Router (3): sử dụng ủy nhiệm hàm

Trong bài học này chúng ta sẽ làm quen với ủy nhiệm hàm (delegate) và hoàn thiện lớp Router.

NỘI DUNG CỦA BÀI [Ẩn]

Thực hành: hoàn thiện lớp Router

Bước 1. Viết code cho lớp Router

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Framework
{
    /* đây không phải là lệnh sử dụng không gian tên
     * mà là tạo biệt danh cho một kiểu dữ liệu
     * ở đây đang tạo một biệt danh cho kiểu Dictionary<string, ControllerAction>.
     * trong cả file này có thể sử dụng tên kiểu RoutingTable
     * thay cho Dictionary<string, ControllerAction>
     * Lưu ý rằng khai báo này nằm trực tiếp trong namespace
     */
    using RoutingTable = Dictionary<string, ControllerAction>;

    // Lưu ý khai báo delegate này là khai báo kiểu, nằm trong namespace
    /// <summary>
    /// delegate này đại diện cho tất cả các phương thức có:
    /// - kiểu ra là void,
    /// - danh sách tham số vào là (Parameter)
    /// </summary>
    /// <param name="parameter"></param>
    public delegate void ControllerAction(Parameter parameter = null);

    /// <summary>
    /// lớp cho phép ánh xạ truy vấn với phương thức
    /// </summary>
    public class Router
    {
        // nhóm 3 lệnh dưới đây biến Router thành một singleton
        private static Router _instance;
        private Router()
        {
            _routingTable = new RoutingTable();
            _helpTable = new Dictionary<string, string>();
        }

        // để ý: constructor là private
        // người sử dụng class thông qua property này để truy xuất các phương thức của c
        // chỉ khi nào _instance == null mới tạo object. Một khi đã tạo object, _instanc
        // không có giá trị null nữa.
        // vì là biến static, _instance một khi được khởi tạo sẽ tồn tại suốt chương trì
        public static Router Instance => _instance ?? (_instance = new Router());

```

```
        // lưu ý: ở đây đang sử dụng alias của Dictionary<string, ControllerAction> cho
        private readonly RoutingTable _routingTable;
        private readonly Dictionary<string, string> _helpTable;

        public string GetRoutes()
        {
            StringBuilder sb = new StringBuilder();
            foreach (var k in _routingTable.Keys)
                sb.AppendFormat("{0}, ", k);
            return sb.ToString();
        }

        public string GetHelp(string key)
        {
            if (_helpTable.ContainsKey(key))
                return _helpTable[key];
            else
                return "Documentation not ready yet!";
        }

        /// <summary>
        /// đăng ký một route mới, mỗi route ánh xạ một chuỗi truy vấn với một phương th
        /// </summary>
        /// <param name="route"></param>
        /// <param name="action"></param>
        public void Register(string route, ControllerAction action, string help = "")
        {
            // nếu _routingTable đã chứa route này thì bỏ qua
            if (!_routingTable.ContainsKey(route))
            {
                _routingTable[route] = action;
                _helpTable[route] = help;
            }
        }

        /// <summary>
        /// phân tích truy vấn và gọi phương thức tương ứng với chuỗi truy vấn
        /// <para>chuỗi truy vấn bao gồm hai phần: route và parameter, phân tách bởi ký
        /// </summary>
        /// <param name="request">chuỗi truy vấn, bao gồm hai phần:
        /// route, paramete; phân tách bởi ký tự ?</param>
        public void Forward(string request)
        {
            var req = new Request(request);
            if (!_routingTable.ContainsKey(req.Route))
                throw new Exception("Command not found!");
            if (req.Parameter == null)
                _routingTable[req.Route]?.Invoke();
            else
                _routingTable[req.Route]?.Invoke(req.Parameter);
        }

        // Code của lớp Request (làm trong buổi trước) nằm ở đây và tạm ẩn đi cho gọn
    }
}
```

Bước 2. Điều chỉnh code của lớp Program

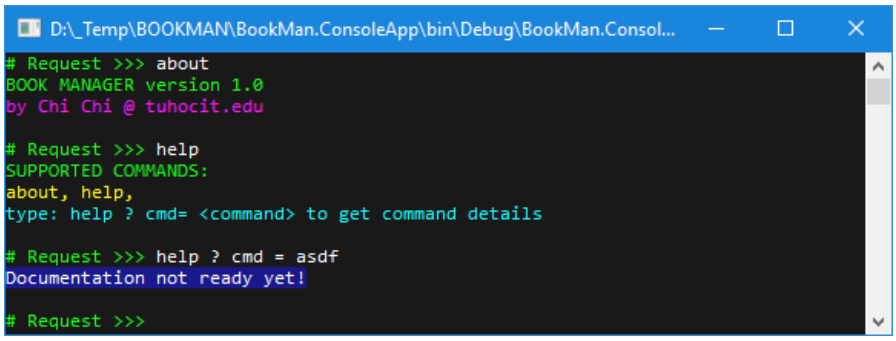
Điều chỉnh code của lớp Program (file Program.cs) như sau:

```
using System;
namespace BookMan.ConsoleApp
{
    using Controllers;
    using Framework;
    using DataServices;
    internal class Program
    {
        private static void Main(string[] args)
        {
            Console.OutputEncoding = System.Text.Encoding.UTF8;

            var context = new SimpleDataAccess();
            BookController controller = new BookController(context);

            Router.Instance.Register("about", About);
```

```
            Router.Instance.Register("help", Help);

            while (true)
            {
                ViewHelp.Write("# Request >>> ", ConsoleColor.Green);
                string request = Console.ReadLine();

                Router.Instance.Forward(request);

                Console.WriteLine();
            }
        }

        private static void About(Parameter parameter)
        {
            ViewHelp.WriteLine("BOOK MANAGER version 1.0", ConsoleColor.Green);
            ViewHelp.WriteLine("by ChiChi@TuHocIct.com", ConsoleColor.Magenta);
        }

        private static void Help(Parameter parameter)
        {
            if (parameter == null)
            {
                ViewHelp.WriteLine("SUPPORTED COMMANDS:", ConsoleColor.Green);
                ViewHelp.WriteLine(Router.Instance.GetRoutes(), ConsoleColor.Yellow);
                ViewHelp.WriteLine("type: help ? cmd= <command> to get command details",
                return;
            }
            Console.BackgroundColor = ConsoleColor.DarkBlue;
            var command = parameter["cmd"].ToLower();
            ViewHelp.WriteLine(Router.Instance.GetHelp(command));

        }
    }
}
```

Bước 3. Dịch và chạy thử chương trình

Dịch và chạy thử chương trình với lệnh `about` và `help`



Kết quả chạy chương trình

Phân tích code

Delegate

Trong phần thực hành trên chúng ta gặp một dạng khai báo:

```
/// <summary>
    /// delegate này đại diện cho tất cả các phương thức có:
    /// - kiểu ra là void,
    /// - danh sách tham số vào là (Router.Parameter)
    /// </summary>
    /// <param name="parameter"></param>
```

```
7.      public delegate void ControllerAction(Parameter parameter = null);
```

Đây là định nghĩa một kiểu dữ liệu *ủy nhiệm* (delegate).

Ví dụ, ở phần thực hành trên chúng ta đã định nghĩa một kiểu Ủy nhiệm:

```
1.  public delegate void ControllerAction(Parameter parameter = null);
```

Lệnh này định nghĩa một kiểu Ủy nhiệm tên là `ControllerAction`. Kiểu `ControllerAction` này dùng để tạo ra các biến chứa tham chiếu tới tất cả các phương thức có tham số đầu vào thuộc kiểu `Parameter` và không trả về dữ liệu. Nói theo cách khác, tất cả các phương thức có tham số đầu vào thuộc kiểu `Parameter` và không trả về dữ liệu đều có thể gán cho biến thuộc kiểu `ControllerAction`.

Biệt danh (alias) của kiểu dữ liệu

Kiểu `RoutingTable` bạn nhìn thấy trong phần đầu của file code thực chất chỉ là một biệt danh (alias) của một `Dictionary` với khóa kiểu `string` và giá trị thuộc kiểu `ControllerAction`.

```
1.  using RoutingTable = Dictionary<string, ControllerAction>;
2.  public delegate void ControllerAction(Parameter parameter = null);
3.
4.  private readonly RoutingTable _routingTable;
```

Dictionary này được sử dụng để khai báo ra biến `_routingTable` nhằm chứa những cặp route / phương thức. Trong đó, phương thức phải tuân thủ theo định nghĩa của `ControllerAction`.

Mỗi cặp này được sử dụng để ánh xạ một route tới một phương thức cụ thể. Các phương thức này bắt buộc phải tiếp nhận chuỗi tham số của người dùng làm tham số đầu vào. Ở trong mỗi phương thức này sẽ tách các tham số đó ra và sử dụng để gọi tới một action tương ứng của controller.

Vai trò của delegate trong Router

Lớp Router của chúng ta có nhiệm vụ lưu lại một danh sách các phương thức tương ứng với chuỗi truy vấn của người dùng. Về sau, khi người dùng nhập một truy vấn nào đó, phương thức tương ứng sẽ được thực hiện. Mỗi phương thức lưu trong Router sẽ tiếp tục gọi một phương thức (action) của controller. Nhờ đó, chúng ta có thể ánh xạ mỗi truy vấn của người dùng tới một action của controller.

Tuy nhiên, khi xây dựng lớp Router chúng ta chưa xác định được các phương thức sẽ lưu trong nó, cũng như chưa thể xác định hết các phương thức (action) của controller.

Trong tương lai, khi bổ sung thêm các chức năng mới, danh sách phương thức lưu trữ trong Router lại tiếp tục tăng lên theo.

Do đó, trong class Router chúng ta phải sử dụng Ủy nhiệm. Mỗi khi trong controller xuất hiện một action mới (bổ sung thêm chức năng cho chương trình) thì trong Router cần đăng ký một chuỗi truy vấn cùng với một phương thức mới chứa lời gọi action này.

Như vậy, Ủy nhiệm cho phép một class uyển chuyển và linh động hơn trong việc sử dụng phương thức. Theo đó, nội dung cụ thể của một phương thức không được định nghĩa sẵn trong class mà sẽ do người dùng class đó tự định nghĩa trong quá trình khởi tạo object. Điều này giúp phân chia logic của một class ra các phần khác nhau và do những người khác nhau xây dựng.

Kết luận

Trong bài học này chúng ta đã học cách sử dụng ủy nhiệm hàm (delegate) trong C# và vận dụng để hoàn thiện lớp Router.

> Nếu muốn biết chi tiết hơn, bạn có thể đọc thêm bài viết về delegate trong C#.