文件

- 文件和流 I/O（输入/输出）是指在存储媒介中传入或传出数据。
- 在 .NET 中，System.IO 命名空间包含允许以异步方式和同步方式对数据流和文件进行读取和写入操作的类型

两种操作文件的方法

- C#中操作文件的方法主要有两种，一种是使用静态类File的方法，另一种是使用FileStream对象

File

- File提供了许多操作文件的方法，包括创建文件、删除文件、读取文件、写入文件等

函数或方法	描述	参数	返回值
AppendAllLines(string path, string[] contents)	将指定的字符串数组追加到指定的文件中。	path：要追加内容的文件路径。contents：要追加的字符串数组。	无
AppendAllText(string path, string contents)	将指定的字符串追加到指定的文件中。	path：要追加内容的文件路径。contents：要追加的字符串。	无
Copy(string sourceFileName, string destFileName)	将指定的文件复制到另一个文件。	sourceFileName：要复制的文件路径。destFileName：目标文件路径。	无
Create(string path)	创建一个新文件。	path：要创建的文件路径。	无
CreateText(string path)	创建一个新文本文件。	path：要创建的文件路径。	无
Delete(string path)	删除指定的文件。	path：要删除的文件路径。	无
Exists(string path)	判断指定的文件是否存在。	path：要检查的文件路径。	true：文件存在；false：文件不存在。

函数或方法	描述	参数	返回值
GetAttributes(string path)	获取指定文件的属性。	path：要获取属性的文件路径。	FileAttributes类型的对象，包含文件的属性。
GetCreationTime(string path)	获取指定文件的创建时间。	path：要获取创建时间的文件路径。	DateTime类型的对象，包含文件的创建时间。
GetLastAccessTime(string path)	获取指定文件的最后访问时间。	path：要获取最后访问时间的文件路径。	DateTime类型的对象，包含文件的最后访问时间。
GetLastWriteTime(string path)	获取指定文件的最后写入时间.	path：要获取最后写入时间的文件路径。	DateTime类型的对象，包含文件的最后写入时间。
Move(string sourceFileName, string destFileName)	移动指定的文件到另一个位置。	sourceFileName：要移动的文件路径。destFileName：目标文件路径。	无
Open(string path, FileMode mode)	打开指定的文件。	path：要打开的文件路径。mode：要打开的文件模式。	FileStream类型的对象，用于操作文件。
OpenRead(string path)	以只读模式打开指定的文件。	path：要打开的文件路径。	FileStream类型的对象，用于读取文件。
OpenText(string path)	以只读模式打开指定文本文件。	path：要打开的文件路径。	StreamReader类型的对象，用于读取文本文件。
OpenWrite(string path)	以写入模式打开指定的文件。	path：要打开的文件路径。	FileStream类型的对象，用于写入文件。
ReadAllBytes(string path)	读取指定文件的全部字节内容。	path：要读取的文件路径。	byte[]类型的数组，包含文件的全部字节内容。

函数或方法	描述	参数	返回值
ReadAllLines(string path)	读取指定文件的全部行内容。	path：要读取的文件路径。	string[]类型的数组，包含文件的全部行内容。
ReadAllText(string path)	读取指定文件的全部文本内容。	path：要读取的文件路径。	string类型的字符串，包含文件的全部文本内容。
Replace(string sourceFileName, string destinationFileName, string backupFileName)	替换指定的文件。	sourceFileName：要替换的文件路径。destinationFileName：目标文件路径。backupFileName：备份文件路径。	无
SetAttributes(string path, FileAttributes fileAttributes)	设置指定文件的属性。	path：要设置属性的文件路径。fileAttributes：要设置的文件属性。	无
WriteAllBytes(string path, byte[] bytes)	将指定的字节数组写入指定的文件。	path：要写入的文件路径。bytes：要写入的字节数组。	无
WriteAllLines(string path, string[] contents)	将指定的字符串数组写入指定的文件。	path：要写入的文件路径。contents：要写入的字符串数组。	无
WriteAllText(string path, string contents)	将指定的字符串写入指定的文件。	path：要写入的文件路径。contents：要写入的字符串。	无

示例:

```
// 创建一个名为“test.txt”的新文件
File.Create("test.txt");

// 将“Hello, world!”写入文件
File.WriteAllText("test.txt", "Hello, world!");

// 读取文件的内容
string text = File.ReadAllText("test.txt");

// 将“Hello, world!”追加到文件
```

```
File.AppendAllText("test.txt", "Hello, world!");

// 删除文件
File.Delete("test.txt");
```

FileStream

- FileStream对象提供了更细粒度的文件操作，例如设置文件的访问权限、读取和写入文件的字节流等

函数或方法	描述
FileStream(string path, FileMode mode, FileAccess access, FileShare share, FileOptions options)	创建一个FileStream对象，用于打开指定文件。
FileStream(Stream stream)	创建一个FileStream对象，用于包装指定的Stream对象。
CanRead	指示是否可以从文件中读取数据。
CanWrite	指示是否可以向文件中写入数据。
Length	获取文件的长度。
Position	获取文件的当前位置。
FileName	获取文件的名称。
FileType	获取文件的类型。
Attributes	获取文件的属性。
Handle	获取文件的句柄。
Read(byte[] buffer, int offset, int count)	从文件中读取指定长度的字节数据，并将其存储到指定的字节数组中。
Write(byte[] buffer, int offset, int count)	向文件中写入指定长度的字节数据。
ReadByte()	从文件中读取一个字节的数据。
WriteByte(byte value)	向文件中写入一个字节的数据。
ReadChar()	从文件中读取一个字符的数据。
WriteChar(char value)	向文件中写入一个字符的数据。
ReadLine()	从文件中读取一行数据。
WriteLine()	向文件中写入一行数据。
Flush()	将缓冲区中的数据写入文件。

函数或方法	描述
Close()	关闭文件。
Seek(long offset, SeekOrigin origin)	将文件的当前位置设置为指定的偏移量。
SetLength(long length)	设置文件的长度。
GetBuffer()	获取文件的缓冲区。
SetBuffer(byte[] buffer, int offset, int count)	设置文件的缓冲区。
GetLifetimeService()	获取文件的IDisposable接口。

示例:

```
// 创建一个FileStream对象，用于打开名为“test.txt”的文本文件
FileStream fs = new FileStream("test.txt", FileMode.Open);

// 从文件中读取内容
byte[] buffer = new byte[1024];
int count = fs.Read(buffer, 0, buffer.Length);

// 关闭文件
fs.Close();

// 将读取到的内容显示在控制台上
Console.WriteLine(Encoding.UTF8.GetString(buffer, 0, count));
```

File和FileStream的区别

- File是静态类，FileStream是实例类
- File类提供了更高级的抽象，例如创建、删除、读取、写入文件等
- FileStream对象则提供了更细粒度的控制，例如设置文件的访问权限、读取和写入文件的字节流等
- File适合小文件操作，FileStream适合大文件操作

异常处理

- 异常处理可以帮助程序员处理程序运行期间发生的错误，并确保程序能够正常运行

try...catch...finally

```
try {
```

```
  // 可能会发生异常的代码
} catch (Exception e) {
  // 处理异常的代码
} finally {
  // 无论是否发生异常都执行的代码
}
```

- try块中的代码是可能发生异常的代码。如果try块中的代码发生异常，则会将异常抛出到catch块中。catch块用于处理异常。finally块中的代码无论是否发生异常都将会执行
- 通常将"关闭文件"这种必须执行的操作写入finally

using

- using语句用于管理资源，例如文件、网络连接等
- using语句会在代码块结束时自动关闭资源

```
using (FileStream fs = new FileStream("test.txt", FileMode.Open)) {
  // 使用FileStream对象
}
```

在上述代码中，using语句会在代码块结束时自动关闭FileStream对象
这样可以确保无论是否发生异常，FileStream对象都会被正确关闭

- using语句还可以用于管理多个资源

```
using (FileStream fs = new FileStream("test.txt", FileMode.Open)) {
  using (StreamReader sr = new StreamReader(fs)) {
    // 使用StreamReader对象
  }
}
```

在上述代码中，using语句会在代码块结束时自动关闭FileStream对象和StreamReader对象