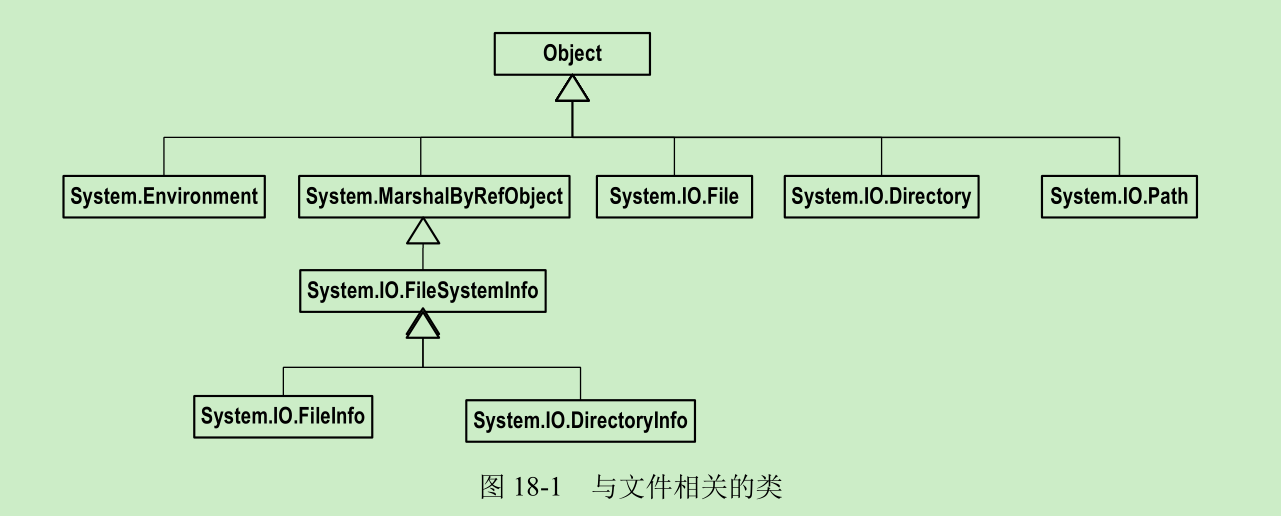
**文 件 和 流**

**8.1 文件系统**

在.NET 框架中，用于文件操作的类位于 System.IO 命名空间中，用这些类可以方便地对文件进行创建、读写、复制、删除、移动、打开等操作。



18.1.1 File类和FileInfo类

命名空间 System.IO 中的 File 类用于对文件进行创建、打开、复制、移动、删除、重命名等典型操作，并能获取或设置文件的属性信息。File类中所以的函数都是静态的，file类也是静态的，不可以实例化。

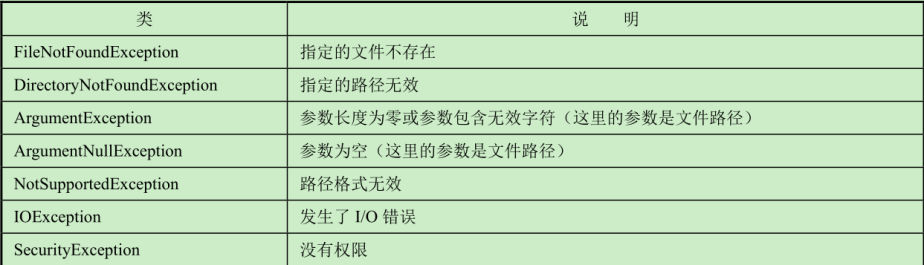
FileInfo 类和 File 类相似，可以创建、复制、移动、删除文件，可以获取或设置文件

的属性，只是少了读写文件的功能。FileInfo 类的成员方法都是非静态的，使用方法前需

要先创建一个 FileInfo 类的对象，然后通过 FileInfo 对象调用方法。当频繁操作某文件时，

使用 FileInfo 类的效率会更高。

**18.1.2 关于文件的异常**



今后我们所有关于文件的操作都会放到 try-catch 结构中，大家编写程序时也要养成这

种良好的习惯，编写出健壮性代码。

**18.1.3 Directory类和DirectoryInfo类**

System.IO 命名空间中的 Directory 类用于执行对目录（文件夹）的操作，比如创建、

移动、删除、重命名等，也可通过它获取或设置目录的属性。和File类一样，所有 Directory 类中的方法都是静态的。

*Directory*.*SetCurrentDirectory*(@"D:\weikai\nutbak");

string workingDir = *Directory*.*GetCurrentDirectory*();

string[] dirs = *Directory*.*GetDirectories*(workingDir);

*Console*.*WriteLine*("当前工作目录：{0}，子目录：", workingDir);

foreach (string dir in dirs)

{

*Console*.*WriteLine*(dir);

}

string[] files = *Directory*.*GetFiles*(workingDir);

foreach (string file in files)

{

*Console*.*WriteLine*(file);

}

*Console*.*ReadKey*();

注意：*GetDirectories*并不获取子目录的子目录，*GetFiles*只能获取文件不能获取目录。

string d = *Environment*.*GetFolderPath*(*Environment*.*SpecialFolder*.*MyDocuments*);

string p = *Environment*.*GetFolderPath*(*Environment*.*SpecialFolder*.*MyPictures*);

string m = *Environment*.*GetFolderPath*(*Environment*.*SpecialFolder*.*MyMusic*);

string f = *Environment*.*GetFolderPath*(*Environment*.*SpecialFolder*.*Favorites*);

**18.2 基于流的文件操作**

18.2.1 FileStream类

使用filestream读写文件简例：

FileStream fileStream = new FileStream(workingDir + @"\lua学习\Lua学习笔记\lua内置对象.md", *FileMode*.*Open*,*FileAccess*.*Read*);

FileStream fileCopy = new FileStream(workingDir + @"\lua学习\Lua学习笔记\lua\_copy.md", *FileMode*.*CreateNew*, *FileAccess*.*Write*);

int readbyte = -1;

while ( (readbyte = fileStream.ReadByte()) != -1)

{

fileCopy.WriteByte(*Convert*.*ToByte*(readbyte));

}

fileStream.*Close*();

fileCopy.*Close*();

问题：为什么readbyte返回int，而writebyte却使用byte类型？是否与readbyte失败返回值有关？

**using语句**

为了防止出现同步问题，当一个程序读写文件时，操作系统通常都会阻止其他程序读

写该文件，因此使用完毕后要及时关闭，否则就会导致其他程序不能使用该文件。除了可

以在 finally 块中关闭文件（流）外，我们还可以用 C#提供的 using 语句进行文件操作。

我们在using语句块中打开一个文件，当退出 using 语句时，系统会自动关闭流流对象。

using (FileStream fs = File.OpenWrite(workingDir))

{

byte[] datas = { 100, 101, 102, 103, 104, 105, 106, 107, 108, 109 };

fs.Write(datas, 0, datas.*Length*);

}// 跳出语句块时，会自动关闭文件流fs指向的文件。

using 语句实际上是对 try 语句的封装。程序先执行 try 块中的代码，不管是否发生异常，系统都会自动执行 finally 块中的代码，调用 obj 对象的 Dispose()方法，销毁对象，释放资源。

**18.2.4 用流读写二进制文件**

我们通过 BinryReader 和 BinaryWriter 类实现二进制读写。