**2. Tạo và đăng ký một route đơn giản**  
Để đăng ký một route, chúng ta mở file global.asax. Sau đó, chúng ta thêm vào method RegisterRoutes đoạn mã sau

Route myRoute = new Route("{controller}/{action}", new MvcRouteHandler());

routes.Add("MyRoute", myRoute);

Hoặc chúng ta có thể dùng phương thức MapRoute của lớp RouteCollection cũng được:

routes.MapRoute("MyRoute", "{controller}/{action}");

**3. Định nghĩa các giá trị mặc định cho Route**  
Như đã nói ở trên, url pattern là một anh chàng khó tính. Anh chàng url pattern chỉ làm việc với một url khi có đủ số segment. Tuy nhiên, chúng ta có thể vượt qua sự khó tính này của url pattern bằng cách gán các giá trị mặc định cho biến segment mà chúng ta mong muốn. Ví dụ đoạn mã sau sẽ gán giá trị mặc định cho biến segment action của chúng ta

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}", new { action = "Index" });

}

Còn ví dụ sau đây sẽ gán giá trị mặc định cho cả biến segment controller và biến segment action

public static void RegisterRoutes(RouteCollection routes) {

routes.MapRoute("MyRoute", "{controller}/{action}",

        new { controller = "Home", action = "Index" });

**4. Sử dụng url segment tĩnh**  
Đôi khi chúng ta muốn có những segment cố định trong url của chúng ta. Giả sử như chúng ta muốn có một chuỗi “Public” cố định trong url của chúng ta thì đoạn code đó như sau

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}",

        new { controller = "Home", action = "Index" });

    routes.MapRoute("", "Public/{controller}/{action}",

       new { controller = "Home", action = "Index" });

|  |
| --- |
| Không những như thế, chúng ta cũng có thể tạo những segment vừa tĩnh vừa động dạng như:  http://gockinhnghiem.com/XAccount/login |

Đoạn mã để thực thi điều đó như sau:

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("", "X{controller}/{action}");

    routes.MapRoute("MyRoute", "{controller}/{action}",

        new { controller = "Home", action = "Index" });

    routes.MapRoute("", "Public/{controller}/{action}",

       new { controller = "Home", action = "Index" });

Không những như thế, chúng ta cũng có thể gán giá trị mặc định cho các segment tĩnh của chúng ta như sau:

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MusicSchema", "Music/{action}", new { controller = "Home" });

}

Với đoạn code trên của chúng ta, chúng ta có 2 segment. Segment tĩnh đầu tiên của chúng ta là Music. Segment thứ 2 của chúng ta chứa giá trị cho action. Chúng ta không có một biến cho controller, mà giá trị mặc định của segment tĩnh đầu tiên là Home sẽ được sử dụng

**6. Định nghĩa những segment mà có nó hay không cũng được**  
Chúng ta có thể định nghĩa những biến segment mà có nó hay không cũng được. Tuy nhiên, chúng ta sẽ không định nghĩa những giá trị mặc định cho những segment dạng này. Chúng ta chỉ định một segment dạng optional như thế này bằng cách thiết lập giá trị mặc định là UrlParameter.Optional, đoạn code mô phỏng như sau:

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}/{id}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional });

}

**7. Định nghĩa chiều dài segment cho route**  
Như đã nói ở trên, url pattern của chúng ta chỉ làm việc với những incoming url đủ số segment. Tuy nhiên, url pattern của chúng ta có thể chấp nhận một url có số segment tùy ý bằng cách dùng catchall cho một trong những biến segment. Nhớ thêm dấu hoa thị ở đầu segment. Đoạn mã mô phỏng điều đó như sau:

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{\*catchall}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional });

}

**8. Dùng namespace để ưu tiên controller theo ý muốn**  
Khi mà một incoming url khớp với một route thì nó sẽ lấy giá trị của biến segment controller và tìm ra một controller thích hợp. Trường hợp này là trường hợp lý tưởng. Giả sử chúng ta có nhiều controller cùng tên thì lỗi sau đây sẽ xuất hiện vì nó không biết phải dùng controller nào

multiple types were found that match the controller named...

Chính vì có nhiều controller cùng tên nên lỗi này sẽ xảy ra  
Để khắc phục tình trạng đụng controller này, chúng ta phải dùng namespace  
Giả sử chúng ta có 2 controller cùng tên là Home. HomeController thứ 1 nằm trong namespace MyNamespace. HomeController thứ 2 nằm trong namespace có tên là OtherNamespace. Và chúng ta muốn ưu tiên HomeController nằm trong namespace MyNamespace, đoạn mã thực hiện điều này như sau:

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{\*catchall}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional },

        new[] { "MyNamespace"});

}

Chúng ta đưa tên các namespace vào mảng chuỗi. Với câu lệnh mới, chúng ta như nói với MVC framework hiểu rằng “Anh hãy ưu tiên tìm HomeController trong namespace MyNamespace, nếu không có, anh mới đi nơi khác tìm”

Tất cả các namespace trong mảng có cùng độ ưu tiên, ví dụ như ta thêm namespace OtherNamespace vào danh sách

routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{\*catchall}",

    new { controller = "Home", action = "Index", id = UrlParameter.Optional },

    new[] { "MyNamespace", "OtherNamespace "});

Với đoạn mã trên, chúng ta sẽ gặp lại lỗi “multiple types were found that match the controller named…”, vì MVC framework không ưu tiên cho ai cả mà nó đi lôi tất cả các controller class name có mặt trong các namespace ra mà nói chuyện, thế là nó lại phân vân và…sinh ra bug

Như đã nói ở trên, MVC framework tìm kiếm trong một namespace nào đó, nếu không thấy thì nó sẽ tìm tiếp trong các namespace khác, đôi khi chúng ta muốn ngăn chặn điều này lại như sau. Đoạn mã tiến hành việc đó như sau:

public static void RegisterRoutes(RouteCollection routes) {

Route myRoute = routes.MapRoute("AddContollerRoute", "Home/{action}/{id}/{\*catchall}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional },

        new[] { "MyNamespace" });

    myRoute.DataTokens["UseNamespaceFallback"] = false;

}

Chúng ta set giá trị cho UseNamespaceFallback của collection DataTokens thành false thì nó sẽ ngăn tình trạng tìm controller trong các namespace khác

**9. Bắt buộc một route**  
a. Dùng regular expression để ép buộc việc khớp một route

public static void RegisterRoutes(RouteCollection routes) {

     routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{\*catchall}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional },

        new { controller = "^H.\*"},

        new[] { "MyNamespace"});

}

Ở đoạn mã trên, ta ép các controller phải bắt đầu bằng chữ H

b. Dùng một tập giá trị cho trước ép buộc việc khớp một route  
Chúng ta có thể ép buộc một biến segment có giá trị thuộc một tập giá trị cho trước như sau

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{\*catchall}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional }

        new { controller = "^H.\*", action = "^Index$|^About$"},

        new[] { "MyNamespace"});

}

Ở đoạn mã trên, chúng ta ép buộc biến segment action có giá trị phải là Index hoặc là About

c. Ep buộc việc khớp một route sử dụng Http method  
Chúng ta có thể ép một route sử dụng một dạng http method theo ý mình. Ví dụ như đoạn mã sau chúng ta sẽ ép route sử dụng dạng GET

public static void RegisterRoutes(RouteCollection routes) {

    routes.MapRoute("MyRoute", "{controller}/{action}/{id}/{\*catchall}",

        new { controller = "Home", action = "Index", id = UrlParameter.Optional },

        new { controller = "^H.\*", action = "Index|About",

            httpMethod = new HttpMethodConstraint("GET") },

        new[] { "MyNamespace" });

}