**Tập tin và thư mục**

Các lớp đề cập trong chương này thuộc về vùng tên System.IO. Các lớp này bao gồm lớp File mô phỏng cho một tập tin trên đĩa, và lớp Directory mô phỏng cho một thư mục trên đĩa.

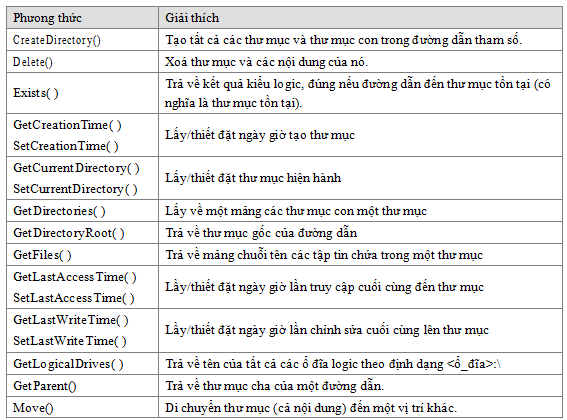
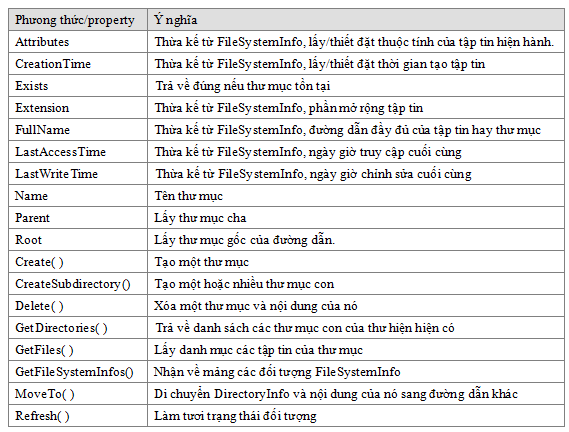
**Làm việc với thư mục**

Lớp **Directory** có nhiều phương thức dành cho việc tạo, di chuyển, duyệt thư mục. Các phương thức trong lớp Directory đều là phương thức tĩnh;vì vậy không cần phải tạo một thể hiện lớp Directory mà có thể truy xuất trực tiếp từ tên lớp.

Lớp **DirectoryInfo** là lớp tương tự như lớp Directory. Nó cung các tất cả các phương thức mà lớp Directory có đồng thời bổ sung nhiều phương thức hữu ích hơn cho việc duyệt cấu trúc cây thư mục. Lớp DirectoryInfo thừa kế từ lớp **FileSystemInfo**, và vì vậy cũng thừa kế lớp **MarshalByRefObj**. Lớp DirectoryInfo không có phương thức tĩnh, vì vậy cần tạo một thể hiện lớp trước khi sử dụng các phương thức.

Có một khác biệt quan trong giữa Directory và DirectoryInfo là các phương thức của lớp Directory sẽ được kiểm tra về bảo mật mỗi khi được gọi trong khi đối tượng DirectoryInfo chỉ kiểm tra một lần vào lúc khởi tạo, các phương thức vì vậy sẽ thực hiện nhanh hơn.

Dưới đây là bảng liệt kê các phương thức quan trọng của hai lớp

Các phương thức lớp DirectoryCác phương thức/property lớp DirectoryInfo

**Tạo đối tượng DirectoryInfo**

Để duyệt cấu trúc cây thư mục, ta cần tạo một thể hiện của lớp DirectoryInfo. Lớp DirectoryInfo không chỉ cung cấp phương thức lấy về tên các tập tin và thư mục con chứa trong một thư mục mà còn cho phép lấy về các đối tượng **FileInfo** và **DirectoryInfo**, cho phép ta thực hiện việc quản lý các cấu trúc cây thư mục, hay thực hiện các thao tác đệ qui.

Khởi tạo một đối tượng DirectoryInfo bằng tên của thư mục muốn tham chiếu.

DirectoryInfo dir = new DirectoryInfo(@"C:\winNT");

Ta có thể thực hiện các phương thức đã liệt kê ở bảng trên. Dưới đây là đoạn mã nguồn ví dụ.

**Duyệt các thư mục con**

using System;

using System.IO;

namespace Programming\_CSharp

{

class Tester

{

public static void Main()

{

Tester t = new Tester( );

// một một thư mục

string theDirectory = @"c:\WinNT";

// duyệt thư mục và hiển thị ngày truy cập gần nhất

// và tất cả các thư mục con

DirectoryInfo dir = new DirectoryInfo(theDirectory);

t.ExploreDirectory(dir);

// hoàn tất. in ra số lượng thống kê

Console.WriteLine( "\n\n{0} directories found.\n", dirCounter);

}

// với mỗi thư mục tìm thấy, nó gọi chính nó

private void ExploreDirectory(DirectoryInfo dir)

{

indentLevel++; // cấp độ thư mục

// định dạng cho việc trình bày

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

Console.Write(" "); // hai khoảng trắng cho mỗi cấp

// in thư mục và ngày truy cập gần nhất

Console.WriteLine("[{0}] {1} [{2}]\n",

indentLevel, dir.Name, dir.LastAccessTime);

// lấy tất cả thư mục con của thư mục hiện tại

// đệ quy từng thư mục

DirectoryInfo[] directories = dir.GetDirectories( );

foreach (DirectoryInfo newDir in directories)

{

dirCounter++; //tăng biến đếm

ExploreDirectory(newDir);

}

indentLevel--; // giảm cấp độ thư mục

}

// các biến thành viên tĩnh cho việc thống kê và trình bày

static int dirCounter = 1;

static int indentLevel = -1; // so first push = 0

}

}

**Kết quả:**

[2] logiscan [5/1/2001 3:06:41 PM]

[2] miitwain [5/1/2001 3:06:41 PM]

[1] Web [5/1/2001 3:06:41 PM]

[2] printers [5/1/2001 3:06:41 PM]

[3] images [5/1/2001 3:06:41 PM]

[2] Wallpaper [5/1/2001 3:06:41 PM]

363 directories found.

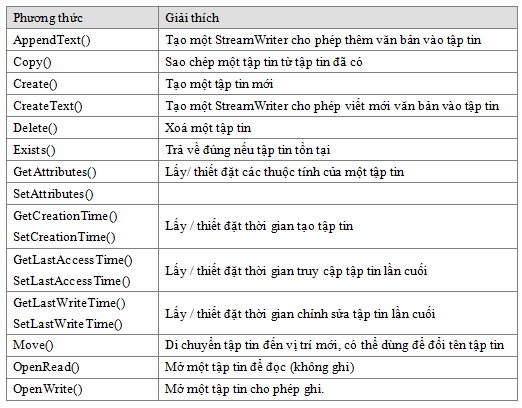
Chương trình tạo một đối tượng DirectoryInfo gắn với thư mục WinNT. Sau đó gọi hàm ExploreDirectory với tham số là đối tượng DirectoryInfo vừa tạo. Hàm sẽ hiển thị các thông tin về thư mục này và sau đó lấy tất cả các thư mục con.

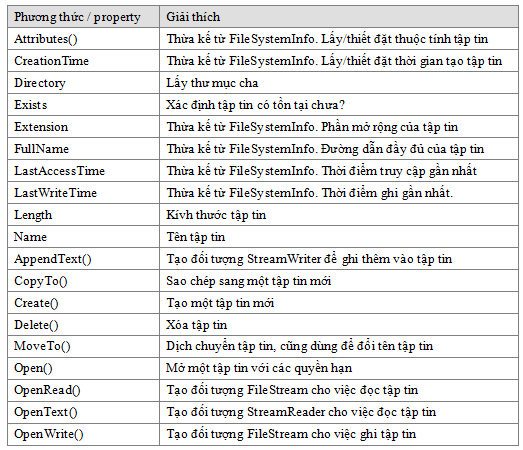
Để liệt kê danh sách các thư mục con, hàm gọi phương thức GetDirectories. Phương thức này trả về mảng các đối tượng DirectoryInfo. Bằng cách gọi đệ qui chính nó, hàm liệt kê xuống các thư mục con và thư mục con của thư mục con … Kết quả cuối cùng là cấu trúc cây thư mục được hiển thị.

**Làm việc với tập tin.**

Đối tượng DirectoryInfo cũng trả về danh sách các đối tượng FileInfo là các tập tin chứa trong thư mục. Các đối tượng này mô tả thông tin về tập tin. Thư viện .NET cũng cung cấp hai lớp File và FileInfo tương tự như với trường hợp thư mục. Lớp File chỉ có các phương thức tĩnh và lớp FileInfo thì không có phương thức tĩnh nào cả.

Hai bảng dưới đây liệt kê các phương thức cũa hai lớp này

Các phương thức lớp File

Các phương thức / property lớp FileInfo

[[link]](https://voer.edu.vn/c/luong-du-lieu/9f00d32d/af9046a5#eip-473) sửa lại từ [[link]](https://voer.edu.vn/c/luong-du-lieu/9f00d32d/af9046a5#eip-391), thêm đoạn mã lấy FileInfo của mỗi thư mục.

Đối tượng này dùng để hiển thị tên, kích thước và ngày truy cấp cuối cùng của tập tin

**Duyệt tập tin và thư mục con**

using System;

using System.IO;

namespace Programming\_CSharp

{

class Tester

{

public static void Main( )

{

Tester t = new Tester( );

string theDirectory = @"c:\WinNT";

DirectoryInfo dir = new DirectoryInfo(theDirectory);

t.ExploreDirectory(dir);

Console.WriteLine(

"\n\n{0} files in {1} directories found.\n",

fileCounter,dirCounter );

}

private void ExploreDirectory(DirectoryInfo dir)

{

indentLevel++;

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

Console.Write(" ");

Console.WriteLine("[{0}] {1} [{2}]\n",

indentLevel, dir.Name, dir.LastAccessTime);

// lấy tất cả các tập tin trong thư mục và

// in tên, ngày truy cập gần nhất, kích thước của chúng

FileInfo[] filesInDir = dir.GetFiles( );

foreach (FileInfo file in filesInDir)

{

// lùi vào một khoảng phía dưới thư mục

// phục vụ việc trình bày

for (int i = 0; i < indentLevel+1; i++)

Console.Write(" "); // hai khoảng trắng cho mỗi cấp

Console.WriteLine("{0} [{1}] Size: {2} bytes",

file.Name, file.LastWriteTime, file.Length);

fileCounter++;

}

DirectoryInfo[] directories = dir.GetDirectories( );

foreach (DirectoryInfo newDir in directories)

{

dirCounter++; ExploreDirectory(newDir);

}

indentLevel--;

}

// các biến tĩnh cho việc thống kê và trình bày

static int dirCounter = 1; static int indentLevel = -1; static int fileCounter = 0;

}

}

**Kết quả (Một phần):**

[0] WinNT [5/1/2001 3:34:01 PM]

ActiveSetupLog.txt [4/20/2001 10:42:22 AM] Size: 10620 bytes actsetup.log [4/20/2001 12:05:02 PM] Size: 8717 bytes

Blue Lace 16.bmp [12/6/1999 4:00:00 PM] Size: 1272 bytes

[2] Wallpaper [5/1/2001 3:14:32 PM]

Boiling Point.jpg [4/20/2001 8:30:24 AM] Size: 28871 bytes

Chateau.jpg [4/20/2001 8:30:24 AM] Size: 70605 bytes

Windows 2000.jpg [4/20/2001 8:30:24 AM] Size: 129831 bytes

8590 files in 363 directories found.

**Chỉnh sửa tập tin**

Đối tượng FileInfo có thể dùng để tạo, sao chép, đổi tên và xoá một tập tin. Ví dụ dưới đậy tạo một thư mục con mới, sao chép một tập tin, đổi tên vài tập tin, và sau đó xóa toàn bộ thư mục này.

**Tạo thư mục con và thao tác các tập tin**

using System;

using System.IO;

namespace Programming\_CSharp

{

class Tester

{

public static void Main( )

{

Tester t = new Tester( );

string theDirectory = @"c:\test\media";

DirectoryInfo dir = new DirectoryInfo(theDirectory);

t.ExploreDirectory(dir);

}

private void ExploreDirectory(DirectoryInfo dir)

{

// tạo mới một thư mục con

string newDirectory = "newTest"; DirectoryInfo newSubDir =

dir.CreateSubdirectory(newDirectory);

// lấy tất cả các tập tin trong thư mục và

// sao chép chúng sang thư mục mới

FileInfo[] filesInDir = dir.GetFiles( );

foreach (FileInfo file in filesInDir)

{

string fullName = newSubDir.FullName + "\\" + file.Name;

file.CopyTo(fullName); Console.WriteLine("{0} copied to newTest",

file.FullName);

}

// lấy các tập tin vừa sao chép

filesInDir = newSubDir.GetFiles( );

// hủy hoặc đổi tên một vài tập tin

int counter = 0;

foreach (FileInfo file in filesInDir)

{

string fullName = file.FullName;

if (counter++ %2 == 0)

{

file.MoveTo(fullName + ".bak"); Console.WriteLine("{0} renamed to {1}",

fullName,file.FullName);

}

else

{

file.Delete( );

Console.WriteLine("{0} deleted.", fullName);

}

}

newSubDir.Delete(true); // hủy thư mục con này

}

}

}

**Kết quả (Một phần) :**

c:\test\media\Bach's Brandenburg Concerto No. 3.RMI copied to newTest

c:\test\media\Beethoven's 5th Symphony.RMI copied to newTest

c:\test\media\Beethoven's Fur Elise.RMI copied to newTest

c:\test\media\canyon.mid copied to newTest

c:\test\media\newTest\Bach's Brandenburg Concerto No. 3.RMI renamed to

c:\test\media\newTest\Bach's Brandenburg Concerto No. 3.RMI.bak

c:\test\media\newTest\Beethoven's 5th Symphony.RMI deleted.

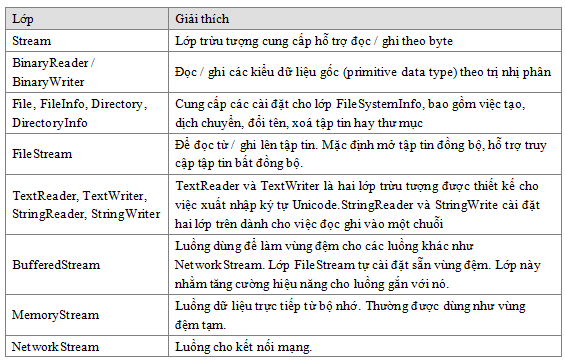
c:\test\media\newTest\Beethoven's Fur Elise.RMI renamed to

c:\test\media\newTest\Beethoven's Fur Elise.RMI.bak

c:\test\media\newTest\canyon.mid deleted.

**Đọc và ghi dữ liệu**

Đọc và ghi dữ liệu là nhiệm vụ chính của các luồng, Stream. Stream hỗ trợ cả hai cách đọc ghi đồng bộ hay bất đồng bộ. .NET Framework cung cấp sẵn nhiều lớp thừa kế từ lớp Stream, bao gồm FileStream, MemoryStream và NetworkStream. Ngoài ra còn có lớp BufferedStream cung cấp vùng đệm xuất nhập được dùng thêm với các luồng khác. Bảng dưới đây tóm tắt ý nghĩa sử dụng của các luồng

Ý nghĩa các luồng

**Tập tin nhị phân**

Phần này sẽ bắt đầu sử dụng lớp cơ sở Stream để đọc tập tin nhị phân. Lớp Stream có rất nhiều phương thức nhưng quan trọng nhất là năm phương thức Read(), Write(), BeginRead(), BeginWrite() và Flush().

Để thao tác tập tin nhị phân (hay đọc tập tin theo kiểu nhị phân), ta bắt đầu tạo một cặp đối tượng Stream, một để đọc, một để viết.

Stream inputStream = File.OpenRead(@"C:\test\source\test1.cs"); Stream outputStream = File.OpenWrite(@"C:\test\source\test1.bak");

Để mở một tập tin để đọc và viết, ta sử dụng hai hàm tĩnh **OpenRead()** và **OpenWrite()** của lớp File với tham số là đường dẫn tập tin.

Tiếp theo ta đọc dữ liệu từ inputStream cho đến khi không còn dữ liệu nữa và sẽ ghi dữ liệu đọc được vào outputStream. Hai hàm lớp Stream phục vụ việc đọc ghi dữ liệu là Read() và Write().

while( (bytesRead = inputStream.Read(buffer,0,SIZE\_BUFF)) > 0 )

{

outputStream.Write(buffer,0,bytesRead);

}

Hai hàm có cùng một số lương và kiểu tham số truyền vào. Đầu tiên là một mảng các byte (được gọi là vùng đệm buffer) dùng để chứa dữ liệu theo dang byte. Tham số thứ hai cho biết vị trí bắt đầu đọc hay ghi trên vùng đệm, tham số cuối cùng cho biết số byte cần đọc hay ghi. Đối với hàm Read() còn trả về số byte mà Stream đọc được, có thể bằng hay khác giá trị tham số thứ ba.

**Cài đặt việc đọc và ghi tập tin nhị phân**

using System;

using System.IO;

namespace Programming\_CSharp

{

class Tester

{

const int SizeBuff = 1024;

public static void Main( )

{

Tester t = new Tester( );

t.Run( );

}

private void Run( )

{

// đọc từ tập tin này

Stream inputStream = File.OpenRead(

@"C:\test\source\test1.cs");

// ghi vào tập tin này

Stream outputStream = File.OpenWrite(

@"C:\test\source\test1.bak");

// tạo vùng đệm chứa dữ liệu

byte[] buffer = new Byte[SizeBuff];

int bytesRead;

// sau khi đọc dữ liệu xuất chúng ra outputStream

while ( (bytesRead =

inputStream.Read(buffer,0,SizeBuff)) > 0 )

{

outputStream.Write(buffer,0,bytesRead);

}

// đóng tập tin trước khi thoát

inputStream.Close( );

outputStream.Close( );

}

}

}

Kết quả sau khi chay chương trình là một bản sao của tập tin đầu vào (test1.cs) được tạo trong cùng thư mục với tên test1.bak

**Luồng có vùng đệm**

Trong ví dụ trước ta thực hiện việc ghi lên tập tin theo từng khối buffer, như vậy hệ điều hành sẽ thực thi việc ghi tập tin ngay sau lệnh Write(). Điều này có thể làm giảm hiệu năng thực thi do phải chờ các thao tác cơ học của đĩa cứng vốn rất chậm.

Luồng có vùng đệm sẽ khắc phục nhược điểm này bằng cách sau: khi có lệnh Write() dữ liệu, luồng sẽ không gọi hệ điều hành ngay mà sẽ giữ trên vùng đệm (thực chất là bộ nhớ), chờ cho đến khi dữ liệu đủ lớn sẽ ghi một lượt lên tập tin. Lớp BufferedStream là cài đặt cho luồng có vùng đệm.

Để tạo một luồng có vùng đệm trước tiên ta vẫn tạo luồng Stream như trên

Stream inputStream = File.OpenRead(@"C:\test\source\folder3.cs"); Stream outputStream = File.OpenWrite(@"C:\test\source\folder3.bak");

Sau đó truyền các luồng này cho hàm dựng của BufferedStream BufferedStream bufferedInput = new BufferedStream(inputStream); BufferedStream bufferedOutput = new BufferedStream(outputStream);

Từ đây ta sử dụng **bufferedInput** và **bufferedOutput** thay cho inputStream và outputStream. Cách sử dụng là như nhau: cũng dùng phương thức **Read()** và **Write()**

while((bytesRead = bufferedInput.Read(buffer,0,SIZE\_BUFF))>0 )

{

bufferedOutput.Write(buffer,0,bytesRead);

}

Có một khác biệt duy nhất là phải nhớ gọi hàm Flush() để chắc chắn dữ liệu đã được "tống" từ vùng buffer lên tập tin.

bufferedOutput.Flush( );

Lệnh này nhằm yêu cầu hệ điều hành sao chép dữ liệu từ vùng nhớ buffer lên đĩa cứng.

**Cài đặt luồng có vùng đệm**

using System;

using System.IO;

namespace Programming\_CSharp

{

class Tester

{

const int SizeBuff = 1024;

public static void Main( )

{

Tester t = new Tester( );

t.Run( );

}

private void Run( )

{

// tạo một luồng nhị phân

Stream inputStream = File.OpenRead( @"C:\test\source\folder3.cs");

Stream outputStream = File.OpenWrite( @"C:\test\source\folder3.bak");

// tạo luồng vùng đệm kết buộc với luồng nhị phân

BufferedStream bufferedInput = new BufferedStream(inputStream);

BufferedStream bufferedOutput = new BufferedStream(outputStream);

byte[] buffer = new Byte[SizeBuff]; int bytesRead;

while ( (bytesRead = bufferedInput.Read(buffer,0,SizeBuff)) > 0 )

{

bufferedOutput.Write(buffer,0,bytesRead);

}

bufferedOutput.Flush( ); bufferedInput.Close( ); bufferedOutput.Close( );

}

}

}

Với tập tin có dung lượng lớn, chương trình này sẽ chạy nhanh hơn chương trình ví dụ trước.

**Làm việc với tập tin văn bản**

Đối với các tập tin chỉ chứa văn bản, ta sử dụng hai luồng StreamReader và StreamWriter cho việc đọc và ghi. Hai lớp này được thiết kế để thao tác với văn bản dễ dàng hơn. Ví dụ như chúng cung cấp hàm ReadLine() và WriteLine() để đọc và ghi một dòng văn bản.

Để tạo một thể hiện StreamReader ta gọi phương thức OpenText() của lớp FileInfo.

FileInfotheSourceFile = new FileInfo (@"C:\test\source\test1.cs");

StreamReader stream = theSourceFile.OpenText( );

Ta đọc từng dòng văn bản của tập tin cho đến hết

do

{

text = stream.ReadLine( );

} while (text != null);

Để tạo đối tượng StreamWriter ta truyền cho hàm khởi dựng đường dẫn tập tin

StreamWriter writer = new StreamWriter(@"C:\test\source\folder3.bak",false);

tham số thứ hai thuộc kiểu bool, nếu tập tin đã tồn tại, giá trị true sẽ ghi dữ liệu mới vào cuối tập tin, giá trị false sẽ xóa dữ liệu cũ, dữ liệu mới sẽ ghi đè dữ liệu cũ.

**Đọc và ghi tập tin văn bản**

using System;

using System.IO;

namespace Programming\_CSharp

{

class Tester

{

public static void Main( )

{

Tester t = new Tester( );

t.Run( );

}

private void Run( )

{

// mở một tập tin

FileInfo theSourceFile = new FileInfo(@"C:\test\source\test.cs");

// tạo luồng đọc văn bản cho tập tin

StreamReader reader = theSourceFile.OpenText( );

// tạo luồng ghi văn bản cho tập tin xuất

StreamWriter writer = new StreamWriter(@"C:\test\source\test.bak",false);

// tạo một biến chuỗi lưư giữ một dòng văn bản

string text;

// đọc toàn bộ tập tin theo từng dòng

// ghi ra màn hình console và tập tin xuất

do

{

text = reader.ReadLine( ); writer.WriteLine(text); Console.WriteLine(text);

} while (text != null);

// đóng tập tin

reader.Close( );

writer.Close( );

}

}

}

Khi thực thi chương trình nội dung tập tin nguồn được ghi lên tập tin mới đồng thời xuất ra màn hình console.

**Bất đồng bộ nhập xuất**

Các ví dụ được trình bày ở trên sử dụng kỹ thuật đồng bộ hóa trong nhập xuất dữ liệu (synchronous I/O), có nghĩa là chương trình sẽ tạm ngưng trong lúc hệ điều hành thực hiện việc đọc hay ghi dữ liệu. Điều này có thể làm chương trình tốn thời gian vô ích, đặc biệt khi làm việc với các ổ đĩa có tốc độ chậm hay dung lượng đường truyền mạng thấp.

Kỹ thuật bất đồng bộ nhập xuất (asynchronous I/O) được dùng để giải quyết vấn đề này. Ta có thể thực hiện các công việc khác trong khi chờ hệ thống hập xuất đọc/ghi dữ liệu. Kỹ thuật này được cài đặt trong phương thức BeginRead() và BeginWrite() của lớp Stream.

Mấu chốt của phương thức Begin\*() là khi được gọi một tiểu trình mới sẽ được tạo và làm công việc nhập xuất, tiểu trình cũ sẽ thực hiện công việc khác. Sau khi hoàn tất việc đọc/ghi, thông báo được gởi đến hàm callback thông qua một deleagte. Ta có thể thao tác với các dữ liệu vừa được đọc/ghi, thực hiện một công việc đọc/ghi khác và lại quay đi làm công việc khác.

Phương thức BeginRead() yêu cầu năm tham số, ba tham số tương tự hàm Read, hai tham số (tùy chọn) còn lại là: delegate AsyncCallback để gọi hàm callback và tham số còn lại là object dùng để phân biệt giữa các thao tác nhập xuất bất đồng bộ khác nhau.

Trong ví dụ dụ này ta sẽ tạo một mảng byte làm vùng đệm, và một đối tượng

Stream

public class AsynchIOTester

{

private Stream inputStream;

private byte[] buffer;

const int BufferSize = 256;

một biến thành viên kiểu delegate mà phương thức BeginRead() yêu cầu

private AsyncCallback myCallBack; // delegated method

Delegate AsyncCallback khai báo trong vùng tên System như sau

public delegate void AsyncCallback (IAsyncResult ar);

Tạo một hàm callback để đóng gói trong delegate

void OnCompletedRead(IAsyncResult asyncResult)

Dưới đây là cách hoạt động của ví dụ. Trong hàm Main() ta khởi tạo và cho thực thi lớp kiểm thử AsyncIOTester

public static void Main( )

{

AsynchIOTester theApp = new AsynchIOTester( );

theApp.Run( );

}

Hàm dựng khởi tạo các biến thành viên

AsynchIOTester( )

{

inputStream = File.OpenRead(@"C:\test\source\AskTim.txt");

buffer = new byte[BufferSize];

myCallBack = new AsyncCallback(this.OnCompletedRead);

}

Phương thức Run() sẽ gọi BeginRead()

inputStream.BeginRead(buffer, // chứa kết quả

0, // vị trí bắt đâu

buffer.Length, // kích thước vùng đệm

myCallBack, // callback

delegate null); // đối tượng trạng thái

Sau đó thực hiện công việc khác, trường hợp này là vòng lặp for thực hiện 500.000 lần.

for (long i = 0; i < 500000; i++)

{

if (i%1000 == 0)

{

Console.WriteLine("i: {0}", i);

}

}

Sau khi việc đọc hoàn tất hàm callback được gọi EndRead()

void OnCompletedRead(IAsyncResult asyncResult)

{

Điều đầu tiên là phải biết số lượng byte thật sự đọc được bằng cách gọi hàm

int bytesRead = inputStream.EndRead(asyncResult);

Sau đó thao tác trên dữ liệu đọc được (in ra console), và lại gọi tiếp một BeginRead() để thực hiện nhập xuất bất đồng bộ một lần nữa,

BeginRead() để thực hiện nhập xuất bất đồng bộ một lần nữa,

if (bytesRead > 0)

{

string s = Encoding.ASCII.GetString (buffer, 0, bytesRead);

Console.WriteLine(s);

inputStream.BeginRead(buffer, 0, buffer.Length,

myCallBack, null);

}

Hiệu quả của chương trình là ta có thể thực hiện các công việc không cần kết quả của việc đọc dữ liệu khác. Ví dụ hoàn chỉnh liệt kê dưới đây

**cài đặt nhập xuất bất đồng bộ**

using System;

using System.IO;

using System.Threading;

using System.Text;

namespace Programming\_CSharp

{

public class AsynchIOTester

{

private Stream inputStream;

// delegated

private AsyncCallback myCallBack;

// vùng nhớ buffer lưu giữ liệu đọc được

private byte[] buffer;

// kích thước buffer

const int BufferSize = 256;

AsynchIOTester( )

{

// mở một luồng nhập

inputStream = File.OpenRead(

@"C:\test\source\AskTim.txt");

// cấp phát vùng buffer

buffer = new byte[BufferSize];

// gán một hàm callback

myCallBack = new AsyncCallback(this.OnCompletedRead);

}

public static void Main( )

{

AsynchIOTester theApp = new AsynchIOTester();

theApp.Run( );

}

void Run()

{

inputStream.BeginRead(

buffer, // chứa kết quả

0, // vị trí bắt đầu trên buffer

buffer.Length, // kích thước buffer

myCallBack, // callback delegate

null); // đối tượng trạng thái cục bộ

// làm chuyện gì đó trong lúc đọc dữ liệu

for (long i = 0; i < 500000; i++)

{

if (i%1000 == 0)

{

Console.WriteLine("i: {0}", i);

}

}

}

// hàm callback

void OnCompletedRead(IAsyncResult asyncResult)

{

int bytesRead = inputStream.EndRead(asyncResult);

// nếu đọc có dữ liệu

if (bytesRead > 0)

{

// chuyển nó thành chuỗi

String s = Encoding.ASCII.GetString(buffer, 0, bytesRead); Console.WriteLine(s);

inputStream.BeginRead( buffer, 0, buffer.Length, myCallBack, null);

}

}

}

}

**Kết quả (Một phần) :**

i: 47000

i: 48000

i: 49000

Date: January 2001

From: Dave Heisler

To: Ask Tim

Subject: Questions About O'Reilly

Dear Tim,

I've been a programmer for about ten years. I had heard of

O'Reilly books,then...

Dave,

You might be amazed at how many requests for help with

school projects I get;

i: 50000

i: 51000

i: 52000

Trong các ứng dụng thực tế, ta sẽ tương tác với người dùng hoặc thực hiện các tính toán trong khi công việc nhập xuất tập tin hay cơ sở dữ liệu được thực hiện một cách bất đồng bộ ở một tiểu trình khác.

**Serialization**

Serialize có nghĩa là sắp theo thứ tự. Khi ta muốn lưu một đối tượng xuống tập tin trên đĩa từ để lưu trữ, ta phải định ra trình tự lưu trữ của dữ liệu trong đối tượng. Khi cần tái tạo lại đối tượng từ thông tin trên tập tin đã lưu trữ, ta sẽ "nạp" đúng theo trình tự đã định trước đó. Đây gọi là quá trình Serialize.

Nói chính xác hơn, serialize là tiến trình biến đổi trạng thái của đối tượng theo một định dạng có thể được lưu trữ hay dịch chuyển (transfer).

.NET Framework cung cấp 2 kỹ thuất serialize:

* Binary serialize (serialize nhị phân): cách này giữ nguyên kiểu dữ liệu, thích hợp cho việc giữ nguyên cấu trúc đối tượng. Có thể dùng kỹ thuật này để chia sẻ đối tương giữa các ứng dụng bằng cách serialize vào vùng nhớ clipboard; hoặc serialize vào các luồng, đĩa từ, bộ nhớ, trên mạng …; hoặc truyền cho máy tính ở xa như một tham trị ("by-value object")
* XML và SOAP Serialize: chỉ serialize các thuộc tính public, và không giữ nguyên kiểu dữ liệu. Tuy nhiên XML và SOAP là các chuẩn mở nên kỹ thuất không bị các hạn chế về giao tiếp giữa các ứng dụng.

Các đối tượng cớ sở đều có khả năng serialize. Để đối tượng của ta có thể serialize, trước tiên cần thêm khai báo attribute [Serialize] cho lớp đối tượng đó. Nếu đối tượng có chứa các đối tượng khác thì các đối tượng đó phải có khả năng serialize.

**Làm việc với Serialize**

Trước tiên, ta tạo một đối tượng Sumof làm ví dụ cho việc Serialize. Đối tượng có các biến thành viên sau:

private int startNumber = 1; private int endNumber; private int[] theSums;

mảng theSums đuợc mô tả: phần tử theSum[i] chứa giá trị là tổng từ startNumber cho đến startNumber + i.

**Serialize đối tượng**

Trước tiên thêm attribute [Serialize] vào trước khai báo đối tượng

[Serializable]

class SumOf

Ta cần một tập tin để lư trữ đối tượng này, tạo một FileStream

FileStream fileStream = new FileStream("DoSum.out",FileMode.Create);

Sau khi tạo một Formatter, gọi phương thức Serialize của nó.

binaryFormatter.Serialize(fileStream,this);

Đối tượng Sumof đã được Serialize.

**Deserialize đối tượng**

Deserialize là tiến trình ngược với serialize, tiến trình này đọc dữ liệu được serialize để tái tạo lại đối tượng.

Khai báo phương thức tĩnh DeSerialize cho tiến trình này

public static SumOf DeSerialize( )

{

FileStream fileStream = new FileStream("DoSum.out",FileMode.Open); BinaryFormatter binaryFormatter = new BinaryFormatter( );

return (SumOf) binaryFormatter.Deserialize(fileStream);

fileStream.Close( );

}

**Serialize và Deserialize đối tượng**

using System;

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

namespace Programming\_CSharp

{

[Serializable]

class SumOf

{

public static void Main( )

{

Console.WriteLine("Creating first one with new...");

SumOf app = new SumOf(1,10);

Console.WriteLine("Creating second one with deserialize...");

SumOf newInstance = SumOf.DeSerialize( );

newInstance.DisplaySums( );

}

public SumOf(int start, int end)

{

startNumber = start;

endNumber = end;

ComputeSums( ); DisplaySums( );

Serialize( );

}

private void ComputeSums( )

{

int count = endNumber - startNumber + 1;

theSums = new int[count];

theSums[0] = startNumber;

for (int i=1,j=startNumber + 1;i < count;i++,j++)

{

theSums[i] = j + theSums[i-1];

}

}

private void DisplaySums( )

{

foreach(int i in theSums)

{

Console.WriteLine("{0}, ",i);

}

}

private void Serialize( )

{

Console.Write("Serializing...");

// tạo một file stream để đọ hay ghi

FileStream fileStream = new FileStream("DoSum.out",FileMode.Create);

//sử dung binary formatter

BinaryFormatter binaryFormatter = new BinaryFormatter( );

// serialize binaryFormatter.Serialize(fileStream,this); Console.WriteLine("...completed"); fileStream.Close( );

}

public static SumOf DeSerialize( )

{

FileStream fileStream = new FileStream("DoSum.out",FileMode.Open);

BinaryFormatter binaryFormatter = new BinaryFormatter( );

return (SumOf) binaryFormatter.Deserialize(fileStream);

fileStream.Close( );

}

private int startNumber = 1;

private int endNumber;

private int[] theSums;

}

}

**Kết quả:**

Creating first one with new...

1,

3,

6,

10,

15,

21,

28,

36,

45,

55,

Serializing......completed

Creating second one with deserialize...

1,

3,

6,

10,

15,

21,

28,

36,

45,

55,

**Handling Transient Data**

Theo cách nhìn nào đó thì serialize kiểu [[link]](https://voer.edu.vn/c/luong-du-lieu/9f00d32d/af9046a5#eip-631) rất lãng phí. Giá trị các phần tử trong mảng có thể tính bằng thuật toán vì vậy không nhất thiết phải serialize mảng này (và làm giảm đáng kể dung lượng tập tin lưu trữ).

Để CLR biết ta không muốn Serialize biến thành viên này, ta đặt attribute

[NonSerialize] trước khai báo:

[NonSerialized] private int[] theSums;

Theo logic, khi deserialize, ta không thể có ngay mảng và vì vậy cần thực hiện lại công việc tính toán một lần nữa. Ta có thể thực hiện trong hàm Deserialize, nhưng CLR cung cấp giao diện IDeserializationCallback, ta sẽ cài đặt giao diện này

[Serializable]

class SumOf : IDeserializationCallback

Giao diện này có một phương thức duy nhất là OnDeserialization() mà ta phải cài đặt:

public virtual void OnDeserialization (Object sender)

{

ComputeSums( );

}

Khi tiến trình Deserialize, phương thức này sẽ được gọi và mảng theSums được tính toán và khởi gán. Cái giá mà ta phải trả chính là thời gian dành cho việc tính toán này.

**Làm việc với dối tượng nonserialize**

using System;

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

namespace Programming\_CSharp

{

[Serializable]

class SumOf : IDeserializationCallback

{

public static void Main( )

{

Console.WriteLine("Creating first one with new..."); SumOf app = new SumOf(1,5); Console.WriteLine("Creating second one with

deserialize...");

SumOf newInstance = SumOf.DeSerialize( );

newInstance.DisplaySums( );

}

public SumOf(int start, int end)

{

startNumber = start; endNumber = end; ComputeSums( ); DisplaySums( ); Serialize( );

}

private void ComputeSums( )

{

int count = endNumber - startNumber + 1;

theSums = new int[count];

theSums[0] = startNumber;

for (int i=1,j=startNumber + 1;i < count;i++,j++)

{

theSums[i] = j + theSums[i-1];

}

}

private void DisplaySums( )

{

foreach(int i in theSums)

{

Console.WriteLine("{0}, ",i);

}

}

private void Serialize( )

{

Console.Write("Serializing..."); FileStream fileStream = new FileStream("DoSum.out",FileMode.Create);

BinaryFormatter binaryFormatter = new BinaryFormatter();

binaryFormatter.Serialize(fileStream,this);

Console.WriteLine("...completed");

fileStream.Close( );

}

public static SumOf DeSerialize( )

{

FileStream fileStream = new FileStream("DoSum.out",FileMode.Open);

BinaryFormatter binaryFormatter = new BinaryFormatter( );

return (SumOf) binaryFormatter.Deserialize(fileStream);

fileStream.Close( );

}

public virtual void OnDeserialization( Object sender )

{

ComputeSums( );

}

private int startNumber = 1;

private int endNumber;

[NonSerialized] private int[] theSums;

}

}

**Kết quả :**

Creating first one with new...

1,

3,

6,

10,

15, Serializing......completed

Creating second one with deserialize...

1,

3,

6,

10,

15,

**Isolate Storage**

Outlook Express (OE) là trình nhận/chuyển thư điện tử của Microsoft. Khi chạy trên môi trường đa người dùng như Windows 2000, nó cung cấp cho mỗi người dùng một hộp thư riêng. Các hộp thư này lưu trữ trên đĩa cứng thành nhiều tập tin khác nhau ở các thư mục thuộc quyền của người dùng tương ứng. Ngoài ra OE còn lưu giữ cảc các thiết đặt (như các cửa sổ hiển thị, tài khỏan kết nối …) của từng người dùng.

.NET Framework cung cấp các lớp thực hiện các công việc này. Nó tương tự như các tập tin .ini của Windows cũ, hay gần đây hơn là khóa **HKEY\_CURRENT\_USER** trong Registry. Lớp thực hiện việc này là luồng IsolatedStorageFileStream. Cách sử dụng tương tự như các luồng khác. Ta khởi tạo bằng cách truyền cho hàm dựng tên tập tin, các công việc khác hoàn toàn do luồng thực hiện.

**Isolated Storage**

using System;

using System.IO;

using System.IO.IsolatedStorage;

namespace Programming\_CSharp

{

public class Tester

{

public static void Main( )

{

Tester app = new Tester( );

app.Run( );

}

private void Run( )

{

// tạo một luồng cho tập tin cấu hình

IsolatedStorageFileStream configFile = new IsolatedStorageFileStream("Tester.cfg",FileMode.Create);

// tạo một writer để ghi lên luồng

StreamWriter writer = new StreamWriter(configFile);

// ghi dữ liệu lr6n tập tin config

String output;

System.DateTime currentTime = System.DateTime.Now;

output = "Last access: " + currentTime.ToString( );

writer.WriteLine(output);

output = "Last position = 27,35";

writer.WriteLine(output);

// tống sạch dữ liệu

writer.Flush( );

writer.Close( );

configFile.Close( );

}

}

}

Sau khi chạy đoạn mã này ta, thực hiện việc tìm kiếm tập tin test.cfg, ta sẽ thấy nó trong đường dẫn sau:

c:\Documents and Settings\Administrator\ApplicationData\ Microsoft\COMPlus\IsolatedStorage\0.4\ Url.wj4zpd5ni41dynqxx1uz0x0aoaraftc\ Url.wj4zpd5ni41dynqxx1uz0ix0aoaraftc\files

Mở tập tin này bằng Notepad, nội dung tập tin như sau

Last access: 5/2/2001 10:00:57 AM Last position = 27,35

Ta cũng có thể đọc tập tin này bằng chính luồng IsolatedStorageFileStream