File

Informatica
Prof. Viglietti Francesco
a.s. 2012-13

Definizioni

Un file può essere definito in modo generale come un:

un contenitore di informazioni a carattere persistente. In un percorso assoluto il punto di partenza è sempre rappresentato dalla radice e cioè dalla barra,

preceduta opzionalmente dal simbolo di unità. Dunque, esso si presenta sempre nella forma:

<UNITÀ>:\...

Un percorso relativo comincia invece dalla "directory corrente".

Ad esempio, il percorso assoluto del file "alimenti.doc" è "D:\casa\spese\alimenti.doc", mentre il suo percorso relativo alla directory "casa" è "spese\alimenti.doc".

Nell'ambito di una LAN esiste una convenzione che consente di fare riferimento ad oggetti che risiedono in un computer remoto; si chiama UNC. Esso è per definizione un percorso assoluto e inizia sempre con due barre, il primo elemento è il nome del PC nel quale si trova l'oggetto da raggiungere. Segue l'elenco delle directory condivise separate dalla barra come in un percorso qualsiasi.

Ad es., supponiamo che l'unità "D:" si trovi in un server di nome "trixie" e che essa sia condivisa con il nome D$; ecco il percorso per accedere al file "Info.txt" della cartella scuola:

"\\trixie\d$\scuola\info.txt"

File di testo

Innanzitutto, il termine "file di testo" è convenzionale e non significa che i file di questo tipo debbano contenere solo dati di natura testuale, oppure che siano prodotti da programmi di video scrittura. Esso fa riferimento al formato, "formato testo" appunto, mediante il quale sono memorizzate le informazioni nel file.

Il formato testo di un file implica che:

- il suo contenuto, qualunque sia la sua natura, è comunque espresso in forma testuale;
- esso è di norma suddiviso in righe, e cioè stringhe di caratteri di lunghezza arbitraria il cui termine è indicato dalla sequenza di caratteri "\r\n" (CR e LF) o dal solo '\n'.

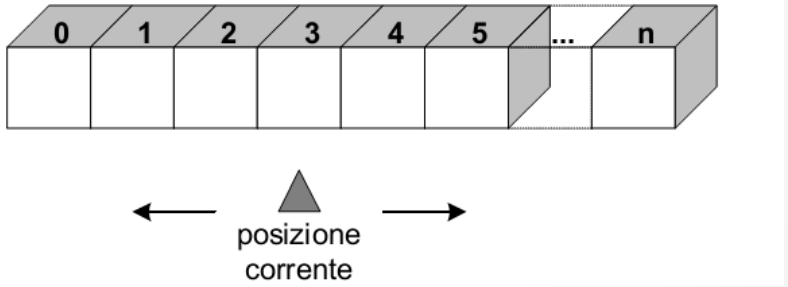
Memorizzare i dati in forma testuale significa convertirli in stringhe e quindi scrivere queste ultime nel file.

Stream

Perché un programma possa processare un file è necessario prima stabilire un collegamento con esso. Ciò avviene attraverso un oggetto (**stream**) che si interfaccia con il file system e si connette al file in questione, rendendo possibili le operazioni richieste. Il concetto di stream è indipendente da quello di file, è infatti un:

oggetto astratto che fornisce l'accesso a una generica sequenza di byte indipendentemente dalla sua reale natura e provenienza.

Uno stream può supportare una o più delle seguenti operazioni:



- lettura: i dati vengono letti dallo stream e caricati in memoria;
- scrittura: i dati vengono memorizzati nello stream;
- posizionamento: viene modificata la "posizione corrente" senza lettura o scrittura dei dati.

Classe filestream

Tutte le classi esaminate, sono definite all'interno del namespace ***System.IO***.

La connessione con il file viene stabilita mediante l'invocazione del costruttore. Passando gli appropriati argomenti al costruttore è possibile specificare alcune informazioni sulla modalità di accesso richiesta:

- percorso del file;
- modo di connessione;
- tipo di accesso;
- livello di condivisione.

L'invocazione del costruttore assume la seguente forma:

oggetto-stream= new FileStream (stringpercorso, FileMode modo, FileAccess accesso*, FileShare condivisione*);

Esempio: FileStream fs = new FileStream(@"c:\temp\prova.txt", FileMode.Open);

(*)opzionali

opzioni

Percorso del file → stringa con il percorso del file con il quale stabilire la connessione.

Modo di connessione → Consente di specificare se il file dev'essere creato o aperto:

Append →Se il file esiste lo apre e si posiziona alla fine, se non esiste lo crea. Questa modalità di connessione è a sola scrittura e dev'essere usata con FileAccess.Write

Create → Crea il file. Se il path si riferisce a un file esistente il contenuto viene cancellato.

CreateNew → Crea il file. Se il percorso specificato fa riferimento a un file già esistente viene sollevata l'eccezione IOException.

Open → Apre il file. Se il file non esiste viene sollevata l'eccezione FileNotFoundException.

OpenOrCreate →Apre il file se questo esiste, altrimenti lo crea.

Truncate →Apre il file, ne azzera il contenuto e imposta Position a 0. Se il file non esiste viene sollevata l'eccezione FileNotFoundException.

Tipo di accesso → si specifica se l'accesso al file è in lettura, scrittura o lettura/scrittura:

Il valore specificato per l'argomento FileAccess dev'essere coerente con la modalità di connessione richiesta (argomento FileMode). Il valore di default FileAccess.ReadWrite.

Es. Tentativo di apertura in lettura di un file già esistente con accesso in sola lettura:

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

Tentativo di apertura in modalità "append" di un file già esistente:

```
FileStream fs = new FileStream("esiste.txt", FileMode.Append); // errore!
```

produce un'eccezione, poiché il tipo di accesso, assunto per default FileAccess.ReadWrite, è incompatibile con la modalità di connessione FileMode.Append.

Accesso

La classe FileStream mette a disposizione funzionalità poco sofisticate per quanto riguarda le operazioni di lettura e scrittura del file.

Per processare il file esistono fondamentalmente 2 metodi, Read()e Write(), i quali consentono di leggere o scrivere una sequenza di byte.

Metodo Read():

Int Read(byte[] vett, int indice, int lunghezza)

Il metodo legge dal file una sequenza di byte pari a lunghezza, memorizzandoli in vettore a partire dalla posizione specificata da indice. L'array vett dev'essere già stato creato prima dell'invocazione del metodo. Read()ritorna come valore il numero di byte letti, (minore di lunghezza se viene raggiunta la fine del file). Dopo l'operazione, la posizione corrente dello stream, viene incrementata del numero di byte letti.

Metodo Write():

void Write(byte[] vettore, int indice, int lunghezza)

Il metodo scrive sul file i byte memorizzati in vettore, partendo da indice e per un numero di byte pari a lunghezza. Dopo l'operazione, Position viene incrementata del numero di byte scritti.

Accesso 2

metodo Seek():

La proprietà Position consente di conoscere, di modificare, la posizione corrente dello stream. In questo senso, lo stream, e il file connesso, può essere visto come un vettore di byte il cui indice del primo elemento è zero. In sostanza, la proprietà Position consente un accesso casuale al file esattamente come avviene per l'indice di un array.

La posizione corrente viene aggiornata in modo automatico conseguentemente alle operazioni di lettura e scrittura, ma può essere impostata direttamente sia assegnando un valore alla proprietà Position, scrivendo ad esempio:

fs.Position = 0; // imposta posizione corrente all'inizio del file sia invocando il metodo Seek():

Long Seek(long distanza, SeekOrigin origine)

Mediante Seek()è possibile spostare la posizione corrente di un valore pari a distanza, relativamente:

- all'inizio: argomento origine uguale a SeekOrigin.Begin;
- alla fine: argomento origine uguale a SeekOrigin.End;
- all'attuale posizione: argomento origine uguale a SeekOrigin.Current;

Il metodo ritorna la nuova posizione corrente. Ecco un esempio d'uso:

Seek(0, SeekOrigin.Begin); // imposta posizione corrente a inizio file

Accesso 3

metodo SetLength():

La proprietà Length è a sola lettura e ritorna il numero dei byte memorizzati nel file. Da notare che il tipo della proprietà è long e dunque è necessario usare il cast se si vuole utilizzarla come un intero. Ad esempio:

Int lunghezzaFile = (int) fs.Length;

Esiste un metodo, SetLength(), che consente di impostare la lunghezza del file:

Void SetLength(long lunghezza)

Invocando SetLength(), è dunque possibile modificare la dimensione del file in accordo alle seguenti regole:

- se lunghezza è minore dell'attuale dimensione del file, questo viene troncato;
- se lunghezza è maggiore dell'attuale dimensione de file, il file viene espanso. Il contenuto di questo, a partire dalla vecchia posizione fino alla nuova, è indefinito.

File di testo

Classe StreamWriter

La classe StreamWriter è un writer che collegandosi a uno stream consente di scrivere su di esso in formato testo. Essa mette a disposizione un nutrito insieme di metodi sia per la scrittura dei tipi di dati predefiniti che per la scrittura di una generica sequenza di caratteri, memorizzata su array. Ogni dato viene convertito in stringa prima di essere scritto nello stream; la stringa risultante viene scritta in base alla codifica – UTF-8, ASCII, ecc. – adottata.

Elenco delle proprietà e dei metodi d'uso più comune.

PROPRIETÀ	DESCRIZIONE

`AutoFlush`
 	
`bool`
	Se true fa sì che il writer svuoti il buffer interno dopo ogni istruzione di scrittura, altrimenti i dati saranno effettivamente scritti su disco soltanto quando il buffer è pieno. (get)

`BaseStream`
 	
`stream`
	Riferimento allo stream sottostante allo StreamWriter. (get)

`Encoding`
 	
`encoding`
	Consente di stabilire il tipo di codifica dei caratteri (Unicode, UTF-8, ASCII, eccetera). Il tipo predefinito è UTF-8. (get)

`NewLine`
 	
`string`
	Definisce il terminatore di linea utilizzato. Il terminatore di default è la combinazione "\r\n" (ritorno carrello + nuova linea). (get e set)

Costruttori StreamWriter

StreamWriter(Stream s)

Crea uno StreamWriter che scrive sullo stream specificato. Lo stream dev'essere abilitato alla scrittura (vedi FileAccess).

StreamWriter (string nomeFile)

Crea uno StreamWriter che scrive su uno stream creato automaticamente e associato al file specificato. Se nomeFile fa riferimento a un file già esistente il suo contenuto viene cancellato, altrimenti il file viene creato.

StreamWriter (string nomeFile, Encoding codifica)

Crea uno StreamWriter che scrive su uno stream creato automaticamente e associato al file specificato. Se nome file fa riferimento a un file già esistente il suo contenuto viene cancellato, altrimenti il file viene creato.

Viene applicata la codifica specificata (Unicode, ASCII, UTF-8, UTF-7).

StreamWriter (string nomeFile, bool aggiungi)

Crea uno StreamWriter che scrive su uno stream associato automaticamente al file specificato. Se nomeFile fa riferimento a un file già esistente, se aggiungi è :

- true: i dati vengono aggiunti al precedente contenuto del file;
- false: il precedente contenuto del file viene cancellato;

Per default viene impostata il tipo di codifica UTF-8.

Metodi StreamWriter

void Close()

Chiude il writer e lo stream sottostante.

void Write(tipo valore)

Scrive il valore sullo stream dopo averlo convertito in stringa. La sequenza di caratteri risultante dipende oltre che dal valore anche dal tipo di codifica impiegata.

void Write(char[] buffer, int indice, int lunghezza)

Scrive sullo stream i caratteri contenuti in buffer, partendo da indice e per un numero di caratteri pari a lunghezza. La sequenza di byte effettivamente scritti dipende dalla codifica impiegata.

void Write(string formattazione, lista-parametri)

Scrive sullo stream i valori presenti nella lista-parametri dopo averli convertiti in accordo alla stringa formattazione.

WriteLine()

Questo metodo presenta le stesse forme e produce lo stesso effetto del metodo Write(). In più, dopo aver scritto gli argomenti sullo strema, scrive un terminatore di linea.

Classe StreamReader

La classe StreamReader è un reader che collegandosi a uno stream consente di leggere da esso in formato testo. Il contenuto del file viene interpretato in base a una determinata codifica, (UTF-8 di default oppure specificata come argomento nel costruttore) la quale deve coincidere con quella effettivamente impiegata per scrivere il file; in caso contrario il risultato delle successive operazioni di lettura sarà imprevedibile.

PROPRIETÀ	DESCRIZIONE

`BaseStream`
 	
`Stream`
	Riferimento allo stream sottostante allo StreamWriter. (get)

`CurrentEncoding`
 	
`Encoding`
	Restituisce il tipo di codifica dei caratteri utilizzato per leggere dallo stream. (get)

Costruttori StreamReader

StreamReader(Stream s)

Crea uno StreamReader che legge dallo stream specificato.

StreamReader(string nomeFile)

Crea uno StreamReader che legge da uno stream creato automaticamente e associato al file specificato. Se nomeFile non fa riferimento a un file già esistente viene sollevata un'eccezione.

StreamReader (string nomeFile, Encoding codifica)

Crea uno StreamReader che legge da uno stream creato automaticamente e associato al file specificato. Se nomeFile non fa riferimento a un file già esistente viene sollevata un'eccezione.

Durante la lettura viene applicata la codifica specificata (Unicode, ASCII, UTF-8, UTF-7).

StreamReader (string nomeFile, bool rilevaCodifica)

Crea uno StreamReader che legge da uno stream creato automaticamente e associato al file specificato. Se nomeFile non fa riferimento a un file già esistente viene sollevata un'eccezione.

Se rilevaCodifica vale true, il reader usa l'eventuale prologo memorizzato nel file per determinare in modo automatico la codifica impiegata per scriverlo.

Metodi StreamReader

void Close()

Chiude il reader e lo stream sottostante.

int Peek ()

Legge e ritorna un carattere dallo stream, o ritorna –1 se è stata raggiunta la fine dello stream. Non avanza l'indicatore di posizione (BaseStream.Position).

int Read()

Legge e ritorna un carattere dallo stream, o ritorna –1 se è stata raggiunta la fine dello stream. Avanza di 1 l'indicatore di posizione (BaseStream.Position).

int Read(char[] buffer, int indice, int lunghezza)

Legge dallo stream un numero di caratteri pari a lunghezza e li memorizza in buffer a partire da indice. Ritorna il numero di caratteri effettivamente letti (0 se era stata raggiunta la fine dello stream).

string ReadLine()

Legge una linea dallo stream e la ritorna come stringa. Se era stata raggiunta la fine dello stream ritorna una stringa nulla. (null). Una linea è definita come una sequenza di caratteri seguita da un terminatore di linea; questo è per default definito dalla sequenza di caratteri "\r\n"(ritorno carrello + nuova linea), la quale non viene comunque memorizzata nella stringa.

string ReadToEnd()

Legge dalla posizione corrente fino alla fine dello stream. Ritorna i caratteri letti sotto forma di stringa. Nella stringa vengono memorizzati anche gli eventuali terminatori di linea.

Primi passi…

Aggiunta di **using System.IO;**

```
Class Program
staticList<string> righe;
static void LeggiFile(stringpercorso)
{
StreamReader sr = new StreamReader(percorso); //dich.e istanz. reader
righe = new List<string>();
String linea;
Do
        {
        linea = sr.ReadLine();          //lettura reader
        righe.Add(linea);
        }
while(linea != null)
sr.Close();                     //chiusura reader

static void ScriviFile(string percorso)
{
 StreamWriter sw = new StreamWriter(percorso); //dich.e istanz. writer
Elaborare file di testo   35
Foreach (string linea in righe)
sw.WriteLine(linea);                    //scrittura writer
sw.Close();                     //chiusura writer
}
```