File di testo

Elaborazione dei file di testo

Ambiente: .NET 2.0+/C# 2.0+

Anno 2013/2014

Indice generale

1 Introduzione ai file di testo....3

1.1 Conversione dei contenuti in stringa....3

1.1.1 Esempio: codifica dei dati numerici....3

2 Leggere un file di testo....4

2.1 Lettura di un file di testo: StreamReader....4

2.1.1 Apertura del file....4

2.1.2 Chiusura del file....4

2.1.3 Lettura del contenuto....5

2.2 Lettura di un file di testo: metodi statici della classe File....5

3 Scrittura di un file di testo....6

3.1 Scrittura di un file di testo: StreamWriter....6

3.1.1 Aggiungere testo ad un file esistente....6

3.2 Scrittura di un file di testo: metodi classe File....6

1 Introduzione ai file di testo

Un file di testo memorizza le informazioni in forma testuale (***formato testo***), codificandole come un flusso di caratteri.

Di norma è suddiviso in *righe di testo:* stringhe di caratteri di lunghezza arbitraria separate dalla sequenza “\r\n” (oppure dal solo carattere “\n”).

Il ***formato testo*** è universalmente riconosciuto da qualsiasi tipo di software e sistema operativo e viene impiegato per memorizzare contenuti di varia natura:

- il codice sorgente dei programmi;
- pagine HTML e documenti XML;
- file di comandi e di script;
- semplici testi privi di formattazione, ad esempio prodotti con Notepad;
- file di configurazione e file di log.

1.1 Conversione dei contenuti in stringa

Memorizzare i dati in formato testo significa convertirli in stringhe e scriverle su file. Naturalmente ciò non è necessario per valori stringa e carattere, poiché sono già nel giusto formato.

1.1.1 Esempio: codifica dei dati numerici

Si ipotizzi di voler scrivere su file tre valori numerici, uno per ogni riga:

```
10
1,5
-2,13
```

In memoria, ognuno di questi valori occupa 8 byte (tipo double), e usa il formato IEEE 754. Ma nella scrittura in formato testo, il risultato effettivamente prodotto è il seguente flusso di caratteri:

'1'	'0'	'\r'	'\n'	'1'	','	'5'	'\r'	'\n'	'-'	'2'	','	'1'	'3'
49	48	13	10	48	44	53	13	10	45	50	44	49	51

Come mostra la seconda riga, su disco non vengono memorizzati caratteri, ma i byte corrispondenti in base al codice alfanumerico utilizzato. Questi dipendono dal set di caratteri utilizzato: ASCII, UTF8, UTF16, eccetera. (nell'esempio è UTF8)

Inoltre, il concetto di riga è puramente logico: in realtà esiste un'unica sequenza di byte. Un programma può interpretare questa sequenza come un elenco di righe utilizzando la coppia di caratteri “\r\n”, che determina la fine di una riga.

2 Leggere un file di testo

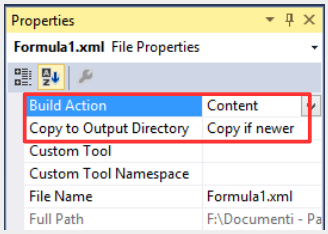
Un file di testo può essere letto mediante uno **StreamReader**.

Copiare un file del progetto nella cartella BIN/DEBUG

Di seguito ipotizzo che il file sia collocato nella "cartella di output", e cioè la cartella BIN/DEBUG del progetto.

Un modo semplice per collocare un file in questa cartella è quello di aggiungere il file al progetto; nel Solution Explorer, eseguire "clic destro" sul file e selezionare **Properties**.

Dentro Properties impostare **Build Action** su "*Content*" e **Copy to Ouput Directory** su "*Copy if newer*" (oppure su "*Copy always*").



2.1 Lettura di un file di testo: StreamReader

La lettura si svolge in tre fasi:

- Apertura del file: creazione dell'oggetto **StreamReader**.
- Lettura del testo: uso dei metodi di lettura (tipicamente: **ReadLine()**).
- Chiusura del file: esecuzione del metodo **Close()**.

2.1.1 Apertura del file

L'operazione di apertura associa lo StreamReader al file:

```
string filePath = "Classe.txt";
StreamReader sr = new StreamReader(filePath);
...
```

Il costruttore esiste in molte varianti, una delle quali consente di specificare la codifica dei caratteri da utilizzare. (Il valore predefinito è **Encoding.UTF8**).

```
string filePath = "Classe.txt";
StreamReader sr = new StreamReader(filePath, Encoding.ASCII);
...
```

2.1.2 Chiusura del file

Dopo l'apertura, il file viene bloccato dal sistema operativo in modo che non possa essere modificato da altri programmi. Per rilasciare questo blocco è necessario chiudere l'oggetto **StreamReader** associato al file.

```
string filePath = "Classe.txt";
StreamReader sr = new StreamReader(filePath, Encoding.ASCII);
...
sr.Close();
```

2.1.3 Lettura del contenuto

StreamReader definisce vari metodi di lettura, ma il più comune è **ReadLine()**, che legge una riga di testo.

Poiché non c'è modo di sapere in anticipo il numero di righe che compongono un file, è necessario leggere una riga per volta fino a quando non ci sono più righe da leggere; in questo caso **ReadLine()** ritorna il valore null.

```
string filePath = "Classe.txt";
StreamReader sr = new StreamReader(filePath);
string line = sr.ReadLine();
while (line != null)
{
    //... elabora la riga
    line = sr.ReadLine();
}
sr.Close();
```

2.2 Lettura di un file di testo: metodi statici della classe File

A volte il contenuto di un file di testo deve essere semplicemente caricato in memoria, dove sarà elaborato successivamente. In questo caso sono utili due metodi della classe **File**: **ReadAllLines()** e **ReadAllText()**; entrambi utilizzano internamente uno **StreamReader**.

Il primo carica il file in un vettore di stringhe, una riga per ogni elemento del vettore.

Il secondo legge l'intero contenuto in una stringa, compresi i terminatori di riga (“\r\n”).

```
string filePath = "Classe.txt";
string[] lineList = File.ReadAllLines(filePath);
...
string text = File.ReadAllText(filePath);
...
```

Entrambi i metodi sono utili quando non è necessaria alcuna elaborazione durante la fase di lettura e tutto il contenuto del file deve essere caricato in memoria.

3 Scrittura di un file di testo

Un file di testo può essere scritto mediante uno **StreamWriter**.

3.1 Scrittura di un file di testo: StreamWriter

Il processo è analogo a quello di lettura:

- Apertura del file: creazione dell'oggetto **StreamWriter**.
- Scrittura del testo: uso dei metodi di lettura (tipicamente: **WriteLine()**).
- Chiusura del file: esecuzione del metodo **Close()**.

Il codice di seguito scrive tre nominativi su un file, uno per ogni riga:

```
string[] lineList = { "Bianchi Andrea", "Giorni Aldo", "Minghi Brenda" };

string filePath = "Classe.txt";
StreamWriter sw = new StreamWriter(filePath);
foreach (var line in lineList)
{
    sw.WriteLine(line);
}
sw.Close();
```

Nota bene: il metodo **WriteLine()** aggiunge automaticamente la coppia di caratteri “\r\n” al termine di ogni riga.

3.1.1 Aggiungere testo ad un file esistente

Il comportamento predefinito di **StreamWriter** è quello di creare il file oppure cancellarne il contenuto se il file esiste già.

Se si desidera aggiungere del testo ad un file esistente, occorre stabilirlo durante la creazione dell'oggetto:

```
string[] lineList = { "Parri Filippo", "Zazzi Elena"};

string filePath = "Classe.txt";
StreamWriter sw = new StreamWriter(filePath, true);
...
```

Il valore **true** passato al costruttore indica che il testo scritto si aggiungerà al precedente contenuto. Se il file non esiste, viene creato normalmente.

3.2 Scrittura di un file di testo: metodi classe File

La classe **File** definisce dei metodi scrittura analoghi a quelli di lettura: **WriteAllLines()** e **WriteAllText()**.

```
string filePath = "Classe.txt";
string[] lineList = { "Bianchi Andrea", "Giorni Aldo", "Minghi Brenda" };
File.WriteAllLines(filePath, lineList);

string text = "Bianchi Andrea\r\nGiorni Aldo\r\rMinghi Brenda";
File.WriteAllText(filePath, text);
```

Sono utili quando non è necessario avere il controllo durante il processo di scrittura.