Cải tiến view (4): lớp trừu tượng, phương thức trừu tượng

Trong bài học này chúng ta sẽ xem xét vấn đề cuối cùng có liên quan đến kế thừa: lớp trừu tượng và phương thức trừu tượng. Chúng ta cũng sẽ vận dụng kỹ thuật này để hoàn thiện tất cả các lớp view hiện có.

NỘI DUNG CỦA BÀI [Ẩn]

Thực hành 1: Cải tiến lớp ViewBase và ViewBase<T> thành lớp abstract

```
using System.IO;

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

        public ViewBase() { }

        public abstract void Render();
    }

    public abstract class ViewBase<T> : ViewBase
    {
        protected T Model;
        public ViewBase(T model) => Model = model;

        public virtual 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!");
        }
    }
}
```

Khi xây dựng lớp ViewBase, bạn không biết được người sử dụng sẽ render object nào. Nói cách khác, phần thân của Render sẽ do class kế thừa nó quyết định thông qua cơ chế ghi đè. Do vậy, trong phần cải tiến trên bạn chuyển phương thức Render thành phương thức abstract. Cũng do đó, lớp ViewBase cũng phải chuyển thành lớp abstract.

Bởi vì lớp `ViewBase<T>` kế thừa `ViewBase` nhưng lại không ghi đè phương thức `Render` nên cũng phải đặt là *`abstract`*.

Cơ chế này tạo ra một "hợp đồng" ràng buộc giữa các lớp view và `ViewBase` / `ViewBase<T>`: Từ bây giờ, mọi class kế thừa `ViewBase` và `ViewBase<T>` đều bắt buộc phải ghi đè phương

thức `Render`. Visual Studio sẽ thông báo lỗi nếu không nhìn thấy phương thức ghi đè của Render trong các lớp dẫn xuất.

Trong phần thực hành trước chúng ta đã ghi đè phương thức `Render` trong tất cả các lớp dẫn xuất của `ViewBase` và `ViewBase<T>` nên không cần điều chỉnh ở các lớp này.

Thực hành 2: Xây dựng nhóm class hỗ trợ gửi thông báo ra console

Bước 1. Xây dựng mới lớp Message

Tạo file mã nguồn mới Message.cs trong thư mục Framework cho lớp `Message` và thêm code như sau:

```
using System;

namespace Framework
{
    public enum MessageType { Success, Information, Error, Confirmation }
    public class Message
    {
        public MessageType Type { get; set; } = MessageType.Success;
        public string Label { get; set; }
        public string Text { get; set; } = "Your action has completed successfully";
        public string BackRoute { get; set; }
    }
    public class MessageView : ViewBase<Message>
    {
        public MessageView(Message model) : base(model)
        {
        }

        public override void Render()
        {
            switch (Model.Type)
            {
                case MessageType.Success:
                    ViewHelp.WriteLine(Model.Label != null ? Model.Label.ToUpper() : "SU
                    break;
                case MessageType.Error:
                    ViewHelp.WriteLine(Model.Label != null ? Model.Label.ToUpper() : "ER
                    break;
                case MessageType.Information:
                    ViewHelp.WriteLine(Model.Label != null ? Model.Label.ToUpper() : "IN
                    break;
                case MessageType.Confirmation:
                    ViewHelp.WriteLine(Model.Label != null ? Model.Label.ToUpper() : "CO
                    break;
            }
            if (Model.Type != MessageType.Confirmation)
                ViewHelp.WriteLine(Model.Text, ConsoleColor.White);
            else
            {
                ViewHelp.Write(Model.Text, ConsoleColor.Magenta);
                var answer = Console.ReadLine().ToLower();
                if (answer == "y" || answer == "yes")
                    Router.Forward(Model.BackRoute);
            }
        }
    }
}
```

Nhiệm vụ của lớp `Message` và `MessageView` là gửi các thông báo ngắn từ controller tới người dùng.

Trong quá trình sử dụng giao diện về sau, mỗi hành động thành công hoặc thất bại đều phải được thông báo trở lại cho người dùng.

Ví dụ, khi cập nhật thành công một cuốn sách sẽ phải thông báo trở lại cho người dùng, trước khi muốn xóa một cuốn sách phải hỏi ý kiến người dùng, v.v..

Tất cả các thông báo này đều tương đối độc lập với nghiệp vụ của bài toán quản lý và có thể tái sử dụng qua nhiều dự án khác. Ngoài ra, cơ chế thông báo này cũng giúp hạn chế phải viết những lớp view nhỏ lẻ chỉ để hiển thị một vài thông báo.

Khi nhập code cho lớp `MessageView` có thể để ý, ngay sau khi gõ

```
public class MessageView : ViewBase<Message>
{
}
```

Visual Studio sẽ hiện thông báo lỗi (gạch chân đỏ ở tên lớp `MessageView`). Nếu sử dụng Quick Action => Implement abstract class, Visual Studio nhanh chóng giúp tạo ra cái khung của phương thức cần ghi đè `Render`:

```
public override void Render()
{
      throw new NotImplementedException();
}
```

Nếu dùng Quick Action => Generate constructor sẽ sinh ra luôn hộ chúng ta hàm tạo

```
public MessageView(Message model) : base(model)
{
}
```

Bước 2. Điều chỉnh lớp ControllerBase

```
namespace Framework
{
    public class ControllerBase
    {
        public virtual void Render(ViewBase view) { view.Render(); }

        public virtual void Render<T>(ViewBase<T> view, string path = "", bool both = fa
        {
            if (string.IsNullOrEmpty(path)) { view.Render(); return; }

            if (both)
            {
                view.Render();
                view.RenderToFile(path);
                return;
            }

            view.RenderToFile(path);
        }

        public virtual void Render(Message message) => Render(new MessageView(message));

        public virtual void Success(string text, string label = "SUCCESS") => Render(new

        public virtual void Inform(string text, string label = "INFORMATION") => Render(

        public virtual void Error(string text, string label = "ERROR!") => Render(new Me

        public virtual void Confirm(string text, string route, string label = "CONFIRMAT
    }
}
```

Bước điều chỉnh này bổ sung cho lớp `ControllerBase` thêm một số phương thức tiện ích hỗ trợ cho việc hiển thị các loại thông báo cho người dùng. Các phương thức tiện ích này sẽ được sử dụng nhiều trong bài tiếp theo khi chúng ta hoàn thiện các chức năng của chương trình ứng dụng.

Kết luận

Trong bài này chúng ta đã vận dụng lớp trừu tượng để xây dựng hoàn thiện các lớp view (BookSingleView, BookListView, BookCreateView, BookUpdateView) cũng như xây dựng thêm một cặp class mới giúp hiển thị các thông báo ngắn (Message, MessageView).

Đến giai đoạn này, tất cả các lớp hỗ trợ của ứng dụng đã hoàn thiện. Trong bài tiếp theo, chúng ta sẽ sử dụng các lớp hỗ trợ này để hoàn thiện các chức năng hiện có cũng như xây dựng bổ sung thêm một số chức năng như đã phân tích.

> + 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!