C# 异常处理 C# 文本文件的读写

C# 文件的输入与输出

一个 **文件** 是一个存储在磁盘中带有指定名称和目录路径的数据集合。当打开文件进行读写时，它变成一个 **流。**

从根本上说，流是通过通信路径传递的字节序列。有两个主要的流：**输入流** 和 **输出流。输入流**用于从文件读取数据（读操作），**输出流**用于向文件写入数据（写操作）。

C# I/O 类

System.IO 命名空间有各种不同的类，用于执行各种文件操作，如创建和删除文件、读取或写入文件，关闭文件等。

下表列出了一些 System.IO 命名空间中常用的非抽象类：

I/O 类	描述
BinaryReader	从二进制流读取原始数据。
BinaryWriter	以二进制格式写入原始数据。
BufferedStream	字节流的临时存储。
Directory	有助于操作目录结构。
DirectoryInfo	用于对目录执行操作。
DriveInfo	提供驱动器的信息。
File	有助于处理文件。
FileInfo	用于对文件执行操作。
FileStream	用于文件中任何位置的读写。
MemoryStream	用于随机访问存储在内存中的数据流。
Path	对路径信息执行操作。
StreamReader	用于从字节流中读取字符。
StreamWriter	用于向一个流中写入字符。
StringReader	用于读取字符串缓冲区。
StringWriter	用于写入字符串缓冲区。

FileStream 类

System.IO 命名空间中的 **FileStream** 类有助于文件的读写与关闭。该类派生自抽象类 Stream。

您需要创建一个 **FileStream** 对象来创建一个新的文件，或打开一个已有的文件。创建 **FileStream** 对象的语法如下：

```
FileStream <object_name> = new FileStream( <file_name>,
<FileMode Enumerator>, <FileAccess Enumerator>, <FileShare Enumerator>);
```

例如，创建一个 FileStream 对象 **F** 来读取名为 **sample.txt** 的文件：

```
FileStream F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read, FileShare.Read);
```

参数	描述
FileMode	**FileMode** 枚举定义了各种打开文件的方法。FileMode 枚举的成员有： ● **Append**：打开一个已有的文件，并将光标放置在文件的末尾。如果文件不存在，则创建文件。 ● **Create**：创建一个新的文件。如果文件已存在，则删除旧文件，然后创建新文件。 ● **CreateNew**：指定操作系统应创建一个新的文件。如果文件已存在，则抛出异常。 ● **Open**：打开一个已有的文件。如果文件不存在，则抛出异常。 ● **OpenOrCreate**：指定操作系统应打开一个已有的文件。如果文件不存在，则用指定的名称创建一个新的文件打开。 ● **Truncate**：打开一个已有的文件，文件一旦打开，就将被截断为零字节大小。然后我们可以向文件写入全新的数据，但是保留文件的初始创建日期。如果文件不存在，则抛出异常。
FileAccess	**FileAccess** 枚举的成员有：**Read**、**ReadWrite** 和 **Write**。
FileShare	**FileShare** 枚举的成员有： ● **Inheritable**：允许文件句柄可由子进程继承。Win32 不直接支持此功能。 ● **None**：谢绝共享当前文件。文件关闭前，打开该文件的任何请求（由此进程或另一进程发出的请求）都将失败。 ● **Read**：允许随后打开文件读取。如果未指定此标志，则文件关闭前，任何打开该文件以进行读取的请求（由此进程或另一进程发出的请求）都将失败。但是，即使指定了此标志，仍可能需要附加权限才能够访问该文件。 ● **ReadWrite**：允许随后打开文件读取或写入。如果未指定此标志，则文件关闭前，任何打开该文件以进行读取或写入的请求（由此进程或另一进程发出）都将失败。但是，即使指定了此标志，仍可能需要附加权限才能够访问该文件。 ● **Write**：允许随后打开文件写入。如果未指定此标志，则文件关闭前，任何打开该文件以进行写入的请求（由此进程或另一进过程发出的请求）都将失败。但是，即使指定了此标志，仍可能需要附加权限才能够访问该文件。 ● **Delete**：允许随后删除文件。

实例

下面的程序演示了 **FileStream** 类的用法：

实例

```
using System;
using System.IO;

namespace FileIOApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream F = new FileStream("test.dat",
            FileMode.OpenOrCreate, FileAccess.ReadWrite);

            for (int i = 1; i <= 20; i++)
            {
                F.WriteByte((byte)i);
            }

            F.Position = 0;

            for (int i = 0; i <= 20; i++)
            {
                Console.Write(F.ReadByte() + " ");
            }
            F.Close();
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

C# 高级文件操作

上面的实例演示了 C# 中简单的文件操作。但是，要充分利用 C# System.IO 类的强大功能，您需要知道这些类常用的属性和方法。

在下面的章节中，我们将讨论这些类和它们执行的操作。请单击链接详细了解各个部分的知识：

主题	描述
文本文件的读写	它涉及到文本文件的读写。**StreamReader** 和 **StreamWriter** 类有助于完成文本文件的读写。

二进制文件的读写	它涉及到二进制文件的读写。**BinaryReader** 和 **BinaryWriter** 类有助于完成二进制文件的读写。
Windows 文件系统的操作	它让 C# 程序员能够浏览并定位 Windows 文件和目录。

← C# 异常处理 C# 文本文件的读写 →

2 篇笔记

写笔记

文件属性操作

File类与FileInfo都能实现。静态方法与实例化方法的区别！

```
//use File class
Console.WriteLine(File.GetAttributes(filePath));
File.SetAttributes(filePath,FileAttributes.Hidden | FileAttributes.ReadOnly);
Console.WriteLine(File.GetAttributes(filePath));

//user FilInfo class
FileInfo fi = new FileInfo(filePath);
Console.WriteLine(fi.Attributes.ToString());
fi.Attributes = FileAttributes.Hidden | FileAttributes.ReadOnly; //隐藏与只读
Console.WriteLine(fi.Attributes.ToString());

//只读与系统属性，删除时会提示拒绝访问
fi.Attributes = FileAttributes.Archive;
Console.WriteLine(fi.Attributes.ToString());
```

aim 2年前 (2017-07-20)

文件路径

文件和文件夹的路径操作都在Path类中。另外还可以用Environment类，里面包含环境和程序的信息。

```
string dirPath = @"D:\TestDir";
string filePath = @"D:\TestDir\TestFile.txt";
Console.WriteLine("<<<<<<<<<<{0}>>>>>>>>>>", "文件路径");
//获得当前路径
Console.WriteLine(Environment.CurrentDirectory);
//文件或文件夹所在目录
Console.WriteLine(Path.GetDirectoryName(filePath));     //D:\TestDir
Console.WriteLine(Path.GetDirectoryName(dirPath));      //D:\
//文件扩展名
Console.WriteLine(Path.GetExtension(filePath));         //.txt
//文件名
Console.WriteLine(Path.GetFileName(filePath));          //TestFile.txt
```

```
Console.WriteLine(Path.GetFileName(dirPath));           //TestDir
Console.WriteLine(Path.GetFileNameWithoutExtension(filePath)); //TestFile
//绝对路径
Console.WriteLine(Path.GetFullPath(filePath));          //D:\TestDir\TestFile.txt
Console.WriteLine(Path.GetFullPath(dirPath));           //D:\TestDir
//更改扩展名
Console.WriteLine(Path.ChangeExtension(filePath, ".jpg"));//D:\TestDir\TestFile.jpg
//根目录
Console.WriteLine(Path.GetPathRoot(dirPath));           //D:\
//生成路径
Console.WriteLine(Path.Combine(new string[] { @"D:\", "BaseDir", "SubDir", "TestFile.txt"
 })); //D:\BaseDir\SubDir\TestFile.txt
//生成随即文件夹名或文件名
Console.WriteLine(Path.GetRandomFileName());
//创建磁盘上唯一命名的零字节的临时文件并返回该文件的完整路径
Console.WriteLine(Path.GetTempFileName());
//返回当前系统的临时文件夹的路径
Console.WriteLine(Path.GetTempPath());
//文件名中无效字符
Console.WriteLine(Path.GetInvalidFileNameChars());
//路径中无效字符
Console.WriteLine(Path.GetInvalidPathChars());
```

aim 2年前 (2017-07-20)