

Cải tiến view (3): che giấu, ghi đè, kế thừa và generic

Hướng dẫn tự học lập trình C# toàn tập > Cải tiến view (3): che giấu, ghi đè, kế thừa và generic

Trong bài học này chúng ta sẽ xem xét khái niệm và kỹ thuật ghi đè, che giấu phương thức, và cách sử dụng lớp generic trong kế thừa. Chúng ta sẽ vận dụng để tiếp tục cải tiến các lớp view. Ngoài ra, chúng ta sẽ tiếp tục xem xét một số vấn đề khác của kế thừa trước khi đi vào cải tiến các lớp giao diện vận dụng các kỹ thuật mới.

Để hiểu được bài thực hành này, bạn cần nắm được các vấn đề liên quan đến [kế thừa](#) (cụ thể là vấn đề che giấu và ghi đè thành viên) và [generic](#).

NỘI DUNG CỦA BÀI [Ấn]

1. Nhắc lại quan hệ giữa kế thừa và đa hình
2. Thực hành 1: cải tiến lớp ViewBase sử dụng ghi đè
 - 2.1. Bước 1. Thay đổi lớp ViewBase
 - 2.2. Bước 2. Điều chỉnh khai báo của phương thức Render
 - 2.3. Bước 3. Xây dựng lớp ControllerBase
 - 2.4. Bước 4. Điều chỉnh lớp BookController
3. Thực hành 2: kết hợp kế thừa và generic
 - 3.1. Bước 1. Điều chỉnh file ViewBase.cs
 - 3.2. Bước 2. Điều chỉnh các lớp view
 - 3.3. Bước 3. Điều chỉnh lớp ControllerBase
 - 3.4. Bước 4. Dịch và chạy thử chương trình với tất cả các lệnh đã biết
4. Kết luận

Nhắc lại quan hệ giữa kế thừa và đa hình

Trong lập trình hướng đối tượng, kế thừa và đa hình là hai nguyên lý khác nhau.

Đa hình thiết lập mối quan hệ "**là**" (is-a relationship) giữa kiểu cơ sở và kiểu dẫn xuất. Ví dụ, nếu chúng ta có lớp cơ sở Bird và lớp dẫn xuất Parrot thì một object của Parrot cũng là object của Bird, kiểu Parrot cũng là kiểu Bird (đương nhiên rồi, vẹt là chim mà!). Mối quan hệ này nhìn rất giống như quan hệ kế thừa ở trên.

Trong khi đó, *kế thừa* liên quan chủ yếu đến **tái sử dụng code**: code của lớp con thừa hưởng code của lớp cha. Một cách nói khác, đa hình liên quan tới quan hệ về ngữ nghĩa, còn kế thừa liên quan tới cú pháp.

Trong các ngôn ngữ như C++, C#, Java, hai khái niệm này hầu như được đồng nhất, thể hiện ở chỗ:

1. class con thừa hưởng các thành viên của class cha (kế thừa, tái sử dụng code);
2. một object thuộc kiểu con có thể gán cho biến thuộc kiểu cha, tức là kiểu cơ sở có thể dùng để thay thế cho kiểu dẫn xuất (đa hình).

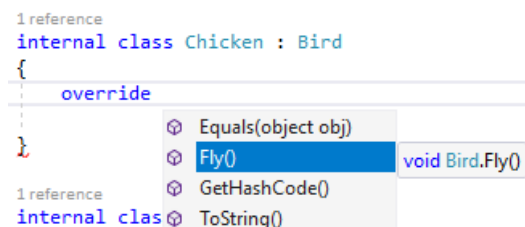
Để áp dụng được cơ chế ghi đè, cả lớp cha và lớp con cần phải phối hợp:

1. lớp cha phải cho phép phương thức được phép ghi đè bằng cách thêm từ khóa `virtual` trước khai báo phương thức;
2. lớp con phải thông báo rõ việc ghi đè bằng cách thêm từ khóa `override` trước định nghĩa phương thức.

Mặc định các phương thức của class không cho ghi đè mà chỉ cho phép che giấu.

Tuy nhiên, các phương thức `Equals`, `GetHashCode`, `ToString` của lớp tổ tiên `System.Object` đều cho phép ghi đè ở lớp hậu duệ.

Để xác định những phương thức nào cho phép ghi đè, chỉ cần viết từ khóa `override` trong thân class (bên ngoài phương thức).



Ghi đè (override) phương thức

Che dấu được sử dụng chủ yếu để đảm bảo tương thích ngược giữa các class. Cơ chế này không được sử dụng nhiều trong thực tế.

Ở phía khác, ghi đè được sử dụng rất phổ biến cùng với đa hình giúp tạo ra một class đại diện cho các biến thể khác nhau.

Thực hành 1: cải tiến lớp `ViewBase` sử dụng ghi đè

Bước 1. Thay đổi lớp `ViewBase`

```
1. public class ViewBase
2. {
3.     protected object Model;
4.     protected Router Router = Router.Instance;
5.
6.     public ViewBase() { }
7.     public ViewBase(object model) => Model = model;
8.     // bổ sung phương thức virtual Render, cho phép ghi đè
9.     public virtual void Render() { }
10.    // chuyển phương thức RenderToFile sang virtual
11.    public virtual void RenderToFile(string path)
12.    {
13.        ViewHelp.WriteLine($"Saving data to file '{path}'");
14.        var json = Newtonsoft.Json.JsonConvert.SerializeObject(Model);
15.        File.WriteAllText(path, json);
16.        ViewHelp.WriteLine("Done!");
17.    }
18. }
```

Ở bước này, chúng ta bổ sung phương thức `Render` với đánh dấu `virtual` để các lớp con có thể ghi đè phương thức này. Chúng ta cũng chuyển phương thức `RenderToFile` sang

`virtual` để các lớp con nếu cần có thể ghi đè (ví dụ, để xuất sang một định dạng khác như xml hoặc plain text).

Bước 2. Điều chỉnh khai báo của phương thức Render

Thay đổi phương thức Render trên cả 4 lớp view như sau:

```
1. public override void Render() ...
```

Bước điều chỉnh này chỉ đơn giản là thêm từ khóa `override` vào trước khai báo của phương thức `Render` trong từng lớp giao diện.

Bước 3. Xây dựng lớp ControllerBase

Tạo file `ControllerBase.cs` trong thư mục Framework cho lớp `ControllerBase` với code như sau:

```
1. namespace Framework
2. {
3.     public class ControllerBase
4.     {
5.         public virtual void Render(ViewBase view, string path = "", bool both = false)
6.         {
7.             if (string.IsNullOrEmpty(path)) { view.Render(); return; }
8.
9.             if (both)
10.            {
11.                view.Render();
12.                view.RenderToFile(path);
13.                return;
14.            }
15.            view.RenderToFile(path);
16.        }
17.    }
18. }
```

Lớp `ControllerBase` định nghĩa một phương thức `Render` giúp gọi tới phương thức `Render` hoặc `RenderToFile` của các lớp view một cách tiện lợi.

Bước 4. Điều chỉnh lớp BookController

```
1. namespace BookMan.ConsoleApp.Controllers
2. {
3.     using DataServices;
4.     using Framework;
5.     using Views;
6.
7.     internal class BookController : ControllerBase
8.     {
9.         protected Repository Repository;
10.
11.         public BookController(SimpleDataAccess context)
12.         {
13.             Repository = new Repository(context);
14.         }
15.
16.         public void Single(int id, string path = "")
17.         {
18.             var model = Repository.Select(id);
19.             Render(new BookSingleView(model), path);
20.         }
21.
22.         public void Create()
23.         {
24.             Render(new BookCreateView());
25.         }
26.     }
```

```

27.     public void List(string path = "")
28.     {
29.         var model = Repository.Select();
30.         Render(new BookListView(model), path);
31.     }
32.
33.     public void Update(int id)
34.     {
35.         var model = Repository.Select(id);
36.         Render(new BookUpdateView(model));
37.     }
38. }
39. }

```

Trong điều chỉnh này chúng ta vận dụng phương thức `Render` kế thừa từ `ControllerBase` giúp đơn giản hóa làm việc với các lớp view.

Có thể để ý thấy rằng, phương thức `Render` kế thừa từ `ControllerBase` yêu cầu kiểu đầu vào là `ViewBase` nhưng chúng ta có thể cung cấp bất kỳ object nào của các lớp view kế thừa từ `ViewBase`. Điều này đạt được nhờ cơ chế đa hình kết hợp kế thừa mà chúng ta đã xem xét ở phần lý thuyết trên.

Do cơ chế ghi đè, phương thức `Render` được gọi không phải là `Render` của lớp cha `ViewBase` mà là phương thức `Render` của từng lớp con. Chúng ta có thể thấy ghi đè là cơ chế rất mạnh giúp chúng ta chỉ cần viết code một lần cho kiểu cha nhưng có thể vận dụng cho các kiểu con và sử dụng được những đặc thù riêng của kiểu con. Nhờ cơ chế này chúng ta không cần viết code xử lý cho từng lớp con cụ thể.

Thực hành 2: kết hợp kế thừa và generic

Bước 1. Điều chỉnh file `ViewBase.cs`

```

1.  using System.IO;
2.
3.  namespace Framework
4.  {
5.      public class ViewBase
6.      {
7.          protected Router Router = Router.Instance;
8.
9.          public ViewBase() { }
10.
11.         public virtual void Render() { }
12.     }
13.
14.     public class ViewBase<T> : ViewBase
15.     {
16.         protected T Model;
17.         public ViewBase(T model) => Model = model;
18.
19.         public virtual void RenderToFile(string path)
20.         {
21.             ViewHelp.WriteLine($"Saving data to file '{path}'");
22.             var json = Newtonsoft.Json.JsonConvert.SerializeObject(Model);
23.             File.WriteAllText(path, json);
24.             ViewHelp.WriteLine("Done!");
25.         }
26.     }
27. }

```

Ở bước này trong file `ViewBase.cs` chúng ta tách một phần lớp `ViewBase` ra thành một lớp generic riêng `ViewBase<T>` và cho lớp này kế thừa từ `ViewBase`.

Trong lớp `ViewBase<T>` sẽ tập trung những phương thức phải sử dụng thông tin từ model. Kiểu dữ liệu của model sẽ được quyết định khi các lớp con kế thừa từ lớp `ViewBase<T>`.

Bước 2. Điều chỉnh các lớp view

Lớp `BookListView`

```
1. using System;
2.
3. namespace BookMan.ConsoleApp.Views
4. {
5.     using Framework;
6.     using Models;
7.
8.     internal class BookListView : ViewBase<Book[]>
9.     {
10.         public BookListView(Book[] model) : base(model) { }
11.
12.         public override void Render()
13.         {
14.             if (Model.Length == 0)
15.             {
16.                 ViewHelp.WriteLine("No book found!", ConsoleColor.Yellow);
17.                 return;
18.             }
19.
20.             Console.ForegroundColor = ConsoleColor.Green;
21.             Console.WriteLine("THE BOOK LIST");
22.             Console.ForegroundColor = ConsoleColor.Yellow;
23.
24.             foreach (Book b in Model)
25.             {
26.                 ViewHelp.Write($"[{b.Id}] ", ConsoleColor.Yellow);
27.                 ViewHelp.WriteLine($" {b.Title}", b.Reading ? ConsoleColor.Cyan : Console
28.             }
29.
30.             ViewHelp.WriteLine($" {Model.Length} item(s)", ConsoleColor.Green);
31.         }
32.     }
33. }
```

Lớp `BookSingleView`

```
1. using System;
2.
3. namespace BookMan.ConsoleApp.Views
4. {
5.     using Framework;
6.     using Models;
7.
8.     internal class BookSingleView : ViewBase<Book>
9.     {
10.         public BookSingleView(Book model) : base(model) { }
11.
12.         public override void Render()
13.         {
14.             if (Model == null)
15.             {
16.                 ViewHelp.WriteLine("NO BOOK FOUND. SORRY!", ConsoleColor.Red);
17.                 return;
18.             }
19.
20.             ViewHelp.WriteLine("BOOK DETAIL INFORMATION", ConsoleColor.Green);
21.
22.             Console.WriteLine($"Authors:      {Model.Authors}");
23.             Console.WriteLine($"Title:      {Model.Title}");
24.             Console.WriteLine($"Publisher:  {Model.Publisher}");
25.             Console.WriteLine($"Year:      {Model.Year}");
26.             Console.WriteLine($"Edition:   {Model.Edition}");
27.             Console.WriteLine($"Isbn:      {Model.Isbn}");
28.             Console.WriteLine($"Tags:      {Model.Tags}");
29.             Console.WriteLine($"Description: {Model.Description}");
30.             Console.WriteLine($"Rating:    {Model.Rating}");
31.         }
32.     }
33. }
```

```

31.         Console.WriteLine($"Reading: {Model.Reading}");
32.         Console.WriteLine($"File: {Model.File}");
33.         Console.WriteLine($"File Name: {Model.FileName}");
34.     }
35. }
36. }

```

Lớp BookUpdateView

```

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

```

Chúng ta thấy, lớp generic cũng có thể đóng vai trò lớp cha để tạo ra các lớp con.

Trong bước này chúng ta điều chỉnh để các lớp view (trừ lớp `BookCreateView` không cần dữ liệu từ `BookController`) kế thừa từ lớp cha generic `ViewBase<T>`, trong đó `T` được thay thế bằng các kiểu dữ liệu mà view chờ đợi từ controller: `BookListView` chờ đợi một mảng `Book[]`, `BookSingleView` và `BookUpdateView` chờ đợi một object kiểu `Book`.

Do đó, `Book[]` và `Book` được sử dụng cho vị trí của `T` khi dùng `ViewBase<T>` làm lớp cơ sở. Các lớp con kế thừa từ `ViewBase<T>` trong trường hợp này lại không phải là lớp generic nữa, vì chúng ta đã cung cấp kiểu dữ liệu cụ thể cho lớp cha trước khi cho lớp con kế thừa.

Lưu ý rằng `ViewBase` và `ViewBase<T>` nhìn tương tự nhau nhưng đây là hai lớp khác nhau (thể hiện rằng chúng có thể kế thừa nhau). Thông thường các lớp non-generic và generic hay đi thành cặp với nhau.

Bước 3. Điều chỉnh lớp ControllerBase

```

1.  namespace Framework
2.  {
3.
4.      public class ControllerBase
5.      {
6.          public virtual void Render(ViewBase view) { view.Render(); }
7.
8.          public virtual void Render<T>(ViewBase<T> view, string path = "", bool both = fa

```

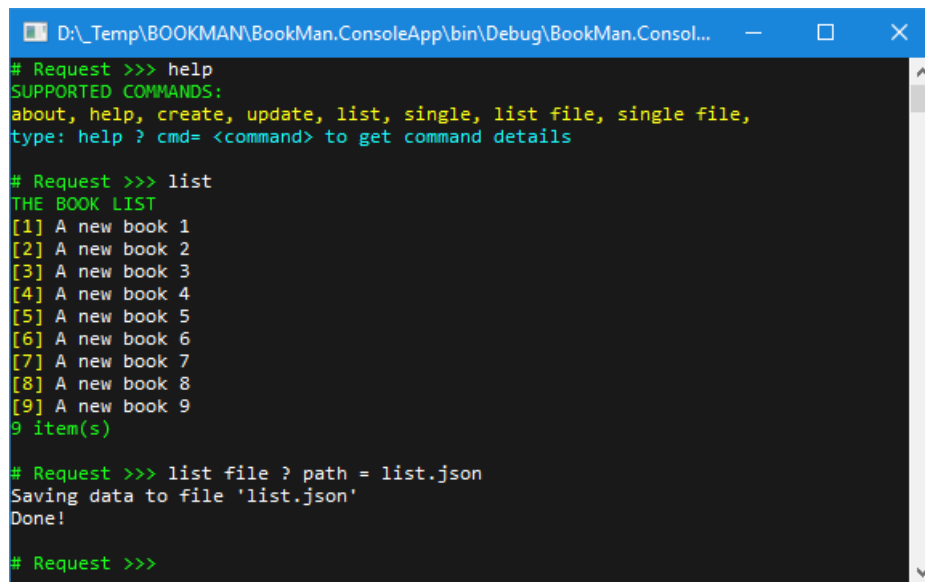
```

9.         {
10.             if (string.IsNullOrEmpty(path)) { view.Render(); return; }
11.
12.             if (both)
13.             {
14.                 view.Render();
15.                 view.RenderToFile(path);
16.                 return;
17.             }
18.
19.             view.RenderToFile(path);
20.         }
21.     }
22. }

```

Bước điều chỉnh này giúp `ControllerBase` “thích nghi” với hai lớp `ViewBase` và `ViewBase<T>`. Phương thức `Render` thứ nhất chỉ làm việc với object kiểu `ViewBase`; phương thức thứ hai thuộc loại generic và làm việc với object của `ViewBase<T>`.

Bước 4. Dịch và chạy thử chương trình với tất cả các lệnh đã biết



```

# Request >>> help
SUPPORTED COMMANDS:
about, help, create, update, list, single, list file, single file,
type: help ? cmd= <command> to get command details

# Request >>> list
THE BOOK LIST
[1] A new book 1
[2] A new book 2
[3] A new book 3
[4] A new book 4
[5] A new book 5
[6] A new book 6
[7] A new book 7
[8] A new book 8
[9] A new book 9
9 item(s)

# Request >>> list file ? path = list.json
Saving data to file 'list.json'
Done!

# Request >>>

```

Kết quả thực hiện chương trình

Kết luận

Trong bài này chúng ta đã xem xét hai vấn đề quan trọng liên quan tới kế thừa: ghi đè và che giấu phương thức. Chúng ta cũng áp dụng các kỹ thuật này 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!