Cải tiến view (2): kế thừa, boxing, ép kiểu

Ở bài trước chúng ta đã xây dựng thêm chức năng xuất dữ liệu ra file. Bạn đã nhận thấy có sự trùng lặp code giữa các class. Một trong những giải pháp chúng ta sẽ áp dụng là sử dụng cơ chế kế thừa. Trong bài này bạn sẽ học cách vận dụng kế thừa vào cải tiến các lớp view giúp chống lặp code. Ngoài ra bạn cũng sẽ xem xét thêm về vấn đề boxing/unboxing dữ liệu và cách ép kiểu.

NỘI DUNG CỦA BÀI [Ẩn]

1. Thực hành: áp dụng kế thừa để cải tiến các lớp view
   1.1. Bước 1. Xây dựng lớp ViewBase
   1.2. Bước 2. Điều chỉnh lớp BookListView
   1.3. Bước 3. Điều chỉnh lớp BookSingleView
   1.4. Bước 4. Điều chỉnh lớp BookUpdateView
   1.5. Bước 5. Điều chỉnh lớp BookCreateView
2. Một số kỹ thuật khác
   2.1. Boxing/Unboxing
   2.2. Type casting
3. Kết luận

Thực hành: áp dụng kế thừa để cải tiến các lớp view

Lưu ý: do khối lượng code lớn, từ giờ về sau các ghi chú cũ trước đây sẽ bị xóa bỏ khi đưa vào bài, chỉ giữ lại những ghi chú mới quan trọng để tránh rối.

Bước 1. Xây dựng lớp ViewBase

Tạo file ViewBase.cs trong thư mục Framework cho lớp ViewBase và viết code như sau:

```
using System.IO;

namespace Framework
{
    public class ViewBase
    {
        protected object Model;
        protected Router Router = Router.Instance;

        public ViewBase() { }

        public ViewBase(object model) => Model = model;

        public void RenderToFile(string path)
        {
            ViewHelp.WriteLine($"Saving data to file '{path}'");
            var json = Newtonsoft.Json.JsonConvert.SerializeObject(Model);
            File.WriteAllText(path, json);
            ViewHelp.WriteLine("Done!");
        }
    }
}
```

ViewBase sẽ là lớp cha của tất cả các lớp view khác. Các bước sau đây sẽ lần lượt điều chỉnh để các lớp view đã xây dựng kế thừa từ ViewBase.

Bước 2. Điều chỉnh lớp BookListView

Điều chỉnh code của lớp BookListView như sau:

```
using System;
using System.IO;

namespace BookMan.ConsoleApp.Views
{
    using Framework;
    using Models;

    internal class BookListView : ViewBase
    {
        public BookListView(Book[] model) : base(model) { }

        public void Render()
        {
            if (((Book[])Model).Length == 0)
            {
                ViewHelp.WriteLine("No book found!", ConsoleColor.Yellow);
                return;
            }

            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("THE BOOK LIST");
            Console.ForegroundColor = ConsoleColor.Yellow;

            foreach (Book b in Model as Book[])
            {
                ViewHelp.Write($"[{b.Id}]", ConsoleColor.Yellow);
                ViewHelp.WriteLine($" {b.Title}", b.Reading ? ConsoleColor.Cyan : Consol
            }

            Console.ResetColor();
        }
    }
}
```

Ở bước này chúng ta xóa bỏ phương thức `RenderToFile` và dòng khai báo `protected Book[] Model;` . Phương thức và dòng khai báo này đã được thực hiện ở lớp cha `ViewBase` và lớp `BookListView` đang kế thừa từ lớp cha này.

Ngoài ra, constructor của BookListView cũng gọi tới constructor của lớp ViewBase: `public BookListView(Book[] model) : base(model) { }`

Bước 3. Điều chỉnh lớp BookSingleView

Tương tự, ở bước này chúng ta cũng xóa bỏ phương thức RenderToFile và lời khai báo protected Book Model;.

```
using System;

namespace BookMan.ConsoleApp.Views
{
    using Framework;
    using Models;

    internal class BookSingleView : ViewBase
    {
        public BookSingleView(Book model) : base(model) { }

        public void Render()
        {
            if (Model == null)
            {
                ViewHelp.WriteLine("NO BOOK FOUND. SORRY!", ConsoleColor.Red);
                return;
            }
```

```
19.
20.             ViewHelp.WriteLine("BOOK DETAIL INFORMATION", ConsoleColor.Green);
21.
22.             // chuyển đổi kiểu từ object sang Book, chỉ áp dụng với kiểu class
23.             var model = Model as Book;
24.             Console.WriteLine($"Authors:     {model.Authors}");
25.             Console.WriteLine($"Title:       {model.Title}");
26.             Console.WriteLine($"Publisher:   {model.Publisher}");
27.             Console.WriteLine($"Year:        {model.Year}");
28.             Console.WriteLine($"Edition:     {model.Edition}");
29.             Console.WriteLine($"Isbn:        {model.Isbn}");
30.             Console.WriteLine($"Tags:        {model.Tags}");
31.             Console.WriteLine($"Description: {model.Description}");
32.             Console.WriteLine($"Rating:      {model.Rating}");
33.             Console.WriteLine($"Reading:     {model.Reading}");
34.             Console.WriteLine($"File:        {model.File}");
35.             Console.WriteLine($"File Name:   {model.FileName}");
36.         }
37.     }
38. }
```

Bước 4. Điều chỉnh lớp BookUpdateView

Điều chỉnh tương tự như hai class ở trên.

```
1.  using System;
2.
3.  namespace BookMan.ConsoleApp.Views
4.  {
5.      using Framework;
6.      using Models;
7.
8.      internal class BookUpdateView : ViewBase
9.      {
10.
11.         public BookUpdateView(Book model) : base(model) {}
12.
13.         public void Render()
14.         {
15.             ViewHelp.WriteLine("UPDATE BOOK INFORMATION", ConsoleColor.Green);
16.             // chuyển đổi kiểu từ object sang Book, chỉ áp dụng với kiểu class
17.             var model = Model as Book;
18.
19.             var authors = ViewHelp.InputString("Authors", model.Authors);
20.             var title = ViewHelp.InputString("Title", model.Title);
21.             var publisher = ViewHelp.InputString("Publisher", model.Publisher);
22.             var isbn = ViewHelp.InputString("Isbn", model.Isbn);
23.             var tags = ViewHelp.InputString("Tags", model.Tags);
24.             var description = ViewHelp.InputString("Description", model.Description);
25.             var file = ViewHelp.InputString("File", model.File);
26.             var year = ViewHelp.InputInt("Year", model.Year);
27.             var edition = ViewHelp.InputInt("Edition", model.Edition);
28.             var rating = ViewHelp.InputInt("Rate", model.Rating);
29.             var reading = ViewHelp.InputBool("Reading", model.Reading);
30.         }
31.     }
32. }
```

Bước 5. Điều chỉnh lớp BookCreateView

```
1.  using System;
2.
3.  namespace BookMan.ConsoleApp.Views
4.  {
5.      using Framework;
6.
7.      internal class BookCreateView : ViewBase
8.      {
9.          public BookCreateView() { }
10.
11.         public void Render()
12.         {
13.             ViewHelp.WriteLine("CREATE A NEW BOOK", ConsoleColor.Green);
14.
15.             var title = ViewHelp.InputString("Title");
16.             var authors = ViewHelp.InputString("Authors");
```

```
            var publisher = ViewHelp.InputString("Publisher");
            var year = ViewHelp.InputInt("Year");
            var edition = ViewHelp.InputInt("Edition");
            var tags = ViewHelp.InputString("Tags");
            var description = ViewHelp.InputString("Description");
            var rate = ViewHelp.InputInt("Rate");
            var reading = ViewHelp.InputBool("Reading");
            var file = ViewHelp.InputString("File");
        }
    }
}
```

Trong phần thực hành vừa rồi chúng ta đã xây dựng lớp `ViewBase` chứa biến thành viên `Model` và `Router` (đều đánh dấu là protected), hai hàm tạo và một phương thức public mới `RenderToFile`.

Biến thành viên Router chỉ để tiện lợi hơn khi sử dụng (thay vì phải gõ đầy đủ là Router.Instance) ở các lớp giao diện kế thừa từ ViewBase, bởi vì hầu hết các lớp giao diện về sau này đều có sử dụng đến Router để kích hoạt phương thức của controller.

Biến thành viên Model chính là nơi chứa dữ liệu để về sau có thể hiển thị qua phương thức Render hoặc ghi vào file qua phương thức RenderToFile. Biến này có thể nhận giá trị thông qua một hàm tạo của ViewBase.

Để ý rằng, `Model` có kiểu là `object`. Như chúng ta đã biết ở phần trên, System.Object là kiểu dữ liệu cha của mọi class. Do đó, biến kiểu System.Object có thể tham chiếu tới bất kỳ kiểu dữ liệu nào. Nói cách khác, biến Model ở đây có thể nhận giá trị là object của bất kỳ class nào.

Như bạn đã biết, các kiểu dữ liệu của C# cũng là kiểu dữ liệu của .NET. Tuy nhiên, C# cung cấp một số từ khóa kiểu để giúp đơn giản hóa việc sử dụng các kiểu dữ liệu của .NET. Do đó `object` tương đương với `System.Object`, `int` tương đương với `System.Int32`, `bool` tương đương với `System.Boolean`.

> C# khuyến khích sử dụng từ khóa kiểu thay cho kiểu .NET.

Một số kỹ thuật khác

Boxing/Unboxing

Tuy rằng biến thuộc kiểu `Object` có thể nhận giá trị thuộc bất kỳ kiểu nào, có sự khác biệt quan trọng giữa đối tượng kiểu tham chiếu (reference type) và kiểu giá trị (value type). Nếu biến thuộc kiểu `Object` nhận đối tượng thuộc kiểu giá trị sẽ xảy ra một quá trình gọi là *boxing*. Quá trình ngược lại (chuyển đổi từ object về kiểu cụ thể) được gọi là *unboxing*.

Khi một số nguyên, số thực, logic (nói chung thuộc nhóm kiểu giá trị) được gán cho một biến kiểu `Object`, .NET tạo một object mới trong vùng heap và để biến kiểu này trỏ vào đó. Quá trình này gọi là boxing.

Như vậy, boxing là quá trình chuyển đổi kiểu giá trị thành kiểu tham chiếu. Quá trình này phức tạp và hiệu suất thấp, vì vậy nên hạn chế sử dụng boxing. Ở các bài sau chúng ta sẽ đưa ra một giải pháp khác.

Ở lớp BookSingleView, BookListView, BookUpdateView, trước khi có thể sử dụng biến Model trong các lệnh, chúng ta đều phải thực hiện quá trình unboxing này thông qua phép toán ép kiểu (type casting).

Type casting

Trong phần thực hành trên bạn gặp dạng chuyển đổi kiểu *type casting*, thường gọi là *ép kiểu* cho dễ phân biệt.

Casting là quá trình chuyển đổi Kiểu của một biến nhưng không làm biến đổi giá trị của biến đó.

> Để phân biệt, conversion làm biến đổi giá trị cùng với kiểu. Khi chuyển đổi từ chuỗi "12345" thành số 12345, bản thân giá trị đã thay đổi cùng với kiểu. Đây là dạng conversion. Trong một số bài thực hành trước đây bạn đã gặp dạng chuyển đổi type conversion, cụ thể là chuyển đổi từ chuỗi ký tự về số và logic.

Unboxing ở trên là một quá trình kết hợp giữa tạo một giá trị mới trong stack, copy dữ liệu từ heap vào đó và ép kiểu (casting).

C# có hai cú pháp khác nhau để ép kiểu:

1. Đối với cả hai loại kiểu (giá trị và tham chiếu): có thể sử dụng lệnh `(Book) Model` để ép kiểu về `Book`. Trong cách này, nếu ép kiểu không thành công sẽ gây lỗi.
2. Riêng đối với kiểu tham chiếu: có thể sử dụng lệnh `Model as Book`. Cách sử dụng này có lợi thế hơn ở chỗ, nếu việc ép kiểu bị lỗi (không ép kiểu được), biến đích nhận giá trị null và không gây lỗi.

Kết luận

Trong bài học này chúng ta đã học về khái niệm kế thừa và một số đặc điểm của kế thừa trong C#. Chúng ta cũng đã bước đầu vận dụng kỹ thuật kế thừa cơ bản để cải tiến các lớp view nhằm tránh lặp code.

Trong bài tiếp theo chúng ta sẽ xem xét chi tiết hơn các vấn đề khác của kế thừa và cách áp dụng để tiếp tục cải tiến các lớp view.

> + 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!