lezione 27 di 27

Gestire il formato JSON

di Vito Gentile 30 aprile 2018

Nell'ultimo decennio, si è affermato sempre di più **JSON** (acronimo che sta per **J**ava**S**cript **O**bject **N**otation), formato di file di testo usato per la strutturazione dei dati, e basato sulla sintassi per la definizione degli oggetti su Javascript.

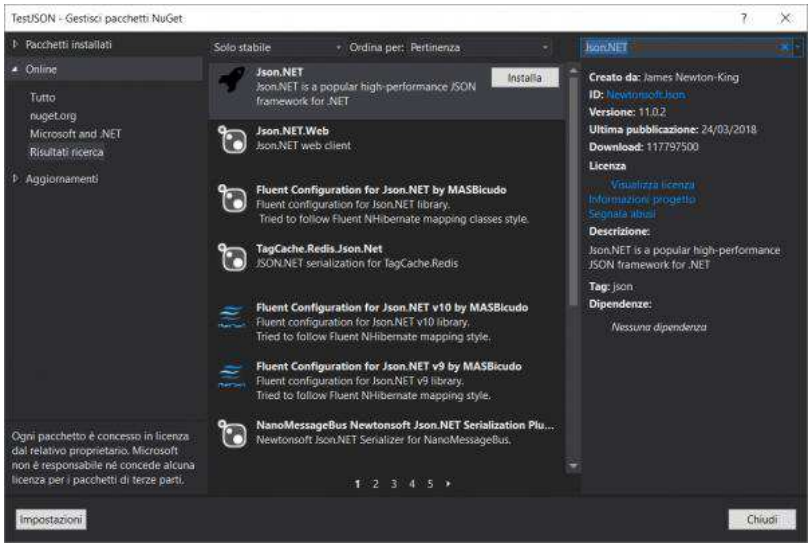
Il formato JSON, in maniera del tutto analoga ad **XML**, è spesso usato per lo scambio di dati tra moduli di applicazioni web, nonché per la gestione di file di configurazione o, più semplicemente, per archiviare dati in formato testuale. Data la grande diffusione di questo formato, è bene capire come leggere e manipolare dati di questo tipo anche tramite il linguaggio di programmazione C#. In questa lezione ci occuperemo proprio di questo, supponendo che il lettore conosca già questo formato di dati; se così non fosse, rimandiamo ad un apposito **approfondimento** di HTML.it su questo specifico argomento.

Installare Json.NET

Esistono molti modi per gestire i file JSON su C#. Tra questi, quello più semplice consiste nell'uso della libreria **Json.NET**, prodotta da Newtonsoft e facilmente integrabile in qualsiasi progetto tramite l'uso del package manager **NuGet**, come spiegato sul **sito ufficiale**. Vediamo rapidamente come installarla tramite **Visual Studio**; noi faremo riferimento alla versione 2013, ma per tutte quelle successive il procedimento è pressocché analogo.

Dal pannello *Esplora soluzioni*, clicchiamo con il tasto destro su *Riferimenti* e selezioniamo la voce *Gestisci pacchetti NuGet…*. Si aprirà quindi una nuova finestra, dove scriveremo (nel campo di ricerca in alto a destra), la stringa "Json.NET". Premendo il tasto *Invio*, la prima opzione visualizzata sarà quella che dovremo selezionare per l'installazione.

Figura 1. Installazione di Json.NET tramite NuGet (click per ingrandire)



Clicchiamo quindi sul pulsante *Installa* mostrato di fianco al nome della libreria, e saremo pronti ad utilizzare Json.NET.

Convertire un oggetto C# in una stringa JSON

La potenza di Json.NET consiste nella capacità di trasformare praticamente qualsiasi oggetto C# in una stringa JSON, in maniera quasi del tutto trasparente al programmatore. Per capire meglio, vediamo uno degli esempi tratti dalla **documentazione ufficiale di Json.NET**. Innanzitutto, definiamo una classe *Account* come segue:

```
public class Account
{
    public string Email { get; set; }
    public bool Active { get; set; }
    public DateTime CreatedDate { get; set; }
    public IList<string> Roles { get; set; }
}
```

Creare quindi un'istanza della classe *Account*:

```
Account account = new Account
{
```

```
    Email = "james@example.com",
    Active = true,
    CreatedDate = new DateTime(2013, 1, 20, 0, 0, 0, DateTimeKind.Utc),
    Roles = new List<string>
    {
        "User",
        "Admin"
    }
};
```

Volendo **convertire** l'oggetto *account* in una stringa JSON, ci basterà una semplice riga di codice che richiami il metodo statico `JsonConvert.SerializeObject` :

```
string json = JsonConvert.SerializeObject(account, Formatting.Indented);
```

L'output che verrà salvato nella stringa *json* sarà il seguente:

```
{
  "Email": "james@example.com",
  "Active": true,
  "CreatedDate": "2013-01-20T00:00:00Z",
  "Roles": [
    "User",
    "Admin"
  ]
}
```

Possiamo utilizzare il metodo `JsonConvert.SerializeObject` anche passando il solo oggetto *account* come unico argomento. L'uso del secondo parametro (in questo caso la costante `Formatting.Indented`) ci permette di specificare come vogliamo formattare la stringa: questo esempio ne genera una indentata, ma potevamo generarne una senza indentazione utilizzando la costante `Formatting.None` .

Possiamo specificare molte altre opzioni, sfruttando ad esempio un oggetto **JsonSerializerSettings** come secondo argomento. Poiché questo livello di dettaglio esula dagli scopi di questa lezione, ci limitiamo a segnalare l'**apposita pagina della documentazione ufficiale**, dedicata a tutte le varianti del metodo `JsonConvert.SerializeObject` , per tutti gli approfondimenti del caso.

Per chiudere il discorso relativo alla conversione di un oggetto C# in JSON, vale la pena menzionare che possiamo facilmente salvare la stringa appena generata in un **file di testo**. Anche in questo caso, bastano un paio di righe di codice:

```
string json = JsonConvert.SerializeObject(account, Formatting.Indented);
System.IO.File.WriteAllText(@"C:\path\to\my\folder\output.txt", json);
```

Convertire una stringa JSON in un oggetto C#

Non resta, a questo punto, che imparare ad implementare l'operazione inversa, ovvero quella che permette di convertire una stringa JSON in un oggetto C#. Supponiamo di avere definito la stessa classe *Account* vista in precedenza, e di volere convertire una stringa in una istanza di tale classe. Il codice di cui necessiteremo è ancora una volta abbastanza semplice:

```
string json = @"{
  ""Email"": ""james@example.com"",
  ""Active"": true,
  ""CreatedDate"": ""2013-01-20T00:00:00Z"",
  ""Roles"": [
    ""User"",
    ""Admin""
  ]
}";
Account account = JsonConvert.DeserializeObject<Account>(json);
```

Abbiamo specificato la classe di output *Account*, e Json.NET ha utilizzato questa informazione per convertire la nostra stringa in una istanza di tale classe. In realtà, a partire da .NET 4, possiamo sfruttare la parola chiave `dynamic` per rendere la conversione appena vista ancora più trasparente al programmatore.

Supponiamo infatti di non volere definire la classe *Account* (o, meglio, di non conoscere l'esatta strutturazione della stringa JSON che vogliamo convertire in oggetto C#). Potremo procedere come mostrato nel codice seguente:

```
dynamic obj = JsonConvert.DeserializeObject("{ \"Nome\": \"Vito Gentile\"
string nome = obj.Nome;
string citta = obj.Indirizzo.Citta;
```

Come si vede, l'oggetto *obj* viene automaticamente generato senza dover preventivamente specificare la struttuazione dei dati. In qualche caso, questo tipo di procedura può risultare estremamente comoda, mentre in altre situazioni affidarsi alla classica definizione delle classi che modellino precisamente la strutturazione dei dati può risultare la soluzione migliore.

Per approfondire l'uso del metodo `JsonConvert.DeserializeObject` , è possibile consultare l'**apposita pagina della documentazione ufficiale**.

Percorsi formativi correlati

per iOS Java

Ti consigliamo anche

DEVELOPMENT

Quarkus, guida allo stack Java Kubernetes

12 lezioni facile

Guida a Red Hat Quarkus, uno stack Java Kubernetes nativo pensato per applicazioni serverless e per la realizzazione di microservizi rapidi ed ottimizzati. Una piattaforma concepita per accedere in ambiente Cloud mettendo a disposizione un framework orientato ai microservizi, come ad esempio Spring Boot o Micronaut

Testare applicazioni Quarkus: JUnit

Sviluppare Blockchain in Java, la guida

DATABASE

Gestire database MongoDB con Python

Come creare database e collection, inserire, estrarre, aggiornare e rimuovere dati da una base di dati MongoDB con Python

Visual Studio Code: editing su Web browser con Python

Python 3.10: le novità in arrivo

DEVELOPMENT

PHP non è morto e domina il segmento server side

DEVELOPMENT

Thumby: Game Boy in miniatura programmabile in Python

DEVELOPMENT

Java 17 LTS: nuove feature e funzionalità rimosse