目录、文件和注册表

王忠民

本章主要内容：

1　目录管理

2　文件管理

3　文件的读写

4　注册表的读写

1　目录管理

在 System.IO 命名空间下，有两个类对磁盘和目录进行操作管理 ：

- DirectoryInfo 类
- Directory 类

相同：　均能对目录进行操作

区别：　DirectoryInfo 必须被实例化后才能使用；而 Directory 是静态类，所以不能被实例化，只提供静态的方法和属性。

DirectoryInfo

1. DirectoryInfo 类的构造函数形式：

   public DirectoryInfo(string path);

2. DirectoryInfo 类的主要属性：
   - Attributes　　获取或设置当前 FileSystemInfo 的 FileAttributes 。
   - Exists　获取指示目录是否存在的布尔值
   - FullName　获取或设置当前路径的完整目录
   - Parent　　获取指定子目录的父目录
   - Root　　获取路径的根目录
   - CreationTime　　获取或设置当前目录创建时间
   - LastAccessTime　获取或设置上一次访问当前目录的时间
   - LastWriteTime　获取或设置上一次写入当前目录的时间

3. DirectoryInfo 类的常用方法：

   Create（ ）　创建目录

   CreateSubDirectory　创建一个或多个子目录

   Delete（ ），GetFiles（ ），MoveTo（ ），GetDirectories（ ）

Directory

Directory 类提供的静态方法

- 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
- 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
- SetCurrentDirectory　将应用程序的当前工作目录设置为指定的目录
- SetLastWriteTime　设置上次写入目录的日期和时间

1. 目录的创建

◆ CreateDirectory 方法（ Directory 类中 ）
用于创建指定路径中的各级目录

如：

```
using System.IO;
Directory.CreateDirectory(“c:\\test”);
Directory.CreateDirectory("c:\\test\\t1\\t2");
```

2．目录的删除

◆ Delete 方法（ Directory 类）

用于删除指定的目录，该方法有两种重载的形式：

1) public static void Delete (string path)

path 为要移除的空目录的名称。此目录必须为可写或为空。

2) public static void Delete (string path,bool recursive)

path 为要移除的目录的名称，不区分大小写；

recursive 是一个布尔值，若要移除 path 中的目录、子目录和文件，则为 true ；否则为 false 。

例 创建并删除指定的目录

```
string  path = @"c:\mytestdir\SubDir1";
try
 {
     if (Directory.Exists(path))
       {   Console.WriteLine(" 目录已存在 ");
           return;     }
     DirectoryInfo   di = Directory.CreateDirectory(path);
     Console.WriteLine(" 成功创建目录时间 ： {0}“, Directory.GetCreationTime(path));
     di.Delete();
     Console.WriteLine(" 目录已删除 ");
  }
  catch (Exception e)
  { Console.WriteLine(" 程序异常 : {0}", e.ToString()); }
```

DirectoryInfo 和 Directory 的使用特点：如果多次使用某个对象一般使用前者；如果仅执行某一个操作则使用后者提供的静态方法效率更高一些。

3．目录的移动

◆ Move 方法（ Directory 类） 移动目录

public static void Move (string sourceDirName, string destDirName)

sourceDirName 为要移动的文件或目录的路径； destDirName 为指向 sourceDirName 的新位置的目标路径。

```
static void dirmove()
        {
            Directory.Move(@"c:\olddir\SubDir2",  @"c:\newdir");
        }
```

例：列出指定路径下的子目录清单。

```
class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo  di = new DirectoryInfo("d:\\");
            DirectoryInfo[] dis = di.GetDirectories();
            foreach (DirectoryInfo diitem in dis)
            {
               Console.WriteLine(diitem.FullName);
            }
            Console.Read();
        }
    }
```

2 文件管理

FileInfo 类和 File 类

提供用于创建、复制、删除、移动和打开文件的方法，并协助创建 FileStream 对象。

相同点：都能完成对文件的操作。

不同点： FileInfo 类必须实例化，并且每个 FileInfo 的实例必须对应于系统中一个实际存在的文件。

使用特点：如果打算多次重用某个对象，可考虑使用 FileInfo 的实例方法，而不是 File 类的相应静态方法。

File 类

File 类的常用方法如表所示。

方法名	说明
Create()	在指定路径中创建文件
Delete()	删除指定的文件
Exists()	判断指定的文件是否存在
Open()	打开指定的文件
OpenRead()	打开文件进行读取
OpenWrite()	打开文件进行写入
OpenText()	打开文本文件进行读取
Move()	将指定文件移到新位置，并提供指定新文件名的选项
Copy()	将指定文件复制到新文件
ReadAllText	
WriteAllText	

2.1 文件复制、删除与移动

1. 文件复制

File 类的 Copy 方法用于将现有文件复制到新文件。方法原型为：

public static void Copy (string sourceFileName,string destFileName)

参数： sourceFileName 为要复制的文件， destFileName 为目标文件的名称，它不能是一个目录或现有文件。

```
File.Copy("e:\\mytestfile.txt","e:\\mytestfile2.txt");
```

等效于：

```
FileInfo  fi = new  FileInfo("e:\\mytestfile.txt");
fi.CopyTo("e:\\mytestfile2.txt");
```

2. 文件删除

File 类的 Delete 方法用于删除指定的文件。方法定义：

public static void Delete (string path)

参数： path 为要删除的文件的名称。

```
string[]  files = Directory.GetFiles(“c:\\testdir”);   // 获取目录下全部文件名
foreach (string  filename  in  files)       //  遍历目录文件
 {
     Console.WriteLine(filename);
     File.Delete(filename);
 }
```

3. 文件移动

File 类的 Move 方法用于将指定文件移到新位置，并提供指定新文件名的选项。方法原型为：

public static void Move (string sourceFileName, string destFileName)

参数： sourceFileName 为要移动的文件的名称， destFileName 为文件的新路径。

```
string  file1 = @"c:\testdir\MyTest.txt";
string  file2 = @"c:\temp2\MyTest.txt";
File.Move(file1,file2);
```

2.2 文件属性与设置

成员名	说明
Archive	文件的存档状态。应用程序使用此属性为文件加上备份或移除标记。
Compressed	文件已压缩
Encrypted	该文件或目录是加密的。对于文件来说，表示文件中的所有数据都是加密的。对于目录来说，表示新创建的文件和目录在默认情况下是加密的。
Hidden	文件是隐藏的，因此没有包括在普通的目录列表中。
System	文件为系统文件。文件是操作系统的一部分或由操作系统以独占方式使用。
Temporary	文件是临时文件。文件系统试图将所有数据保留在内存中以便更快地访问，而不是将数据刷新回大容量存储器中。不再需要临时文件时，应用程序会立即将其删除。

文件属性的设置

对文件的属性 FileAttributes 进行设置，可以使用 File 类的 SetAttributes 方法。方法原型为：

public static void SetAttributes (string path, FileAttributes fileAttributes)

参数： path 为该文件的路径， fileAttributes 为所需的 FileAttributes 。

文件属性的获取

获取指定路径上文件的属性 FileAttributes ，可以使用 File 类的 GetAttributes 方法，方法定义：

public static FileAttributes GetAttributes (string path)

参数： path 为该文件的路径。

```
string  filename= "e:\\testfile.txt";
FileAttributes  fa = File.GetAttributes(filename);   // 获取文件属性
File.SetAttributes(filename, fa|FileAttributes.Hidden);  // 设置为隐藏
//    File.SetAttributes(filename, fa &(~FileAttributes.Hidden));
```

3 文件的读写

文件和流

System.IO 命名空间提供了多种类型用于数据文件和数据流的操作。

文件（ file ）和流（ stream ）既有区别又有联系。文件是在各种媒质上（可移动磁盘、硬盘 等）永久存储的数据的有序集合。它是一种进行数据读写操作的基本对象。通常情况下，文件按照树状目录进行组织，每个文件都有文件名、文件所在路径、创建时间、访问权限等属性。

流是字节序列的抽象概念，流提供一种向后备存储器写入字节和从后备存储器读取字节的方式。如文件、输入输出设备或者 TCP/IP 套接字等均可以看成流。

除了和磁盘文件直接相关的文件流以外，还有多种其它类型的流，如分布在网络中、内存中的流，分别称为网络流、内存流。

所有表示流的类都是从抽象基类 Stream 继承的：

FileStream 类： 文件流 （写入和读出的都是字节数组 byte[] ）

MemoryStream 类： 内存流

BufferedStream 类：缓存流

FileStream 文件流

FileStream 类常见属性如表所示

属性名	说明
IsAsync	当前流是异步打开还是同步打开
CanRead	获取指示当前流是否支持读取的值
CanSeek	获取指示当前流是否支持查找功能的值
CanTimeOut	获取一个值，该值确定当前流是否可以超时
CanWrite	获取指示当前流是否支持写入功能的值
Length	获取用字节表示的流长度
Position	获取或设置当前流中的位置
ReadTimeout	获取或设置一个值（以毫秒为单位），该值确定流在超时前尝试读取多长时间
WriteTimeout	获取或设置一个值（以毫秒为单位），该值确定流在超时前尝试写入多长时间

FileStream 常用方法：

Read（ ）：从当前流读取字节序列，并将流中位置提升读取的字节数

Seek（ ）：设置当前流中的位置，以读取指定位置的字节。

long Seek(int offset, SeekOrigin origin)

偏移量

参考点（ Begin, End, Current ）

例： fs.Seek (2, SeekOrigin.Begin) ;

ReadByte（ ）：从流中读取一个字节，将流中位置向前推进一个字节

WriteByte（ ）：将一个字节写入流中当前位置，流中位置向前推进一个字节

Write（ ）：向当前流中写入字节序列，并将流中位置提升写入的字节数

Write(byte[] array, int offset,int count); array 中从 offset 开始的 Count 个字节

写入文件

例：单击按钮时实现写入文件。

```
private void button1_Click(object sender, EventArgs e)
{   // FileMode.Append  文件若不存在将自动创建
    FileStream  filestream = new  FileStream("E:\\ 数据 .txt", FileMode.Append);
    byte[] content = {71,72,73,74,75,76,77,78,79};
    filestream.Write(content, 0, content.Length);
    filestream.Close();
}
```

FileStream 可以由 File 辅助创建，也可独立创建

```
FileStream  filestream = File.OpenWrite("E:\\ 数据 .txt");
// 可读写，不是追加。文件若不存在将自动创建
```

读取文件

例：单击时实现读取功能。

```
private void button2_Click(object sender, EventArgs e)
{
    FileStream  fs = new  FileStream("E:\\ 数据 .txt", FileMode.Open);
    byte[]  bt = new  byte[(int)fs.Length];
    fs.Read(bt, 0, (int)fs.Length);
    foreach (byte  x  in  bt)
    {
        textBox1.Text += x;
    }
    fs.Close();
}
```

Read(字节数组，数组中起始位置，读入的长度)

FileMode： Open , Append, OpenOrCreate, Create, CreateNew, Truncate

文本文件的读写： StreamWriter 和 StreamReader 类

StreamWriter 类也叫写入器，它用于将数据写入文件流。

StreamWriter 类中包含多个方法，可以调用它的方法将内容写入文件流，主要方法如下：

- Write() 写入流，即用户创建好的流。可写入多种类型
- WriteLine() 用于写入一行数据，写入某些数据后换行
- Close() 关闭写入器

StreamReader 类也叫读取器，主要用于读取流中的数据。除非特意指定，否则 StreamReader 类的默认编码为 UTF8 。

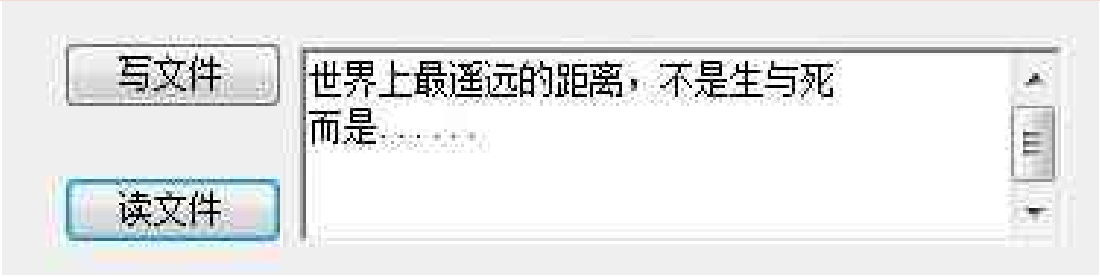
StreamReader 类中也包含多个方法，常用的方法如下：

- Read （ ）读取输入流中下一个字符并使指针移一位
- ReadLine() 读取文件流中的一行数据，并返回字符串
- ReadToEnd() 从当前位置读到末尾，返回字符串
- Close() 用于关闭读取器

没有 Seek

```
private void button1_Click(object sender, EventArgs e)
      {   StreamWriter  sw = new  StreamWriter("G:\\ 世界上 .txt");
          string  content = " 世界上最遥远的距离，不是生与死 \r\n 而是 .....\r\
n";
          sw.WriteLine(content);
          sw.Close();
      }

private void button2_Click(object sender, EventArgs e)
      {   StreamReader  sr = new  StreamReader("G:\\ 世界上 .txt");
          richTextBox1.Text = sr.ReadToEnd();
          sr.Close();
      }
```



二进制文件的读写： BinaryWriter 和 BinaryReader 类

C# 中除了字节类型以外，还有许多其它基本数据类型，例如 int 、 bool 、 float 等，读写这些基本数据类型要使用 BinaryReader 和 BinaryWriter 类，访问二进制文件的若干个字节。

BinaryWriter 类常用的方法：

方法名	说明
Close()	关闭当前文件
Write()	将值写入当前流。参数： int,long,bool,char,string, 数组等
Seek()	设置当前流的位置 long Seek(int offset, SeekOrigin origin)
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

BinaryReader 类常用的方法：

方法名	说明
Close()	关闭当前文件
Read()	从文件中读取字符并提升一个字节的位置
PeekChar()	返回下一个可用字符，不提升字符或字节的位置

BinaryWriter 有 Seek, BinaryReader 没有 Seek ，但可以：

br.BaseStream.Seek()

例：用 BinaryWriter 往文件写入 Int16 类型数据。

```
using System.IO;
FileStream fs=new FileStream("E:\\ 数据 .txt",FileMode.Append);
BinaryWriter w=new BinaryWriter(fs); // 括号中必须是 FileStream
for(Int16 i =70; i < 82; i++ )
w.Write(i);
w.Close();
```

4　注册表的读写

注册表是 Windows 中的一个重要的数据库，用于存储系统和应用程序的设置信息。从 Windows 95 开始成为 Windows 用户经常接触的内容，并在其后的操作系统中继续沿用至今。

注册表是 Windows 建造的一个复杂的信息数据库，它是多层次式的。Windows 操作系统中打开注册表有两种方式，在【菜单】|【开始】|【运行】下输入 regedit 或 regedt32 即可。

注册表类

Microsoft.Win32 命名空间下有两个类操作注册表： Registry 类和 RegistryKey 类。

1.Registry

Registry 类是一个静态类，提供访问键 / 值对的静态方法。它是一个存储设备，包含有关应用程序、用户和默认系统设置的信息。例如，可以存储颜色选项、屏幕位置和窗口大小等；也可以使用注册表存储应用程序关闭后需要保留的信息，并在应用程序重新加载时访问这些信息。常用字段（只读）：

ClassesRoot	定义文档的类型以及那些类型相关联属性。读取基项 HKEY_CLASSES_ROOT
CurrentConfig	包含有关非用户特定的硬件的配置信息。读取基项 HKEY_CURRENT_CONFIG
CurrentUser	包含有关当前用户首选项的信息。读取基项 HKEY_CURRENT_USER
LocalMachine	包含本地计算机的配置数据。读取基项 HKEY_LOCAL_MACHINE
Users	包含有关默认用户配置的信息。读取基项 HKEY_USERS

Registry 类提供了两个常用的方法： GetValue() 方法和 SetValue() 方法。GetValue() 方法用于检索与指定注册表项中与名称关联的值； SetValue() 方法用于设置指定注册表项的指定值，如果指定的项不存在，则创建该项。

2.RegistryKey

RegistryKey 类是一个密封类，它封装了对注册表项的操作方法，包括读取、写入和删除等。 RegistryKey 类最常用的属性有 3 个：

- Name 属性 检索项的名称
- SubKeyCount 属性 检索当前项的子项数目
- ValueCount 属性 检索项中值的计数

RegistryKey 类中也包含多个方法：

方法名	说明
Close()	关闭该项，如果该项的内容已修改，则将该项刷新到磁盘
CreateSubKey()	创建一个新子项或打开一个现有项
DeleteSubKey()	删除指定的项，字符串 subkey 不区分大小写
DeleteValue()	从此项中删除指定值
GetValue()	检索与指定名称关联的值
GetSubKeyNames()	检索包含所有子项名称的字符串数组
OpenSubKey()	检索指定的项
SetValue()	设置注册表项中的名称对应的值

例：写入注册表

```
using Microsoft.Win32;
private void button1_Click(object sender, EventArgs e)
    {    WTRegedit(textBox1.Text,  textBox2.Text);
         MessageBox.Show(" 添加成功 ", " 成功提示窗 ");
    }
private void WTRegedit (string name, string value)
    {    RegistryKey  hklm = Registry.CurrentUser;
         RegistryKey  software = hklm.OpenSubKey("Software",true);
         RegistryKey mysoftware = software.CreateSubKey("my_test_software");
         RegistryKey mycolor = mysoftware.CreateSubKey("color");
         RegistryKey mysize = mysoftware.CreateSubKey("size");
         mycolor.SetValue(name, value);
    }
```

例：从注册表读出键值

```
private void button1_Click(object sender, EventArgs e)
 {
     RegistryKey hklm = Registry.CurrentUser;
     RegistryKey software = hklm.OpenSubKey("Software");
     RegistryKey mycolor = software.OpenSubKey("my_test_software\\
color");
     label1.Text = mycolor.GetValue(textBox1.Text).ToString();

 }
```

删除 subkey

DeleteSubKey() 删除不包括子键的键及其所有值 ,
DeleteSubKeyTree() 不仅删除键及其值 , 还删除键下所有子键

```
private void button1_Click(object sender, EventArgs e)
      {
          RegistryKey  hklm = Registry.CurrentUser;
          RegistryKey  software = hklm.OpenSubKey("SoftwAre",true);
          software.DeleteSubKeyTree("my_test_Software");
      }
```