

A close-up, low-angle shot of a white sailboat's hull and rigging on the left side of the frame. The boat is moving through a deep blue sea with white-capped waves. In the background, a range of mountains is visible under a warm, orange-hued sky, suggesting a sunset or sunrise. The sun is partially obscured by the boat's mast, creating a lens flare effect.

文件操作

siki 微信: devsiki QQ:804632564

下面要学习的知识点

- 1, 通过FileInfo和DirectoryInfo类来读取文件和文件夹属性
查看文件属性, 创建文件, 移动文件, 重命名文件
判断路径是否存在, 创建目录
- 2, 通过File读写文件
读写文件
- 3, 使用流来读写文件

FileStream

StreamReader (读取流-读取数据) StreamWriter (写入流-向别人传输)



文件系统

下面的类用于浏览文件系统和执行操作，比如移动，复制和删除文件。

`System.MarshalByRefObject` 这个是.NET类中用于远程操作的基对象类，它允许在应用程序域之间编组数据。

`FileSystemInfo` 这是表示任何文件系统对象的基类

`FileInfo`和`File` 这些类表示文件系统上的文件

`DirectoryInfo`和`Directory` 表示文件系统上的文件夹

`Path` 包含用于处理路径名的一些静态方法

`DriveInfo` 它的属性和方法提供了指定驱动器的信息



表示文件和文件夹的.NET类

我们有两个用于表示文件夹的类和两个用于表示文件的类

Directory（文件夹）和File（文件）类只包含静态方法，不能被实例化。如果只对文件夹或文件执行一个操作，使用这些类就 very 有效，省去了去实例化.NET类的系统开销。

DirectoryInfo类和FileInfo类实现与Directory和File相同的公共方法，他们拥有一些公共属性和构造函数，这些类的成员都不是静态的。需要实例化这些类，之后把每个实例与特定的文件夹或者文件关联起来。如果使用同一个对象执行多个操作，使用这些类比较合适，这是因为在构造的时候他们将读取合适文件系统对象的身份验证和其他信息，无论每个对象调用了多少方法，都不需要再次读取这些信息。



FileInfo和DirectoryInfo类

对于FileInfo和DirectoryInfo类中的很多方法也可以使用File和Directory中的很多方法实现。

1, 完成一个文件的拷贝

```
FileInfo myFile = new FileInfo(@"c:\pxx\xx\xxx\xxx.txt");  
myFile.CopyTo(@"d:\xx\xx.txt");//拷贝文件
```

对应的File处理方式

```
File.Copy(@"c:\xxx\xx\xx\xx.txt", @"d:\xx\xx\xx.txt");
```

2, 判断一个文件夹是否存在

```
DirectoryInfo myFolder = new DirectoryInfo(@"c:\program files");  
myFolder.Exists
```

对于FileInfo, 或者DirectoryInfo进行构造的时候, 如果传递了一个不存在的文件或者文件夹路径, 这个时候不会出现异常, 只有当你使用这个文件或者文件夹的时候才会出现问题。

FileInfo和DirectoryInfo的对象都可以通过Exists属性判断这个文件或者文件夹是否存在



FileInfo和DirectoryInfo的属性列表

CreationTime	创建文件或文件夹的时间
DirectoryName(用于FileInfo)	包含文件夹的完整路径
Parent(用于DirectoryInfo)	指定子目录的父目录
Exists	文件或文件夹是否存在
Extension	文件的扩展名，对于文件夹，它返回空白
FullName	文件或文件夹的完整路径名
LastAccessTime	最后一次访问文件或文件夹的时间
LastWriteTime	最后一个修改文件或文件夹的时间
Name	文件或文件夹的名称
Root（仅用于DirectoryInfo）	路径的根部分
Length（仅用于FileInfo）	返回文件的大小（以字节为单位）



FileInfo和DirectoryInfo的方法列表

Create() 创建给定名称的文件夹或者空文件，对于FileInfo, 该方法会返回一个流对象，以便于写入文件。

Delete() 删除文件或文件夹。对于文件夹有一个可以递归的Delete选项

MoveTo() 移动或重命名文件或文件夹

CopyTo() (只用于FileInfo)复制文件，文件夹没有复制方法，如果想要复制完整的目录树，需要单独复制每个文件和文件夹

GetDirectories() (只适用于DirectoryInfo)返回DirectoryInfo对象数组，该数组表示文件夹中包含的所有文件夹

GetFiles() (只适用于DirectoryInfo)返回FileInfo对象数组，该数组表示文件夹中所有的文件

GetFileSystemInfos() (只适用于DirectoryInfo)返回FileInfo和DirectoryInfo对象，它把文件夹中包含的所有对象表示为一个FileSystemInfo引用数组



小案例

创建时间，最后一次访问时间和最后一次写入时间都是可写入的。

```
FileInfo test = new FileInfo(@"c:\myfile.txt");  
Console.WriteLine(test.Exists);  
Console.WriteLine(test.CreationTime);  
test.CreationTime = new DateTime(2010, 1, 1, 7, 20, 0);  
Console.WriteLine(test.CreationTime);
```



Path类

我们不能去实例化Path类，Path类提供了一些静态方法，可以更容易的对路径名执行操作。

```
Console.WriteLine(Path.Combine(@"c:\my documents", "Readme.txt"));
```

在不同的操作系统上，路径的表示是不一样的 windows上是 \ ， 在Unix就是/ ，我们可以使用Path.Combine连接两个路径，不用关心在哪个系统上。



Path类的一些静态字段

<code>AltDirectorySeparatorChar</code>	提供分割目录的字符，在windows上使用 \ 在Unix上用 /
<code>DirectorySpeparatorChar</code>	提供分割目录的字符，在windows上使用 / 在Unix上用 \
<code>PathSeparator</code>	提供一种与平台无关的方式，来指定划分环境变量的路径字符串，默认为分号
<code>VolumeSepartorChar</code>	提供一种与平台无关的方式，来指定容量分割符，默认为冒号



读写文件

在.NET 2.0扩展了File类，通过File可以读写文件。



读取文件

在.NET 2.0扩展了File类，通过File可以读写文件。

1, `File.ReadAllText (FilePath)`; 根据文件路径读取文件中所有的文本

2, `File.ReadAllText (FilePath, Encoding)`; //Encoding可以指定一个编码格式
`Encoding.ASCII;`

3, `File.ReadAllBytes ()` 方法可以打开二进制文件把内容读入一个字节数组

4, `File.ReadAllLines ()` 以行的形式读取文件，一行一个字符串，返回一个字符串的数组



写入文件

我们读取文件有ReadAllText() ReadAllLines()和ReadAllBytes()这几个方法，对应的写入文件的方法有WriteAllText() WriteAllLines()和WriteAllBytes()



流

流是一个用于传输数据的对象，数据可以向两个方向传输：

如果数据从外部源传输到程序中，这就是读取流

如果数据从程序传输到外部源中，这就是写入流

外部源可能是

一个文件

网络上的数据

内存区域上

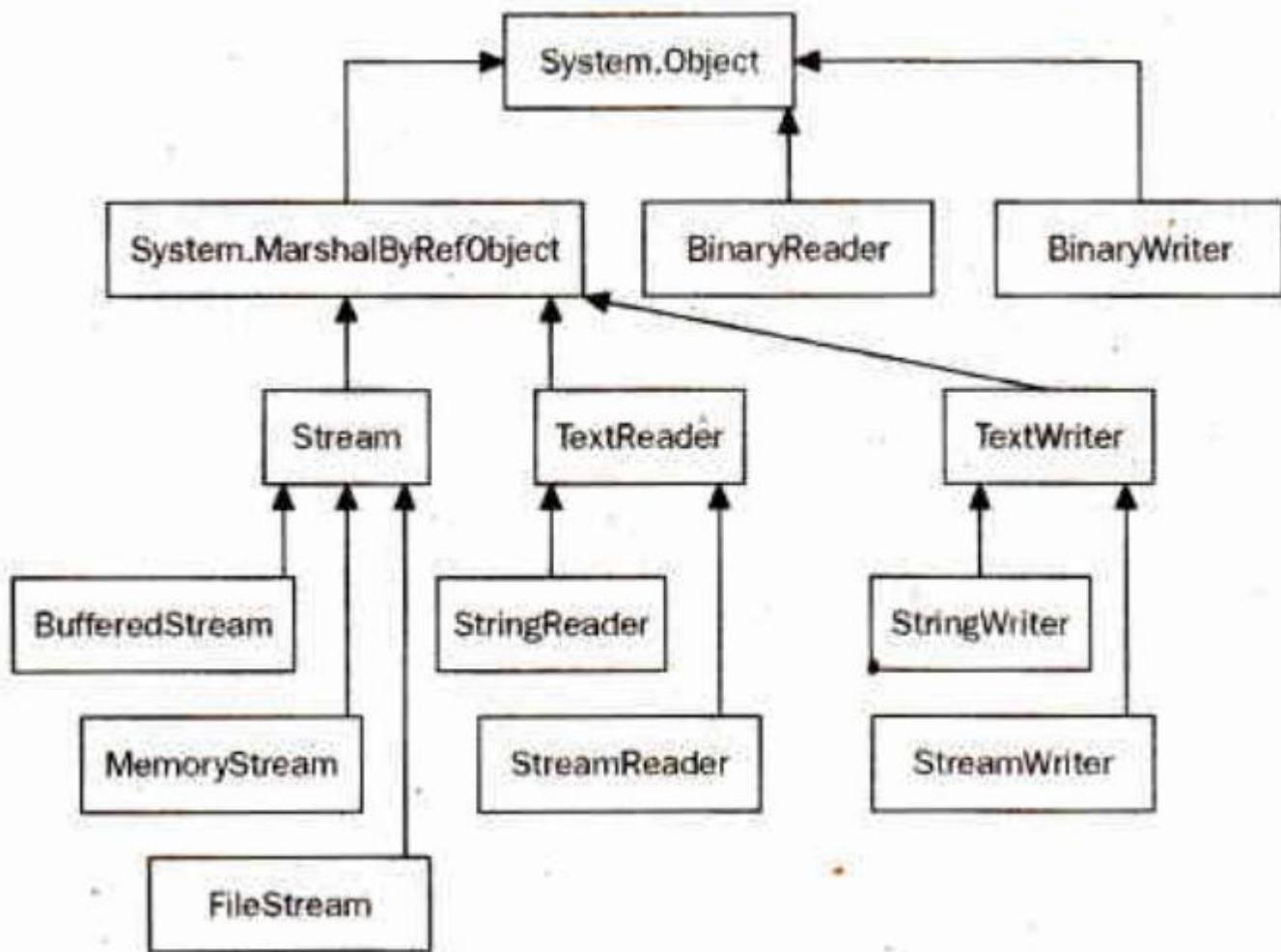
读写到命名管道上

读写内存使用`System.IO.MemoryStream`

处理网络数据使用`System.Net.Sockets.NetworkStream`



与流相关的类



`FileStream`(文件流) 这个类主要用于二进制文件中读写，也可以使用它读写任何文件。
`StreamReader`(流读取器)和
`StreamWriter`(流写入器)专门用于读写文本文件



使用FileStream类读写二进制文件

FileStream实例用于读写文件中的数据，要构造FileStream实例，需要提供下面的4中信息：

要访问的文件 - 一般提供一个文件的完整路径名

表示如何打开文件的模式 - 新建文件或打开一个现有文件，如果打开一个现有的文件，写入操作是覆盖文件原来的内容，还是追加到文件的末尾？

表示访问文件的方式 - 只读 只写 还是读写

共享访问 - 表示是否独占访问文件，如果允许其他流同时访问文件，则这些流是只读 只写 还是读写文件



构造函数的参数

构造函数的参数的取值

FileMode(打开模式)

Append, Create, CreateNew, Open, OpenOrCreate和

Truncate

FileAccess(读取还是写入) Read, ReadWrite和Write

FileShare(文件共享设置) Delete, Inheritable, None, Read, ReadWrite和Write

注意

如果文件不存在 Append Open和Truncate会抛出异常

如果文件存在 CreateNew会抛出异常

Create 和 OpenOrCreate Create会删除现有的文件, 新建一个空的文件, OpenOrCreate会判断当前是否有文件, 没有的话才会创建



创建FileStream文件流

```
FileStream fs = new FileStream(@"c:\xx\xx.doc", FileMode.Create);  
FileStream fs2 = new FileStream(@"c:\xx\xx.doc", FileAccess.Write);  
FileStream fs3 = new FileStream(@"c:\xx\xx.doc", FileAccess.Write, FileShare.None);
```



通过FileInfo打开文件流

```
FileInfo myfile1 = new FileInfo(@"c:\xx\xx.doc");  
FileStream fs4= myfile1.OpenRead();  
FileInfo myfile2 = new FileInfo(@"c:\xx\xx.doc");  
FileStream fs5= myfile2.OpenWrite();  
FileInfo myfile3 = new FileInfo(@"c:\xx\xx.doc");  
FileStream fs6= myfile3.Open(FileMode.Append, FileAccess.Write, FileShare.None);  
FileInfo myfile4 = new FileInfo(@"c:\xx\xx.doc");  
FileStream fs7= myfile4.Create();
```

FileStream文件流的关闭

当我们使用完了一个流之后，一定要调用`fs.Close()`方法去关闭流，关闭流会释放与它相关联的资源，允许其他应用程序为同一个文件设置流。这个操作也会刷新缓冲区。



从文件流中读取内容和写入内容

从文件流读取内容

1, `int nextByte = fs.ReadByte();` 读取一个字节 (0-255) 的数据, 返回结果, 如果达到流的末尾, 就返回-1

2, `int nBytesRead = fs.Read(ByteArray, 0, nBytes);` // 读取多个字节, 第一个是存放的数组, 第二个参数是开始存放的索引, 第三个参数是要读取多少个字节。返回的结果是读取的自己的实际个数, 如果达到流的末尾 返回-1

向文件流写入内容

1, `byte NextByte = 100; fs.WriteByte(NextByte);` 把一个字节写入文件流中

2, `fs.Write(ByteArray, 0, nBytes);` 把一个自己数组写入文件流中 参数同上



读写文本文件

我们对文本文件的读写一般使用StreamReader和StreamWriter, 因为不同的文本有不同的编码格式, 这个StreamReader会帮我们自动处理, 所以我们不需要关心文本文件的编码是什么

1, 创建文本的读取流(会检查字节码标记确定编码格式)

```
StreamReader sr = new StreamReader(@"c:\xx\ReadMe.txt");
```

2, 指定编码格式

```
StreamReader str = new StreamReader(@"c:\xx\xx.txt", Encoding.UTF8);
```

(可取的编码格式 ASCII Unicode UTF7 UTF8 UTF32)

3, 在文件流的基础上创建文本读取流

```
FileStream fs = new FileStream(@"c:\xx\xx.txt", FileMode.Open, FileAccess.Read, FileShare.None);
```

```
StreamReader sr = new StreamReader(fs);
```

4, 通过文件信息创建文本读取流-第二种方式

```
FileInfo myFile = new FileInfo(@"c:\xx\xx.txt");
```

```
StreamReader sr = myFile.OpenText();
```

流的关闭 sr.Close();



读取文本文件

1, string nextLine = sr.ReadLine(); // 读取一行字符串

2, string restOfStream = sr.ReadToEnd(); // 读取流中所有剩余的文本内容

3, int nextChar = sr.Read(); // 只读取一个字符

4, int nChars = 100;

char[] charArray = new char[nChars];

int nCharsRead = sr.Read(charArray, 0, nChars); 读取多个字符，第一个参数是要存放的字符数组，第二个参数是从数组的哪一个索引开始放，第三个参数是读取多少个字符 返回值是实际读取的字符的个数



文本写入流StreamWriter-创建

StreamWriter的创建

1, StreamWriter sw = new StreamWriter(@"c:\xx\xx.txt"); (默认使用UTF-8编码)

2, StreamWriter sw = new StreamWriter(@"c:\xx\xx.txt", true, Encoding.ASCII)

第二个参数表示是否以追加的方式打开，第三个参数是编码方式

3, 通过FileStream创建StreamWriter

```
F i l e S t r e a m f s = n e w  
FileStream(@"c:\xx\xx.txt", FileMode.CreateNew, FileAccess.Write, FileShare.Read);  
StreamWriter sw = new StreamWriter(fs);
```

4, 通过FileInfo创建StreamWriter

```
FileInfo myFile = new FileInfo(@"c:\xx\xx.txt");  
StreamWriter sw = myFile.CreateText();  
所有流用完之后关闭 sw.Close();
```



文本写入流StreamWriter-写入

1, 写入一行字符

```
string nextLine = "x xx x x x x ";sw.Write(nextLine);
```

2, 写入一个字符

```
char nextChar = 'a';  
sw.Write(nextChar);
```

3, 写入字符数组

```
char[] charArray = ...;  
sw.Write(charArray);
```

4, 写入字符数组的一部分

```
sw.Write(charArray, StartIndex, Length);
```

1: 要写入的数组 2: 开始索引 3: 写入长度





关注公众号

发布最新的视频，文章和教学资源

THANK

@siki 微信: devsiki QQ:804632564
