**Controller**

# ASP.NET MVC Routing Overview (C#)

First, ASP.NET Routing is enabled in your application's Web configuration file (Web.config file). There are four sections in the configuration file that are relevant to routing: the system.web.httpModules section, the system.web.httpHandlers section, the system.webserver.modules section, and the system.webserver.handlers section. Be careful not to delete these sections because without these sections routing will no longer work.

Second, and more importantly, a route table is created in the application's Global.asax file. The route table is created during the Application Start event.

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

"Default", // Route name

"{controller}/{action}/{id}", // URL with parameters

new { controller = "Home", action = "Index", id = "" } // Parameter defaults

);

}

protected void Application\_Start()

{

RegisterRoutes(RouteTable.Routes);

}

The RegisterRoutes() method creates the route table.

For the following Controller

public class HomeController : Controller

{

public ActionResult Index(string id)

{

return View();

}

# }

URL /Home Index() method

URL /Home/Index/3 Index() method (the Id is ignored).

# Understanding Action Filters (C#)

An action filter is an attribute that you can apply to a controller action -- or an entire controller -- that modifies the way in which the action is executed. The ASP.NET MVC framework includes several action filters:

* OutputCache – This action filter caches the output of a controller action for a specified amount of time.
* HandleError – This action filter handles errors raised when a controller action executes.
* Authorize – This action filter enables you to restrict access to a particular user or role.

You also can create your own custom action filters. For example, you might want to create a custom action filter in order to implement a custom authentication system.

### Using an Action Filter

An action filter is an attribute. You can apply most action filters to either an individual controller action or an entire control

[OutputCache(Duration = 10)]

public string Time()

{

return DateTime.Now.ToString();

}

# The Different Types of Filters

The ASP.NET MVC framework supports four different types of filters:

1. Authorization filters – Implements the IAuthorizationFilter attribute.

to implement authentication and authorization for controller actions.

1. Action filters – Implements the IActionFilter attribute.

logic that is executed before and after a controller action executes.

1. Result filters – Implements the IResultFilter attribute.

logic that is executed before and after a view result is executed.

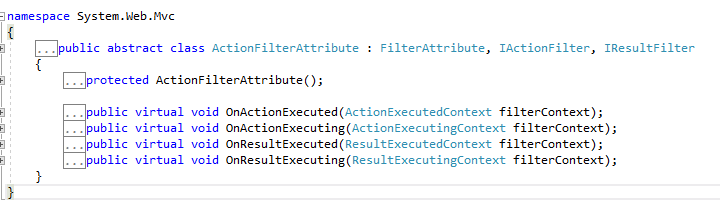
1. Exception filters – Implements the IExceptionFilter attribute.

to handle errors raised by either your controller actions or controller action results.

Filters are executed in the order listed above.

# The Base ActionFilterAttribute Class

In order to make it easier for you to implement a custom action filter, the ASP.NET MVC framework includes a base ActionFilterAttribute class. This class implements both the IActionFilter and IResultFilter interfaces and inherits from the Filter class.



# Improving Performance with Output Caching (C#)

The output cache enables you to cache the content returned by a controller action. That way, the same content does not need to be generated each and every time the same controller action is invoked.

You enable output caching by adding an [OutputCache] attribute to either an individual controller action or an entire controller class.

## **Where Content is Cached**

By default, when you use the [OutputCache] attribute, content is cached in three locations: the web server, any proxy servers, and the web browser. You can control exactly where content is cached by modifying the Location property of the [OutputCache] attribute.

You can set the Location property to any one of the following values:

· Any

· Client

· Downstream

· Server

· None

· ServerAndClient

By default, the Location property has the value Any. However, there are situations in which you might want to cache only on the browser or only on the server. For example, if you are caching information that is personalized for each user then you should not cache the information on the server. If you are displaying different information to different users then you should cache the information only on the client.

## **Varying the Output Cache**

In some situations, you might want different cached versions of the very same content. Imagine, for example, that you are creating a master/detail page. The master page displays a list of movie titles. When you click a title, you get details for the selected movie.

If you cache the details page, then the details for the same movie will be displayed no matter which movie you click. The first movie selected by the first user will be displayed to all future users.

You can fix this problem by taking advantage of the VaryByParam property of the [OutputCache] attribute. This property enables you to create different cached versions of the very same content when a form parameter or query string parameter varies.

[OutputCache(Duration=int.MaxValue, VaryByParam="none")]

public ActionResult Master()

{

ViewData.Model = (from m in \_dataContext.Movies

select m).ToList();

return View();

}

[OutputCache(Duration = int.MaxValue, VaryByParam = "id")]

public ActionResult Details(int id)

{

ViewData.Model = \_dataContext.Movies.SingleOrDefault(m => m.Id == id);

return View();

}

The Master() action includes a VaryByParam property with the value "none". When the Master() action is invoked, the same cached version of the Master view is returned.

The Details() action includes a VaryByParam property with the value "Id". When different values of the Id parameter are passed to the controller action, different cached versions of the Details view are generated.

## **Creating a Cache Profile**

As an alternative to configuring output cache properties by modifying properties of the [OutputCache] attribute, you can create a cache profile in the web configuration (web.config) file.

<caching>

<outputCacheSettings>

<outputCacheProfiles>

<add name="Cache1Hour" duration="3600" varyByParam="none"/>

</outputCacheProfiles>

</outputCacheSettings>

## </caching>

## **Adding Dynamic Content to a Cached Page (C#)**

#### **Using Post-Cache Substitution**

Using post-cache substitution requires two steps. First, you need to define a method that returns a string that represents the dynamic content that you want to display in the cached page. Next, you call the HttpResponse.WriteSubstitution() method to inject the dynamic content into the page.

public static class PostCache

{

public static string RenderUpdatedCache(HttpContext context)

{

return DateTime.Now.ToString();

}

}

@{

Response.WriteSubstitution(PostCache.RenderUpdatedCache);

}

## **ASP.NET MVC Controller Overview**

## **Understanding Action Results**

A controller action returns something called an *action result*. An action result is what a controller action returns in response to a browser request.

The ASP.NET MVC framework supports several types of action results including:

1. ViewResult - Represents HTML and markup.
2. EmptyResult - Represents no result.
3. RedirectResult - Represents a redirection to a new URL.
4. JsonResult - Represents a JavaScript Object Notation result that can be used in an AJAX application.
5. JavaScriptResult - Represents a JavaScript script.
6. ContentResult - Represents a text result.
7. FileContentResult - Represents a downloadable file (with the binary content).
8. FilePathResult - Represents a downloadable file (with a path).
9. FileStreamResult - Represents a downloadable file (with a file stream).

All of these action results inherit from the base ActionResult class.