# NET Core CSharp 文件操作

## 简介

## 文件操作在我们C#里还是比较常见的，例如我们读取Excel、Txt文件的内容，在程序中，这些文件都是以流的方式读取进入我们内存中，本节内容会详细的讲解有关文件的操作的方法。

## 我们从最大的驱动器开始往文件这个最小的开始讲解

### 驱动器读取

### 在Windows操作系统中，存储介质统称为驱动器，硬盘由于可以划分为多个区域，每一个区域称为一个驱动器。.NET提供DriveInfo类和 DriveType枚举型，以方便在程序中直接使用驱动器。DriveInfo类的常用字段成员有DriveFormat(文件系统格式，如NTFS或FAT32)、DriveType(驱动器类型)、Name（驱动器名）、TotalSize(总空间)、TotalFreeSpace(获得驱动器可用空间)。常用的方法成员有GetDrives(获得可用驱动器列表)。

### DriveType枚举型的枚举值有CDRom(光驱)、Fixed(硬盘)、Network(网络驱动器)和Removeable(软盘或U盘)等。例如，以下代码可以输出每一个硬盘驱动器的剩余空间信息。

DriveInfo[] drivers = DriveInfo.GetDrives();

foreach(DriveInfo driver in drivers)

{

if(driver.DriveType == DriveType.Fixed && driver.DriveFormat == "NTFS")

{

Console.WriteLine("在{0}驱动器上还有{1}字节的剩余空间。", driver.Name, driver.AvailableFreeSpace);

}

}

Console.ReadLine();

### 目录

### 为了方便检索文件，需要在驱动器中先创建目录，然后把文件保存到这个目录中。在Windows操作系统中，目录又称文件夹。每个驱动器都有一个根目录，使用”\”表示，如”C:\”表示C驱动器的根目录。创建在根目录中的目录称为一级子目录。在一级子目录中创建的目录称为二级子目录，依此类推。文件系统的目录结构是一种树形结构。

### .NET提供了Directory类和DirectoryInfo类，以方便在程序中直接操作目录。

### Directory类的常用方法成员有CreateDirectory(创建新目录)、Delete(删除目录)、Exists(判断目录是否存在)、Move(移动目录)、GetFiles(获得目录的文件列表)、GetDirectories(获得子目录列表)等。

### DirectoryInfo类的常用字段成员有Name(提取目录名)、Exists（判断目录是否存在）、Parent(父目录)、Root(根目录)、MoveTo(移动目录)、GetFiles(获得目录的文件列表)、GetDirectories(获得子目录列表)等。例如，以下代码分别展现了Directory类和DirectoryInfo类的基本方法。

Directory.CreateDirectory(@"d:\C#程序设计");

if(Directory.Exists(@"d:\C#程序设计"))

{

Console.WriteLine("创建成功");

}

Directory.Delete(@"d:\C#程序设计");

if (!Directory.Exists(@"d:\C#程序设计"))

{

Console.WriteLine("删除成功");

}

DirectoryInfo dir = new DirectoryInfo(@"d:\C#程序设计");

if (!dir.Exists)

{

dir.Create();

}

else

{

Console.WriteLine("该目录已经存在");

}

### 文件

### 文件的操作主要是分为两个类，一个是File类，一个是FileInfo类。File和FileInfo类位于System.IO命名空间，都可以用来实现创建、复制、移动、打开文件等操作。File类和FileInfo类与Directory类和DirectoryInfo类的工作方式相似。File类是一个静态类，可直接调用其方法成员。FileInfo类不是静态类，需要先创建实例。

### File类

### File类是一个文件的基本操作类，提供用于创建、复制、删除、移动和打开文件的静态方法，并协助创建 FileStream 对象。里面包含的大多都是实际操作文件的方法，我用一个列表表示它内部常用的方法

#### Open(string path, FileMode mode)+3重载 打开指定路径文件

#### Create(string path)+3重载，在指定路径创建文件，可以指定缓冲区大小

#### Copy(string sourceFileName, string destFileName)，复制文件

#### Delete(string path),删除文件

#### Exists(),判断文件是否存在

#### Move(string sourceFileName, string destFileName)，移动文件

#### Replace(string sourceFileName,string destinationFileName, string destinationBackupFileName)，替换文件

#### AppendAllText(string path, string contents)新建文件并添加文本

#### ReadAllText(string path, Encoding encoding)+2重载，打开并读取文本内容（可以指定打开的编码格式）

#### 这些方法的使用也是非常简单，例如：

string path = @"E:\test.txt";

*//以路径为参数创建文件*

#### File.Create(path);

#### 更多的使用你最好自己去尝试，我们这几节的内容都是一些常用的类和方法的实际运用，并没有过多的知识点，你只需要按着这里的操作一直尝试即可。

**FileInfo类**

#### FileInfo类也可以对文件进行操作，文件信息类FileInfo与File类不同，它虽然也提供类创建、复制、删除、移动和打开文件的方法，并且帮助创建FileStream对象，但是它提供的仅仅是实例方法他是我们最常用的文件操作类。

#### 它里面含有的属性有：

#### Name 提取文件名

#### Directory 所属目录

#### Exists 是否存在（继承自父类FileSystemInfo）

#### Extension 文件扩展名

#### Length 文件长度

#### IsReadOnly 是否为只读

#### 常用的方法和File类类似，这里添加几个之前没有讲过的

#### EnCrypt() 加密文件

#### Decrypt() 解密文件

#### CopyTo() 复制到新文件

#### MoveTo() 移动文件

#### 与文件类File和文件夹类Directory相比，文件信息类FileInfo和文件夹信息类DirectoryInfo具有其大部分功能。

#### File类和Directory类适合对不同的对象进行单一的处理。此种特殊情况下，静态方法的调用速度比较快，不用进行实例化。

#### FileInfo类和DirectoryInfo类适合用于对同一文件或文件夹进行多种操作的情况。此种情况下，实例化后的对象不需要每次都寻找文件，可以直接对该文件进行操作。

**文件流**

#### 文件流对我们而言是一个陌生的概念，事实上，你仔细思考一下，我们是如何通过File类和FileInfo类读取文件中的内容呢？它其实就是以文件流的形式进行读写。流包括以下基本操作：

#### 读取（read）：把数据从流传输到某种数据结构中，如输出到字符数组中。

#### 写入（write）：把数据从某种数据结构传输到流中，如把字节数组中的数据传输到流中。

#### 定位（seek）：在流中查找或重新定位当前位置。

#### 在C#中，所有的流操作都继承于Stream类，Stream类的主要属性有

#### CanRead（是否支持读取）

#### CanSeek（是否支持查找）

#### CanTimeout（是否可以超时）

#### CanWrite（是否支持写入）

#### Length（流的长度）

#### Position（获取或设置当前流中的位置）

#### ReadTimeout（获取或设置读取操作的超时时间）

#### WriteTimeout（获取或设置写操作的超时时间）

#### 主要方法有

#### BeginRead（开始异步读操作）

#### BeginWrite（开始异步写操作）

#### Close（关闭当前流）

#### EndRead（结束异步读操作）

#### EndWrite（结束异步写操作）

#### Flush（清除流的所有缓冲区并把缓冲数据写入基础设备）

#### Read（读取字节序列）

#### ReadByte（读取一个字节）

#### Seek（设置查找位置）

#### Write（写入字节序列）

#### WriteByte（写入一个字节）。

**FileStream、MemoryStream和BufferedStream类**

### 文件流类FileStream以流的形式读、写、打开、关闭文件。另外，它还可以用来操作诸如：管道、标准输入/输出等其他与文件相关的操作系统句柄。

### 内存流MemoryStream类用来在内存中创建流，以暂时保持数据，因此有了它就无须在硬盘上创建临时文件。它将数据封装为无符号的字节序列，可以直接进行读、写、查找操作。

### 缓冲流BufferedStream类表示把流先添加到缓冲区，再进行数据的读/写操作。缓冲区是存储区中用来缓存数据的字节块。使用缓冲区可以减少访问数据时对操作系统的调用次数，增强系统的读/写功能。

### 注意，FileStream类也有缓冲功能，在创建FileStream类的实例时，只需要指定缓冲区的大小即可。

示例

*//要写入文件的字符数组*

char[] m\_cDataWrite = new char[100];

*//包含要写入该流的数据的缓冲区*

byte[] m\_bDataWrite = new byte[100];

try

{

*//创建d:\file.txt的FileStream对象*

FileStream m\_FileStream = new FileStream(@"d:\file.txt", FileMode.OpenOrCreate);

*//将要写入的字符串转换成字符数组*

m\_cDataWrite = "test filestream".ToCharArray();

*//通过UTF-8编码方法将字符数组转成字节数组*

Encoder m\_Enc = Encoding.UTF8.GetEncoder();

m\_Enc.GetBytes(m\_cDataWrite, 0, m\_cDataWrite.Length, m\_bDataWrite, 0, true);

*//设置流当前位置为文件开始位置*

m\_FileStream.Seek(0, SeekOrigin.Begin);

*//将字节数组中的内容写入文件*

m\_FileStream.Write(m\_bDataWrite, 0, m\_bDataWrite.Length);

if (m\_FileStream != null)

{

*//清除此流的缓冲区，使得所有缓冲的数据都写入到文件中*

m\_FileStream.Flush();

m\_FileStream.Close();

}

}

catch (Exception ex)

{

Console.WriteLine("There is an IOException");

Console.WriteLine(ex.Message);

}

Console.WriteLine("Write to File Succeed!");

### StreamReader和StreamWriter类

#### 流读取器StreamReader类用来以一种特定的编码（如：UTF-8）从字节流中读取字符，流写入器StreamWriter类用来以一种特定的编码（如：UTF-8）向流中写入字符。StreamReader和StreamWriter类一般用来操作文本文件。

#### 示例

try

{

*//保留文件现有数据，以追加写入的方式打开d:\file.txt文件*

StreamWriter m\_SW = new StreamWriter(@"d:\file.txt", true);

*//向文件写入新字符串，并关闭StreamWriter*

m\_SW.WriteLine("Another File Operation Method");

m\_SW.Close();

}

catch (Exception ex)

{

Console.WriteLine("There is an IOException");

Console.WriteLine(ex.Message);

}

StreamWriter类提供了另一种从文件中读取数据的方法，下面演示其用法：

try

{

*//以绝对路径方式构造新的StreamReader对象*

StreamReader m\_SR = new StreamReader(@"d:\file.txt");

*//用ReadToEnd方法将d:\file.txt中的数据全部读入到字符串m\_Data中，并关闭StreamReader*

string m\_Data = m\_SR.ReadToEnd();

m\_SR.Close();

Console.WriteLine(m\_Data);

}

catch (Exception ex)

{

Console.WriteLine("There is an IOException");

Console.WriteLine(ex.Message);

}

### BinaryReader和BinaryWriter类

#### BinaryReader类用特定的编码将基元数据类型读作二进制。BinaryWriter类以二进制形式将基元类型写入流，并支持用特定的编码写入字符串。如果你需要查看写入的内容，你需要使用专门的二进制浏览器而不是记事本

示例

FileStream m\_FS = new FileStream(@"d:\data.dat", FileMode.Create);

*//通过文件流创建相应的BinaryWriter*

BinaryWriter m\_BW = new BinaryWriter(m\_FS);

for(int i = 0; i < 11; i++)

{

*//向d:\data.dat中写入数据*

m\_BW.Write((int)i);

}

m\_BW.Close();

m\_FS.Close();

FileStream m\_FS = new FileStream(@"d:\data.dat", FileMode.Open, FileAccess.Read);

*//通过文件流创建相应的BinaryReader*

BinaryReader m\_BR = new BinaryReader(m\_FS);

*//从d:\data.dat中读取数据*

for(int i = 0; i < 11; i++)

{

Console.WriteLine(m\_BR.ReadInt32());

}

m\_BR.Close();

m\_FS.Close();

Console.ReadLine();