Routing in ASP.NET Core MVC

Posted by **Code Maze** | Updated Date Sep 2, 2020 | **0**

Want to build **great APIs?** Or become **even better** at it? Check our program **Ultimate ASP.NET Core 3 Web API** and learn how to create a full production-ready ASP.NET Core API using only the **latest .NET technologies**. Bonus materials included!

In this article, we're going to discuss the routing capabilities in ASP.NET Core MVC. We're also going to learn how to set up routes in two different ways.

If you've missed some of the previous articles in the series we recommend visiting the series page: **ASP.NET Core MVC Series.**

To download this article's source code visit: **Routing in ASP.NET Core MVC.**

We have divided this article into the following sections:

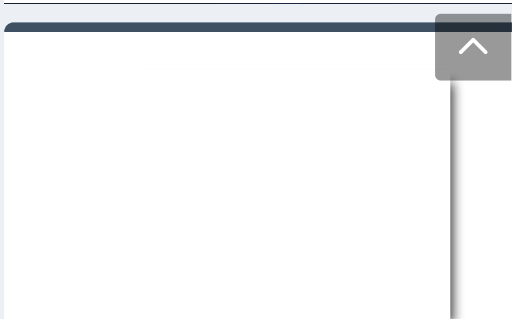
Let's dive into the material.

Routing in ASP.NET Core MVC

Routing is the process through which the application matches an incoming URL path and executes the corresponding action methods. ASP.NET Core MVC uses a routing middleware to match the URLs of incoming requests and map them to specific action methods.

We can define the routes either in the startup code or as attributes. They describe how we can match the URL paths with the action methods. We can also use routes to generate URLs for links that are sent out in responses.

There are two types of routing for action methods:



- Conventional Routing
- Attribute Routing

Conventional Routing

When we create a new ASP.NET Core MVC application using the default template, the application configures a default routing. Let's create a new project and examine this.

After creating a new project with the default ASP.NET Core MVC template, let's have a look at the `startup.cs` class. We can see that the application has configured a default routing in the `Configure()` method:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Inside the call to `UseEndpoints()`, we use the `MapControllerRoute()` method to create a route by giving the name `default`.

MVC configures the default route template as `{controller=Home}/{action=Index}/{id?}`. This will match the `Index()` method in `HomeController` with an optional parameter `id` by default. This can also match a URL path like `/Books/Details/5` and will extract the route values `{ controller = Books, action = Details, id = 5 }` by tokenizing the path. MVC will attempt to locate a controller named `BooksController` and run the action method `Details()` by passing the id parameter as `5`.

Let's run the application and place a breakpoint in the `Index()` method of `HomeController`. We can see that this method is executed by default. This is because we have defined these as the default values in the route.

Now let's change the default route. For that, let's add a new controller `BooksController` with an action method `Details()` and an optional parameter `id`:

```
public class BooksController : Controller
{
    public IActionResult Details(int id)
    {
        ViewBag.Id = id;
        return View();
    }
}
```

Then let's create a view for the `Details()` action method:

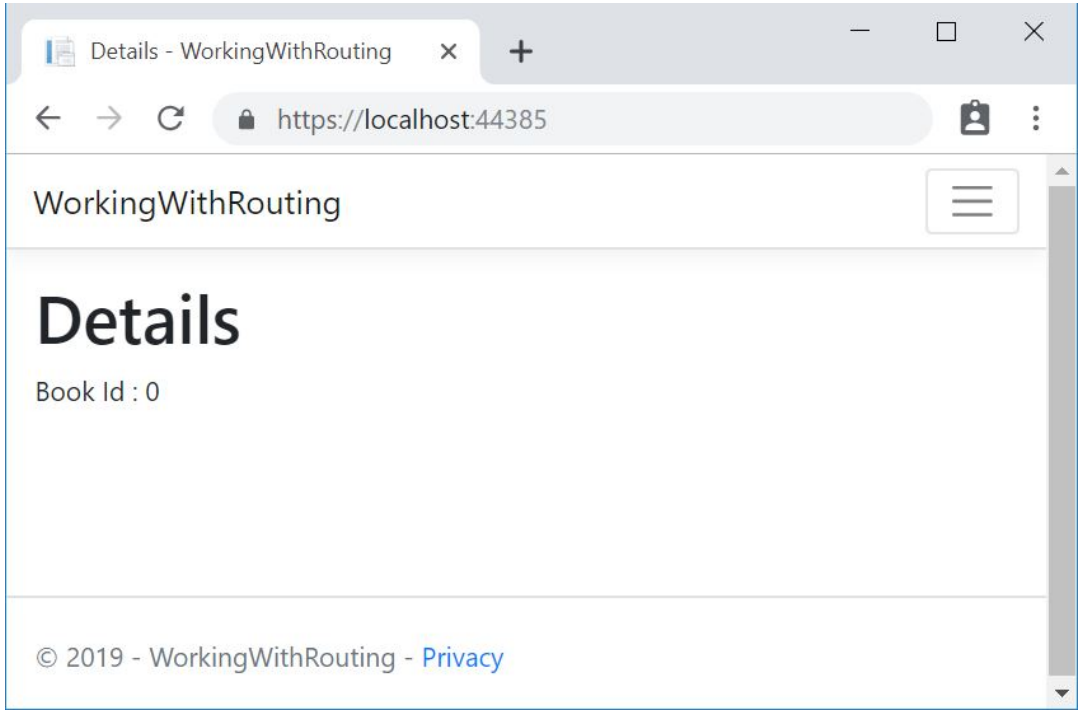
```
@{
    ViewData["Title"] = "Details";
    int id = ViewBag.Id;
}

<h1>Details</h1>

Book Id : @id
```

Once these are in place, let's change the default route in the startup class:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Books}/{action=Details}/{id?}");
```

```
});
```

Now let's run the application once again. We can see that the call goes to `Details()` method of the `BooksController` by default:



In this section, we have learned how Conventional Routing works and how to configure the default route for an application. Now let's see how to configure multiple routes.

Multiple Routes

We can add multiple routes inside `UseMvc()` by calling `MapRoute()` multiple times. This allows us to define multiple conventions or to add routes that are dedicated to a specific action:

```
app.UseEndpoints(endpoints =>
```

```
{
    endpoints.MapControllerRoute(
        name: "blog",
        pattern: "blog/{*article}",
        defaults: new { controller = "Blog", action = "Article" })

    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Books}/{action=Details}/{id?}");
});
```

The `blog` route here is a dedicated conventional route. It uses the conventional routing system but is dedicated to a specific action. Since controller and action don't appear in the route template as parameters, they can only have the default values, and thus this route will always map to the action method `BlogController.Article()`.

Routes in the route collection are ordered and will be processed in the order we add them. So in this example, MVC will try the `blog` route before the `default` route.

It is always good to specify a route name while creating the routes. It gives the route a logical name so that we can use the named route for URL generation.

So far, we have looked at how to configure the Conventional Routing. Next, let's learn about Attribute Routing.

Attribute Routing

By placing a route on the controller or the action method, we can make use of the Attribute Routing feature.

Let's modify the `Configure()` method in the `startup.cs` class and remove the default routes that we had defined earlier.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
```

Now let's modify the `BooksController` by giving custom attributes as routes:

```
public class BooksController : Controller
{
    [Route("")]
    [Route("Home")]
    [Route("Home/Index")]
    public IActionResult Index()
    {
        return View();
    }

    [Route("Home/Details/{id:int}")]
    public IActionResult Details(int id)
    {
        ViewBag.Id = id;
        return View();
    }
}
```

We have specified the URL paths `/`, `/Home`, or `/Home/Index` for the `BooksController.Index()` action and `/Home/Details/{id}` for the `Details()` method. When using Attribute Routing, the controller name and action method name plays no role in selecting which action method to execute.

We can also use `Http[Verb]` attributes for Attribute Routing:

```
[HttpGet("/books")]
public IActionResult ListBooks()
{
    // ...
}

[HttpPost("/books")]
public IActionResult CreateBook(...)
{
    // ...
}
```

For the URL path `/books`, MVC will execute the `ListBooks()` action when the HTTP verb is `GET` and `CreateBook()` when the HTTP verb is `POST`.

Route attributes defined on the controller are combined with route attributes on the individual action methods to form the complete URL. Any route templates defined on the controller are prepended to route templates on all of its action methods. When we place a route attribute on the controller, all actions in the controller use attribute routing.

Attribute routes support token replacement by enclosing a token in square-braces ([,]). The tokens `[action]`, `[area]`, and `[controller]` are replaced with the values of the `action name`, `area name`, and `controller name` from the action where the route is defined:

```
[Route("[controller]/[action]")]
public class BooksController : Controller
{
    [HttpGet]
    public IActionResult List()
    {
        // ...
    }
```

```
    [HttpGet("{id}")]
    public IActionResult Edit(int id)
    {
        // ...
    }
}
```

The `List()` action method matches the URL `/Books/List` and `Edit()` method matches the URL `/Books/Edit/{id}`.

In this section, we've learned how to configure Attribute Routing. Now let's look at how to specify multiple routes as attributes.

Multiple Routes

Attribute routing supports defining multiple routes that reach the same action. The most common usage of this is to achieve the functionality of the default conventional route:

```
[Route("[controller]")]
public class BooksController : Controller
{
   [Route("")]     // Matches 'Books'
   [Route("Index")] // Matches 'Books/Index'
   public IActionResult Index()
}
```

By defining two routes, the `Index()` method matches the URL paths `/Books` and `Books/Index`.

That's it, let's summarize what we've learned.

Conclusion

In this article we have learned the following topics:

- Creating a Conventional Routing pattern for an ASP.NET Core MVC application
- Configuring Attribute Routing for controllers and action methods
- Setup multiple routes for both conventional and attribute routing

In the next part of this series, we'll look into file upload in ASP.NET Core MVC, so stay tuned.

SHARE:

RELATED POSTS

C# Design Patterns – Command

Updated Date Mar 22, 2020

How to Configure PostgreSQL in Entity Framework Core

Updated Date Sep 4, 2020

CQRS and MediatR in ASP.NET Core

Updated Date Sep 19, 2020

C# Back to Basics – Conditions

Updated Date Aug 27, 2020