Research of Routing System which applied in ASP.NET MVC Application

Xiangjun Li $^{1,2}$, Liang Huang $^1$, Zhenrong Lin $^1$, and ZiLong Sai $^1$

$^1$Computer Department, College of Information Engineering, Nanchang University, Nanchang, China
Email: brilliance_h@live.cn

$^2$Colleges of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
Email: lixiangjun@chinalmtc.com

Abstract—Most URLs have been correspond directly to files on the server side in traditional ASP.NET WebForms and many other web platforms. Though, it’s very easy to access the files that existed on the server, through inputs its name directly in the URL, which is easy to understand, but it also makes the files in insecurity. This paper has analyzed the control from Routing System to URL schemas, which constructs agility, concise and beautiful URLs, Then apply it to an Online Store System that designed in ASP.NET MVC application as a demonstration which aims to make search engine optimization.

Index Terms—ASP.NET MVC, Routing System, Routing mechanism, URL Schema

I. INTRODUCTION

Routing System is all about the Universal Resource Locator (URL) and how it is used as an external input to the applications. The URL has led a short but troubled life and the HTTP URL is currently being tragically misused by current web technologies. As the web began to change from being a collection of hyperlinked static documents into dynamically created pages and applications, the URL has been kidnapped by web technologies and it often emerges undesirable mistakes. The URL is in trouble and as the web becomes more dynamic, software developers have a chance to introduce a rescue operation and bring back the simple, logical, readable and flexibility URL.

Routing System brings the chance to software developers to take URL back and have controlling over the URLs of applications. The core conception of routing in traditional ASP.NET WebForms and many other web platforms has been that URLs correspond directly to files on the server. The server implements and serves the page or file corresponding to the incoming URL, while in ASP.NET MVC; URLs are not expected to correspond to files on the web server. In fact, that wouldn’t even make sense—since ASP.NET MVC’s requests are handled by controller classes (compiled into a .NET assembly), there are no particular files corresponding to incoming URLs [1].

While routing is not core to all implementations of the MVC pattern but is often found implements as a convenient extra way to add an extra level of separation between external inputs and the controllers and actions which make up an application.

II. BASIC THEORY OF ROUTING SYSTEM

In ASP.NET MVC application, software developers are given complete controlling of their URL schema—that is, the set of URLs that are accepted, and whose mappings to controllers and actions. This schema isn’t restricted to any predefined pattern and doesn’t need to contain any file name extensions or the names of any of the classes or code files.

A. Routing mechanism

The routing mechanism runs early in the framework’s request processing pipeline. Its job is to take an incoming URL and use it to obtain an IHttpHandler object that will handle the request.

A routing system in ASP.NET MVC framework is responsible for managing the decoupling of the URL with the application logic [2, 3]. It must manage this in both directions so that it can:

- Map URLs to a controller/action and any additional parameters.
- Construct URLs which match the URL schema from a controller, action and additional parameters.

This is more commonly referred to as inbound routing (Figure 1) and outbound routing (Figure 2), respectively. Inbound routing describes when a URL ends up invoking a controller action, whereas outbound routing describes the framework generating URLs for links and other elements on the web site.

![Figure 1. Inbound routing takes an HTTP Request (a URL) and maps it to a controller and action](http://example.com)

![Figure 2. Outbound routing generate appropriate URLs from a given set of route data (controller & action)](http://example.com)
constructing links in a view then the URL schema should be trivial to change independently of the application logic.

B. Three main characters of routing

Routing configurations are built up of three main elements: they are RouteBase, Route and RouteCollection [4-6].

- RouteBase is the abstract base class for a routing entry. Software developers can implement unusual routing behaviors by deriving a custom type from it.
- Route is the standard, commonly used subclass of RouteBase that brings in the notions of URL template, defaults, and constraints.
- A RouteCollection is a complete routing configuration. It’s an ordered list of RouteBase derived objects (e.g. Route objects).

RouteTable.Routes is a special static instance of RouteCollection. It represents the application’s actual, live routing configuration. Typically, it populated just once, when the application first starts, during the Application_Start() method in Global.asax.cs. It’s a static object, so it remains live throughout the application’s lifetime, and is not recreated at the start of each request [7].

Normally, the configuration code isn’t actually inline in Application_Start(), but is factored out into a public static method called RegisterRoutes(). That makes the configuration easy to access from the automated tests.

To see how routing system works clearly. It creates a blank new ASP.NET MVC project, which executing codes are as the following:

```csharp
Route myRoute = new Route
("{controller}/#{action}/#{id}", new MvcRouteHandler());
Defaults = new RouteValueDictionary(new {
    controller = "Home", action = "Index", id = ""
});
routes. Add ("Default", myRoute);
```

Now, the software developers are given complete controlling of their URL schema—that is, the set of URLs that are accepted, and their mappings to controllers and actions. This schema isn’t restricted to any predefined pattern and doesn’t need to contain any file name extensions or the names of any of classes or code files. Table I shows an example:

C. Routing configuration

There are five properties software developers can configure on each RouteTable entry. These affect whether or not it matches a given URL, and if it does, what happens to the request, see Table II.

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Url</td>
<td>The URL to be matched, with any parameters in curly braces (required).</td>
<td>string</td>
<td>&quot;Browse/{category}/{pageIndex}&quot;</td>
</tr>
<tr>
<td>RouteHandler</td>
<td>The handler used to process the request (required).</td>
<td>IRouteHandler</td>
<td>new MvcRouteHandler()</td>
</tr>
</tbody>
</table>
| Defaults      | Makes some parameters optional, giving their default values. | RouteValueDictionary | new RouteValueDictionary(new {
controller = "Products", action = "List", category = "Fish", pageIndex = 3 }) |
| Constraints   | A set of rules that request parameters must satisfy. | RouteValueDictionary | new RouteValueDictionary(new {
pageIndex = @"d(0,8)" }) |
| DataTokens    | A set of arbitrary extra configuration options that will be passed to the route handler (usually not required). | RouteValueDictionary | new RouteValueDictionary(new {
pageIndex = @"d(0,8)" }) |

D. The order of Route Entries

RouteCollection is an ordered list, and the order in which we add route entries is critical to the route-matching process. The system does not attempt to find the “most specific” match for an incoming URL; its algorithm is to start at the top of the route table, check each entry in turn, and stop when it finds the first match [10]. So put more specific route entries before less specific ones which configure as follows:

```csharp
routes.MapRoute ("Specials", // Route name
    new { controller = "Catalog", action = "ShowSpecials" }); // Parameter defaults
```

```
routes.MapRoute ("DailySpecials/{date}", // URL with parameters
    new {
        controller = "Catalog", action = "ShowSpecials" }, // Parameter defaults
); // Parameter defaults
```

```
routes.MapRoute ("Default", // Route name
    new { controller = "Home", action = "Index", id = "" }); // Parameter defaults
```
E. Unit Testing Routes

One of the core design principles of the ASP.NET MVC Framework is enabling great testing support. Ref. [11] like the rest of the MVC framework, programmers can easily unit test routes and route matching rules. The MVC Routing system can be instantiated and run independent of ASP.NET—which means programmers can load and unit test route patterns within any unit test library and using any unit test framework (NUnit, MBUnit, MSTest, etc)[12].

Although programmers can unit test an ASP.NET MVC Application’s global RouteTable mapping collection directly within their unit tests, in general it is usually a bad idea to have unit tests ever change or rely on global state [13]. A better pattern that programmers can use is to structure their route registration logic into a RegisterRoutes () helper method like below that works against a RouteCollection that is passed in as an argument (the unit tests code below is compiled based on the mock):

```csharp
private string GetOutboundUrl(object routeValues){
    //Get route configuration and mock request context
    RouteCollection routes = new RouteCollection ();
    MvcApplication.RegisterRoutes (routes);
    var mockHttpContext = new Moq.Mock(HttpContextBase());
    var mockRequest = new Moq.Mock(HttpRequestBase());
    var fakeResponse = new FakeResponse();
    mockHttpContext.Setup(x=>x.Request).Returns(mockRequest.Object);
    mockHttpContext.Setup(x=>x.Response).Returns(fakeResponse);
    mockRequest.Setup(x=>x.ApplicationPath).Returns("/");
    //Generate the outbound URL
    var ctx=new RequestContext(mockHttpContext.Object,
    new RouteData());
    return routes.GetVirtualPath(ctx, new RouteValueDictionary(routeValues));
}
```

Then programmers can write unit tests that create their own RouteCollection instance and call the Application's RegisterRoutes () helper method like below that works against a RouteCollection that is passed in as an argument (the unit tests code below is compiled based on the mock):

```csharp
private string GetOutboundUrl(object routeValues){
    //Get route configuration and mock request context
    RouteCollection routes = new RouteCollection ();
    MvcApplication.RegisterRoutes (routes);
    var mockHttpContext = new Moq.Mock(HttpContextBase());
    var mockRequest = new Moq.Mock(HttpRequestBase());
    var fakeResponse = new FakeResponse();
    mockHttpContext.Setup(x=>x.Request).Returns(mockRequest.Object);
    mockHttpContext.Setup(x=>x.Response).Returns(fakeResponse);
    mockRequest.Setup(x=>x.ApplicationPath).Returns("/");
    //Generate the outbound URL
    var ctx=new RequestContext(mockHttpContext.Object,
    new RouteData());
    return routes.GetVirtualPath(ctx, new RouteValueDictionary(routeValues)).VirtualPath;
}
```

A. Add RouteTable entries

The default route (matching{controller}/ {action}/ {id}) is so general in purpose that we could build an entire application around it without needing any other routing configuration entry[17, 18]. However, if it does want to handle URLs that don’t bear any resemblance to the names of controllers or actions, then it will need other configuration entries, these route entries show in follows (Just show representative ones):

```csharp
routes.MapRoute (null,
"{category}",  //Matches ~/Football or
~/Anything with no slash
new { controller = "Products", action = "List", page = 1 });

routes.MapRoute (null,
"{Category}/Page {page}" ,//Matches
~/Football/Page567
new { controller = "Products", action = "List" },
//Defaults
new { page = @"@\d\+" }); //Constraints: Page

This entry will match /Catalog or /Catalog? some=querystring, but not /Catalog/Anythingelse. It’s understood which parts of a URL are significant to a Route entry.

B. URL Patterns Match the Path Portion of a URL

When a Route object decides whether it matches a certain incoming URL, it only considers the path portion of that incoming URL. That means it doesn’t consider the domain name (also called host) or any query string values. Figure 3 depicts the path portion of a URL [19, 20].

![Figure 3. Identifying the path portion of a URL](http://www.example.com/some/url?abc=def&ghi=jkl)

The URL pattern "Catalog" would therefore match both http://a.b.c.d:1234/Catalog?query=string and http://example.com/Catalog. If programmers deploy to a virtual directory, their URL patterns are understood to be relative to that virtual directory root.

C. Order Route Entries

Order all of the RouteTable Entries according to the golden rule of routing—put more specific route entries before less specific ones [21]:

```csharp
new { controller = "Products", action = "List", category = (string)null, page = 1 },
new { controller = "Products", action = "List", category = (string)null },
new { controller = "Products", action = "List", page = 1 },
new { controller = "Products", action="List" }
```

III. EXAMPLES OF APPLICATION

Now, it’s been aware of the principle of the Routing mechanism and the theory behind its configuration. It’s necessary to put the mechanism into action for demonstration and see how those benefits work out in a demonstration application.

The demonstration application, SportsStore, is designed by ASP.NET MVC Framework for online shopping. There is a product catalog browsable by category and page index between which would be used by routing system to navigation or redirection from one product to another.
D. Demonstrate

After adding route entries, they are ordered and configured in ASP.NET MVC application, it has broken the traditional URL schemas, convert the common URL to be human-friendly URL, See Figure 4:

![Figure 4. Clear URL designed by routing system in ASP.NET MVC Application](image)

It shows that the URL address is at the Page2 with clear, human-friendly and simple, it presents the URL with the correspond information about the product, which let users understand immediately, when redirected to a product detail page.

E. Comparison

In Figure 5, we can see the common URL which contains extension name of asp file in the ASP.NET WebForm application.

![Figure 5. URL in the traditional ASP.NET WebForm](image)

![Figure 6. Contrast Results from the Route System of MVC Application and the traditional ASP.NET WebForm by LoadRunner Analysis Software](image)
Compared to ASP.NET WebForm, the routing system of MVC application is better than ASP.NET’s at the following:

- It's doesn't use the storage path on the server as the URL, MVC routing system configures the URL by controller.
- It hides the complex .aspx extensions which make sure the files on the server’s security.
- It could be designed a completely independent web application and attached to the core routing system with its placeholder, defaults, route validation, and URL generation features.

Through LoadRunner analysis software, it can analyze the ASP.NET MVC application and the traditional ASP.NET WebForm. Compared to them both in the reaction time and interaction by users. It shows that the routing system of MVC application is better than the ASP.NET’s, See Figure 6.

IV. CONCLUSION

To design the URL schema for an application is a great challenge which covered in this paper. There is never a definitive answer to that route will need to be implemented and that the code needed to generate routes and URLs from routes is very simple, while the process of designing of schema is not. Ultimately every application is different, some will be effective that the default routes created by the project template where others will have route manager classes stretching to hundreds of lines.

All descriptions above shows that order of definition matters and that careful consideration have to be made when adding new routes to the application. As more routes are defined, the risk of breaking existing URLs increases.

Separation of the URL schema from the underlying code architecture gives ultimate flexibility and allows software developers to focus on what would make sense for the users on the URL rather than what the layout of our source code requires.

ACKNOWLEDGMENT

This paper is supported by the 2009 Jiangxi Natural Science Foundation (No.2009JX02289) and by the 2008 Technical Plan Project of Jiangxi Education Department (No.GJJ08036).

REFERENCES