

# Router (4): phương thức vô danh, hàm lambda

Hướng dẫn tự học lập trình C# toàn tập > Router (4): phương thức vô danh, hàm lambda

Trong bài học này chúng ta tiếp tục các nội dung liên quan đến lớp Router, bao gồm việc sử dụng phương thức vô danh, hàm lambda, hàm cục bộ, và mẫu thiết kế singleton.

## NỘI DUNG CỦA BÀI [ Ẩn ]

- Thực hành: sử dụng lớp Router vừa tạo để đăng ký thêm các truy vấn mới
  - Bước 1. Viết thêm code cho lớp Program
  - Bước 2. Dịch và chạy thử chương trình
- Mẫu thiết kế, singleton, mediator
  - Mẫu thiết kế
  - Singleton
  - Mediator
- Kết luận

## Thực hành: sử dụng lớp Router vừa tạo để đăng ký thêm các truy vấn mới

### Bước 1. Viết thêm code cho lớp Program

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

Trong đoạn code trên, bạn đã sử dụng hàm lambda để gán cho các tham số action của phương thức Register:

```
p => controller.Create()
p => controller.Update(p["id"].ToInt())
```

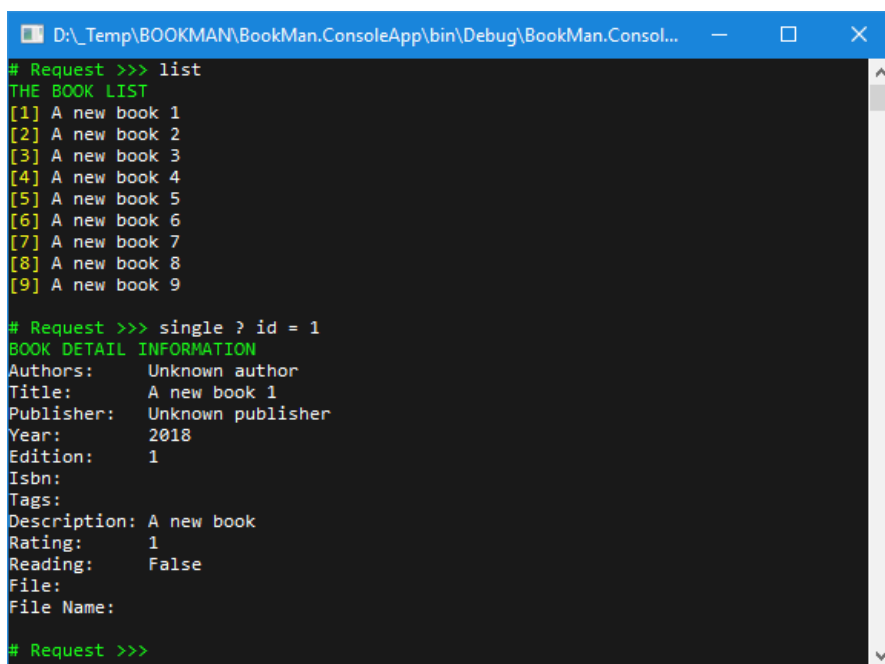
```
p => controller.List()
p => controller.Single(p["id"].ToInt())
```

Bạn hoàn toàn có thể sử dụng anonymous ở đây để tránh phải xây dựng các phương thức “mini” làm rối code. Thực tế, các phương thức này bạn chỉ xây dựng và sử dụng một lần duy nhất làm tham số cho phương thức Register. Bạn không có nhu cầu tái sử dụng code của nó. Do đó, việc xây dựng các phương thức thành viên ở đây không có ý nghĩa.

## Bước 2. Dịch và chạy thử chương trình

Dịch và chạy thử chương trình với các truy vấn sau:

```
Single ? id = 2
Create
Update ? id = 1
List
```

A screenshot of a Windows console application window titled "D:\\_Temp\BOOKMAN\BookMan.ConsoleApp\bin\Debug\BookMan.Consol...". The console output shows the results of four API requests. The first request is "list", which returns a list of 9 books. The second request is "single ? id = 1", which returns detailed information for a book with ID 1. The third request is "Create", and the fourth is "Update ? id = 1". The output is as follows:

```
# 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

# Request >>> single ? id = 1
BOOK DETAIL INFORMATION
Authors:      Unknown author
Title:        A new book 1
Publisher:    Unknown publisher
Year:         2018
Edition:      1
Isbn:
Tags:
Description:  A new book
Rating:       1
Reading:      False
File:
File Name:
```

Kết quả chạy chương trình

## Mẫu thiết kế, singleton, mediator

Khi xây dựng lớp Router chúng ta đã vận dụng hai mẫu thiết kế: *singleton* và *mediator*.

### Mẫu thiết kế

Khi học toán ở trường phổ thông chúng ta thường gặp những mẫu bài tập điển hình. Khi gặp một bài toán lạ chúng ta thường cố gắng quy nó về những mẫu mình biết, từ đó giúp việc giải bài toán đơn giản hơn.

Trong lập trình ứng dụng cũng có tình trạng tương tự. Có những vấn đề chung lặp lại trong nhiều dự án khác nhau đưa đến những kinh nghiệm rằng, nếu gặp lại vấn đề tương tự trong

một dự án mới, chúng ta hoàn toàn có thể áp dụng lại giải pháp đó.

Ví dụ, trong các dự án thường gặp một yêu cầu: làm sao để một class trong chương trình chỉ cho phép sinh ra một object duy nhất.

Lớp Router ở trên là một ví dụ. Khi tạo ra object của lớp Router, chúng ta đăng ký một loạt lệnh và action tương ứng với nó. Danh mục lệnh-action này rõ ràng là phải sử dụng chung trong toàn bộ chương trình, vì nếu chúng ta tạo ra một object mới của lớp Router, chúng ta không thể sử dụng danh mục lệnh đã đăng ký trong object trước đó.

Như vậy rất tự nhiên chúng ta gặp phải yêu cầu: chỉ được phép tạo ra và sử dụng một object duy nhất của lớp Router trong toàn bộ chương trình.

Một ví dụ khác. Trong chương trình thường yêu cầu tính năng log để ghi lại hoạt động của chương trình (ví dụ, để tìm lỗi). Tính năng log thường ghi lại các hoạt động ra file. Nếu một file được mở trong một object nào đó để ghi dữ liệu, object khác không thể mở lại file này để ghi dữ liệu nữa vì nó đã bị chương trình khóa lại. Chỉ khi nào object thứ nhất giải phóng kết nối tới file, object thứ hai mới có thể mở được.

Một giải pháp tự nhiên là chỉ cho phép một object duy nhất làm việc với file này, và bất kỳ ở đâu và lúc nào trong chương trình, khi cần ghi log thì chỉ cần sử dụng object này.

Những vấn đề lặp lại nhiều lần như trên được đúc rút ra thành các hướng dẫn về giải pháp, gọi là các mẫu thiết kế (design pattern). Như vậy, mẫu thiết kế rất tương đồng với mẫu bài tập toán. Nó không phải là một thuật toán mà là một hướng dẫn (guideline) để có thể áp dụng vào các bài toán cụ thể.

## Singleton

Đối với lớp Router và yêu cầu đã nói, chúng ta sử dụng *mẫu thiết kế Singleton* để giải quyết.

Mẫu thiết kế singleton giúp giải quyết vấn đề: chỉ cho phép tạo ra duy nhất một object của một class trong toàn bộ ứng dụng.

Nhóm lệnh sau giúp biến lớp Router thành một singleton:

```
1. // nhóm 3 lệnh dưới đây biến Router thành một singleton
2. private static Router _instance;
3. private Router() => _routingTable = new RoutingTable(); // để ý: constructor là private
4. // người sử dụng class thông qua property này để truy xuất các phương thức của class
5. // chỉ khi nào _instance == null mới tạo object. Một khi đã tạo object, _instance sẽ
6. // không có giá trị null nữa.
7. // vì là biến static, _instance một khi được khởi tạo sẽ tồn tại suốt chương trình
8. public static Router Instance => _instance ?? (_instance = new Router());
```

Nếu sau này có nhu cầu biến một class bất kỳ thành singleton, chỉ cần vận dụng nhóm 3 lệnh này với class mới.

Khi cần sử dụng lớp Router, chúng ta sử dụng thuộc tính tĩnh Instance:

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

```
2. Router.Instance.Register("help", Help);
3. Router.Instance.Register("single", delegate (Router.Parameter p) { controller.Single(p["
4. Router.Instance.Register("update", p => controller.Update(p["id"].ToInt());
```

Đây là cách xây dựng và sử dụng singleton có thể vận dụng đối với bất kỳ class nào nếu có yêu cầu tương tự đặt ra.

## Mediator

Trên thực tế, lớp Router còn vận dụng một mẫu thiết kế nữa, gọi là mẫu *Mediator*.

Mẫu thiết kế này dùng để giải quyết vấn đề kích hoạt một phương thức từ một phương thức khác nhưng không cho hai class chứa các phương thức đó phụ thuộc vào nhau. Thiết kế này hướng tới mục tiêu là *phụ thuộc lỏng* (loosely coupling) giữa các class.

Ở đây chúng ta vận dụng mẫu thiết kế Mediator để các lớp giao diện gọi các phương thức của controller nhưng trong điều kiện là các lớp giao diện không được biết về sự tồn tại của lớp controller (theo quy ước của MVC).

Trong cuốn sách nổi tiếng "Patterns of Enterprise Application Architecture", Martin Fowler đã hệ thống hóa các kiến trúc và thiết kế áp dụng cho phát triển ứng dụng hướng đối tượng. Các bạn có thể tìm đọc cuốn sách này để hiểu chi tiết hơn về các mẫu thiết kế và mẫu kiến trúc.

## Kết luận

Trong bài này chúng ta đã xây dựng hoàn thiện lớp Router. Nhờ lớp này chúng ta có thể cung cấp một chuỗi ký tự vào chương trình vào gọi một phương thức tương ứng của lớp controller. Qua bài này chúng ta đã làm quen phương thức vô danh, hàm lambda, hàm cục bộ. Chúng ta cũng nhắc tới khái niệm mẫu thiết kế và vận dụng mẫu singleton cho lớp Router.

Trong các bài còn lại của phần này, chúng ta sẽ lần lượt hoàn thiện các chức năng cơ bản của ứng dụng sử dụng các công cụ đã được phát triển.

+ 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!