ξ2. LUỒNG TRONG C#

Ta đã biết về hai kiểu hoạt động cơ bản của các luồng C#, gồm:

- Luồng đồng bộ, và
- Luồng bất đồng bộ

Tiết này thực hành với một luồng C# cơ bản, FileStream, một luồng con của luồng Stream.

3. Giới thiệu luồng FileStream

Luồng FileStream rất hữu dụng trong đọc ghi các tập tin và nó sẽ rất cần trong hoạt động của luồng Stream cùng các luồng con vào khi lập trình mạng.

Đường dẫn tập tin

Với lớp FileStream, ta còn có thể sử dụng truy cập các tập tin của hệ điều hành giống như truy cập các thiết bị nhập xuất chuẩn. Khi truy cập file qua mạng cũng có thể dùng lớp FileStream này nếu ta chuyển file thành luồng ở phía server; phía client ta sẽ chuyển ngược luồng thành file ban đầu.

Constructor[1] FileStream hỗ trợ một số cách tạo đối tượng FileStream. Dù là cách nào thì chúng cũng cần có chuỗi đường dẫn chỉ đến nơi tập tin lưu trữ.

Constructor sau đây tạo đối tượng FileStream có tên inF:

```
// Chỉ ra đường dẫn tập tin và chế độ truy xuất tập tin này
FileStream inF = new FileStream("C:\\Net\\NewStream.txt", FileMode.CreateNew);
```

Giá trị FileMode	Mô tả
Append	Mở/tạo file để ghi dữ liệu vào cuối. Nếu giá trị này còn được dùng thì file chưa thể đọc
Create	Tạo file mới. Nếu đã tồn tại file thì nó bị ghi đè.
CreateNew	Tạo file mới. Nếu đã tồn tại thì báo cáo và hủy lệnh.
OpenOrCreate	Mở file có sẵn. Nếu file chưa tồn tại thì báo cáo và hủy lệnh.

1 Constructor (được gọi là hàm tạo) là những hàm đặc biệt cho phép thực thi, điều khiển chương trình ngay khi khởi tạo đối tượng. Trong C#, Constructors có tên giống như tên của Class và không có giá trị trả về.

Truncate	Hủy nội dung có sẵn của file. Con trỏ file được chỉ về đầu.

Chế độ truy xuất tập tin

Ngoài ra còn có thêm các tham số chỉ ra cách thức truy xuất tập tin: Read, Write và ReadWrite. Theo đó, đối tượng FileStream inF có thể được khởi tạo theo cách sau:

```
FileStream inF = new FileStream("C:\\Net\\NewStream.txt",
                                                FileMode.Open, FileAccess.Write);
```

Quyền chia xẻ

Các tham số quy định quyền chia xẻ tập tin gồm: None, Read, Write, ReadWrite và Inheritable. Tham số Inheritable không được hỗ trợ trong Win32.

```
FileStream inF = new FileStream("C:\\Net\\NewStream.txt",
                                FileMode.Open, FileAccess.Write, FileShare.Read);
```

Kích thước bộ đệm

Kết hợp thêm tham số kích thước của bộ đệm luồng. Nếu không chỉ ra kích thước bộ đệm thì C# ngầm định là 8192 byte.

```
FileStream outF = new FileStream("C:\\Net\\NewStream.txt",
                            FileMode.Open, FileAccess.Write, FileShare.Read, 1000);
```

Trạng thái thi hành là đồng bộ hay bất đồng bộ

Thêm một tham số Boolean nữa, ta chỉ ra chế độ thi hành là đồng bộ hay bất đồng bộ. Nếu là true thì đó là thi hành bất đồng bộ, false là thi hành đồng bộ. Chẳng hạn, để tạo một đối tượng FileStream có tên OutF theo cách sau sẽ cho phép thi hành bất đồng bộ:

```
FileStream outF = new FileStream("C:\\Net\\NewStream.txt",
                        FileMode.Open, FileAccess.Write, FileShare.Read, 1000, true);
```

4. Lập trình luồng với FileStream

Lập trình thi hành đồng bộ

Luồng Stream cung cấp hai phương thức là Read() và Write() để thi hành hai hành động là đọc và ghi theo thể thức đồng bộ với một luồng. Ví dụ sau sẽ gắn đối tượng luồng với một tập tin. Chương trình nhỏ này còn dùng Seek() để thiết lập vị trí truy xuất trong luồng.

Ta cần có các không gian tên gồm System.IO cho các hoạt động I/O; cần có System.Text để triệu gọi các phương thức chuyển đổi một chuỗi ký tự thành một mảng byte.

Tạo luồng FileStream, chỉ định cho nó bằng tham số FileMode là OpenOrCreate. Tham số này sẽ mở tập tin nếu nó đã tồn tại, còn không thì sẽ tạo mới. Ta gọi tập tin này là SyncDemo.txt, nó được ghi vào cùng thư mục với tập tin exe mà ta sẽ biên dịch trước lúc thi hành.

```
using System;
using System.IO;
using System.Text;

class SyncIO
{
   public static void Main(string[] args)
   {
        // Tạo đối tượng FileStream
        FileStream syncF = new FileStream("SyncDemo.txt",FileMode.OpenOrCreate);
```

Bây giờ ta sẵn sàng khảo sát thể thức ghi luồng đồng bộ. Bắt đầu bằng phương thức WriteByte(). Để ghi được ký tự A vào luồng, cần phải đổi ký tự thành byte, rồi ghi vào trong file. Sau khi ghi xong byte này, con trỏ file tự động tăng lên một vị trí.

```
        syncF.WriteByte(Convert.ToByte('A'));
```

Sử dụng phương thức Write(), ta có thể ghi vào luồng nhiều hơn một ký. Trước hết cần phải chuyển đổi chuỗi ký tự thành một mảng byte bằng phương thức GetBytes() có trong lớp Encoding của không gian System.Text. Tiếp đó, ghi mảng byte này vào luồng file bằng phương thức Write(). Phương thức Write() nhận ba tham số gồm: tên của mảng byte cần ghi vào luồng, vị trí hay còn gọi là chỉ số mảng mà từ đó ta bắt đầu đọc ra để ghi, và tham số thứ ba là chiều dài cần ghi vào file kể từ vị trí chỉ ra ở tham số thứ hai - ở ví dụ này ta lấy cả mảng.

```
        Console.WriteLine("--Write method demo--");
        byte[] writeBytes = Encoding.ASCII.GetBytes(" is a first character.");
        syncF.Write(writeBytes, 0, writeBytes.Length);
```

Cho đến đây, tập tin SyncDemo.txt chứa dòng chữ A is a first character. Giờ ta sẽ đọc toàn bộ dòng này ra từ file bằng phương thức Read() và hiển thị nó lên màn hình. Để làm thế, ta cần chuyển con trỏ file đang ở cuối trở về đầu.

```
        syncF.Seek (0,SeekOrigin.Begin);
        Console.WriteLine ("--Readbyte method demo--");
```

Con trỏ đã ở đầu file, và ta đọc byte đầu tiên bằng phương thức ReadByte(). Byte này sẽ được chuyển đổi thành ký tự trước khi in ra màn hình.

```
    // Đọc byte và hiển thị
    Console.WriteLine("First character is ->" + Convert.ToChar(syncF.ReadByte()));
```

Có thể đọc nội dung bằng phương thức ReadByte(). Phương thức này cho phép đọc một đoạn dài kể từ vị trí con trỏ file hiện hành. Chiều dài của đoạn cần đọc sẽ được thiết lập thông qua một bộ đệm. Trường hợp này ta lấy toàn bộ phần còn lại của luồng. Theo đó, chiều dài bộ đệm chính là syncF.Length – 1.

```
    // Đọc bằng phương thức Read
    Console.WriteLine("----Read method demo----");
    // Khai báo bộ đệm
    byte[] readBuf = new byte[syncF.Length-1];
    // Đọc file
    syncF.Read(readBuf,0,(Convert.ToInt32(syncF.Length))-1);
    // Hiển thị nội dung đã được lưu trong bộ đệm
    Console.WriteLine("The rest of the file is : " +
                      Encoding.ASCII.GetString(readBuf));
  }
}
```

Lập trình thi hành bất đồng bộ

Constructor FileStream cung cấp một cờ có tên IsAsync mà nó có thể là true hay false để biểu thị trạng thái thi hành là đồng bộ hay bất đồng bộ. Các phiên bản Windows NT trở đi đã hỗ trợ cả hai chế độ thi hành này.

Như đã trình bày ở tiết học trước, ủy nhiệm hàm – hay delegate - AsyncCallback hỗ trợ một cơ chế hữu dụng trong lập trình bất đồng bộ. Nó thực hiện báo cho ứng dụng client khi công việc hoàn thành. Ủy nhiệm hàm này có thể làm việc với cả hai phương thức BeginRead() và BeginWrite().

Như thường lệ, ta bắt đầu với các không gian tên cần thiết, một biến đối tượng FileStream và một mảng byte.

Tiếp đó, khai báo một delegate tĩnh Callback. Trong thân chương trình chính, Callback được khởi tạo và trỏ tới hàm AsyncCallback(). Đây chính là cách chung để gọi phương thức thi hành vào lúc kết thúc công việc.

```
using System;
using System.IO;
using System.Text;
using System.Threading;
public class AsyncDemo
{
   static FileStream fileStm;          // Khai báo đối tượng luồng đọc
   static byte[] readBuf;              // Khai báo bộ đệm để đọc
   static AsyncCallback Callback;      // Khai báo delegate AsyncCallback
public static void Main(String[] args)
{
   Callback = new AsyncCallback(CallBackFunction);
   //Khởi tạo đối tượng FileStream dưới dạng bất đồng bộ
   fileStm = new FileStream(@"C:\Networking\Streams\Test.txt",
             FileMode.Open, FileAccess.Read, FileShare.Read, 64, true);
   readBuf= new byte[fileStm.Length];
```

Bây giờ ta có thể sử dụng phương thức BeginRead() để đọc nội dung luồng tập tin theo dạng bất đồng bộ. Delegate callback được truyền cho BeginRead() dưới dạng tham số áp chót.

```
   // Đọc luồng theo dạng bất đồng bộ. Callback được gọi lúc hoàn thành
   fileStm.BeginRead(readBuf, 0, readBuf.Length, Callback, null);
```

Dữ liệu được đọc lần lượt từ luồng FileStream. Trong lúc đó, các hành động khác vẫn được thi hành mà không phải chờ đợi. Để chứng minh điều đó, ta bổ sung thêm một đoạn mã có chứa một vòng lặp. Khi FileStream hoàn thành công việc nó sẽ tự đóng lại.

```
   for (long i = 0; i < 5000; i++)
   {
     if (i % 1000 == 0)
     {
        Console.WriteLine("Executing in Main  - " + i.ToString());
        Thread.Sleep(10);
     }
```

```
    }
     fileStm.Close();
}
```

Để cho đơn giản, ví dụ này tạm thời chưa bàn tới một vấn đề quan trọng ở đây. Cụ thể là, nếu vòng lặp đã thi hành xong mà hành động đọc bất đồng bộ vẫn chưa hoàn thành. Dù thế, đối tượng luồng FileStream vẫn đóng lại mà không chờ lệnh BeginRead() hoàn thành công việc. Tiết học sau sẽ giải quyết vấn đề này.

Hàm callback, mà ở đây ra gọi là CallBackFunction(), sẽ được gọi khi bộ đệm đã đầy. Phương thức EndRead() được gọi để báo hoàn thành công việc đọc bất đồng bộ. Nếu con trỏ chưa đến được dấu cuối file thì hàm BeginRead() lại được gọi thêm lần nữa để tiếp tục đọc.

```
   static void CallBackFunction(IAsyncResult asyncResult)
   {
      // Hàm thực hiện chạy khi bộ đệm đầy
      int readB = fileStm.EndRead(asyncResult);
      if (readB > 0)
      {
         fileStm.BeginRead(readBuf, 0, readBuf.Length, Callback, null);
         Console.WriteLine(Encoding.ASCII.GetString(readBuf, 0, readB));
      }
   }
}
```