How to use attribute routing in ASP.NET Core

## **Introduction**

Take advantage of attribute-based routing in ASP.NET Core to decouple the controller and action names from the route template and allow more flexibility in your routing.

The routing middleware in ASP.NET Core is adept at mapping incoming requests to the respective route handlers. You can set up routing in ASP.NET Core in two different ways: attribute-based routing and convention-based routing.

Unlike convention-based routing, in which the routing information is specified at a single location, attribute routing enables you to implement routing by decorating your action methods with attributes. This article presents a discussion of how we can work with attribute-based routing in ASP.NET Core MVC.

## Create an ASP.NET Core MVC project in Visual Studio.NET

First off, let’s create an ASP.NET Core project in Visual Studio .NET. Follow the steps outlined below to create a new ASP.NET Core project in Visual Studio.

1. Launch the Visual Studio IDE.
2. Click on “Create new project.”
3. In the “Create new project” window, select “ASP.NET Core Web App (Model-View-Controller)” from the list of templates displayed.

We’ll use this project in the sections below to illustrate how we can work with attribute routing.

## Create a controller class in ASP.NET Core MVC

Create a new controller named DefaultController and replace the default source code of the DefaultController with the following code:

**public** **class** **DefaultController** : **Controller**  
    {  
        [**Route**("")]  
        [**Route**("Default")]  
        [**Route**("Default/Index")]  
        **public** **ActionResult** **Index**()  
        {  
            **return** **new** **EmptyResult**();  
        }  
        [**Route**("Default/GetRecordsById/{id}")]  
        **public** **ActionResult** **GetRecordsById**(**int** id)  
        {  
            **string** str = **string**.**Format**  
            ("The id passed as parameter is: {0}", id);  
            **return** **Ok**(str);  
        }  
    }

## Use attribute routing at the controller level in ASP.NET Core

Attribute routing can be used both at the controller and action method levels. If we apply the route attribute at the controller level, then the route is applicable to all action methods of that controller.

If you examine our DefaultController class, you’ll observe that the Default route is used multiple times when specifying the route template for the action methods. The following code snippet shows how you can specify different route attributes at the controller level to make more flexible use of attribute routing.

[**Route**("Default")]     
**public** **class** **DefaultController** : **Controller**  
{  
  [**Route**("")]  
  [**Route**("Index")]  
  **public** **ActionResult** **Index**()  
  {  
      **return** **new** **EmptyResult**();  
   }  
  [**HttpGet**]  
  **Route**("Default/GetRecordsById/{id}")]  
  **public** **ActionResult** **GetRecordsById**(**int** id)  
  {  
      **string** str = **string**.**Format**("The id passed as parameter is: {0}", id);  
      **return** **Ok**(str);  
   }  
}

When using route attributes both at the controller and the action method levels, the route template applied at the controller level is prepended to the route template specified at the action method level.

You might often need a common prefix for your controller. When you do, you should use the [RoutePrefix] attribute as shown in the code snippet given below.

[**RoutePrefix**("services")]  
**public** **class** **HomeController** : **Controller**  
{  
   *//Action methods*  
}

## Use attribute routing at the action method level in ASP.NET Core

Refer to the DefaultController class shown above. As you can see, we’ve specified three routes in the Index method of the DefaultController class. This implies that each of the following URLs will invoke the Index() action method of the DefaultController.

http://localhost:11277  
http://localhost:11277/home  
http://localhost:11277/home/index

As in convention-based routing, you can specify parameters in attribute-based routing as well. In other words, attribute-based routing allows you to specify route attributes with parameters. The GetRecordsById action method of the DefaultController class shown earlier is an example.

Note that "{id}" in the specified route represents a parameter or a place holder. The id parameter in this example can be anything, such as a string or an integer. What if you would like to restrict the parameter to only integers? You can achieve this by using constraints.

## Use attribute route constraints in an action method

Route constraints are used to thwart invalid requests to controller actions. For example, you might want to ensure that the parameter passed to an action method is always an integer. The syntax of using route constraints is {parameter:constraint}. The following code snippet illustrates this. Note that the id parameter here is always an integer.

[**Route**("Default/GetRecordsById/{id:int}")]  
**public** **ActionResult** **GetRecordsById**(**int** id)  
{  
  **string** str = **string**.**Format**("The id passed as parameter is: {0}", id);  
  **return** **Ok**(str);  
}

## Use optional parameters in attribute route specifications

You can use optional parameters in your route specification too. The following code snippet shows how this can be achieved. Note that the action method in this case would be executed even if the id parameter is not passed.

[**Route**("Sales/GetSalesByRegionId/{id?}")]

It’s important to understand that when using attribute routing neither the controller name nor the action method name plays any role in selecting which action method will be executed. Let’s see this with an example. The following code snippet illustrates how the URL has been changed in the route specification for the GetRecordsById action method.

[**Route**("Home/GetRecordsById/{id:int}")]  
**public** **ActionResult** **GetRecordsById**(**int** id)  
{  
   **string** str = **string**.**Format**("The id passed as parameter is: {0}", id);  
   **return** **Ok**(str);  
}

You can now invoke the GetRecordsById action method using the following URL:

http://localhost:11277/home/GetRecordsById/1

## Use multiple attribute route constraints in an action method

It is also possible to apply multiple constraints to a parameter. The following code snippet illustrates how this can be achieved. Note the minimum value of the id parameter should be 1, else a 404 error will be returned.

[**Route**("Default/GetRecordsById/{id:int:min(1)}")]  
**public** **ActionResult** **GetRecordsById**(**int** id)  
{  
    **string** str = **string**.**Format**("The id passed as parameter is: {0}", id);  
    **return** **Ok**(str);  
}

## Use HTTP verbs in attribute routes in an action method

You can even use HTTP verbs in attribute routing. The following code snippet shows how this can be achieved.

[**HttpGet**]  
[**Route**("Default/GetRecordsById/{id:int:min(1)}")]  
**public** **ActionResult** **GetRecordsById**(**int** id)  
{  
  **string** str = **string**.**Format**("The id passed as parameter is: {0}", id);  
  **return** **Ok**(str);  
}

## Commonly used attribute route constraints

Here is a list of the most commonly used route constraints in ASP.NET Core.

* bool - used to match a Boolean value
* datetime - used to match a DateTime value
* decimal - used to match a decimal value
* double - used to match a 64-bit floating-point value
* float - used to match a 32-bit floating-point value
* guid - used to match a GUID value
* int - used to match a 32-bit integer value
* long - used to match a 64-bit integer value
* max - used to match an integer with a maximum value
* min - used to match an integer with a minimum value
* minlength - used to match a string with a minimum length
* regex - used to match a regular expression

## Create custom attribute route constraints

You can also create your own custom route constraints by creating a class that extends the IRouteConstraint interface and implements the Match method as shown in the code snippet given below.

**public** **class** **CustomRouteConstraint** : **IRouteConstraint**  
    {  
        **public** **bool** **Match**(**HttpContext** httpContext, **IRouter** route,  
        **string** routeKey,  
        **RouteValueDictionary** values, **RouteDirection** routeDirection)  
        {  
            **throw** **new** **NotImplementedException**();  
        }  
    }

## Use token replacement in attribute routes at the controller level

Attribute routing in ASP.NET Core MVC provides support for another interesting feature named token replacement. You can use the tokens [action], [area], and [controller] in your controller, and these tokens will be replaced by the action, area, and controller names respectively. The following code snippet illustrates how this can be achieved.

[**Route**("[controller]/[action]")]  
**public** **class** **HomeController** : **Controller**  
{  
   **private** **readonly** **ILogger**<**HomeController**> \_logger;  
   **public** **HomeController**(**ILogger**<**HomeController**> logger)  
   {  
       \_logger = logger;  
   }  
   **public** **IActionResult** **Index**()  
   {  
       **return** **View**();  
   }  
   *//Other action methods*  
}

Attribute routing in ASP.NET Core gives you more control and flexibility over the URIs in your web application. Although convention-based routing can be configured at a single location, which can in turn be applied to all controllers in your application, it is difficult to support certain URI patterns (such as API versioning) with convention-based routing.

By using attribute routing, you can decouple the controller and action names from the route template. You can even use a combination of convention-based routing and attribute-based routing in your ASP.NET Core applications.