

Cải tiến view (1): NuGet, Newtonsoft, JSON

Hướng dẫn tự học lập trình C# toàn tập > Cải tiến view (1): NuGet, Newtonsoft, JSON

Trong bài học này chúng ta sẽ học cách sử dụng công cụ quản lý gói thư viện NuGet để cài đặt thư viện lớp của bên thứ ba. Chúng ta sẽ xem xét thư viện Newtonsoft Json để thêm chức năng xuất dữ liệu ra file.

Trong các bài trước chúng ta đã xây dựng được một khung chương trình cơ bản đáp ứng một số yêu cầu đặt ra từ phân tích ở bài 1.

Trong các bài còn lại của phần này, chúng ta sẽ lần lượt cải tiến các lớp giao diện sử dụng các kỹ thuật mới.

NỘI DUNG CỦA BÀI [Ấn]

1. Xuất dữ liệu ra file, JSON
2. Thực hành 1: cài đặt thư viện Newtonsoft.Json
 - 2.1. Bước 1. Mở giao diện quản lý các gói thư viện NuGet
 - 2.2. Bước 2. Chọn cài gói thư viện
 - 2.3. Kiểm tra kết quả
3. Những cách khác sử dụng NuGet Packages Manager
 - 3.1. Sử dụng website kết hợp Package Manager Console
 - 3.2. Lưu ý
4. Thực hành 2: thêm chức năng xuất dữ liệu ra file
 - 4.1. Bước 1. Bổ sung phương thức RenderToFile
 - 4.2. Bước 2. Điều chỉnh các phương thức của lớp BookController
 - 4.3. Bước 3. Điều chỉnh lớp Program
 - 4.4. Bước 4. Dịch và chạy thử chương trình
5. Kết luận

Xuất dữ liệu ra file, JSON

Trước hết chúng ta cải tiến các lớp giao diện bằng cách bổ sung thêm khả năng xuất dữ liệu ra file ở dạng json. Đây là yêu cầu có trong phần phân tích ca sử dụng ở bài đầu tiên (ca sử dụng số 8).

Vì đây là một ứng dụng dạng console, mọi thông tin đều viết ra màn hình. Giao diện console không tiện lợi lắm để xem danh sách dữ liệu quá dài. Ngoài ra chúng ta có thể muốn xuất thông tin ra để sử dụng trong những chương trình khác (chẳng hạn import vào excel, import vào cơ sở dữ liệu).

Một trong những định dạng dữ liệu được sử dụng và hỗ trợ rộng rãi hiện nay là *JSON* (JavaScript Object Notation).

Các bạn có thể tham khảo thêm về định dạng JSON ở các trang web này: [JSON Introduction](#), [JSON Syntax](#), [JSON.org](#).

Chuỗi ký tự JSON được viết theo cú pháp mô tả object của ngôn ngữ JavaScript và hiện được sử dụng rất rộng rãi để lưu trữ hoặc truyền dữ liệu. JSON được sử dụng đặc biệt phổ biến trong môi trường web và để lưu trữ thông tin cấu hình của ứng dụng.

.NET framework có hỗ trợ Json nhưng không được tiện lợi và hiệu quả như một số bộ thư viện của các bên thứ ba.

Trong phần thực hành dưới đây chúng ta sẽ học cách sử dụng bộ thư viện JSON của Newtonsoft.

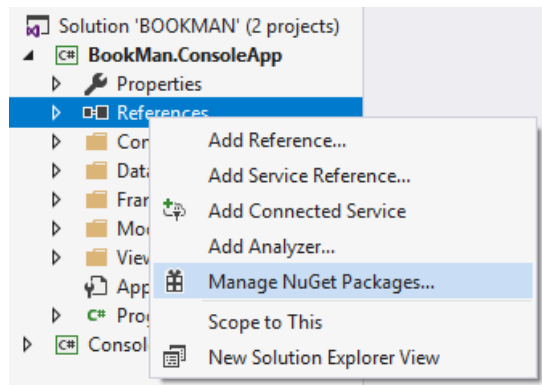
Thực hành 1: cài đặt thư viện Newtonsoft.Json

Bộ thư viện này cho phép chúng ta chuyển đổi một object của C# thành một chuỗi ký tự định dạng theo quy ước của JSON (JavaScript Object Notation) cũng như chuyển đổi ngược chuỗi JSON về object của C#. Đây là một trong những bộ thư viện có lượt download lớn nhất trên NuGet.

Quá trình chuyển đổi này có tên gọi là *serialization* (từ object về JSON) và *deserialization* (từ JSON về object).

Bước 1. Mở giao diện quản lý các gói thư viện NuGet

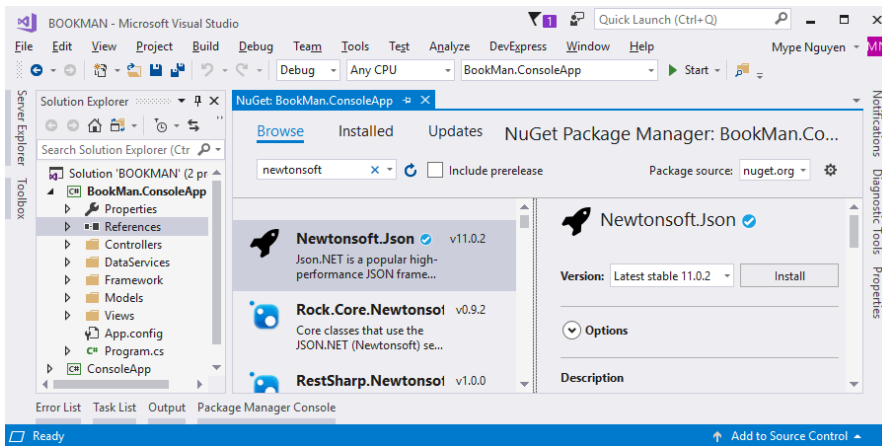
Click phải vào References, chọn Manage NuGet Packages (xem hình dưới đây).



Mở giao diện Manage NuGet Packages

Bước 2. Chọn cài gói thư viện

Trong ô tìm kiếm ở tab Browse gõ *newtonsoft*, chọn gói Newtonsoft.Json và ấn Install.



Giao diện Manage NuGet Packages

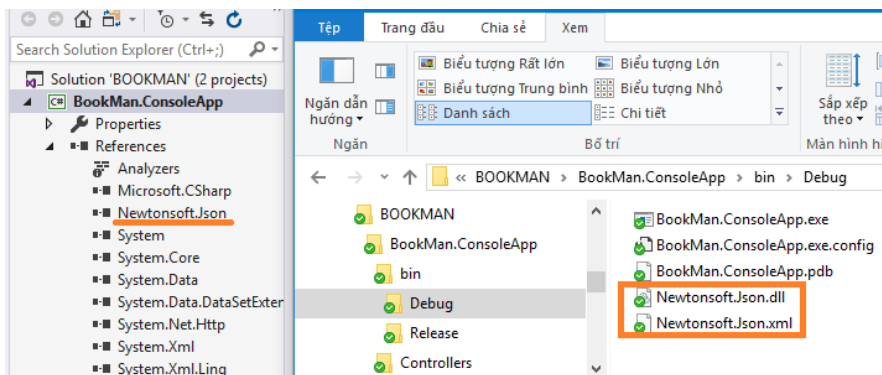
Sau lệnh này, Visual Studio sẽ tải gói thư viện này về và cài đặt lên project tương ứng (trong trường hợp này là BookMan.ConsoleApp).

Kiểm tra kết quả

Sau khi cài đặt thành công bộ thư viện này, trong danh sách References sẽ xuất hiện thêm một mục "NewtonSoft.Json". Trong cấu trúc dự án sẽ xuất hiện thêm file "packages.config" chứa thông tin về các gói thư viện được cài đặt đặt thêm.

Sau khi dịch chương trình (Ctrl + Shift + B) thành công, trong thư mục BinDebug của dự án sẽ xuất hiện file thư việc Newtonsoft.Json.dll.

Khi triển khai ứng dụng cho người dùng cuối, file thư viện này cũng phải đi cùng file chương trình.



File thư viện Newtonsoft.json.dll sau khi cài đặt

Những cách khác sử dụng NuGet Packages Manager

Chúng ta có thể sử dụng theo một số cách khác nhau:

1. sử dụng ứng dụng giao diện đồ họa NuGet Package Manager (như phần thực hành trên),
2. sử dụng giao diện dòng lệnh Package Manager Console,
3. cài đặt tự động với các file mã kịch bản.

Cách đơn giản nhất để tìm và cài đặt các gói thư viện từ NuGet là sử dụng tiện ích mở rộng NuGet Package Manager như đã thực hiện trong phần thực hành trên.


Sử dụng website kết hợp Package Manager Console

Cách thứ hai là sử dụng dịch vụ tìm kiếm trên website <https://www.nuget.org/packages> để tìm gói thư viện phù hợp. Sau đó copy dòng lệnh paste vào Package Manager Console.

Newtonsoft.Json 12.0.1-beta1 

Json.NET

Json.NET is a popular high-performance JSON framework for .NET

 This is a prerelease version of Newtonsoft.Json.

Requires NuGet 2.12 or higher.

Package Manager

.NET CLI

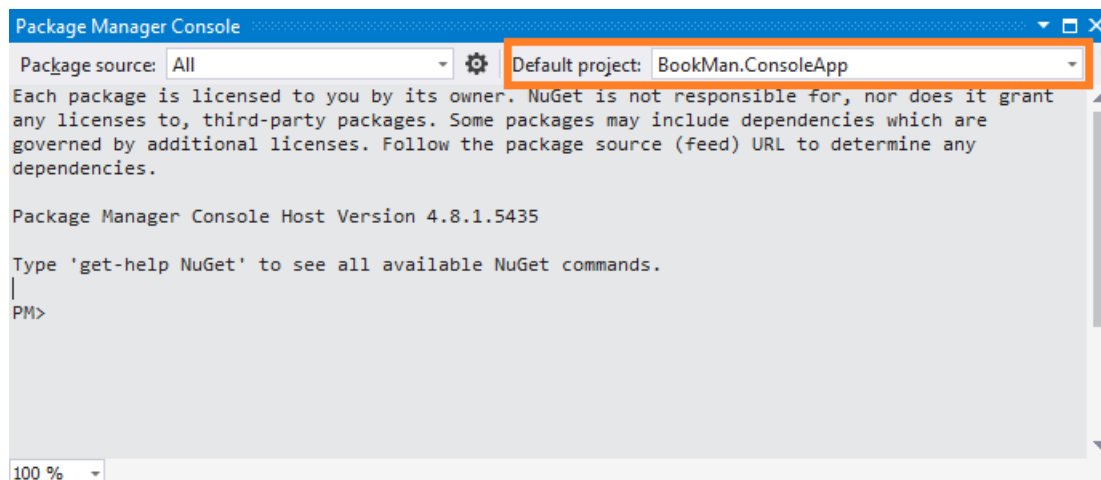
Paket CLI

PM> Install-Package Newtonsoft.Json -Version 12.0.1-beta1



Giao diện tìm kiếm thư viện Newtonsoft.Json trên website

Nếu không nhìn thấy tab Package Manager Console, chọn View => Other Windows => Package Manager Console, hoặc Tools => NuGet Package Manager => Package Manager Console.



Giao diện dòng lệnh của Package Manager Console

Khi sử dụng Package Manager Console lưu ý chọn tham số "Default project" là project mình cần cài đặt gói thư viện.

Lưu ý

Khi sử dụng các gói thư viện trên NuGet cần lưu ý xem xét kỹ sự phụ thuộc của gói thư viện cần dùng.

Lý do là nhiều thư viện trên NuGet sử dụng lẫn nhau, cũng như được xây dựng cho các phiên bản .NET khác nhau.

Khi cài đặt một thư viện mà nó phụ thuộc vào các thư viện khác, các thư viện kia cũng phải được cài đặt theo và phải cài đặt phiên bản mà thư viện chính có thể sử dụng được.

Nếu các gói thư viện có phiên bản mới, NuGet cũng cho phép cập nhật phiên bản đang cài đặt trong dự án lên phiên bản mới. Tuy nhiên, cũng giống như khi cài đặt, phải lưu ý sự phụ thuộc giữa các thư viện trước khi quyết định nâng cấp.

Thực hành 2: thêm chức năng xuất dữ liệu ra file

Bước 1. Bổ sung phương thức RenderToFile

Bổ sung phương thức `RenderToFile` vào lớp `BookListView`

```
1. public void RenderToFile(string path)
2. {
3.     ViewHelp.WriteLine($"Saving data to file '{path}'");
4.     var json = Newtonsoft.Json.JsonConvert.SerializeObject(Model);
5.     System.IO.File.WriteAllText(path, json);
6.     ViewHelp.WriteLine("Done!");
7. }
```

Tương tự, bổ sung phương thức `RenderToFile` vào lớp `BookSingleView`

```
1. public void RenderToFile(string path)
2. {
3.     ViewHelp.WriteLine($"Saving data to file '{path}'");
4.     var json = Newtonsoft.Json.JsonConvert.SerializeObject(Model);
5.     System.IO.File.WriteAllText(path, json);
6.     ViewHelp.WriteLine("Done!");
7. }
```

** để ý thấy rằng hai phương thức này giống hệt nhau*

Bước 2. Điều chỉnh các phương thức của lớp BookController

Điều chỉnh phương thức Single và List của BookController thành dạng như sau:

```
1. /// <summary>
2. /// ghép nối dữ liệu 1 cuốn sách với giao diện hiển thị 1 cuốn sách
3. /// </summary>
4. /// <param name="id">mã định danh của cuốn sách</param>
5. public void Single(int id, string path = "")
6. {
7.     // lấy dữ liệu qua repository
8.     var model = Repository.Select(id);
9.     // khởi tạo view
10.    BookSingleView view = new BookSingleView(model);
11.    // gọi phương thức Render để thực sự hiển thị ra màn hình
12.    if (!string.IsNullOrEmpty(path)) { view.RenderToFile(path); return; }
13.    view.Render();
14. }
15.
16. /// <summary>
17. /// kích hoạt chức năng hiển thị danh sách
18. /// </summary>
```

```

19. public void List(string path = "")
20. {
21.     // lấy dữ liệu qua repository
22.     var model = Repository.Select();
23.     // khởi tạo view
24.     BookListView view = new BookListView(model);
25.     if (!string.IsNullOrEmpty(path)) { view.RenderToFile(path); return; }
26.     view.Render();
27. }

```

Bước 3. Điều chỉnh lớp Program

```

1. private static void Main(string[] args)
2. {
3.     Console.OutputEncoding = System.Text.Encoding.UTF8;
4.
5.     SimpleDataAccess context = new SimpleDataAccess();
6.     BookController controller = new BookController(context);
7.
8.     Router r = Router.Instance;
9.
10.    r.Register("about", About);
11.    r.Register("help", Help);
12.    r.Register(route: "create",
13.        action: p => controller.Create(),
14.        help: "[create]\r\nnhập sách mới");
15.    r.Register(route: "update",
16.        action: p => controller.Update(p["id"].ToInt()),
17.        help: "[update ? id = <value>]\r\ntim và cập nhật sách");
18.    r.Register(route: "list",
19.        action: p => controller.List(),
20.        help: "[list]\r\nhiển thị tất cả sách");
21.    r.Register(route: "single",
22.        action: p => controller.Single(p["id"].ToInt()),
23.        help: "[single ? id = <value>]\r\nhiển thị một cuốn sách theo id");
24.
25.    r.Register(route: "list file",
26.        action: p => controller.List(p["path"]),
27.        help: "[list file ? path = <value>]\r\nhiển thị tất cả sách");
28.    r.Register(route: "single file",
29.        action: p => controller.Single(p["id"].ToInt(), p["path"]),
30.        help: "[single file ? id = <value> & path = <value>]");
31.
32.    while (true)
33.    {
34.        ViewHelp.Write("# Request >>> ", ConsoleColor.Green);
35.        string request = Console.ReadLine();
36.        Router.Instance.Forward(request);
37.
38.        Console.WriteLine();
39.    }
40. }

```

Bước 4. Dịch và chạy thử chương trình

Dịch và chạy thử chương trình với hai lệnh mới

List file ? path = list.json

Single file ? id = 1 & path = single1.json

```

D:\_Temp\BOOKMAN\BookMan.ConsoleApp\bin\Debug\BookMan.Console...
# Request >>> list file ? path = list.json
Saving data to file 'list.json'
Done!

# Request >>> single file ? id = 1 & path = single1.json
Saving data to file 'single1.json'
Done!

# Request >>>

```

Kết quả chạy chương trình

Khi đó trong cùng thư mục với file chạy xuất hiện hai file: list.json và single1.json với nội dung lần lượt như sau:

```
1. [{"Id":1,"Authors":"Unknown author","Title":"A new book 1","Publisher":"Unknown publisher"}]
```

Và

```
1. {"Id":1,"Authors":"Unknown author","Title":"A new book 1","Publisher":"Unknown publisher"}
```

Đây là hai file văn bản định dạng theo kiểu JSON do chương trình tạo ra theo lệnh của người dùng.

Nếu mang đoạn văn bản trên vào file mã nguồn của JavaScript, đoạn văn bản đó là mã nguồn chạy được vì đây chính là các đoạn code để tạo object của ngôn ngữ này!

Đến đây chúng ta có thể thấy, việc bổ sung thêm các tính năng mới cho ứng dụng trở nên rất đơn giản theo quy trình: điều chỉnh/ tạo mới lớp giao diện => điều chỉnh lớp điều khiển => bổ sung phần tử mới cho routes.

Tuy nhiên, trong việc thêm tính năng xuất dữ liệu ra file chúng ta lại gặp một vấn đề: lặp code.

Trong bài trước, chúng ta đã giải quyết tình trạng lặp code bằng cách nhóm các phương thức có chức năng liên quan vào một class mới và chuyển chúng thành các phương thức tĩnh. Tuy nhiên, giải pháp này chỉ áp dụng tốt với các phương thức không sử dụng biến thành viên (tức là không có liên quan đến trạng thái của object).

Đối với các lớp giao diện xuất hiện tình trạng lặp code khác. Hai phương thức trùng nhau RenderToFile ở trên lại bắt buộc phải sử dụng biến thành viên Model.

Ngoài hai phương thức bị lặp này, chúng ta cũng nhận thấy, trong tất cả các lớp giao diện, các thành viên Model, phương thức Render và hàm tạo đều rất tương đồng nhau.

Ở bài tiếp theo chúng ta sẽ học một giải pháp khác để tái sử dụng code giữa các class: [kế thừa](#).

Kết luận

Trong bài học này chúng ta đã học cách sử dụng công cụ NuGet Package Manager để cài đặt bộ thư viện của hãng thứ ba. Chúng ta đã cài đặt thư viện Newtonsoft.Json để thực hiện chức năng xuất dữ liệu ra file.

Trong bài tiếp theo chúng ta sẽ xem xét sử dụng công cụ kế thừa để tái sử dụng code.

- + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
 - + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
 - + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
- Cảm ơn bạn!