C#文件操作

一：文件操作和核心类

文件是任何应用程序开发时必不可少的操作，.Net框架下提供的文件操作基本都位于System.IO命名空间下，下面我们详细的介绍文件操作的核心类。

1.1 File和FileInfo类

这两个类用来操作硬盘上的文件，File类主要通过静态方法实现，而FileInfo类则是通过实例方法实现的。

成员	说明
AppendText	创建一个StreamWrite类型，用于追加文本
Create	指定文件下创建或者覆盖文件
Delete	删除指定文件
Exists	检查文件是不是存在
Open	指定特有的读写权限打开文件
OpenRead	以读取的方式打开现有文件
OpenWrite	打开或者创建一个现有文件，以写入文本
ReadAllText	读取文件的所有行，然后关闭文件
WriteAllText	写入指定的字符串，然后关闭文件

FIleInfo类的实例成员提供了与FIle类差不多的功能，大多数情况下，他们可以相互互换使用，但是由于FIle类提供的方法都是静态方法，使用频率居多。

```
public static void  WriteFile(){
        FileStream fs = null;
        StreamWriter writer = null;
        string path = "test.txt";
```

```
            if (!File.Exists (path)) {
                fs = File.Create (path);
                Console.WriteLine ("新建一个文件:{ 0}",
path);
            }  else {
                fs = File.Open (path, FileMode.Open);
                Console.WriteLine ("文件已经存在，直接打
开");
            }

            writer = new StreamWriter (fs);
            writer.WriteLine ("测试文本");
            writer.Flush ();
            writer.Close ();

            fs.Close ();
        }
```

以上代码通过File.Exists方法判断指定路径下的文件是不是存在，如果文件存在，则直接调用File.Open将文件打开，否则将调用Create方法创建一个文件。然后，代码会初始化一个StreamWriter对象，然后向文本中写入字符串操作。最后，程序会通过调用Flush方法清空缓冲区，然后将所有的缓冲区数据写入文件，并且调用CLose方法关闭数据流。

1.2 Directory和DirectoryInfo类

这两个类都包含了一组用来创建，移动，删除和枚举所有目录或者子目录的成员，如下表所示：

成员	说明
CreateDirectory	在指定路径下创建目录和子目录
Delete	删除目录
Exists	目录存在
GetFiles	获得目录下所有文件名称的数组
GetParent	获得指定目录的父目录

成员	说明
GetCurrentDirectory	获得应用程序当前的工作目录
Move	移动目录

DirectoryInfo功能与这个类似，大多数情况下两者可以互换使用。

```
     public static void directoryOpe(){
               string dirPath =
Directory.GetCurrentDirectory();
               Console.WriteLine (dirPath);
               string filePath = string.Format ("{ 0}/
{ 1}",dirPath,"hello.txt");
               if (!Directory.Exists (dirPath)) {
                    Directory.CreateDirectory (dirPath);
                    Console.WriteLine ("创建一个目录");
               }  else {
                    Console.WriteLine ("存在目录");
               }

               FileInfo file = new FileInfo (filePath);

               if (!file.Exists) {
                    file.Create ();
                    Console.WriteLine ("创建一个文件");
               }else{
                    Console.WriteLine ("存在");

          }
```

1.3遍历一个文件夹下所有的文件

```
          public static void FindAllText(){
               string[] files = Directory.GetFiles ("/Users/a/
Projects/FileOperation","*",SearchOption.AllDirectories);
               Console.WriteLine (files.Length);
               for (int i = 0, len = files.Length; i < len; i+
+) {
                    string filePath = files [i];
                    Console.WriteLine (filePath);
               }
          }
```

1.4 流Stream使用

流可以理解为内存中的字节序列，Stream是所有流的抽象基类，每个具体的存储实体都可以通过Stream派生类来实现。如FileStream就是这种存储实体。同样，流也涉及三个基本操作。

- 对流进行读取：将流中的数据读取到具体的数据结构中
- 对流进行写入：把数据结构中的数据写入到流中
- 对流进行查找：对流内的当前位置进行查询和修改

Stream类的一些常用成员如下表所示：

成员	说明
CanRead	检查当前流是否支持读取操作
CanSeek	检查当前流是否支持查找操作
CanWrite	检查当前流是否支持写入操作
Length	获取用字节表示的流的长度
Position	设置当前流中的位置
BeginRead	开始异步读操作
BeginWrite	开始异步写操作
Close	关闭当前流并且释放资源
EndRead	等待异步读操作完成
EndWrite	等待异步写操作完成
Flush	清除当前流的缓冲区，并且将数据写入存储设备
Write	向当前流写入子节序列

其中Stream有几个常见的派生类，

NetworkStream,FileStream,MemoryStream,GZipStream.

- NetworkStream提供网络通信的基础数据流
- FileStream用于将数据以流的形式写入文件，或者从文件中读取
- MemoryStream用于对内存中的数据进行写入或者读取
- GZipStream提供用于压缩和解压缩的数据流

```
public static void TestStream(){
                string filePath="test.txt";

                using (FileStream fileStream = File.Open (filePath,
FileMode.OpenOrCreate)) {
                        string msg = "HelloWorld";
                        byte[] msgByteArray =
Encoding.Default.GetBytes (msg);
                        Console.WriteLine ("开始写入文件");
                        fileStream.Write (msgByteArray,
0,msgByteArray.Length);
                        fileStream.Seek (0,SeekOrigin.Begin);
                        Console.WriteLine ("写入文件的数据为");
                        byte[] bytesFromFile=new
byte[msgByteArray.Length];

                        fileStream.Read (bytesFromFile,
0,msgByteArray.Length);

                        Console.WriteLine
(Encoding.Default.GetString(bytesFromFile));

                }


        }
```

以上代码首先调用File.Open方法来创建一个FileStream实例对象，然后调用Write方法把字符串的字节数组写入到流中，接着调用Seek方法重置流内部的位置，最后调用Read方法将数据从流中读取到字节数组，并把写入的信息输出到控制台。

System.IO命名空间提供了不同的读写器，以对流中的数据进行操作。这些类通常是成对出现的：一个从流中读取数据，另外一个写入数据。

```
    public static void TestWriter(){
              string filePath="test.txt";

              using (FileStream fileStream = File.Open (filePath,
FileMode.OpenOrCreate)) {
                   string msg = "HelloWorld";
                   StreamWriter streamWriter = new StreamWriter
(fileStream);
                   Console.WriteLine ("开始写入文件");
                   streamWriter.Write (msg);
                   Console.WriteLine ("写入文件的数据为");

                   StreamReader streamReader = new StreamReader
(fileStream);
                   string str = streamReader.ReadToEnd ();

                   Console.WriteLine (str);
                   streamWriter.Close ();
                   streamReader.Close ();

              }

         }
```

1.5 文件异步操作

前面对文件的操作都是同步的，在同步操作中，如果向文件中写入大量数据，方法将一直处于等待状态，直到写入完成。若是使用异步操作，方法就可以在写入操作的同时继续执行后面的操作。下面以FileStream为例子，介绍对文件进行异步操作的方法。

FileStream共有15个构造函数，其中只有一个构造函数可以指定异步操作，该构造函数的定义如下：

FileStream(string path,FileMode mode,FileAccess access,FileShare share,int bufferSize,bool useAsync)

最后一个参数useAsync用于指定程序使用的是异步还是同步，如果设置为true，异步方式来设置FileStream。