MVC

The MVC separates the user interface (UI) of an application into three main aspects:

The Model: A set of classes that describes the data you’re working with as well as the business rules for how the data can be changed and manipulated

The View: Defines how the application’s UI will be displayed

The Controller: A set of classes that handles communication from the user, overall application flow, and application-specific logic

The detail ASP.NET MVC pipeline is given below:

1. Routing - Routing is the first step in ASP.NET MVC pipeline. Typically, it is a pattern matching system that matches the incoming request to the registered URL patterns in the Route Table.

The UrlRoutingModule(System.Web.Routing.UrlRoutingModule) is a class which matches an incoming HTTP request to a registered route pattern in the RouteTable(System.Web.Routing.RouteTable).

1. Controller Initialization - The MvcHandler initiates the real processing inside ASP.NET MVC pipeline by using ProcessRequest method. This method uses the IControllerFactory instance (default is System.Web.Mvc.DefaultControllerFactory) to create corresponding controller.
2. Action Execution – Action execution occurs in the following steps:

 When the controller is initialized, the controller calls its own InvokeAction() method by passing the details of the chosen action method. This is handled by the IActionInvoker.

 After chosen of appropriate action method, model binders(default is System.Web.Mvc.DefaultModelBinder) retrieves the data from incoming HTTP request and do the data type conversion, data validation such as required or date format etc. and also take care of input values mapping to that action method parameters.

 Authentication Filter was introduced with ASP.NET MVC5 that run prior to authorization filter. It is used to authenticate a user. Authentication filter process user credentials in the request and provide a corresponding principal. Prior to ASP.NET MVC5, you use authorization filter for authentication and authorization to a user.

 By default, Authenticate attribute is used to perform Authentication. You can easily create your own custom authentication filter by implementing IAuthenticationFilter.

 Authorization filter allow you to perform authorization process for an authenticated user. For example, Role based authorization for users to access resources.

 By default, Authorize attribute is used to perform authorization. You can also make your own custom authorization filter by implementing IAuthorizationFilter.

 Action filters are executed before (OnActionExecuting) and after (OnActionExecuted) an action is executed. IActionFilter interface provides you two methods OnActionExecuting and OnActionExecuted methods which will be executed before and after an action gets executed respectively. You can also make your own custom ActionFilters filter by implementing IActionFilter. For more about filters refer this article Understanding ASP.NET MVC Filters and Attributes

 When action is executed, it process the user inputs with the help of model (Business Model or Data Model) and prepare Action Result.

4. Result Execution - Result execution occurs in the following steps:

 Result filters are executed before (OnResultExecuting) and after (OnResultExecuted) the ActionResult is executed. IResultFilter interface provides you two methods OnResultExecuting and OnResultExecuted methods which will be executed before and after an ActionResult gets executed respectively. You can also make your own custom ResultFilters filter by implementing IResultFilter.

 Action Result is prepared by performing operations on user inputs with the help of BAL or DAL. The Action Result type can be ViewResult, PartialViewResult, RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResult and EmptyResult.

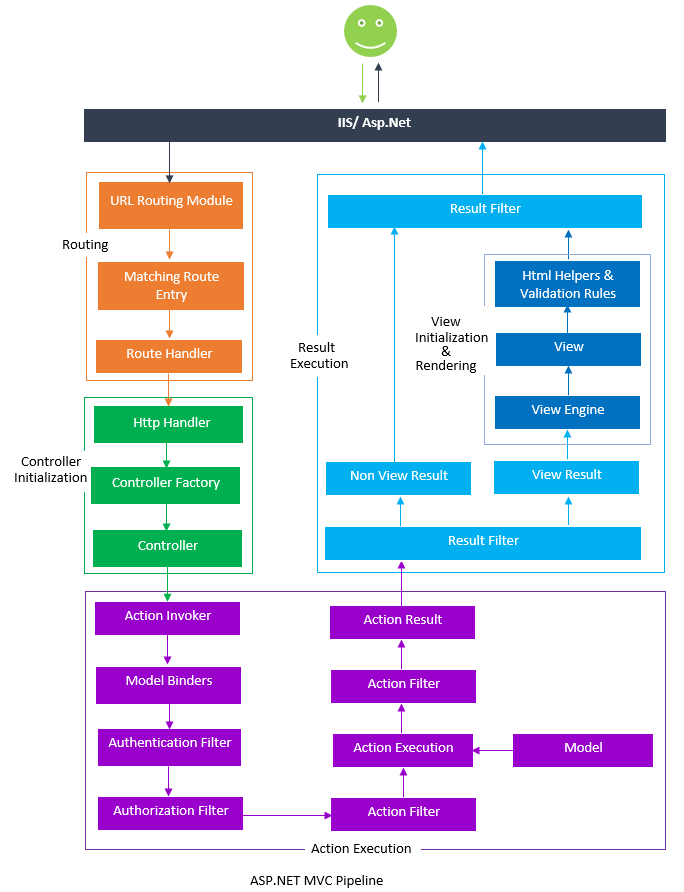
 Various Result type provided by the ASP.NET MVC can be categorized into two category- ViewResult type and NonViewResult type. The Result type which renders and returns an HTML page to the browser, falls into ViewResult category and other result type which returns only data either in text format, binary format or a JSON format, falls into NonViewResult category.

4.1 View Initialization and Rendering - View Initialization and Rendering execution occurs in the following steps:

 ViewResult type i.e. view and partial view are represented by IView (System.Web.Mvc.IView) interface and rendered by the appropriate View Engine.

 This process is handled by IViewEngine (System.Web.Mvc.IViewEngine) interface of the view engine. By default ASP.NET MVC provides WebForm and Razor view engines. You can also create your custom engine by using IViewEngine interface and can registered your custom view engine in to your ASP.NET MVC application as shown below:

 Html Helpers are used to write input fields, create links based on the routes, AJAX-enabled forms, links and much more. Html Helpers are extension methods of the HtmlHelper class and can be further extended very easily. In more complex scenario, it might render a form with client side validation with the help of JavaScript or jQuery.



DropDownList allows single item selection, whereas ListBox allows for multiple item selection (by setting the multiple attribute to multiple in the rendered markup

|  |  |  |
| --- | --- | --- |
| ViewData | ViewBag | TempData |
| It is Key-Value (ViewDataDictionary)Dictionary collection | It is a type object | It is Key-Value Dictionary collection |
| ViewData is a dictionary object and it is property of ControllerBase class | ViewBag is Dynamic property of ControllerBase class. | TempData is a dictionary object of TempDataDictionary class |
| ViewData is Faster than ViewBag | ViewBag is slower than ViewData | NA |
| ViewData is introduced in MVC 1.0 and available in MVC 1.0 and above | ViewBag is introduced in MVC 3.0 and available in MVC 3.0 and above | TempData is also introduced in MVC1.0 and available in MVC 1.0 and above. |
| ViewData  is also work with .net framework 3.5 and above | ViewBag  is only  work with .net framework 4.0 and above | TempData  is also work with .net framework 3.5 and above |
| Type Conversion code is required while enumerating | In depth, ViewBag is used dynamic, so there is no need to type conversion while enumerating. | Type Conversion code is required while enumerating |
| It value become null if redirection is occurred. | Same as ViewData | TempData is used to pass data between two consecutive requests. |
| It lies only during the current request. | Same as ViewData | TempData is only work during the current and subsequent request |

Global.asax contains :

AreaRegistration.RegisterAllAreas(); WebApiConfig.Register(GlobalConfiguration.Configuration);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

In ASP.NET MVC, we have many built-in *ActionResults* type:

* ViewResult
* PartialViewResult
* RedirectResult
* RedirectToRouteResult
* ContentResult
* JsonResult
* EmptyResult
* JavascriptResult
* FileResult
* HttpStatusCodeResult
* ViewResultBase

ViewData Sample:

public ActionResult Index()

{

      List<string> Student = new List<string>();

      Student.Add("Jignesh");

      Student.Add("Tejas");

      Student.Add("Rakesh");

      ViewData["Student"] = Student;

      return View();

}

//page code

<ul>

    <% foreach (var student in ViewData["Student"] as List<string>)

        { %>

    <li><%: student%></li>

    <% } %>

</ul>

//Controller Code

public ActionResult Index()

{

      List<string> Student = new List<string>();

      Student.Add("Jignesh");

      Student.Add("Tejas");

      Student.Add("Rakesh");

      ViewBag.Student = Student;

      return View();

}

//page code

<ul>

    <% foreach (var student in ViewBag.Student)

        { %>

    <li><%: student%></li>

    <% } %>

</ul>

public class FirstController : Controller

{

    // GET: First

    public ActionResult Index()

    {

        TempData["Message"] = "Hello MVC!";

        return new RedirectResult(@"~\Second\");

    }

}

<https://www.facebook.com/shivprasad.koirala/videos/689393794478113/>

1.If tempdata value is not read it is persisted for the subsequent request.

2. If tempdata value is read then it is not persisted for the next request.

3.If Keep() is called on TempData then the value will be maintained for the next request.

4.

we add test data in TempData in the first request and in the second subsequent request we access test data from TempData which we stored in the first request. However, you can't get the same data in the third request because TempData will be cleared out after second request.

Call TempData.Keep() to retain TempData values in a third consecutive request.

ASP.NET MVC TempData Vs Sessions

Although ASP.NET MVC TempData stores it’s content in Session state(internally) but it gets destroyed earlier than a session object. TempData gets destroyed immediately after it’s used in subsequent HTTP request, so no explicit action required. If we use a Session object for this purpose, we would have to destroy the object explicitly.

It’s best fit for scenarios when:

* we need to preserve data from current to subsequent request.
* passing error message to an error page.

|  |  |
| --- | --- |
| ASP.NET Web Forms | ASP.NET MVC |
| ASP.NET Web Forms uses Page controller pattern approach for rendering layout. In this approach, every page has it’s own controller i.e. code-behind file that processes the request. | ASP.NET MVC uses Front Controller approach. That approach means ,a common controller for all pages, processes the requests. |
| No separation of concerns. As we discussed that every page (.aspx) has it’s own controller (code behind i.e. aspx.cs/.vb file), so both are tightly coupled. | Very clean separation of concerns. View and Controller are neatly separate. |
| Because of this coupled behavior, automated testing is really difficult. | Testability is key feature in ASP.NET MVC. Test driven development is quite simple using this approach. Please [follow here](http://msdn.microsoft.com/en-us/magazine/dd942838.aspx) for demo on building testable applications. |
| In order to achieve stateful behavior, viewstate is used. Purpose was to give developers, the same experience of a typical WinForms application. | ASP.NET MVC approach is stateless as that of the web. So here no concept of viewstate. |
| Statefulness has a lots of problem for web environment in case of excessively large viewstate. Large viewstate means increase in page size. | As controller and view are not dependent and also no viewstate concept in ASP.NET MVC, so output is very clean. |
| ASP.NET WebForms model follows a Page Life cycle. | No Page Life cycle like WebForms. Request cycle is simple in ASP.NET MVC model. |
| Along with statefulness, microsoft tries to introduce server-side controls as in Windows applications. Purpose was to provide  somehow an abstraction to the details of HTML. In ASP.NET Web Forms, minimal knowledge of HTML, JavaScript and CSS is required. | In MVC, detailed knowledge of HTML, JavaScript and CSS is required. |
| Above abstraction was good but provides limited control over HTML, JavaScript and CSS which is necessary in many cases. | Full control over HTML, JavaScript and CSS. |
| With a lots of control libraries availability and limited knowledge of other related technologies, ASP.NET WebForms is RAD(Rapid Application Development) approach. | It’s a step back. For developers decrease in productivity. |
| It’s good for small scale applications with limited team size. | It’s better as well as recommended approach for large-scale applications where different teams are working together. |

Html Helpers in MVC

1. 1.Inline html helpers - **@foreach (string item in items)**
2. **{**
3. **<li>@item</li>**
4. **}**
5. **</ol>**
6. **}**

2.Built in html helpers

a. Standard html helpers -These helpers are used to render the most common types of HTML elements like as HTML text boxes, checkboxes etc.   
@Html.TextBox("Textbox1", "val")

@Html.DropDownList (“DropDownList1”, new SelectList(new [] {"Male", "Female"}))

b. Strongly typed html helpers - The strongly typed HTML helpers work on lambda expression. The model object is passed as a value to lambda expression, and you can select the field or property from model object to be used to set the id, name and value attributes of the HTML helper.

@Html.TextBoxFor(m=>m.Name)

@Html.DropDownListFor(m => m.Gender, new SelectList(new [] {"Male", "Female"}))

c. Template HTML helpers -

3.Custom Html Helpers -

1. **public static MvcHtmlString SubmitButton(this HtmlHelper helper, string**
2. **buttonText)**
3. **{**
4. **string str = "<input type=\"submit\" value=\"" + buttonText + "\" />";**
5. **return new MvcHtmlString(str);**
6. **}**

public class Item

{

public int ItemId { get; set; }

[Display(Name = "Current name")]

[Required(ErrorMessage = "Name is required")]

[StringLength(160)]

public string Name { get; set; }

[Required(ErrorMessage = "Price is required")]

[Range(0.01, 100.00,

ErrorMessage = "Price must be between 0.01 and 100.00")]

public decimal Price { get; set; }

}

@Html.DisplayNameFor(m => m.Name) would show 'Current name'.

@Html.DisplayNameFor(m => m.Price) would just show Price.

The Html.DisplayFor shows the value of the field.

**EditorFor** -If the property is typed as a primitive type (integer, string, and so on), the method renders an **HTML input** element for a text box  
  
**DisplayFor** - If the property is typed as a primitive type (integer, string, and so on), the method renders a **string** that represents the property value

**For EditorFor will render input control as below -**

Hide   Copy Code

<label for="DateOfJoining">DOJ :</label><input class="text-box single-line" data-val="true" data-val-date="The field DOJ : must be a date." data-val-required="The DOJ : field is required." id="DateOfJoining" name="DateOfJoining" type="datetime" value="7/8/2015 12:00:00 AM" />

**For DisplayFor will render string as below -**

Hide   Copy Code

<label for="DateOfJoining">DOJ :</label>08-00-2015

Action Helper in MVC

@Html.ActionLink : To generate anchor tag with href and anchor text in MVC

@Html.Partial() : for partial html pages

@Html.EditorFor(m => m.Title)

@Html.Label("GenreId")

@Html.DropDownListFor(m => m.GenreId, ViewBag.Genres as SelectList)

<span>@Html.Raw(model.Message)</span> - the Html.Raw method to ensure that the value is not encoded.

Html RouteLink we can provide user specific name to the route.

Difference is Html Partial returns an MvcHtmlString but RenderPartial outputs straight to the response.

@Html.Action, on the other hand, executes a separate controller action and displays the [results.@Html.RenderAction](mailto:results.@Html.RenderAction) writes directly to the response (which can bring a slight effi ciency gain).

@Html.RouteLink(

linkText: "route: Test",

routeName: "test",

routeValues: new {controller="section", action="Index", id=123}

)

As an example, the Html.TextBoxFor helper renders the following HTML for an album’s Title

property:

<input id="Title" name="Title" type="text"

value="For Those About To Rock We Salute You" />

Instead of using Html.TextBoxFor, you can switch to using the following code:

@Html.EditorFor(m => m.Title)

The EditorFor helper will render the same HTML as TextBoxFor; however, you can change the HTML using data annotations. If you think about the name of the helper (Editor), the name is more generic than the TextBox helper (which implies a specifi c type of input element). When using the templated helpers, you are asking the runtime to produce whatever “editor” it sees fi t. Let’s see what happens if you add a DataType annotation to the Title property:

[Required(ErrorMessage = "An Album Title is required")]

[StringLength(160)]

[DataType(DataType.MultilineText)]

public string Title { get; set; }

Now the EditorFor helper renders the following HTML:

<textarea class="text-box multi-line" id="Title" name="Title">

Let There Be Rock

</textarea>

@section Footer {

This is the <strong>footer</strong>.

}

The @section syntax specifi es the contents for a section defi ned in the layout.

<footer>@RenderSection("Footer")</footer>

The \_ViewStart.cshtml page can be used to remove this redundancy. The code within this file is executed before the code in any view placed in the same directory. This file is also recursively applied to any view within a subdirectory.

**IValidatableObject**

A *self*-*validating* model is a model object that knows how to validate itself. A model object can

announce this capability by implementing the IValidatableObject interface.

public class Order : IValidatableObject

{

public IEnumerable<ValidationResult> Validate(

ValidationContext validationContext)

{

if (LastName != null &&

LastName.Split(' ').Length > 10)

{

yield return new ValidationResult("The last name has too many words!",

new []{"LastName"});

}

}

// rest of Order implementation and properties

// ...

}

@using (Html.BeginForm("Search", "Home", FormMethod.Get)) {

<input type="text" name="q" />

<input type="submit" value="Search" />

}

Checkbox list in asp.net mvc code sample  
<http://www.c-sharpcorner.com/UploadFile/4d9083/creating-simple-checkboxlist-in-mvc-4-using-razor/>

Caching in ASP.Net MVC

public class TestController : Controller

{

[OutputCache(Duration=30, VaryByParam="none",**Location = OutputCacheLocation.Client**)]

public ActionResult Index()

{

return View();

}

}

#### Cache Location

There is one more important property of the OutputCache attribute class which is Location. This property will determine where will the cached data be stored on the server. Possible values for this property can be:

* Any (Default): Content is cached in three locations: the web server, any proxy servers, and the web browser.
* Client: Content is cached on the web browser.
* Server: Content is cached on the web server.
* ServerAndClient: Content is cached on the web server and and the web browser.
* None: Content is not cached anywhere.

In case we have multiple action methods across controllers needing the same caching behavior, we can put this caching values in the web.config and create a cacheprofile for it.

Hide   Copy Code

<caching>

<outputCacheSettings>

<outputCacheProfiles>

<add name="MyCacheProfile"

duration="30"

varyByParam="id"

location="Any" />

</outputCacheProfiles>

</outputCacheSettings>

</caching>

And to use these values in the action methods we just need to specify the CacheProfile name in the action method.

Hide   Copy Code

[OutputCache(CacheProfile = "MyCacheProfile")]

public ActionResult Dummy()

{

return View();

Partial Page caching is implemented using partial views

public ActionResult Index2()

{

return View();

}

[OutputCache(Duration = 10, VaryByParam = "none")]

public PartialViewResult PartialTest()

{

return PartialView("SamplePartial");

}

Html.ValidationSummary

The ValidationSummary helper displays an unordered list of all validation errors in the ModelState dictionary.

**Authentication in MVC**

Data Annotation in ASP.Net MVC – For validations

System.ComponentModel.DataAnnotations

[Required]

[StringLength(160, MinimumLength=3)]

public string FirstName { get; set; }

[RegularExpression(@"[A-Za-z0-9.\_%+-][+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}](mailto:+@[A-Za-z0-9.-]+\.%5bA-Za-z%5d%7b2,4%7d)",ErrorMessage="Email doesn't look like a valid email address.")]

public string Email { get; set; }

[Compare("Email")]

public string EmailConfirm { get; set; }

By default, the ASP.NET MVC framework executes validation logic during model binding. You can also explicitly request model binding using the UpdateModel or TryUpdateModel methods of a controller

If there are any errors in model state, ModelState.IsValid returns false.

Custom Annotation Example:

public class MaxWordsAttribute : ValidationAttribute

{

public MaxWordsAttribute(int maxWords)

{

\_maxWords = maxWords;

}

protected override ValidationResult IsValid(object value, ValidationContext validationContext)

{

if (value != null)

{

var valueAsString = value.ToString();

if (valueAsString.Split(' ').Length > \_maxWords)

{

return new ValidationResult("Too many words!");

}

}

return ValidationResult.Success;

}

private readonly int \_maxWords;

}

**Authorization in MVC**

[Authorize(Roles = "Administrator")]

public class AdministrationController : Controller

{

}

[Authorize]

public class ControlPanelController : Controller

{

public ActionResult SetTime()

{

}

[AllowAnonymous]

public ActionResult Login()

{

}

}

If you want to specify multiple allowed roles in a requirement then you can specify them as parameters to the RequireRole method; This normally occurs in ConfigureServices() in your Startup.cs file.

Copy

C#

options.AddPolicy("ElevatedRights", policy =>

policy.RequireRole("Administrator", "PowerUser", "BackupAdministrator"));

**AntiforgeryToken in MVC** - for CSRF

On client side use @Html.AntiForgeryToken()

On controller use [ValidateAntiForgeryToken()]  to check whether the request to the related controller comes with token or not which is checked internally by ASP.net MVC otherwise System.web.mvc.httpAntiforgeryToken exception is raised.

XSS attack in MVC –

[ValidateInput(false)]

public ActionResult PostProduct(Product obj)

{

return View(obj);

}

 by passes the HTML and Script tag checks for that action.

public class Product

{

public string ProductName { get; set; }

[AllowHtml]

public string ProductDescription { get; set; }

}

AllowHtml allow only Product description to have html tags if productname contains html tags then potentially dangerous form request is raised.

URL rewriting is focused on mapping one URL (new url) to another URL (old url) while routing is focused on mapping a URL to a resource.

|  |  |
| --- | --- |
| **Razor View Engine** | **Webform View Engine** |
| Razor Engine is an advanced view engine that was introduced with MVC3. This is not a new language but it is a new markup syntax. | Web Form Engine is the default view engine for the Asp.net MVC that is included with Asp.net MVC from the beginning. |
| Razor Engine is an advanced view engine that was introduced with MVC3. This is not a new language but it is a new markup syntax. | Web Form Engine is the default view engine for the Asp.net MVC that is included with Asp.net MVC from the beginning. |
| The namespace for Razor Engine is System.Web.Razor. | The namespace for Webform Engine is System.Web.Mvc.WebFormViewEngine. |
| The file extensions used with Razor Engine are different from Web Form Engine. It has .cshtml (Razor with C#) or .vbhtml (Razor with VB) extension for views, partial views, editor templates and for layout pages. | The file extensions used with Web Form Engine are also like Asp.net Web Forms. It has .aspx extension for views, .ascx extension for partial views & editor templates and .master extension for layout/master pages. |
| Razor has new and advance syntax that are compact, expressive and reduces typing. | Web Form Engine has the same syntax like Asp.net Web Forms uses for .aspx pages. |
| Razor syntax are easy to learn and much clean than Web Form syntax. Razor uses @ symbol to make the code like as:  @Html.ActionLink("SignUp", "SignUp") | Web Form syntax are borrowed from Asp.net Web Forms syntax that are mixed with html and sometimes make a view messy. Webform uses <% and %> delimiters to make the code like as:  <%: Html.ActionLink("SignUp", "SignUp") %> |
| By default, Razor Engine prevents XSS attacks (Cross-Site Scripting Attacks) means it encodes the script or html tags like <, > before rendering to view. | Web Form Engine does not prevent XSS attacks means any script saved in the database will be fired while rendering the page |
| Razor Engine is little bit slow as compared to Webform Engine. | Web Form Engine is faster than Razor Engine. |
| Razor Engine, doesn't support design mode in visual studio means you cannot see your page look and feel. | Web Form engine support design mode in visual studio means you can see your page look and feel without running the application. |
| Razor Engine support TDD (Test Driven Development) since it is not depend on System.Web.UI.Page class. | Web Form Engine doesn't support TDD (Test Driven Development) since it depend on System.Web.UI.Page class which makes the testing complex. |

**URL Helpers**

The URL helpers are similar to the HTML ActionLink and RouteLink helpers, but instead of

returning HTML they build URLs and return the URLs as strings.

1. Action - The Action URL helper is exactly like ActionLink, but does not return an anchor tag.

2.Content - The Content helper is particularly helpful because it can convert a relative application path to an

absolute application path.

3. RouteUrl- The RouteUrl helper follows the same pattern as the Action helper, but like RouteLink it accepts a

route name and does not have arguments for controller name and action name.

For optimizing style and script files

System.Web.Optimization.BundleTable.EnableOptimizations = false;

**Attribute based routing in MVC5**

[RoutePrefix("MyHome")]

Controller based routing in ASP.net MVC5

[Route("{action=index}")] //default action

public class HomeController : Controller

{

//new route: /MyHome/Index

public ActionResult Index()

{

return View();

}

//new route: /MyHome/About

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

//new route: /MyHome/Contact

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

}

Action Level Routing

[Route("users/about")] //route" /users/about

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

*routes.MapMvcAttributeRoutes()* method with in *RegisterRoutes()* method of RouteConfig.cs file.

Filters in ASP.Net MVC

1. Authentication Filters :

public interface IAuthenticationFilter  
{  
    void OnAuthentication(AuthenticationContext filterContext);

    void OnAuthenticationChallenge(AuthenticationChallengeContext filterContext);  
}

1. Authorization Filters :

public class CustAuthFilter : AuthorizeAttribute

{

public override void OnAuthorization(AuthorizationContext filterContext)

{

filterContext.Controller.ViewBag.AutherizationMessage = "Custom Authorization: Message from OnAuthorization method.";

}

}

IPrincipal user = httpContext.User;

if (!user.Identity.IsAuthenticated)

{

return false;

}

1. Action Filters :

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

filterContext.Controller.ViewBag.CustomActionMessage1 = "Custom Action Filter: Message from OnActionExecuting method.";

}

public override void OnActionExecuted(ActionExecutedContext filterContext)

{

filterContext.Controller.ViewBag.CustomActionMessage2 = "Custom Action Filter: Message from OnActionExecuted method.";

}

public override void OnResultExecuting(ResultExecutingContext filterContext)

{

filterContext.Controller.ViewBag.CustomActionMessage3 = "Custom Action Filter: Message from OnResultExecuting method.";

}

public override void OnResultExecuted(ResultExecutedContext filterContext)

{

filterContext.Controller.ViewBag.CustomActionMessage4 = "Custom Action Filter: Message from OnResultExecuted method.";

}

**4. Exception Filter:** This filter is used to capture any execptions if raised by controller or an action method. Create a class which will inherit FilterAttribute class and implement IExceptionFilter interface.

public class CustExceptionFilter : FilterAttribute, IExceptionFilter

{

public void OnException(ExceptionContext filterContext)

{

filterContext.Controller.ViewBag.ExceptionMessage = "Custom Exception: Message from OnException method.";

}

}

\_Layout.cshtml – works as master page in mvc

Instead of setting layout property of each and every page set that in \_ViewStart.cshtml

Global Filters:

Global action filters are applied to all actions in web application

protected void Application\_Start()  
{  
    AreaRegistration.RegisterAllAreas();  
   
    // Register global filter  
    GlobalFilters.Filters.Add(new MyActionFilterAttribute());  
      
    RegisterGlobalFilters(GlobalFilters.Filters);  
    RegisterRoutes(RouteTable.Routes);  
}

public class MyActionFilterAttribute : ActionFilterAttribute  
{  
    public override void OnResultExecuting(ResultExecutingContext context)  
    {  
        base.OnResultExecuting(filterContext);  
   
        context.RequestContext.HttpContext.Response.Write("<!-- Buuu! -->");  
    }  
}

### No Authentication

The first option is No Authentication and this option is used when you want to build a website that doesn't care who the visitors are.

### Individual User Accounts

The second option is Individual User Accounts and this is the traditional forms-based authentication where users can visit a website.

### Work and School Accounts

The third option is to use organizational accounts and this is typically used for business applications where you will be using active directory federation services.

A user logs into Windows desktop and can launch a browser to the application that sits inside the same firewall. ASP.NET can automatically pick up the user's identity, the one that was established by active directory. This option does not allow any anonymous access to the site, but again that is a configuration setting that can be changed.