**The routing system has two functions:**

1-Examine an incoming URL and figure out for which controller and action therequest is intended.

2-Generate outgoing URLs.

URLs can be broken down into segments. These are the parts of the URL, excluding the hostname and query string, that are separated by the / character.

In this example **http://mysite.com/Admin/Index**, there are two segments

Admin - First Segment

Index - Second Segment

Having this pattern in RoueConfig.cs url: "{controller}/{action}/{id}" => controller = Admin, action = Index

An MVC application will usually have several routes and the routing system will compare the incoming URL to the URL pattern of each route until it finds a match.

By default, a URL pattern will match any URL that has the correct number of segments.

Request URL Segment Variables

http://mysite.com/Admin/Index controller = Admin action = Index

http://mysite.com/Index/Admin controller = Index action = Admin

http://mysite.com/Apples/Oranges controller = Apples action = Oranges

http://mysite.com/Admin No match No match too few segments

http://mysite.com/Admin/Index/Soccer No match—too many segments

## [Creating and Registering a Simple Route]

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

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

);

name is optional

url defines the URL pattern

defaults sets the default values for controller and action

## [Using Static URL Segments]

Not all of the segments in a URL pattern need to be variables. You can also create patterns that have static segments.

**Example 1:**

routes.MapRoute(

name: "Default1",

url: "**Public/**{controller}/{action}",

defaults: new { controller = "Home", action = "Index" }

);

**Example 2:**

routes.MapRoute(

name: "Default2",

url: "**X**{controller}/{action}",

defaults: new { controller = "Home", action = "Index" }

);

## [Defining Custom Segment Variables]

The controller and action segment variables have special meaning to the MVC Framework and, obviously, they correspond to the controller and action method that will be used to service the request.

we can also define our own variables

routes.MapRoute(

name: "Default",

url: "{controller}/{action**}/{id}**",

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

);

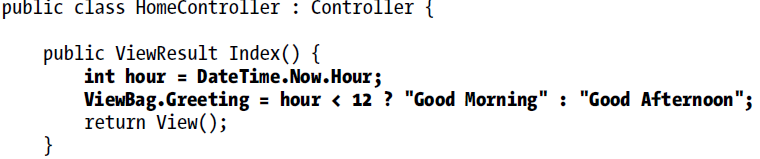
We can access any of the segment variables in an action method by using the RouteData.Values

## [Adding Dynamic Output]

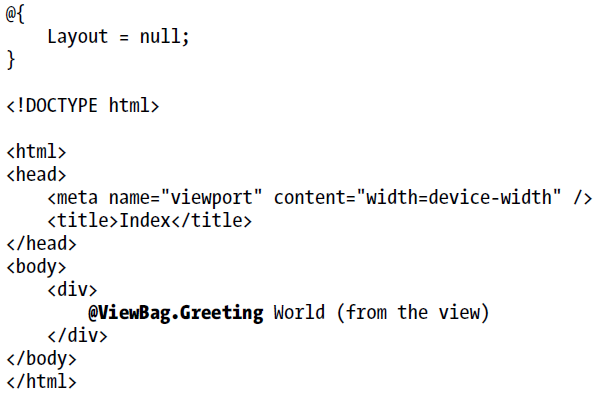
In MVC, it is the controller’s job to construct some data and pass it to the view, which is responsible for rendering it to HTML.

One way to pass data from the controller to the view is by using the **ViewBag** object, which is a

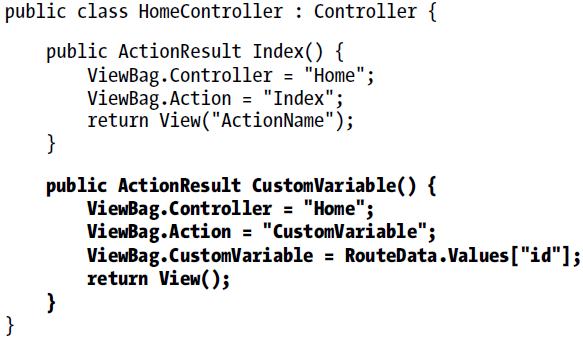
member of the **Controller** base class. **ViewBag** is a dynamic object to which you can assign arbitrary properties.

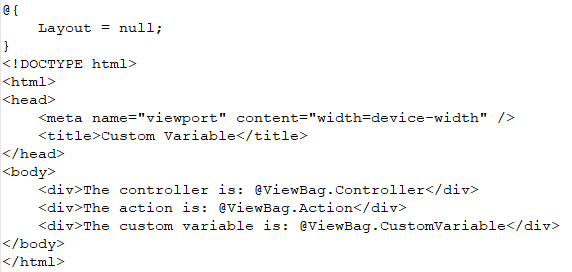


Change we made to the **Index.cshtml**



## [Reading Custom Segment Variables]





## Using Custom Variables as Action Method Parameters

Using the **RouteData.Values** property is only one way to access custom route variables. The other way is much more elegant. If we define parameters to our action method with names that match the URL pattern variables, the MVC Framework will pass the values obtained from the URL as parameters to the action method.

The MVC Framework uses the model binding system to convert the values contained in the URL to .NET

types.