)
{
    fw.WriteByte((byte)fr.ReadByte());
}

fr.Close();
```

```
fw.Close();
```

代码说明

- (1) 首先需要确保 A.bmp 文件存在,否则创建 fr 对象时将引发“未找到文件”的异常。
- (2) 以只读方式创建对象 fr,以新建文件方式创建对象 fw。
- (3) 通过 fr 对象的属性 Length 取得文件的长度,即文件的字节数。在循环体中逐个字节读出 A.bmp 文件的内容,写入到文件 B.bmp 中。注意 ReadByte 方法返回的 int 类型数据,而 WriteByte 方法需要 byte 类型参数,因此需要将 int 类型强制转换为 byte 类型。
- (4) 文件读写完毕,必须关闭文件。

注意: 如果文件读取完毕,则 ReadByte 方法将返回 -1。因此也可以通过判断该返回值是否是 -1 使用 while 循环结构来实现文件的读取。

应用实例 5.10 使用 FileStream 对象进行文件读写。使用字节数组方式读取 C:\A.bmp 图片文件的内容,保存为 C:\B.bmp。

代码片段

```
FileStream fr = new FileStream("c:\\A.BMP", FileMode.Open, FileAccess.Read);
FileStream fw = new FileStream("c:\\B.BMP", FileMode.Create, FileAccess.Write);
byte[] by = new byte[1024];
int num = fr.Read(by, 0, by.Length);
while (num > 0)
{
    fw.Write(by, 0, num);
    num = fr.Read(by, 0, by.Length);
}
fr.Close();
fw.Close();
```

代码说明

- (1) 与应用实例 5.9 的不同之处是将文件内容先读取到字节数组,然后将字节数组写入到文件。
- (2) “byte[] by = new byte[1024];”语句创建了长度为 1024B 的数组 by。数组的长度是任意的,数值越大循环次数越小,一般采用 1K 的倍数,如 2048、4096 等。
- (3) “int num=fr.Read(by, 0, by.Length);”语句是将读取的字节保存到字节数组 by,指定从数组下标为 0 的位置开始存放; by.Length 是 Read 方法每次从文件读取字节的长度; Read 方法的返回值 num 代表实际读取的字节数,如果返回值为 0,表示文件读取完毕或者指定的文件是空文件。
- (4) 需要注意的是,使用 Read 方法读取文件的字节数据时,读取的实际数据长度可能比要求读取的数据长度小(这种情况一般出现在第一次读取和最后一次读取时),因此,每次写入文件的长度是实际读取的长度。

应用实例 5.11 将文本数据写入文件后读取出来。

FileStream 只支持最基本的操作,把数据写入字节数组或者从字节数组写入文件中。

如果是用 FileStream 把数据保存在文件中,首先要把数据转换为 Byte 数组,然后调用 FileStream 的 Write 方法。同样,FileStream 的 Read 方法,返回的也是字节数组。

FileStream 类可以读写任何类型的文件,当然包括文本文件。但文本文件内容是字符串类型的数据,而 FileStream 对象的方法是对字节数据进行读写,这就需要将输入的字符串数据转换为字节数据保存,并且在读取数据时,将读取的字节数据转换为字符串显示。下面是读写文本文件的具体实现。

■ 界面布局

新建项目,保存为 MyFileStream。在窗体 Form1 中添加一个带滚动条的多行文本框和两个按钮,保留默认的名称,界面布局如图 5.17 所示。

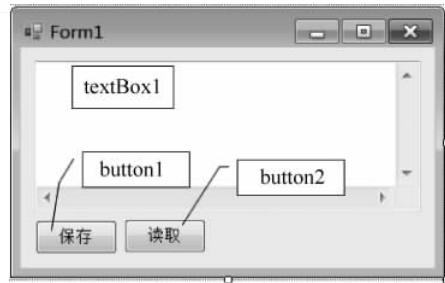


图 5.17 界面布局

■ 代码片段

```
//1."保存"按钮 button1 的事件处理过程
private void button1_Click(object sender, EventArgs e)
{
    byte[] buffer = Encoding.Unicode.GetBytes(textBox1.Text); //2. 转换为字节数组
    FileStream fw = new FileStream("C:\\\\test.txt", FileMode.Create, FileAccess.Write);
    fw.Write(buffer, 0, buffer.Length);
    fw.Close();
}

//3."读取"按钮 button2 的事件处理过程
private void button2_Click(object sender, EventArgs e)
{
    FileStream fr = new FileStream("C:\\\\test.txt", FileMode.Open, FileAccess.Read); //4. 强制转换
    int len = (int)fr.Length; //5. 定义字节数组长度
    byte[] buffer = new byte[len];
    fr.Read(buffer, 0, buffer.Length); //6. 读取内容保存到数组中
    fr.Close();

    string s = Encoding.Unicode.GetString(buffer); //7. 转换为字符串
    textBox1.Text = s;
}
```

■ 代码说明

(1) 注释 1: button1 的 Click 事件处理过程,实现将文本框中录入的内容转换为字节数组,然后将字节数组内容写入指定的文件。

(2) 注释 2 和注释 7: Encoding 类位于命名空间 System.Text,包含 Unicode 类的静态方法 GetBytes,可以将 Unicode 编码的字符串转换为字节数组; Unicode 类的静态方法 GetString 可以将字节数组转换为 Unicode 编码的字符串。字符串和字节数组之间的转换可以通过这两个方法来实现。如果要使用其他编码,如 UTF8,可以使用 Encoding.UTF8

类的静态方法实现字节数组与字符串之间的转换,注意字节数组与字符串之间的相互转换都要使用同一编码方式,否则会出现乱码。

(3) 注释 3: button2 的 Click 事件处理过程,实现将文件的内容读取出来显示到文本框。

(4) 注释 4、注释 5: 由于文件长度属性 Length 是 long 类型,因此需要转换为 int 类型,然后作为创建 buffer 字节数组的长度。

(5) 注释 6: 由于文本文件的内容比较少,可以使用文件长度作为创建字节数组的长度,然后一次将文件内容读取到字节数组。

▶ 运行结果

按 F5 键运行程序,在文本框中输入内容并单击“保存”按钮,将在 C 盘根目录下看到创建的文本文档 Test.txt; 单击“读取”按钮,文件内容将显示到文本框中。运行结果如图 5.18 所示。



图 5.18 运行结果

典型案例 文件合并与拆分

本案例实现将任意多个文件合并成一个新的文件,新的文件同时保存各个被合并文件的名称和大小信息,在需要拆分时,根据读取出的文件信息,将文件重新拆分成各个独立的文件。

1. 案例分析

合并文件的基本思路是,先依次读出各个需要合并的文件的基本信息:文件名和大小,按一定格式连接成“文件信息”字符串,它也是拆分文件的依据。然后使用 FileStream 对象创建一个新文件,该文件的结构如图 5.19 所示,其中,前 8 个字节保存的是代表“文件信息长度”的内容,后面紧跟每个文件的内容,文件尾保存的是“文件信息”的内容。

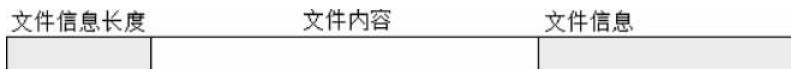


图 5.19 合并后文件结构示意图

文件信息的自定义格式为:

文件名|文件大小|文件名|文件大小|…

而“文件信息长度”是指文件信息字符串转换为字节数组后,字节数组的长度。因为保存到文件的数据都是字节类型,该长度也将转换为字节类型保存在文件头。

拆分文件的基本思路是,首先取得文件头前 8 个字节中代表“文件信息长度”的内容,根据该长度移动文件指针从文件尾部开始取得代表“文件信息”的字节内容,转换为字符串后再进行拆分,从而得到合并前各个文件的名称和大小,最后从文件内容部分取得各个文件的数据并保存为独立的文件。

将“文件信息”保存到合并文件的末尾的好处是改善性能,在以后要添加新的文件到这个合并文件时,不需要再创建新文件,直接从其原文件的“文件信息”位置开始添加新文件内容,再更新“文件信息长度”以及附加新的“文件信息”。而如果将“文件信息”放在“文件信息长度”字节后,在添加新文件时,由于“文件信息”内容会增加,写到该文件时,将覆盖合并文件原有内容,那样就必须重新创建新文件来保存合并后的文件。

2. 具体实现

新建项目,保存为 MyCombineFile。

■ 界面布局

在 Form1 窗体中添加两个按钮,一个用于将给定的一组文件合并成一个新文件,一个用于将合并后的文件拆分成合并前的文件。界面布局如图 5.20 所示。

1) 文件合并的功能实现

在 Form1 中添加一个自定义方法 CombineToFile, 实现将一组文件合并成新的文件。

■ 代码片段

```
//自定义方法,合并到指定的文件
private void CombineToFile(string[] files, string newFile)
{
    string fileInfoStr = "";

    //1.取得文件信息并连接成字符串,格式:文件名|大小|文件名|大小…文件名|大小
    foreach (string s in files)
    {
        FileInfo fi = new FileInfo(s);
        fileInfoStr += fi.Name + "|" + fi.Length + "|";
    }
    //2.去掉最后一个字符|
    fileInfoStr = fileInfoStr.Substring(0, fileInfoStr.Length - 1);
    //3.字符串转换为字节数组
    byte[] fileInfoBytes = Encoding.Unicode.GetBytes(fileInfoStr);
    long fileInfoStrLen = fileInfoBytes.LongLength;           //4.文件信息的字节长度

    //5.创建文件类对象,准备写入数据
    FileStream fs = new FileStream(newFile, FileMode.Create, FileAccess.Write);
    //6.二进制读写器对象,将简单类型数据作为字节写入到文件头
    BinaryWriter bw = new BinaryWriter(fs);
    bw.Write(fileInfoStrLen);                                //头部写入 long 类型长度字节

    //7.写入文件内容数据
    foreach (string s in files)
    {
        FileStream fsRead = new FileStream(s, FileMode.Open, FileAccess.Read);
        byte[] fileByte = new byte[fsRead.Length];
```



图 5.20 界面布局

```

        int realLen = fsRead.Read(fileByte, 0, fileByte.Length);
        fs.Write(fileByte, 0, realLen);
        fsRead.Close();
    }

    //8.文件尾部写入文件信息
    fs.Write(userInfoBytes, 0, userInfoBytes.Length);
    fs.Flush();                                     //9.确保写入剩余字节并关闭文件
    fs.Close();
}

```

代码说明

- (1) 自定义方法 CombineToFile 实现将一组文件 files 合并成新文件 newFile。
- (2) 注释 1、注释 2：取得 files 中的文件名和文件大小，组合成自定义格式：文件名|大小|文件名|大小|…，保存到 fileInfoStr 字符串中。注意在注释 2 中，去掉最后一个多余的“|”。
- (3) 注释 3、注释 4：将字符串 fileInfoStr 转换为字节数组，并将其 long 类型的长度属性保存到 fileInfoStrLen 变量中。数组的 LongLength 属性表示 long 类型的长度，而 Length 属性表示 int 类型的长度。
- (4) 注释 5：创建 FileStream 对象 fs，准备将数据以二进制方式写入文件。
- (5) 注释 6：以 fs 为参数，创建 BinaryWriter 对象，该对象也是以二进制方式写入数据到文件，但 fs 对象只能写入字节类型的数据，而 BinaryWriter 简化了 FileStream 写入简单类型数据的方法，它提供将简单类型，例如整型、实型和字符等类型，以字节方式写入到文件。如果不使用 BinaryWriter 对象的方法，那么写入整型数据到文件，要经过复杂的转换，将整数转换为字节后，才能使用 fs 对象写入。二进制读/写器 (BinaryReader/BinaryWriter) 提供一组为简化 FileStream 对象读写简单类型的方法，从而避免简单类型和字节类型之间的烦琐转换。
- (6) 注释 7：创建文件读取对象 fsRead，将读取的文件数据使用 fs 对象写入文件 newFile。
- (7) 注释 8、注释 9：将转换为字节数组的文件信息写入到文件尾，使用 Flush 将内存中可能没有写入到文件的内容全部写入，最后关闭文件。

2) 文件拆分的功能实现

在 Form1 中添加一个自定义方法 SplitCombineFile，实现将合并后的文件，按原来的文件名，拆分成各自独立的文件，保存到指定的文件夹中。

代码片段

```

//自定义方法：拆分文件
private void SplitCombineFile(string combineFile, string savePath)
{
    FileStream fs = new FileStream(combineFile, FileMode.Open, FileAccess.Read);
    BinaryReader br = new BinaryReader(fs);           //1.方便读写基本类型数据
    long len = br.ReadInt64();                      //2.读取长整型数据,它保存的是文件信息长度
    fs.Seek(-len, SeekOrigin.End);                  //3.移到文件信息位置
}

```

```

byte[ ] FileInfo = new byte[ len ];
fs.Read(FileInfo, 0, FileInfo.Length); //4. 定义保存文件信息长度的数组
string FileList = Encoding.Unicode.GetString(FileInfo); //5. 读取文件信息
string[ ] FileNameArr = FileList.Split(' '); //6. 文件信息字符串
//7. 按分隔符拆分字符串

//8. 移动到真实数据的开始,前面8个字节(long数组)为长度信息
fs.Seek(8, SeekOrigin.Begin); //移动文件开始,不包含长度信息

//9. 按照文件名和信息长度,读取数据保存为文件
for (int i = 0; i < FileNameArr.Length; i += 2) //10. 文件名,大小
{
    FileStream tmpWrite = new FileStream(savePath + "\\\" + FileNameArr[i],
    FileMode.Create, FileAccess.Write);

    byte[ ] tmpByte = new byte[ long.Parse(FileNameArr[i + 1]) ];
    int tmpLen = fs.Read(tmpByte, 0, tmpByte.Length); //读出全部数据

    tmpWrite.Write(tmpByte, 0, tmpLen); //写入文件
    tmpWrite.Flush();
    tmpWrite.Close();
}

fs.Close();
br.Close();
}

```

代码说明

(1) 自定义方法 SplitCombineFile 实现将合并后的文件,按原来的文件名拆分成独立文件,即还原合并前的文件。

(2) 注释 1、注释 2: 以 fs 为参数,创建 BinaryReader 对象,目的是读取文件头中代表合并文件信息长度的长整型 len 的数据,这个长度是在合并文件时写入的。

(3) 注释 3: 根据 len 长度信息,将文件位置从文件尾移动 -len 字节,负数代表往文件后移动的字节数,正数代表往前移动的字节数。枚举类型参数 SeekOrigin 有三个值,分别代表移动到文件位置时,是相对当前位置 Current、文件尾 End 还是文件头 Begin。这里代表从文件尾位置,往后移动 len 字节,从这个位置开始是文件信息内容。

(4) 注释 4、注释 5 和注释 6: 根据文件信息长度,读取文件尾中代表合并后的文件信息的字节数据,保存到字节数组 FileInfo 中,然后转换为字符串数据保存到 FileList。

(5) 注释 7: 将 FileList 字符串按字符“|”拆分成字符串数组,该数组的元素依次是:文件名,文件大小,文件名,文件大小……。

(6) 注释 8: 跳过文件头的 8 个字节,该字节是保存文件长度的字节数据,真正的文件内容是该字节之后的数据。

(7) 注释 9: 从指定的文件位置开始,读取每个文件长度的字节数,保存为原来的文件名。可以从字符串数组 FileNameArr 中得到文件名和长度。

3) 测试文件合并与拆分

为窗体的按钮分别添加事件处理过程。

代码片段

```

//“合并文件”按钮 button1 的事件处理过程
private void button1_Click(object sender, EventArgs e)
{
    string[] files = Directory.GetFiles("c:\\files");
    CombineToFile(files, "c:\\tmp.cmb");
    //合并后的新文件,扩展名任意指定
}
//“拆分文件”按钮 button2 的事件处理过程
private void button2_Click(object sender, EventArgs e)
{
    SplitCombineFile("c:\\tmp.cmb", "c:\\splitFiles");
}

```

代码说明

要正确测试该事件过程,必须在 C:\下建立文件夹 files,并添加任意多个不同类型的文件。在图 5.20 中,单击 button1 按钮时,合并后的文件将保存为: C:\tmp. cmb 文件。同时,必须在 C:\建立 splitFiles 文件夹; 单击 button2 按钮时,将合并后的文件拆分到该文件夹中。

5.6 StreamReader 类和 StreamWriter 类

对于文本文件的读写,为了避免在字节与字符串之间的烦琐转换,可以采用 StreamReader 类和 StreamWriter 类来简化文本文件的读写,StreamReader/StreamWriter 用一种特定的编码输入和输出字符,而 FileStream 则用于字节的输入和输出。除非另外指定,其默认编码为 UTF-8,而不是当前系统的 ANSI 代码页。UTF-8 可以正确处理 Unicode 字符并在操作系统的本地化版本上提供一致的结果。

(1) StreamReader 类常用属性,见表 5.11。

表 5.11 StreamReader 类常用属性

属性	说明
CurrentEncoding	获取当前 StreamReader 对象正在使用的当前字符编码
EndOfStream	是否到了文件结尾

(2) StreamReader 类常用方法,见表 5.12。

表 5.12 StreamReader 常用方法

方法	说明
Read	读取输入流中的下一个字符或下一组字符
ReadLine	从当前流中读取一行字符并将数据作为字符串返回
ReadToEnd	读取文件全部内容
Close	关闭当前的 StreamReader 对象和基础流

(3) StreamWriter 类常用方法,见表 5.13。

表 5.13 StreamWriter 类常用方法

方 法	说 明
Write	写入基类型数据到文件,后不跟行结束符
WriteLine	写入一行数据,后跟行结束符
Close	关闭当前的 StreamWriter 对象和基础流

应用实例 5.12 将文本框的内容保存到文件 C:\mytxt.txt,如果文件不存在,则创建,否则将输入的内容添加到文件尾;读取 C:\mytxt.txt 文件的内容,并为每一行添加行号后显示到信息对话框中。

新建项目,保存为 MyStreamRW。

■ 界面布局

在 Form1 窗体中添加一个带滚动条的多行文本框和两个按钮,保留默认的名称,界面布局如图 5.21 所示。

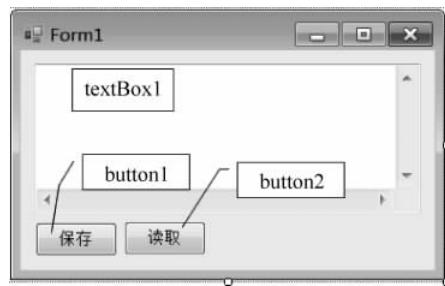


图 5.21 界面布局

■ 代码片段

```
//为“保存”按钮 button1 添加以下的事件处理过程
private void button1_Click(object sender, EventArgs e)
{
    FileStream fs = new FileStream("C:\\\\mytxt.txt", FileMode.Append,
        FileAccess.Write); //1. 创建 FileStream 对象
    StreamWriter sw = new StreamWriter(fs); //2. 创建 StreamWriter 对象
    sw.WriteLine(textBox1.Text); //3. 把文本框内容作为一行保存到文件
    sw.Close(); //4. 关闭文件
}
```

■ 代码说明

(1) 注释 1: 以写方式创建 FileStream 对象 fs,准备对文件 C:\mytxt.txt 进行写入操作。参数 FileMode.Append 表示如果文件存在,将内容以添加方式写入,如果文件不存在,自动创建新文件。

(2) 注释 2: 以 FileStream 对象 fs 为参数创建 StreamWriter 对象 sw,也可以直接使用文件名创建 StreamWriter 对象,如 StreamWriter sw = new StreamWriter("C:\\\\mytxt.txt");。

(3) 注释 3、注释 4: WriteLine 方法实现将文本框中输入的内容作为一行写入到文件中,如果该方法不带参数,则写入空行;最后关闭文件。

实质上,WriteLine 方法和 Write 方法都是将基本类型(如 int、long、float 和 double 等)参数转换为字符串后写入文件的,只是 Write 方法不会写入行结束符。

代码片段

```
//为“读取”按钮 button2 添加以下的事件处理过程
//读出文件内容，并添加行号信息
private void button2_Click(object sender, EventArgs e)
{
    FileStream fs = new FileStream("C:\\mytxt.txt", FileMode.Open, FileAccess.Read);
    StreamReader sr = new StreamReader(fs);
    string s = "";                                //保存读出的内容
    int lineNo = 1;                               //初始行号

    while (!sr.EndOfStream)                      //判断是否读到文件结束
    {
        s += "行" + lineNo + ". " + sr.ReadLine() + "\\r\\n"; //读取一行，添加行号信息
        lineNo++;                                 //行号递增
    }
    sr.Close();                                  //关闭对象
    MessageBox.Show(s);
}
```

代码说明

首先以只读的方式创建 FileStream 对象 fs,准备对 c:\\mytxt.txt 文件进行读操作。然后以 fs 为参数创建 StreamReader 对象 sr,使用 ReadLine 方法读取文件中的每一行,并添加行号。按行读取时,属性 EndOfStream 用于判断是否到达了文件尾,已到达文件尾则为 true,否则为 false。另外,使用 StreamReader 对象的 ReadToEnd 方法可以一次性读取文本文件的全部内容,而 Read 方法每次读取文本文件中的一个字符。

运行结果

按 F5 键运行程序,在文本框中输入内容后单击“保存”按钮,然后单击“读取”按钮,运行结果如图 5.22 所示。



图 5.22 运行结果

小结

本章主要介绍对文件或目录的基本操作,包括创建、复制、移动、删除或重命名,File 类实现对文件的基本操作以及提供了对文本文件的快速读写方法,而要取得具体文件相关信息,可以借助 FileInfo 类; DirectoryInfo 实现对目录的基本操作,要取得目录相关信息,可以借助 DirectoryInfo 类。FileStream 类提供以字节为单位的二进制方式读写任何类型文件,也可以指定文件位置实现随机读写。BinaryReader/BinaryWriter 是二进制读写器,简化了以二进制方式对基类型数据的读写,而 StreamReader/StreamWriter 简化了对文本文件的读写。

上机实践

练习 5.1 新建项目,保存为 Ex5_1。使用 File 类提供的相关方法,将在文本框中输入的内容,使用 UTF8 编码,保存到使用 SaveFileDialog 控件选择的位置;然后使用 OpenFileDialog 控件选择该文件,将内容读取出来显示到另一个文本框中,界面布局如图 5.1 所示。

练习 5.2 新建项目,保存为 Ex5_2。使用文件夹浏览对话框选择任意文件夹,取得该文件夹中的当前文件并将文件名显示到列表框中,运行界面参考图 5.23。

练习 5.3 新建项目,保存为 Ex5_3。设计一个方法 CopyFile,复制指定文件夹中的所有的内容到指定的目标位置,并测试该方法。

练习 5.4 新建项目,保存为 Ex5_4。使用 FileStream 类提供的方法,将任意类型的文件,拆分成大小相等的两部分,分别保存为 1. tmp 和 2. tmp,然后实现将两部分文件合并成一个完整的文件 allFile. file,运行界面如图 5.24 所示。合并后的文件扩展名请自行修改为与原始文件扩展名一致,并核对合并前后的文件字节数是否一致。



图 5.23 练习 5.2 运行界面

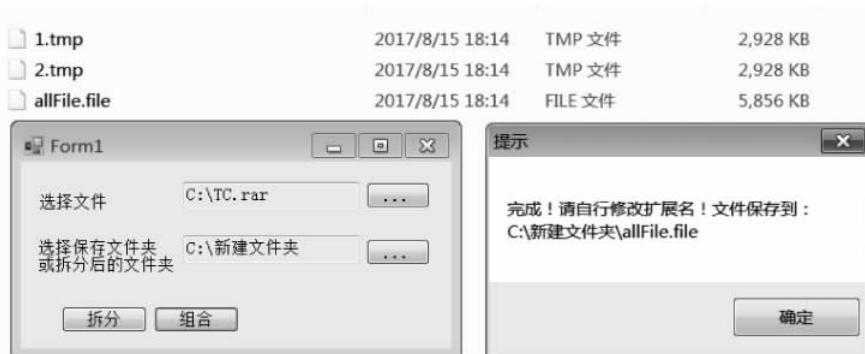


图 5.24 练习 5.4 运行界面