C#에서의 IO

흐흐.. 드디어 입출력관련 클래스입니다. 저는 이런 주제가 좋습니다. 재밌고요…^^;

System.IO네임스페이스에는 입출력에 관한 다양한 클래스가 있습니다.

한번 열거해 볼까요. 이런 기회가 아님 또 언제 이렇게 쭈욱~ 보겠습니까.

지루하게 느껴지신다면 노래라도 부르면서 한번 쭈욱 보시죠.

“시골 영감 처음 타는 기차놀이에~……”

System.IO.BinaryReader

System.IO.BinaryWriter

System.IO.BufferdStream

System.IO.Directory

System.IO.DirectoryInfo

System.IO.DirectoryNotFoundException

System.IO.EndOfStreamException

System.IO.ErrorEventArgs

System.IO.File

System.IO.FileInfo

System.IO.FileLoadException

System.IO.FileNotFoundException

System.IO.FileStream

System.IO.FileSystemEventArgs

System.IO.FileSystemInfo

System.IO.FileSystemWatcher

System.IO.InternalBufferOverflowException

System.IO.IODescriptionAttribute

System.IO.IOException

System.IO.IsolatedStorage.IsolatedStorage

System.IO.IsolatedStorage.IsolatedStorageException

System.IO.IsolatedStorage.IsolatedStorageFile

System.IO.IsolatedStorage.IsolatedStorageFileStream

System.IO.MemoryStream

System.IO.Path

System.IO.PathTooLongException

System.IO.RenamedEventArgs

System.IO.Stream

System.IO.StreamReader

System.IO.StreamWriter

System.IO.StringReader

System.IO.StringWriter

System.IO.TextReader

System.IO.TextWriter

클래스 수만 34개이군요. 자바같이 예제는 별로 없더라도 쉽게 볼 수 있는 API가 있으면 좋으련만… MSDN에도 자바처럼 각 패키지에 관련된 클래스가 쭈욱 나열된 페이지가 있나요? 아시는 분 있으시면 댓글 달아주시면 감사하겠네요. 아무튼 다시 돌아와서…

그 밖에 네임스페이스,인터페이스,열거형,구조체등이 정의되어 있습니다.

자바에서는 Exception클래스를 제외하고도 50개의 입출력관련 클래스가 정의되어 있습니다.

이 입출력관련 클래스를 여기서 다 살펴보기는 무리고요. 스터디의 목적에 맞게끔 각자 한번씩 쭈욱 공부해 보시기 바랍니다.

여기서는 우선 먼저 배우고 자주 사용하게될 파일관련 클래스들과 스트림관련 클래스들에 대해서 알아보도록 하겠습니다.

아, System.IO네임스페이스는 mscorlib.dll에 정의되어 있다고 합니다.

자자 드디어 본격적으로 파일관련 클래스들을 알아보겠습니다.

System.Object

 System.IO.Directory

 System.IO.File

 System.IO.FileSystemInfo

 System.IO.DirectoryInfo

 System.IO.FileInfo

위에 보이는 것처럼 DirectoryInfo, FileInfo클래스는 FileSystemInfo클래스를 상속합니다.

Directory : 디렉터리와 하위 디렉터리에서 만들기, 이동 및 열거를 위한

정적 메서드를 노출합니다

File : 파일에 대한 만들기, 복사, 삭제, 이동 및 열기를 위한

정적 메서드를 제공하고 FileStream 개체 만들기를 지원합니다

FileSystemInfo : FileInfo와 DirectoryInfo개체에 대한 기본 클래스 제공

DirectoryInfo : 디렉터리 및 하위 디렉터리를 만들고, 이동하고, 열거하는

인스턴스 메서드를 노출

FileInfo : 파일을 만들고, 복사하고, 삭제하고, 이동하고, 열기 위한

인스턴스 메서드를 제공하고, FileStream 개체를 만드는 것을 도와 줍니다.

디렉터리의 복사, 이동, 이름 바꾸기, 만들기 및 삭제와 같은 일반적인 작업을 수행하는데 Directory 클래스를 사용할 수 있습니다.

또한 디렉터리의 생성, 액세스 및 쓰기와 관련된 DateTime 정보를 얻고 설정하는 경우에도 Directory 클래스를 사용할 수 있습니다.

모든 Directory 메서드는 static메서드이기 때문에 한 작업만 수행하는 경우에는 해당하는 DirectoryInfo 인스턴스 메서드보다 File 메서드를 사용하는 것이 더욱 효율적일 수 있다고 합니다. 대부분의 Directory 메서드에는 조작하는 디렉터리에 대한 경로가 필요합니다.

그럼 우선 FileSystemInfo클래스의 주요 속성에는 어떤 것이 있는지 알아 보겠습니다.

속성 설명

Attributes 파일과 관련된 FileAttributes속성을 가져오거나 설정

CreateTime 파일,디렉토리 생성 시간을 가져오거나 설정

Exists 파일,디렉토리가 존재하는지 확인

Extension 파일 확장자를 가져옴

FullName 파일,디렉토리의 전체경로를 가져옴

LastAccessTime 파일,디렉토리에 마지막 접근한 시간을 가져옴

LastWriteTime 파일,디렉토리에 마지막 쓴 시간을 가져옴

Name 파일이름을 가져옴

Attributes에서 사용하는 FileAttributes열거형(enum) 의 주요 프로퍼티를 살펴 보겠습니다.

열거형 이름 설명 값

Archive 파일 아카이브상태 표시 32

Compressed 파일이 압축됨 2048

Directory 파일이 디렉토리임 16

Encrypted 파일이 암호화되어 있음 16384

Hidden 파일이 숨겨짐 2

Normal 다른 속성이 없는 일반상태 128

Office 파일이 오프라인상태임 4096

ReadOnly 읽기전용 1

System 시스템파일임 4

Temporary 임시파일 256

FileAttributes열거형(enum)은 파일에 관련한 특성(정보)를 갖고 있으며 각 값을 비트단위로 저장하고 있습니다. 그러므로 나중에 파일에 대한 특성을 수정할 때나 추가할 때 한번에 여러 값을 설정할 수 있다는 거죠.

File.SetAttributes(path, File.GetAttributes(path) | FileAttributes.Hidden);

**디렉토리정보를 확인하는 DirectoryInfo**

디렉토리정보를 알수 있는 DirectoryInfo속성을 보면 우선 FileAttributes속성을 재정의한 속성에 디렉토리정보를 알 수 있는 몇가지가 더 추가되어 있습니다.

Parent 지정된 하위 디렉토리의 부모 디렉토리를 가져옴

Root 경로의 루트부분을 가져옴

메서드 또한 FileSystemInfo클래스의 메서드를 재정의하거나 몇가지 추가한 메서드가 있습니다. MSDN에도 메서드에 대한 정보가 올라와 있고 다른 사이트에도 잘 정의되어 있기 때문에 여기서는 그냥 예제를 통해 확인해 보도록 하겠습니다.

그럼 디렉토리정보를 확인하는 간단한 코드를 작성해 보겠습니다.

using System;

using System.IO;

class Test

{

static void main(string[] args)

{

DirectoryInfo di = new DirectoryInfo(@"c:\Program Files\Internet Explorer");

 if(di.Exists)

 {

 Console.WriteLine("full path: {0}", di.FullName);

 Console.WriteLine("dir name: {0}", di.Name);

 Console.WriteLine("create date: {0}", di.CreationTime);

 Console.WriteLine("dir property: {0}", di.Attributes);

 Console.WriteLine("root path: {0}", di.Root);

 Console.WriteLine("parent dir: {0}", di.Parent);

}

 else

 {

 Console.WriteLine("not exist directory.");

}

}

}

DirectoryInfo의 생성자는 위에 코드와 같이 파일경로입니다.

그런데 좀 특이한 점이 보이죠? 경로명을 쓴 문자열앞에 “@”를 붙였습니다.

DirectoryInfo dinfo = DirectoryInfo(**@**”C:\Program Files\Internet Explorer”);

절대경로명 앞에 @를 추가하면 경로를 구분짓는 슬래시앞에 다시 슬래시를 붙이지 않아도 알아서 경로를 나타내는 슬래시로 해석합니다. 보기 참 좋죠.

그렇다고 이전 프로그래밍방식을 지원하지 않는 것은 아닙니다.

DirectoryInfo dinfo = DirectoryInfo(”C:\\Program Files\\Internet Explorer”);

이것도 똑같이 작동합니다.

**파일정보를 확인하는 FileInfo**

파일 정보는 FileInfo클래스를 통해 알 수 있습니다.

파일을 만들고, 복사하고, 삭제하고, 이동하고, 열기 위한 인스턴스 메서드를 제공하고, FileStream 개체를 만드는 것을 도와 줍니다.

다음 예제에서는 FileInfo개체를 생성해서 만약 해당 파일이 존재하지 않으면 CreateText()를 이용하여 새 텍스트 파일을 쓰는 StreamWriter를 만들고 문자열을 씁니다. 그 후에 AppendText()를 통해FileInfo의 인스턴스가 나타내는 파일에 텍스트를 추가하는 StreamWriter를 만들고 문자열을 쓴 후 파일에 있는 내용을 출력하는 예제입니다.

using System;

using System.IO;

class Test

{

 public static void Main()

 {

 FileInfo fi = new FileInfo(@"c:\temp\MyTest.txt");

 // This text is added only once to the file.

 if (!fi.Exists)

 {

 //Create a file to write to.

 using (StreamWriter sw = fi.CreateText())

 {

 sw.WriteLine("Hello");

 sw.WriteLine("And");

 sw.WriteLine("Welcome");

 }

 }

 // This text will always be added, making the file longer over time

 // if it is not deleted.

 using (StreamWriter sw = fi.AppendText())

 {

 sw.WriteLine("This");

 sw.WriteLine("is Extra");

 sw.WriteLine("Text");

 }

 //Open the file to read from.

 using (StreamReader sr = fi.OpenText())

 {

 string s = "";

 while ((s = sr.ReadLine()) != null)

 {

 Console.WriteLine(s);

 }

 }

 }

}

결과가 어떻게 나올 것 같습니까? 해당 파일이 존재하는 경우이면 “This is Extra Text”만 적혀있고, 해당파일이 존재하지 않은 경우엔 “Hello And Welcome”과 “This is Extra Text”둘 다 기록되어 출력되겠죠.

파일 복사나 이동 또한 FileInfo클래스의 메서드를 이용해서 간단하게 구현할 수 있습니다.

using System;

using System.IO;

class Test

{

public static void Main(string[] args) {

 if(args.Length < 2)

 {

 Console.WriteLine("USAGE: Test.exe [srcfile] [dstfile]");

 return;

 }

 FileInfo srcFile = new FileInfo(args[0]);

 string dstFileName = args[1];

 if(srcFile.Exists)

 {

 Console.WriteLine("Copy from {0} to {1}...", srcFile.Name, dstFileName);

 FileInfo copyFile = srcFile.CopyTo(dstFileName, true);

 copyFile.MoveTo(dstFileName);

 if(copyFile.Exists){

 Console.WriteLine("success to copy!");

 }

 }

}

}

닷넷에서는 스트림처리를 위해 System.IO.Stream클래스를 제공합니다.

<System.IO.Stream 클래스 보기>

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiostreamclasstopic.asp>

역시 자바처럼 Stream클래스는 가장 기본적인 기능들만 제시해 주고 있습니다.

간단히 말해 스트림은 다음 중 하나 이상의 작업을 지원하는 개체입니다.

바이트 읽기

바이트 쓰기

특정 위치 검색

즉, 스트림에서 바이트를 읽고 이를 스트림에 작성하고 스트림의 특정 위치로 이동할 수 있습니다. 이런 역할을 수행하기 위해 Stream클래스에서는 다음과 같은 멤버를 제공하고 있습니다.

<Stream클래스멤버 보기>

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemio.asp>

멤버들을 보시면 알겠지만 abstract입니다. Stream클래스를 상속한 클래스들이 나름대로 정의해 사용해야하죠. 그밖에 public 메서드를 보시면 virtual로 선언되어 있습니다.

저번 스터디 내용이었던 다형성(polymorphism)에서 배웠습니다.

virtual로 선언해 둠으로서 Stream을 상속하는 클래스에서 재정의가능하게 해 두었습니다.

알아봤다시피 Stream클래스 자체는 스트림이 작동하는 방식을 정의하는 추상적인 클래스이지 특정 유형의 스트림을 구현한 클래스는 아닙니다.

즉, 우리가 실제 코드를 작성할 경우에는 상황에 맞는, 스트림에서 파생된 클래스를 이용하여 처리해야겠죠. 일반적인 스트림은 다음과 같습니다.

Stream클래스로부터 파생된 클래스는 다음과 같습니다.

[System.Object](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemobjectclasstopic.asp)
   [System.MarshalByRefObject](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemmarshalbyrefobjectclasstopic.asp%22%20%5Ct%20%22_blank)
      **System.IO.Stream**
         [System.Data.OracleClient.OracleBFile](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemdataoracleclientoraclebfileclasstopic.asp%22%20%5Ct%20%22_blank)
         [System.Data.OracleClient.OracleLob](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemdataoracleclientoraclelobclasstopic.asp%22%20%5Ct%20%22_blank)
         [System.IO.BufferedStream](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiobufferedstreamclasstopic.asp%22%20%5Ct%20%22_blank)
         [System.IO.FileStream](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiofilestreamclasstopic.asp%22%20%5Ct%20%22_blank)
         [System.IO.MemoryStream](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiomemorystreamclasstopic.asp%22%20%5Ct%20%22_blank)
         [System.Net.Sockets.NetworkStream](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemnetsocketsnetworkstreamclasstopic.asp%22%20%5Ct%20%22_blank)
         [System.Security.Cryptography.CryptoStream](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemsecuritycryptographycryptostreamclasstopic.asp%22%20%5Ct%20%22_blank)

System.Data.OracleClient.OracleBFile클래스와

System.Data.OracleClient.OracleLob클래스는

오라클에서 제공하는 BFile이나 Lob데이터형식을 지원하기 위한 클래스이므로

여기서는 대상외로 하겠습니다.

System.Net.Sockets.NetworkStream는 네트워크를 통한 입출력에 관한 클래스입니다.

이 클래스에 대한 내용은 네트워크에 대한 내용에서 다뤄질 것입니다.

System.Security.Cryptography.CryptoStream는 입출력시 파일 암호화를 지원하는 클

래스입니다. 역시 암호화에 대한 내용에서 다뤄질 것 입니다.

처음에 Stream클래스에 대한 기능설명에서도 말씀 드렸다시피 스트림클래스는

바이트로 읽기 쓰기를 합니다. 그러므로 한글이나 일본어같은 유니코드문자를

제대로 읽고 쓰기위해서는 Stream클래스로 파생된 클래스를 사용하는 것이 아닌

다음의 클래스들중 하나를 이용하여 코드를 구현해야 합니다.

**System.IO.**[StreamReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostreamreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[StreamWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostreamwriterclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[StringReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostringreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[StringWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostringwriterclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[TextReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiotextreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[TextWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiotextwriterclasstopic.asp%22%20%5Ct%20%22_blank)

StreamReader, StringReader클래스는 TextReader클래스로부터

StreamWriter, StringWriter클래스는 TextWriter클래스로부터 파생된 클래스입니다.

또한 C#에서는 바이너리데이터를 읽고 쓰기 위해 다음의 클래스를 이용합니다.

**System.IO.**[BinaryReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiobinaryreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[BinaryWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiobinarywriterclasstopic.asp%22%20%5Ct%20%22_blank)

**파일에 대해 스트림을 제공하는 FileStream**

먼저 FileStream클래스부터 차근차근 보도록 하겠습니다.

<FileStream 멤버 보기>

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiofilestreammemberstopic.asp>

파일스트림의 생성자를 보시면 파일의 오픈모드(FileMode열거형)도 지정할 수 있고, 억세스모드(FileAccess열거형)나 다른 프로세스와의 파일 공유여부(FileShare열거형)까지도 함께 지정할 수 있습니다. 호오~~

FileStream s2 = new FileStream(name, FileMode.Open, FileAccess.Read, FileShare.None);

다음과 같은 코드의 경우 name에 해당하는 파일은 현프로세스가 살아있는 동안에는 다른 프로세스에서 접근이 불가능합니다.

C에서의 fopen()을 떠올리게 하는군요. JAVA보다 더 쉽게 파일스트림에 대한 기능설정이 가능하도록 한 것 같습니다. 상당히 다양한 생성자가 존재합니다.

다음 예제는 Test.data라는 파일이 존재하는지 확인하고 존재하지 않는다면FileMode.CreateNew를 통해 Test.data라는 파일명으로 파일스트림을 생성합니다.

FileStream fs = new FileStream(FILE\_NAME, FileMode.CreateNew);

그리고 아직 설명은 안했지만 BinaryWriter로 파일스트림을 감싸 바이너리값으로 1부터 10까지 데이터파일에 int형으로 저장합니다. 그 후에 읽기전용으로 파일스트림을 열어

fs = new FileStream(FILE\_NAME, FileMode.Open, FileAccess.Read);

데이터파일에 있는 바이너리값들을 int형으로 읽어들여 화면에 출력합니다.

여하튼 데이터파일에는 바이너리형태로 숫자들이 기록되어 있기 때문에 파일을 열어도 숫자가 보이지는 않습니다.

using System;

using System.IO;

class MyStream

{

 private const string FILE\_NAME = "Test.data";

 public static void Main(String[] args)

 {

 // Create the new, empty data file.

 if (File.Exists(FILE\_NAME))

 {

 Console.WriteLine("{0} already exists!", FILE\_NAME);

 return;

 }

 FileStream fs = new FileStream(FILE\_NAME, FileMode.CreateNew);

 // Create the writer for data.

 BinaryWriter w = new BinaryWriter(fs);

 // Write data to Test.data.

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

 {

 w.Write((int)i);

 }

 w.Close();

 fs.Close();

 // Create the reader for data.

 fs = new FileStream(FILE\_NAME, FileMode.Open, FileAccess.Read);

 BinaryReader r = new BinaryReader(fs);

 // Read data from Test.data.

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

 {

 Console.WriteLine(r.ReadInt32());

 }

 r.Close();

 fs.Close();

 }

}

Stream클래스에서 파생된 FileStream, BufferedStream, MemoryStream클래스는 파일포인터를 이용한 파일에의 임의 접근이 가능합니다(Seek). 하지만 뒤에 알아볼 문자 데이터를 다루는 스트림 클래스들은 이런 임의 접근을 지원하지 않습니다.

다음 코드 예제에서는 데이터를 바이트 단위로 파일에 쓴 다음 데이터가 올바르게 쓰여졌는지 확인하는 방법을 보여 줍니다. System.Random클래스의 NextBytes메서드를 사용해 지정된 바이트배열을 난수로 채웁니다. 그리고 FileStream을 이용해 파일을 생성하고, 배열의 내용을 바이트형으로 파일에 write합니다. Seek메서드를 이용해 파일 끝을 가리키고 있는 파일포인터를 파일 처음으로 이동시킵니다. 그리고 FileStream의 ReadByte메서드를 이용해 파일에 있는 내용을 바이트단위로 읽어 들여 실제 배열에 있는 내용과 비교해 정합성을 판정하는 코드입니다.

using System;

using System.IO;

class FStream

{

 static void Main()

 {

 const string fileName = "Test#@@#.dat";

 // Create random data to write to the file.

 byte[] dataArray = new byte[100000];

 new Random().NextBytes(dataArray);

 using (FileStream

 fileStream = new FileStream(fileName, FileMode.Create))

 {

 // Write the data to the file, byte by byte.

 for (int i = 0; i < dataArray.Length; i++)

 {

 fileStream.WriteByte(dataArray[i]);

 }

 // Set the stream position to the beginning of the file.

 fileStream.Seek(0, SeekOrigin.Begin);

 // Read and verify the data.

 for (int i = 0; i < fileStream.Length; i++)

 {

 if (dataArray[i] != fileStream.ReadByte())

 {

 Console.WriteLine("Error writing data.");

 return;

 }

 }

 Console.WriteLine("The data was written to {0} " +

 "and verified.", fileStream.Name);

 }

 }

}

**버퍼를 이용하여 오버헤드를 줄여라~ BufferedStream**

다음으로 BufferedStream클래스를 살펴 보겠습니다.

이름만 봐도 아시다시피 입출력시 버퍼를 두어 입출력횟수를 줄여 성능을 향상시킵니다.

자바에도 똑같은 기능을 하는 BufferedReader/Writer클래스가 있습니다만 다시 한번 설명하자면 FileStream같이 바이트단위로 데이터를 읽고 쓰는 경우에는 바이트당 read/write가 발생하기 때문에 오버헤드가 발생합니다. 물론 쓰고 읽는 데이터가 작은 크기라면 별 상관이 없겠지만서도.. 아무튼 이러한 바이트스트림에 BufferedStream을 이용해 버퍼크기만큼 데이터를 저장했다가 한번에 읽고 쓰기를 한다면 물리적인 디스크와 접촉하는 횟수가 줄기 때문에 성능향상을 기대할 수가 있습니다.

<BufferedStream 멤버보기>

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiobufferedstreammemberstopic.asp>

BufferedStream의 생성자를 보면 자바의 BufferedReader/Writer와 같이 Stream형태의 개체를 매개변수로 받아 개체를 생성합니다. 또한 생성시 버퍼사이즈를 설정할 수도 있습니다. 기본 버퍼 사이즈는 (1024\*4)바이트입니다. 버퍼 크기가 128KB일 경우 버퍼링된 작업은 버퍼링되지 않은 작업에 비해 10-20% 정도 속도가 빠르다고 합니다(MSDN).

public BufferedStream(Stream stream);

public BufferedStream(Stream stream, int buffersize);

자바에서와 같이BufferedStream은 자체적으로 스트림을 생성하지 못하기 때문에 우선 사용하기 위해서는 스트림을 생성해서 건네주어야겠죠.

FileStream fs = new FileStream(“test.txt”, FileMode.Open, FileAccess.Write);

BufferedStream bs = new BufferedStream(fs);

bs.Write(……);

다음 코드는 아스키 값을 파일에 쓰고 읽는 간단한 코드입니다. 버퍼를 이용해 파일에 쓰는데 한번, 읽는데 한번의 파일 접근(Access)이 이루어집니다.

using System;

using System.IO;

class BufferedStreamTest

{

 public static void Main(String[] args)

 {

 byte[] data = new byte[127];

 FileStream fs = new FileStream("test.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite);

 BufferedStream bs = new BufferedStream(fs);

 for (int i = 0; i < 127; i++) {

 data[i] = (byte)(i + 1);

 }

 bs.Write(data, 0, data.Length);

 bs.Close();

 fs = new FileStream("test.txt", FileMode.Open, FileAccess.Read);

 bs = new BufferedStream(fs);

 data = new byte[127];

 bs.Read(data, 0, 127);

 Console.WriteLine(System.Text.Encoding.Default.GetString(data));

 bs.Close();

 }

}

위 코드에서는BufferedStream을 사용하여 파일에 쓰는데 한번, 읽는데 한번의 파일 입출력이 이루어 지지만, 그냥 FileStream을 이용했다면 127회 파일에 대한 입출력이 이루어졌을 겁니다.

for(int i=1; i<=127; i++){

 file.WriteByte(Convert.ToByte(i));

}

**백업 저장소가 메모리인 MemoryStream**

다음으로 알아볼 스트림은 MemoryStream입니다.

MSDN에 있는MemoryStream에 대한 설명을 인용해 보기로 하겠습니다.

MemoryStream 클래스에서는 디스크나 네트워크 연결 대신 메모리를 백업 저장소로 사용하는 스트림을 만듭니다. MemoryStream 에서는 MemoryStream 개체 작성 시 초기화되거나 비어 있게 만들어진 부호 없는 바이트 배열로 저장된 데이터를 캡슐화합니다. 캡슐화된 데이터는 메모리에서 직접 액세스할 수 있습니다. 메모리 스트림을 사용하면 응용 프로그램의 임시 버퍼 및 파일의 필요성을 줄일 수 있습니다.

<MemoryStream 멤버 보기>

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiobufferedstreamclasstopic.asp>

생성자를 보면 초기 메모리바이트도 함께 설정할 수 있는데 int형을 이용하는 생성자는 나중에 크기조정 가능한 개체를 생성하지만, byte배열을 이용하는 생성자는 크기 조정이 불가능한 개체로 초기화 합니다.

중요 멤버를 한번 보자면...

멤버 설명

Capacity 이 스트림에 할당된 바이트 수를 가져오거나 설정합니다.

GetBuffer 스트림에 할당된 내용을 바이트 배열로 반환합니다.

ToArray 전체 문자열의 내용을 바이트 배열로 반환합니다.

WriteTo MemoryStream의 모든 내용을 다른 문자열 파생 타입에 출력합니다.

다음 예제는 MemoryStream의 사용법에 대한 MSDN에 나와 있는 예제입니다.

우선 UnicodeEncoding개체를 생성합니다. 첫번째 문자열로는"Invalid file path characters are: "를 작성하고 두번째 문자열에는 System.IO.Path클래스의InvalidPathChars메서드를 이용하여 부적절한 패스문자들을 입력받습니다. 두 문자열을 MemoryStream개체를 이용해 메모리에 쓰고, 메모리로부터 읽어 내어 출력하는 예제입니다.

using System;

using System.IO;

using System.Text;

class MemStream

{

 static void Main()

 {

 int count;

 byte[] byteArray;

 char[] charArray;

 UnicodeEncoding uniEncoding = new UnicodeEncoding();

 // Create the data to write to the stream.

 byte[] firstString = uniEncoding.GetBytes(

 "Invalid file path characters are: ");

 byte[] secondString = uniEncoding.GetBytes(

 Path.InvalidPathChars);

 using (MemoryStream memStream = new MemoryStream(100))

 {

 // Write the first string to the stream.

 memStream.Write(firstString, 0, firstString.Length);

 // Write the second string to the stream, byte by byte.

 count = 0;

 while (count < secondString.Length)

 {

 memStream.WriteByte(secondString[count++]);

 }

 // Write the stream properties to the console.

 Console.WriteLine(

 "Capacity = {0}, Length = {1}, Position = {2}\n",

 memStream.Capacity.ToString(),

 memStream.Length.ToString(),

 memStream.Position.ToString());

 // Set the position to the beginning of the stream.

 memStream.Seek(0, SeekOrigin.Begin);

 // Read the first 20 bytes from the stream.

 byteArray = new byte[memStream.Length];

 count = memStream.Read(byteArray, 0, 20);

 // Read the remaining bytes, byte by byte.

 while (count < memStream.Length)

 {

 byteArray[count++] =

 Convert.ToByte(memStream.ReadByte());

 }

 // Decode the byte array into a char array

 // and write it to the console.

 charArray = new char[uniEncoding.GetCharCount(

 byteArray, 0, count)];

 uniEncoding.GetDecoder().GetChars(

 byteArray, 0, count, charArray, 0);

 Console.WriteLine(charArray);

 }

 }

}

이상으로 System.IO.Stream클래스로부터 파생된 스트림에 대해 알아 보았습니다.

다음에는 문자 스트림에 대해 알아보겠습니다.

이번에는 문자관련 스트림에 대해 알아보겠습니다.

**System.IO.**[StreamReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostreamreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[StreamWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostreamwriterclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[StringReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostringreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[StringWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiostringwriterclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[TextReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiotextreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[TextWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiotextwriterclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[BinaryReader](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiobinaryreaderclasstopic.asp%22%20%5Ct%20%22_blank)

**System.IO.**[BinaryWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiobinarywriterclasstopic.asp%22%20%5Ct%20%22_blank)

우선 StreamReader/Writer, StringReader/Writer클래스의 기본클래스인 TextReader/Writer클래스에 대해 알아 보겠습니다.

**문자열을 읽고 쓰는 기본이 되는TextReader/Writer 클래스**

이전 설명한 스트림클래스들은 스트림을 바이트단위로 받아들이고 그 데이터의 내용이 어떤 지는 전혀 의식하지 않고 데이터를 읽고, 썼습니다. 그렇지만 한국어나 일본어같은 문자들은 한문자가1바이트가 아닌 1바이트이상의 바이트수로 이루어져 있기 때문에 이전의 스트림 같은 경우, 스트림의 사용에 따라 문자가 잘릴 경우가 발생할 수 있습니다.

이런 경우를 위해 C#에서는 기반스트림으로 TextReader/Writer클래스를 제공합니다. 기본스트림이기 때문에 TextReader나 TextWriter객체를 생성하기보다는 하위 클래스로 쓰이거나 하위클래스의 타입으로 쓰이는 경우가 많습니다. 역시 여기서도 다형성의 개념이..^^;

TextReader의 멤버 보기

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiobinaryreaderclassbasestreamtopic.asp>

TextWriter의 멤버 보기

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiobinaryreaderclassbasestreamtopic.asp>

MSDN에 나와 있는 예제를 통해 한번 알아보도록 하겠습니다. 아래 예제는 다형성의 개념을 통해 읽고 쓰는 각각 하나의 메서드(WriteText, ReadText)로 스트링객체와 스트림객체를 함께 받아들이고 있습니다.

다음 예제는 TextReader 및 TextWriter 형식의 다형 동작을 보여 줍니다. stringWriter 와 streamWriter 는 모두 TextWriter 형식이므로 WriteText 메서드는 두 개체를 모두 사용하여 호출되며 각 특정 형식과 관련된 Write 메서드가 실행됩니다. 마찬가지로 ReadText 메서드는 stringReader 와 streamReader를 모두 사용하여 호출되며 올바른 ReadToEnd 메서드가 실행됩니다. stringWriter 및 stringReader의 경우 백업 저장소는 문자열인 반면 파일은 streamWriter 및 streamReader에 대한 백업 저장소입니다.

-------------------------------------------------------------------

using System;

using System.IO;

class TextRW

{

 static void Main()

 {

 TextWriter stringWriter = new StringWriter();

 using (TextWriter streamWriter =

 new StreamWriter("InvalidPathChars.txt"))

 {

 WriteText(stringWriter);

 WriteText(streamWriter);

 }

 TextReader stringReader =

 new StringReader(stringWriter.ToString());

 using (TextReader streamReader =

 new StreamReader("InvalidPathChars.txt"))

 {

 ReadText(stringReader);

 ReadText(streamReader);

 }

 }

 static void WriteText(TextWriter textWriter)

 {

 textWriter.Write("Invalid file path characters are: ");

 textWriter.Write(Path.InvalidPathChars);

 textWriter.Write('.');

 }

 static void ReadText(TextReader textReader)

 {

 Console.WriteLine("From {0} - {1}",

 textReader.GetType().Name, textReader.ReadToEnd());

 }

}

-------------------------------------------------------------------

TextWriter형태로 StringWriter객체와 StreamWriter객체를 생성(다형성)하고,

스태틱메서드인 WriteText메서드를 통해 사용할 수 없는 경로문자열을 스트링과 스트림에 각각 기록합니다.

그 후에 TextReader타입으로 StringReader객체와 StreamReader객체를 생성한 후,

역시 스태틱메서드인 ReadText메서드를 이용하여 StringWriter객체와

StreamWriter객체에서 생성된 파일로부터 데이터를 읽어들여 내용을 화면에 출력합니다.

(Public필드인 InvalidPathChars는 해당 플랫폼에서 경로 문자열 인수에 사용할 수 없는

특정 문자 배열을 건네줍니다.)



**문자열에 읽고 쓰는 StringReader/Writer 클래스**

StringReader/Writer 클래스는 이름을 통해 알 수 있듯이 스트림을 스트링에 쓰거나, 스트링을 읽어내는 기능을 하는 클래스입니다.

StringWriter에 대한 MSDN의 설명을 빌리자면 “정보를 문자열로 쓰기 위한 [TextWriter](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemiotextwriterclasstopic.asp)를

구현합니다. 정보는 내부 [StringBuilder](http://msdn.microsoft.com/library/KOR/cpref/html/frlrfsystemtextstringbuilderclasstopic.asp)에 저장됩니다.”라고 설명하고 있습니다.

즉, TextReader/Writer클래스를 각각 상속받았고, 내부적으로 StringBuffer사용하니만큼

변형가능한 문자열을 메모리에 쓰거나 읽어낼 수 있죠.

그럼 StringReader/Writer의 멤버를 보기로 하죠. 꼭 한번씩은 어떤 메서드가 있는지

확인해 보시기 바랍니다.

StringReader의 멤버보기

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiostringwritermemberstopic.asp>

StringWriter의 멤버보기

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiostringwritermemberstopic.asp>

StringWriter클래스에는 3가지 public 속성이 있습니다.

**Encoding** 출력이 쓰여질 Encoding을 가져옵니다.

**FormatProvider** 형식을 제어하는 개체를 가져옵니다.

**NewLine** 현재 TextWriter에서 사용한 줄 종결자 문자열을 가져오거나 설정.

아래 예제에서는 스트림으로부터 스트링객체 w에 문자열을 쓰고, 스트링객체w가 저장하고 있는 문자열을 출력하는 예제입니다.

-------------------------------------------------------------------

using System;

using System.IO;

using System.Text;

namespace StringReadWrite

{

 class StringReadWriteApp

 {

 [STAThread]

 static void Main(string[] args)

 {

 // Using StringWriter

 StringWriter w = new StringWriter();

 w.WriteLine("StringWriterテストです。");

 string s = "文字列に書きます。";

 w.Write(s);

 w.Write(w.NewLine);

 w.Write(

 String.Format(4 +" + " +20 +" = " ));

 w.Write(new StringBuilder("80"));

 w.WriteLine();

 Console.WriteLine(w);

 Console.WriteLine("The following string is {0} encoded.\n{1}",

 w.Encoding.EncodingName, w.ToString());

 // Using the internal StringBuilder

 StringBuilder sb = w.GetStringBuilder();

 int i = sb.Length;

 sb.Append("abcde");

 sb.Insert(i, "元の文字列の最後に追加します。\n");

 sb.AppendFormat(

 "\nLet's together {0} study!!!", "C#");

 Console.WriteLine(w);

 // Using StringReader

 Console.WriteLine();

 StringReader r = new StringReader(w.ToString());

 string t = r.ReadLine();

 Console.WriteLine(t);

 Console.Write((char)r.Read());

 char[] ca = new char[37];

 r.Read(ca, 0, 19);

 Console.Write(ca);

 r.ReadBlock(ca, 0, 37);

 Console.Write(ca);

 Console.WriteLine(r.ReadToEnd());

 r.Close();

 w.Close();

 }

 }

}

-------------------------------------------------------------------

StringWriter객체 w를 생성한후 Write()메서드를 이용해 문자열을 씁니다. 퍼블릭속성을

통해 새로 라인을 추가할 수도 있습니다(w.NewLine). 물론 문자열 뒤에 “\n”을 붙여도

같은 효과를 냅니다만.

Console.WriteLine(w);

을 통해 객체w에 있는 문자열을 표준출력에 출력합니다.

StringWriter.GetStringBuilder()메서드를 이용해서 StringWriter객체에 있는 문자열을 StringBuilder객체에 단번에 써 넣을 수 있습니다.

StringBuilder sb = w.GetStringBuilder();

ToString()메서드로 객체에 저장된 문자열을 반환하여 StringReader객체를 생성합니다.

StringReader r = new StringReader(w.ToString());

StringReader의 Read()메서드나 ReadBlock(), ReadToEnd()메서드를 이용해 객체에

있는 문자열을 출력할 수 있습니다.

결과는 다음과 같습니다.



**바이너리 데이터를 읽고 쓰는 BinaryReader/Writer클래스**

StreamReader/Writer클래스와 같은 경우에는 데이터를 스트림에 쓸 때 문자열로 변환하여

기록하지만BinaryReader/Writer클래스를 이용하면 이러한 문자열변환 과정을 거치지 않고

데이터를 스트림에 기록할 수 있습니다.

BinaryReader의 멤버 보기

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiostringwritermemberstopic.asp>

BinaryWriter의 멤버 보기

<http://msdn.microsoft.com/library/kor/default.asp?url=/library/KOR/cpref/html/frlrfsystemiostringwritermemberstopic.asp>

BinaryReader클래스의 메서드를 보면 ReadByte, ReadChar, ReadInt, ReadString등의 메서드가 정의되어 있어 원하는 형식에 맞게끔 읽어 출력할 수 있습니다. 당연히 BinaryWriter클래스는 Write메서드(다양한 형태로 오버로드되어 있는)를 통해 모든 형태의 데이터를 바이너리 형태로 기록합니다.

다음 예제는 FileStream객체를 통해 Foo.txt와 Bar.dat 파일을 생성합니다.그 후 Foo.txt에는 StreamWriter객체를 생성해 텍스트데이터를 입력하고 파일을 닫습니다.

Bar.dat에는 BinaryWriter객체를 생성해 바이너리데이터를 입력하고 파일을 닫습니다.

-------------------------------------------------------------------

using System;

using System.IO;

namespace BinReadWrite

{

 class BinReadWriteApp

 {

 [STAThread]

 static void Main(string[] args)

 {

 Stream s =

 new FileStream("Foo.txt", FileMode.Create);

 StreamWriter w = new StreamWriter(s);

 w.Write("Hello World ");

 w.Write(123);

 w.Write(' ');

 w.Write(45.67);

 w.Close();

 s.Close();

 Stream t =

 new FileStream("Bar.dat", FileMode.Create);

 BinaryWriter b = new BinaryWriter(t);

 b.Write("Hello World ");

 b.Write(123);

 b.Write(' ');

 b.Write(45.67);

 b.BaseStream.Position = 0;

 BinaryReader r = new BinaryReader(t);

 int i = 0;

 while (true)

 {

 i = b.BaseStream.ReadByte();

 if (-1 == i)

 {

 Console.WriteLine();

 break;

 }

 Console.Write("{0,-4}", i);

 }

 r.Close();

 b.Close();

 t.Close();

 }

 }

}

-------------------------------------------------------------------

결과는 다음과 같습니다.

Foo.txt의 내용은 텍스트문자열로 나와 있습니다.

Hello World 123 45.67

Bar.dat의 내용은 바이너리로 기록했기 때문에 당연히 바이너리문자열이 들어 있습니다.



(여기서는 토탈코멘더의 Hex형태로 파일보기를 이용하여 파일을 열어 보았습니다.)

제일 처음 값인 0x0C(12)는 마지막 공백을 포함한 문자열의 길이를 나타내고,

값 “48 65 6C 6C 6F 20 57 6F 72 6C 64 20”은 문자열 “Hello World “를 나타냅니다. 아무튼 이렇게 바이너리 값으로 기록이 되고 이 파일을 읽으려면 BinaryReader객체를 생성하여

바이너리 데이터를 읽게 됩니다.

여기서는 바이너리 데이터수만큼 읽기위해

b.BaseStream.Position = 0;

을 통해 바이너리스트림의 위치를 처음으로 되돌립니다.

참고로 데이터를 읽어서 쓰는 부분에서 좀 익숙치 않은 표현이 나올지도 모르겠습니다.

Console.Write("{0,-4}", i);

C언어를 공부를 해본 분들은 익숙한 표현입니다. 첫번째 인자인"{0,-4}"의 -4는 바로 출력 데이터의 정열방법을 나타냅니다. “-“는 왼쪽정렬, “+”는 오른쪽 정렬입니다. 뒤이어 나오는 4는 하나의 데이터를 표현할 행의 수를 나타냅니다.

