第 12 章　文 件 操 作

本章要点

- 文件和目录的创建、复制、移动、删除
- 文件的读写操作
- 异步文件操作

本章主要介绍了如何对文件和目录的操作，即我们通常说的输入/输出操作。我们在实际的应用过程中经常会碰到对文件和目录的管理操作。总体来讲，文件和目录的操作主要包括创建、读取，写入、删除、更新等。本章将对以上内容做逐一的介绍和举例，以便初学者较好地理解这些内容。

12.1 文件和目录

要实现对文件和目录的操作，就必须得到.NET 框架中相关类库的支持。在.NET 框架的命名空间 System.IO 中就提供了 Directory 类和 File 类，通过这些类提供的属性和方法可以完成对文件和目录的创建、移动、浏览、复制、删除等操作。

12.1.1 目录操作

Directory 类提供了创建、查找和移动目录的许多静态方法。因此 Directory 无须创建类的实例即可调用。DirectoryInfo 类与 Directory 很类似，它表示磁盘上的物理目录，具有可以处理此目录的方法，但必须实例化才能调用。

表 12.1 列出了 Directory 类的公共方法。

表 12.1　Directory 类的公共方法

方　法	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
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
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
GetParent	检索指定路径的父目录，包括绝对路径和相对路径
Move	将文件或目录及其内容移到新位置
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

表 12.2 列出了 DirectoryInfo 类的公共属性。

表 12.2 DirectoryInfo 类的公共属性

名 称	说 明
Attributes	获取或设置当前 FileSystemInfo 的 FileAttributes
CreationTime	获取或设置当前 FileSystemInfo 对象的创建时间
Exists	获取指示目录是否存在的值
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
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
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

表 12.3 列出了 DirectoryInfo 类的公共方法。

表 12.3 DirectoryInfo 类的公共方法

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。 指定路径可以是相对于 DirectoryInfo 类的此实例的路径
Delete	从路径中删除 DirectoryInfo 及其内容
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 DirectoryInfo 实例及其内容移动到新路径
Refresh	刷新对象的状态

下面是一个关于目录操作的简单例子。

例 12.1 编写程序，要求判断在指定位置是否存在一个目录，如果存在则删除此目录，否则创建该目录。

程序代码：

```
using System;
 using System.Collections.Generic;
 using System.Linq;
 using System.Text;
 using System.Threading.Tasks;
 using System.IO;//手动加载命名空间
namespace c12_1
{
    class Program
    {
        static void Main(string[] args)
        {
            //指定目录的路径
            string path = @"C:\CrtDire";
            try
            {
                //判断目录是否存在
                if(!Directory.Exists(path))
                {
                    //如果不存在则创建目录
                    Directory.CreateDirectory(path);
                    Console.WriteLine("创建目录成功");
                }
                else
                {
                 //如果目录存在，则删除该目录
                    Directory.Delete(path, true);
                    Console.WriteLine("删除目录成功");
                }
            }
            catch(IOException e)
            {
                Console.WriteLine("处理过程失败：{0}",
e.ToString());
            }
            finally{}
        }
    }
}
```

分析：上面的例子在控制台应用程序中完成，其中用到了@"C:\CrtDire"; 的表达方式，加个@说明后面都是字符串形式，不然就要为"\"这些进行转义。在 System.IO 中提供了各种输入输出的异常，如对异常 IOException 的捕捉，输出提示信息，便于查找任务失败的原因。

12.1.2 DirectoryInfo 对象的创建

要查看目录层次，需要实例化一个 DirectoryInfo 对象。DirectoryInfo 类提供了许多方法，用于典型操作，如复制、移动、重命名、创建和删除目录，可以获得所含文件和目录的名称。如果打算多次重用某个对象，可考虑使用 DirectoryInfo 的实例方法，而不是 Directory 类的静态方法，因为并不总是需要安全检查。

下面的代码示例演示如何利用 DirectoryInfo 实例化一个对象目录，并使用其属性获得信息，使用其方法来操作对象。

例 12.2 设计一个程序，将某个目录（含子目录）移到目标文件夹下

程序代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;//手动加载命名空间

namespace c12_2
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                // 创建一个DirectoryInfo对象
                                DirectoryInfo di = new
DirectoryInfo(@"c:\TempDir");
                //如果不存在的话，建立此目录
                if (di.Exists == false)
                    di.Create();
                // 在这个新建的目录下建立子目录
                                    DirectoryInfo dis =
di.CreateSubdirectory("SubDir");
                 // 如果目标目录不存在，则建立目录，并将刚才的
目录移动至
                  if (Directory.Exists(@"c:\NewTempDir")
== false)
```

```
                    Directory.CreateDirectory(@"c:\NewTe
mpDir");
                di.MoveTo(@"c:\NewTempDir\TempDir");
                    Console.WriteLine("目录已于{0}移动成
功! ",di.CreationTime);
            }
            catch (IOException e)
            {
                    Console.WriteLine("移动失败: {0}",
e.ToString());
            }
            finally { }
        }
    }
}
```

分析：如果试图将 c:\TempDir 移动到 c:\NewTempDir，而 c:\NewTempDir 已经存在，则此方法将引发 IOException。因此必须将“c:\NewTempDir\TempDir”作为 MoveTo 方法的参数。

按 Ctrl+F5 组合键运行后，显示的结果如图 12.1 所示。

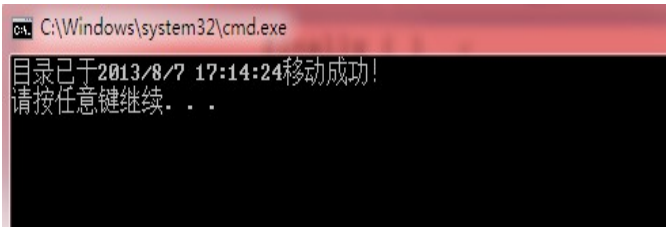

图 12.1　例 12.2 的运行结果

12.1.3 文件操作

File 类通常与 FileStream 类协作完成对文件的创建、删除、复制、移动、打开等操作。与 Directory 的方法一样，所有的 File 方法都是静态的，不需要实例化即可以调用 File 方法。

FileInfo 和 File 对象是紧密相关的，与 DirectoryInfo 一样，FileInfo 的所有方法都是实例方法。所以如果只想执行一个操作，那么使用 File 中的静态方法的效率比使用相应的 FileInfo 中的 实例方法可能更高。所有的 File 方法都要求当前所操作的文件和目录的路径。

表 12.4 列出了 File 类公开的成员。

表 12.4　File 类公开的成员

名　称	说　明

AppendAllText	将指定的字符串追加到文件中，如果文件还不存在则创建该文件
AppendText	创建一个 StreamWriter，它将 UTF-8 编码文本追加到现有文件
Copy	将现有文件复制到新文件
Create	在指定路径中创建文件
CreateText	创建或打开一个文件，用于写入 UTF-8 编码的文本
Delete	删除指定的文件。如果指定的文件不存在，则不引发异常
GetAttributes	获取在此路径上的文件的 FileAttributes
GetCreationTime	返回指定文件或目录的创建日期和时间
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
Move	将指定文件移到新位置，并提供指定新文件名的选项
Open	打开指定路径上的 FileStream
OpenRead	打开现有文件以进行读取
OpenText	打开现有 UTF-8 编码文本文件以进行读取
OpenWrite	打开现有文件以进行写入
ReadAllBytes	打开一个文件，将文件的内容读入一个字符串，然后关闭该文件
ReadAllLines	打开一个文本文件，将文件的所有行都读入一个字符串数组，然后关闭该文件
ReadAllText	打开一个文本文件，将文件的所有行读入到一个字符串中，然后关闭该文件
Replace	使用其他文件的内容替换指定文件的内容，这一过程将删除原始文件，并创建被替换文件的备份
SetAttributes	设置指定路径上文件的指定的 FileAttributes
SetCreationTime	设置创建该文件的日期和时间
SetLastAccessTime	设置上次访问指定文件的日期和时间
SetLastWriteTime	设置上次写入指定文件的日期和时间
WriteAllBytes	创建一个新文件，在其中写入指定的字节数组，然后关闭该文件。如果目标文件已存在，则改写该文件
WriteAllLines	创建一个新文件，在其中写入指定的字符串，然后关闭文件。如果目标文件已存在，则改写该文件
WriteAllText	创建一个新文件，在文件中写入内容，然后关闭文件。如果目标文件已存在，则改写该文件

表 12.5 列出了 FileInfo 类的常用属性。

表 12.5　FileInfo 类的常用属性

名　称	说明
Attributes	获取或设置当前 FileSystemInfo 的 FileAttributes

CreationTime	获取或设置当前 FileSystemInfo 对象的创建时间
Directory	获取父目录的实例
DirectoryName	获取表示目录的完整路径的字符串
Exists	获取指示文件是否存在的值
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
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
Length	获取当前文件的大小
Name	获取文件名

表 12.6 列出了 FileInfo 类的常用方法。

表 12.6 FileInfo 类的常用方法

名 称	说 明
AppendText	创建一个 StreamWriter，它向 FileInfo 的此实例表示的文件追加文本
CopyTo	将现有文件复制到新文件
Create	创建文件
CreateText	创建写入新文本文件的 StreamWriter
Delete	永久删除文件
MoveTo	将指定文件移到新位置，并提供指定新文件名的选项
Open	用各种读/写访问权限和共享特权打开文件
OpenRead	创建只读 FileStream
OpenText	创建使用 UTF8 编码、从现有文本文件中进行读取的 StreamReader
OpenWrite	创建只写 FileStream
Refresh	刷新对象的状态
Replace	使用当前 FileInfo 对象所描述的文件替换指定文件的内容，这一过程将删除原始文件，并创建被替换文件的备份

下面的示例演示了 File 类中部分成员的用法。该示例通过 File 类的 CreateText()方法创建一个文本，接着向文本写入数据，读取文本内容，使用 Move()方法进行移动并重命名文件。注意执行程序前需要在 C 盘新建“temp”文件夹。

例 12.3 创建一个程序，利用 File 类的方法进行文本的创建，数据的读写，以及文本的移动、重命名。

程序代码：

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace c12_3
{
    class Program
    {
        static void Main(string[] args)
        {
//设定创建文件的路径为C盘根目录，文本文件名称为FileTest
            string path = @"c:\FileTest.txt";
            string path2= @"c:\temp\NewFileTest.txt";
            if (!File.Exists(path))
            {
                // 创建一个文件用于写入 UTF-8 编码的文本
                                        StreamWriter  sw  =
File.CreateText(path);
                    sw.WriteLine("You");
                    sw.WriteLine("are");
                    sw.WriteLine("beautiful");
                    sw.Dispose();
            }
            // 打开文件，从里面读出数据
            StreamReader sr = File.OpenText(path);
                string s = "";
                //输出文件里的内容，直到文件结束
                while ((s = sr.ReadLine()) != null)
                {
                    Console.WriteLine(s);
                }
                sr.Dispose();
                try
                {
                    // 确保目标路径中没有NewFileTest文件
                    if (File.Exists(path2))
                        File.Delete(path2);
                    // 移动文件
```

```
                    File.Move(path, path2);
                     Console.WriteLine("{0} 移动至 {1}.",
path, path2);
                    // 判断源文件在不在
                    if (File.Exists(path))
                    {
                        Console.WriteLine("源文件还存在，移
动失败！");
                    }
                    else
                    {
                        Console.WriteLine("没有源文件，移动
成功！");
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("移动失败：{0}",
e.ToString());
                }
            }
        }
        }
```

按 Ctrl+F5 键运行程序，结果如图 12.2 所示。

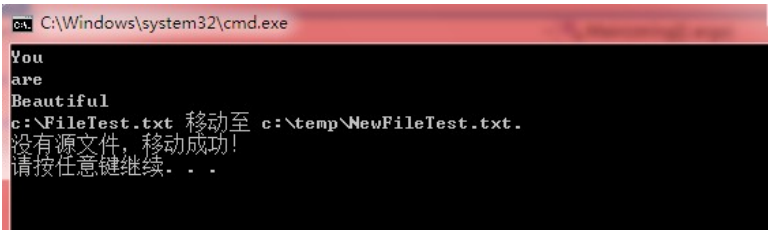

图 12.2　例 12.3 的运行结果

在程序运行后，我们可以在 C 盘根目录下找到新命名的 NewFileTest 文本文件，可以查看里面内容为：

```
You
are
beautiful
```

Move()方法移动范围是整个磁盘，如果尝试通过将一个同名文件移到该目录中来替换文件，将发生 IOException。不能使用 Move()覆盖现有文件。因此此例中要确保目标路径中没有 NewFileTest 文件。另外程序中的 Dispose()

是释放对象所占用的资源，如果程序不用了，就可以调用释放。要实现文件重命名只需修改 Move()中目标文件名参数的文件名即可。此例中是将“FileTest”修改成“NewFileTest”。

下面的例子用 FileInfo 的 GetFiles()方法得到指定文件夹下的所有文件，用 Delete()方法来删除当前目录下的所有文件，并显示文件相关的属性。需要先手工在 C 盘根目录下创建一个名为 Temp 的文件夹，在这个新建的文件夹下创建一些文本文件。

例 12.4 编写一个程序，删除所有指定文件夹中的文件。

程序代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace c12_4
{
    class Program
    {
        static void Main(string[] args)
        {   //创建一个DirectoryInfo实例
            DirectoryInfo d = new
DirectoryInfo("C:\\Temp");
            //创建一个FileInfo数组对象，用于存储指定目录下的
文件对象
            FileInfo[] fis = d.GetFiles();
            foreach (FileInfo fi in fis)
            {
                //遍历删除各文件
                fi.Delete();
Console.WriteLine("{0}文件已删除,文件大小{1} Bytes",
fi.Name,fi.Length);
            }
        }
    }
}
```

分析：GetFiles()方法可以得到指定文件夹下的所有文件，将这些得到的文件作为对象依次存储到 FileInfo[]数组中，利用 foreach 遍历每个数组元素，

将输出每个文件对象的文件名和文件大小的属性信息，并执行 Delete()方法来删除这些文件。

按 Ctrl+F5 键运行程序，运行结果如图 12.3 所示。

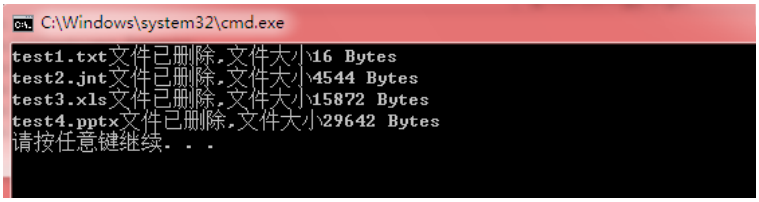

图 12.3 例 12.4 的运行结果

执行完程序再次打开 C:\Temp 文件夹，您将会发现里面已经没有任何文件了。

12.2 数据的读取和写入

在 System. IO 命名空间中，包含几个用于从流中读写数据的类，各有不同的用途。

12.2.1 按文本模式读写

StreamReader 类和 StreamWriter 类提供了按文本模式读写数据的方法。

表 12.7 列出了 StreamReader 类的常用属性。

表 12.7 StreamReader 类的常用属性

名 称	说 明
BaseStream	返回基础流
CurrentEncoding	获取当前 StreamReader 对象正在使用的当前字符编码
EndOfStream	获取一个值，该值表示当前的流位置是否在流的末尾

表 12.8 列出了 StreamReader 类的常用方法。

表 12.8 StreamReader 类的常用方法

名 称	说 明
Close	关闭 StreamReader 对象和基础流，并释放与读取器关联的所有系统资源
DiscardBufferedData	允许 StreamReader 对象丢弃其当前数据
Peek	返回下一个可用的字符，但不使用它
Read	读取输入流中的下一个字符或下一组字符
ReadBlock	从当前流中读取最大 count 的字符并从 index 开始将该数据写入 buffer
ReadLine	从当前流中读取一行字符并将数据作为字符串返回
ReadToEnd	从流的当前位置到末尾读取流

表 12.9 列出了 StreamWriter 类的常用属性。

表 12.9 StreamWriter 类的常用属性

名　称	说　明
AutoFlush	获取或设置一个值，该值指示 StreamWriter 是否在每次调用 StreamWriter.Write 之后，将其缓冲区刷新到基础流
BaseStream	获取与后备存储区连接的基础流
Encoding	获取将输出写入到其中的 Encoding
FormatProvider	获取控制格式设置的对象
NewLine	获取或设置由当前 TextWriter 使用的行结束符字符串

表 12.10 列出了 StreamWriter 类的常用方法。

表 12.10　StreamWriter 类的常用方法

名　称	说　明
Close	关闭当前的 StreamWriter 对象和基础流
Flush	清理当前编写器的所有缓冲区，并使所有缓冲数据写入基础流
Write	写入流
WriteLine	写入重载参数指定的某些数据，后跟行结束符(从 TextWriter 继承)

下面的例子实现了追加文本并读取显示的功能。

例 12.5 使用 StreamReader 类的方法将数据从文本文件中读出并显示。

程序代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace c12_5
{
    class Program
    {
        static void Main(string[] args)
        {
            FileInfo fi = new
FileInfo(@"c:\Test.txt");
            if (!fi.Exists)
            {
                //创建一个文件并写入
               StreamWriter sw1 = fi.CreateText();
                    sw1.WriteLine("Hello");
                    sw1.WriteLine("And");
                    sw1.WriteLine("Welcome");
                    sw1.Dispose();
            }
```

```
            // 在文件中追加内容
            StreamWriter sw2 = fi.AppendText();
                sw2.WriteLine("This");
                sw2.WriteLine("is Extra");
                sw2.WriteLine("Text");
                sw2.Dispose();
            //读取并显示文件
            using (StreamReader sr = fi.OpenText())
            {
                string s = "";
                while ((s = sr.ReadLine()) != null)
                {
                    Console.WriteLine(s);
                }
            }
        }
    }
}
```

按 Ctrl+F5 键运行程序，运行结果如图 12.4 所示。

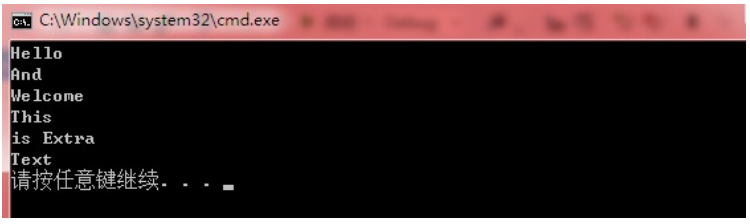

图 12.4　例 12.5 的运行结果

下面的示例演示如何使用 StreamWriter 对象将 C 盘上的所有文件夹名称写入到一个文件中。此例中的 using 标记是用来自动释放资源的，与 Dispose()方法相同，只是在执行完 using 体的语句后会自动执行 Dispose()方法。虽然微软推荐这种用法，但这样使用 using 可能不利于程序的可读性。

例 12.6 使用 StreamWriter 类，把数据写到文本文件中去。

程序代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace c12_6
{
  class Program
  {
   static void Main(string[] args)
   {
```

```
    // 得到 c 盘下的所有文件夹
    DirectoryInfo[] cDirs = new
DirectoryInfo(@"c:\").GetDirectories();
   // 将得到的文件夹名写入指定文件内
  using (StreamWriter sw = new
StreamWriter("CDriveDirs.txt"))
   {
     foreach (DirectoryInfo dir in cDirs)
      {
        sw.WriteLine(dir.Name);
      }
   }
    // 读取并显示这些文件夹名
     string line = "";
     using (StreamReader sr = new
StreamReader("CDriveDirs.txt"))
     {
       while ((line = sr.ReadLine()) != null)
       {
          Console.WriteLine(line);
       }
      }
    }
  }
}
```

12.2.2 按二进制模式读写

FileStream 类提供了最原始的字节级上的文件读写功能，但我们习惯于对字符串操作，于是 StreamWriter 和 StreamReader 类增强了 FileStream，它让我们在字符串级别上操作文件，但有的时候我们还是需要在字节级上操作文件，却又不是一个字节一个字节的操作，通常是 2 个、4 个或 8 个字节这样操作，这便有了 BinaryWriter 和 BinaryReader 类，它们可以将一个字符或数字按指定 个数字节写入，也可以一次读取指定个数字节转为字符或数字。

表 12.11 列出了 BinaryReader 类的常用方法。

表 12.11　BinaryReader 类的常用方法

名　称	说　明
Close	关闭当前阅读器及基础流
PeekChar	返回下一个可用的字符，并且不提升字节或字符的位置
Read	从基础流中读取字符，并提升流的当前位置
ReadBoolean	从当前流中读取 Boolean 值，并使该流的当前位置提升 1 个字节
ReadByte	从当前流中读取下一个字节，并使流的当前位置提升 1 个字节
ReadBytes	从当前流中将 count 个字节读入字节数组，并使当前位置提升 count 个字节
ReadChar	从当前流中读取下一个字符，并根据所使用的 Encoding 和从流中读

	取的特定字符，提升流的当前位置
ReadChars	从当前流中读取 count 个字符，以字符数组的形式返回数据，并根据所使用的 Encoding 和从流中读取的特定字符，提升当前位置
ReadString	从当前流中读取一个字符串。字符串有长度前缀，一次 7 位地被编码为整数

表 12.12 列出了 BinaryWriter 类的常用方法。

表 12.12 BinaryWriter 类的常用方法

名 称	说 明
Close	关闭当前的 BinaryWriter 和基础流
Flush	清理当前编写器的所有缓冲区，使所有缓冲数据写入基础设备
Seek	设置当前流中的位置
Write	将值写入当前流

下面的例子实现了如何向新的空文件流 BinFile.dat 写入数据及从中读取数据。在当前目录中创建了数据文件之后，也就同时创建了相关的 BinaryWriter 类和 BinaryReader 类，BinaryWriter 类用于向 BinFile.dat 写入字符 A~F。通过 BinaryReader 类的 ReadChar()方法读出指定的内容。

例 12.7 使用 BinaryWriter 类和 BinaryReader 类进行二进制文件流的读写。

程序代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace c12_7
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream myFileStream = new
FileStream("c:\\BinFile.dat", FileMode.CreateNew);
            // 为文件流创建二进制写入器
            BinaryWriter myBinaryWriter = new
BinaryWriter(myFileStream);
            // 写入数据
            for (int i=65; i<71; i++)
            {
                myBinaryWriter.Write((char)i);
            }
            myBinaryWriter.Close();
```

```
            myFileStream.Close();
            // 创建 reader
            myFileStream = new
FileStream("c:\\BinFile.dat", FileMode.Open,
FileAccess.Read);
            BinaryReader myBinaryReader = new
BinaryReader(myFileStream);
            // 从 BinFile 读数据
            while (myBinaryReader.PeekChar()!= -1)
            {
                Console.Write("{0}",
myBinaryReader.ReadChar());
            }
            myBinaryReader.Close();
            myFileStream.Close();
        }
    }
}
```

按 Ctrl+F5 键运行程序，运行结果如图 12.5 所示。

图 12.5　例 12.7 的运行结果

12.3 异步文件操作

以上几小节涉及的都是同步 I/O 操作，而异步 I/O 操作在依托计算机高性能的前提下，对程序的执行效率有了较大的提升。在同步 I/O 操作中，方法将一直处于等待状态，直到 I/O 操作完成。而在异步 I/O 操作中，程序的方法仍可以转移去执行其他的操作。在 .NET framework 4 和早期版本中，通过 Stream 类的 BeginRead()、EndRead()、BeginWrite()和 EndWrite()方法提供了异步 I/O。

异步 I/O 的顺序如下：调用文件的读取方法，然后转向其他与此无关的工作，读取过程将在另一线程中进行。当读取完成时，会有一个回调方法进行通知，然后处理读取的数据，再启动一次读取，然后又回到另一项工作上去。

异步操作可以在不必阻止主线程的情况下执行大量占用资源的 I/O 操作。从.NET Framework 4.5 开始，I/O 类型所包括的异步方法简化了异步操作。异步方法在其名称中包含 Async，例如 ReadAsync、WriteAsync、CopyToAsync、FlushAsync、ReadLineAsync 和 ReadToEndAsync。

从 Visual Studio 2012 开始为异步编程提供两个关键字：

(c#) async 修饰符，用于指示方案包含一个异步操作。

(c#) await 运算符，应用于异步方法的结果。

有关更多信息，请参见使用 Async 和 Await 的异步编程的专用教程。

下面的代码演示如何使用复制文件的两 FileStream 对象异步从一个目录到另一个。 请注意 Button 控件的 Click 事件处理程序标记 async 修饰符，因为它调用异步方法。

```
namespace WpfApplication
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private async void Button_Click(object sender,
RoutedEventArgs e)
        {
            string StartDirectory =
@"c:\Users\exampleuser\start";
            string EndDirectory =
@"c:\Users\exampleuser\end";
            foreach (string filename in
Directory.EnumerateFiles(StartDirectory))
```

```
            {
                using (FileStream SourceStream =
File.Open(filename, FileMode.Open))
                {
                    using (FileStream DestinationStream
= File.Create(EndDirectory +
filename.Substring(filename.LastIndexOf('\\'))))
                    {
                        await
SourceStream.CopyToAsync(DestinationStream);
                    }
                }
            }
        }
```

12.4 案 例 实 训

1. 案例说明

本例主要是涉及文件操作的相关内容，完成文本打开、编辑、保存的操作。

2. 编程思路

主要使用文件的读取和写入的方法操作文件，利用对话框控件过滤成文本文件，写入时将所有缓冲区数据写入基础流。

3. 窗体设计

在窗体上添加 3 个 Button 按钮、1 个 TextBox 控件、2 个 GroupBox 控件、1 个 OpenFileDialog 控件、1 个 SaveFileDialog 控件。具体的放置如图 12.6 所示(OpenFileDialog 控件和 SaveFileDialog 不可显示)。

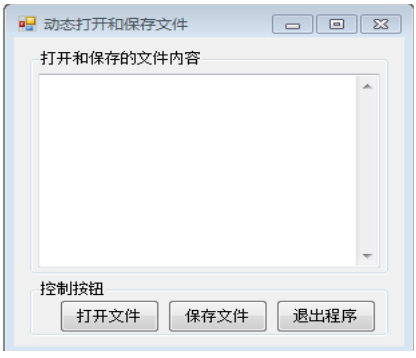

图 12.6 窗体界面

4. 程序代码

具体的程序代码如下：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace Case12_1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        string myfile = "";

        private void button1_Click(object sender,
EventArgs e)
        {
         openFileDialog1.ShowDialog();       //打开对话
框
         //定义StreamReader对象实例
        StreamReader myread = new
StreamReader(openFileDialog1.FileName,
Encoding.Default);
         try
         {
            textBox1.Text = "";
            string mys = myread.ReadLine();      //读取
打开文件的一行
            while (mys != null)             //如果打开文
件不为空，则一行一行读取
           {
            textBox1.Text = textBox1.Text + mys +
"\r\n";
             mys = myread.ReadLine();
            }
          myfile = openFileDialog1.FileName;
         }
         catch (Exception mye)
        {
          MessageBox.Show("读取文件失败！" +
mye.Message);     //提示对话框
        }
         finally
         {
           myread.Close();
        }
     }
    private void button2_Click(object sender,
```

```
EventArgs e)
    {
     if (saveFileDialog1.ShowDialog() ==
DialogResult.OK && saveFileDialog1.FileName != "")
      {
        myfile = saveFileDialog1.FileName;
      }
       //第二个参数为false代表改写内容，不是追加
       StreamWriter Writer = new StreamWriter(myfile,
false, Encoding.Default);
     try
      {
        foreach (string line in textBox1.Lines)
       {
         Writer.Write(line + "\n", Encoding.Default);
       }
           Writer.Flush();//将缓冲区的数据写入流
      }
      catch (Exception ex)
      {
       MessageBox.Show("保存文件失败！" + ex.Message);
      }
      finally
      {
        Writer.Close();
       }

     }
   private void button3_Click(object sender,
EventArgs e)
    {
      this.Close();
       Application.Exit();          //退出程序
    }
  }
}
```

5. 运行结果

程序的运行结果如图 12.7 所示。

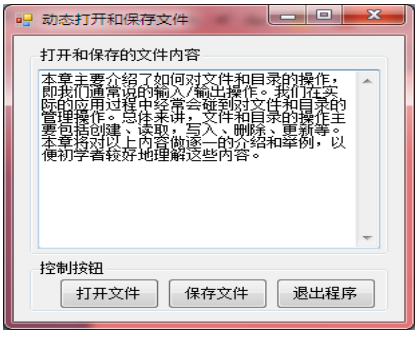

图 12.7　案例的运行结果

12.5 小　　结

在一般应用中经常会涉及到目录和文件的操作，这时就会遇到文件的读写操作。要在C#语言中进行文件操作，只需要利用.NET框架在System.IO命名空间中提供的类就可以实现。其中经常用到的类有File、Stream、FileStream、BinaryReader、BinaryWriter、StreadReader、StreamWriter等。我们通过直接调用或者实例化对象来使用它们的属性和方法。

在本章中我们通过实例详细地介绍了如何以File类和Directory类进行目录和文件的操作，以及如何采用StreamReader、StreamWriter、BinaryReader、BinaryWriter类进行文本模式和二进制模式的文件读写操作。在最后一节，又对文件的异步操作方式做了简单的介绍，需要深入了解的话请参看专项教程。

12.6 习　　题

1.编写一个程序，在窗体中输入指定目录路径，如图12.8所示，点击显示按钮即可以图标的形式显示此目录中的所有文件名称。（需要listView控件和imageList控件）

图12.8　案例的运行结果

2. 在第1题的基础上添加如图12.9所示的创建按钮，点击创建指定位置的文件夹，再点击显示按钮显示全部文件夹。（需要listView控件和imageList控件）

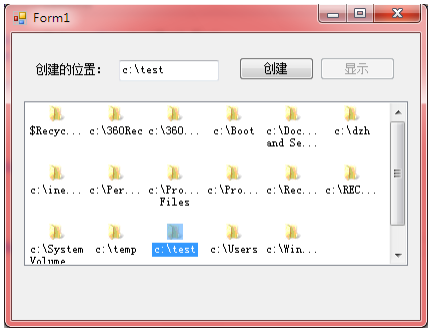

图 12.9　案例的运行结果