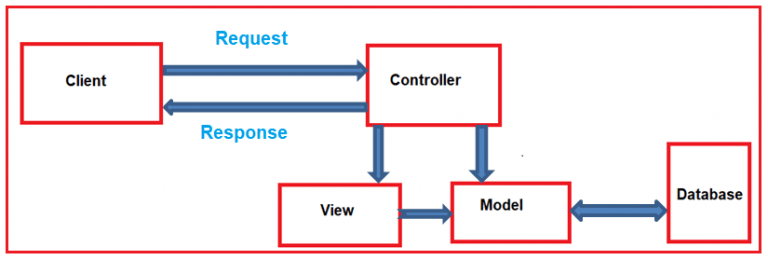
##### ****Explain MVC (Model-View-Controller) in general?****

MVC is an architectural software design pattern that is used for developing interactive applications where there would be user interaction involved and based on the user interaction some event handling occurs. So, this Model View Controller (MVC) design pattern is not only used for developing web-based applications but it can also be used for Desktop or mobile-based applications.

This MVC (Model-View-Controller) design pattern was introduced in the 1970s which basically divides an application into 3 major components such as **Model, View,**and **Controller**. The main objective of the **MVC design pattern** is the **separation of concerns.** It means the **domain model** and **business logic** are separated from the **user interface** (i.e. view). Thus, maintenance and testing of the application become simpler and easier.



##### ****The Model in MVC Design Pattern****:

The Model is the component in the MVC design pattern which manages that data i.e. state of the application in memory. It represents a set of classes that holds the business data as well as business logic, validation logic, and data access logic to manage the business data. For example, an Employee object (Model) might retrieve the data from a database, operate on it, validate the data, and then write the updated information back to an Employee table in the database. So the model is basically used to contain business (domain) data and business logic.

##### ****The View in MVC Design Pattern****:

The view is the component in the MVC design pattern which represents the user interface with which the end-user can interact. Basically, the view is used to render the domain data (i.e. business data). It creates the user interface with data from the model with which the end-user can interact. An example would be an edit view of a product table that displays text boxes, dropdown lists, and checkboxes based on the current state of a Product object.

##### ****The Controller in MVC Design Pattern****:

The Controller is the component in the MVC design pattern which contains the control flow logic. It is the one that will interact with both the models and views to control the flow of application execution. So, the controller component responds to user actions. Based on the user actions, the respective controller works with the model and view and sends responses back to the user.

##### ****What is ASP.NET MVC?****

In many ASP.NET MVC interviews, they asked this question to check whether you know the difference between MVC and ASP.NET MVC. The first thing is first. So, remember both ASP.NET MVC and MVC are two different things altogether.

The **ASP.NET MVC** is an open-source **web application development framework** from Microsoft that is based on the **MVC (Model-View-Controller)** architectural design pattern which provides a clean separation of code. This is the most customizable and extensible platform or framework provided by Microsoft.

So, the point that you need to remember is, **ASP.NET MVC is a Framework** whereas **MVC is a Design Pattern** and the ASP.NET MVC Framework is based on MVC Design Pattern.

##### ****What are Model, View, and Controller from ASP.NET MVC Point of view?****

##### ****Model in ASP.NET MVC****:

1. In MVC, the models are basically C#.NET or VB.NET classes.
2. The Models are basically used to manage the business data and business logic.
3. This is the component that can be accessed by both the controller and view.
4. The model component can be used by a controller to pass the data to a view.
5. It can also be used by a view, in order to display the data the in page (HTML output)

##### ****View in ASP.NET MVC****:

1. In the ASP.NET MVC application, the views are nothing but the cshtml or vbhtml pages without having a code-behind file.
2. All page specific HTML generation and formatting can be done inside view.
3. A request to view can only be made from a controller’s action method.
4. The view is only responsible for displaying the data.
5. By default, views are stored in the Views folder of an ASP.NET MVC application

##### ****The controller in ASP.NET MVC****:

1. In ASP.NET MVC, the controller is basically a C# or VB.NET class that inherits from **System.Web.Mvc.Controller.**
2. This is the component that has access to both the Models and Views in order to control the flow of the application execution.
3. The Controller class contains action methods that are going to respond to the incoming URL.
4. A controller can access and use the model classes to pass the data to a view.
5. By default, controllers have stored in the Controllers folder an ASP.NET MVC application.

**What are the Advantages of ASP.NET MVC?**

This is one of the ASP.NET MVC questions asked in the interview to check whether you know why you are using MVC or not. So let us discuss what the advantages of using ASP.NET MVC are or you can say why we need to use ASP.NET MVC to develop a web application.

1. The ASP.NET MVC Views are lightweight as compared to ASP.NET Web Forms because it does not use view state or any server-based forms or controls to render the data.
2. As we are separating the view from the rest of the application which enables us to change of view (technology) in the future without affecting the rest of the application. For example, we might have views in Silverlight or HTML5.
3. Each developer based on his expertise or experience can work on different parts of the application without stepping on each other toes. For example, one developer can work on the view while the second developer can work on the controller logic and the third developer can focus on the business logic in the model by interacting with the database.
4. With the introduction of Attribute Routing, now it is very easy to create Restful or User-friendly URL’s which enable SEO to rank your page high.
5. The ASP.NET MVC provides better support for test-driven development (TDD). This is because we can focus on one aspect at a time i.e. we can focus on the view without worrying about business logic as each component can be developed independently and also can be mocked easily.
6. As the MVC design pattern divides the application into three major components such asModel, View, and Controller which make it easier to manage the application complexity.
7. MVC framework components were designed to be pluggable and extensibleand therefore can be replaced or customized in the future very easily
8. AS the MVC framework is built on top of ASP.NET, so, we can use mostof the ASP.NET features such as the authentication and authorization scenarios, membership and roles, caching, session, and many more.

**What is the difference between 3-layer architecture and MVC architecture?**

The 3-layer architecture separates the application into 3 components which consist of the Presentation Layer,Business Layer, and Data Access Layer. In 3-layer architecture, the user is going to interacts with the Presentation layer only. 3-layer is a linear architecture.

The MVC architecture separates the application into three major components such as Model, View, and Controller. In MVC architecture, the user is going to interacts with the controller with the help of a view. MVC is a triangle architecture.

MVC does not replace 3-layer architecture. Typically MVC and 3-layer architecture are used together and the MVC Design Pattern acts as the Presentation layer of the application.

**What are the differences between ASP.NET MVC and ASP.NET WebForms?**

This is one of the Frequently asked ASP.NET MVC interview questions and answers.

The **ASP.NET Web Forms** uses the **Page controller** pattern approach for rendering the layout whereas **ASP.NET MVC** uses the **Front controller**approach for rendering the layout. In the case of the Page controller Pattern approach, every aspx page having its own controller i.e. the code-behind file which is used to process the incoming request. Whereas, in ASP.NET MVC, a common controller can process the requests for all pages.

1. **What is the difference between ASP.NET MVC and ASP.NET WebForms?**
2. **Is ASP.NET MVC going to replace ASP.NET WebForms?**

The first and important thing that you need to remember is, the **ASP.NET MVC** Framework is not going to replace the **ASP.NET WebForms**Framework. Both technologies exist and both can be used to develop ASP.NET applications. Here we are going to discuss, what are the advantages and disadvantages of both these technologies one over another.

1. In ASP.NET WebForms there is no separation of concerns as every page (.aspx) has its own controller (i.e. code-behind i.e. aspx.cs/.vb file). As a result, both are tightly coupled. Whereas the main objective of ASP.NET MVC is the clean separation of concerns as View and Controller are cleanly separate and also they are not tightly coupled.
2. Because of the tightly coupled behavior in ASP.NET Web Forms (i.e. .aspx page and its code-behind file), unit testing is really a nightmare for a developer. Whereas in ASP.NET MVC unit testing is achieved very easily as they are not tightly coupled.
3. ASP.NET Web Forms follows a Page Life cycle. Whereas in ASP.NET MVC there is no such Page Life cycle like WebForms. The request cycle is simple in the ASP.NET MVC model.
4. ASP.NET MVC views are lightweight, as they do not use ViewState whereas ASP.NET Web Forms pages are not lightweight as they use ViewState.
5. Complex applications can be easily managed in ASP.NET MVC because of the separation of concerns. Different components of the application can be divided into Model, View, and Controller.
6. ASP.NET web form has file-based URLs means the file name that exists in the URLs must have its physical existence whereas ASP.NET MVC has route-based URLs means URLs are divided into controllers and actions and moreover it is based on controller not on physical file.
7. ASP.NET web form has Master Pages for consistent looks and feels whereas ASP.NET MVC has Layouts for consistent looks and feels.
8. ASP.NET web form has User Controls for code re-usability whereas ASP.NET MVC has Partial Views for code re-usability.
9. ASP.NET Web Form is not Open Source whereas ASP.NET Web MVC is an Open Source.
10. ASP.NET Web Form has server controls whereas ASP.NET MVC has HTML helpers.

**Can you please explain the request flow in the ASP.NET MVC framework?**

**The Request flow for the ASP.NET MVC framework is as follows:**

In the ASP.NET MVC application when the client makes a request (i.e. HTTP Request) for a resource from the browser then that request is first Received by the Routing Engine. Once the Routing engine receives the HTTP Request, then it figures out the URL Pattern of the incoming request and checks if that URL pattern is present in the Route table. If there is no matching URL Pattern found in the routing table for the incoming HTTP request URL Pattern, then it simply returns a 404 HTTP status code to the client. If it found a matching URL pattern for the incoming request in the Route Table, then it fetches the corresponding handler information and forwards the request to the appropriate controller and action method.

The controller then plays its role and executes the action method. While executing the action method if needed then it will work with the model component in order to serve the request. Once it works with the model component, then it selects a view and passes that model data to that view, and that view will transform the model data and generates an appropriate response that is rendered to the client.

The point that you need to be noted is that the difference between WebForms and MVC is that in MVC we don’t have the Page concept whereas, in Legacy like ASP.NET Web Forms application, we do have pages like .aspx. So if your question is, Then how do the requests are been pointed in MVC? The answer is via Controller. This special class is responsible for generating the response and sending the data back to the client.

**What are the important namespaces used in ASP.NET MVC?**

This question is not that important, but you should what are the important namespaces you used in the ASP.NET MVC application. The following are some of the important namespaces:

1. **System.Web.Mvc** – This namespace contains classes and interfaces that support the MVC pattern for ASP.NET Web applications. This namespace includes classes that represent controllers, controller factories, action results, views, partial views, and model binders.
2. **System.Web.Mvc.Ajax** – This namespace contains classes that support Ajax scripting in an ASP.NET MVC application. The namespace includes support for Ajax scripts and Ajax option settings as well.
3. **System.Web.Mvc.Html** – This namespace contains classes that help render HTML controls in an MVC application. This namespace includes classes that support forms, input controls, links, partial views, and validation.

**What is ViewModel in ASP.NET MVC?**

In ASP.NET MVC, ViewModel is a class that contains the fields which are represented in the strongly-typed view. It is used to pass data from the controller to a strongly-typed view.

**Key Points about ViewModel**

1. ViewModel contains fields that are represented in the view (for LabelFor, EditorFor, DisplayFor helpers).
2. ViewModel can have specific validation rules using data annotations.
3. ViewModel can have multiple entities or objects from different data models or data source.

**What are Action methods in ASP.NET MVC?**

All public methods of a Controller in ASP.NET MVC application that respond to the incoming URL are considered as Action Methods.

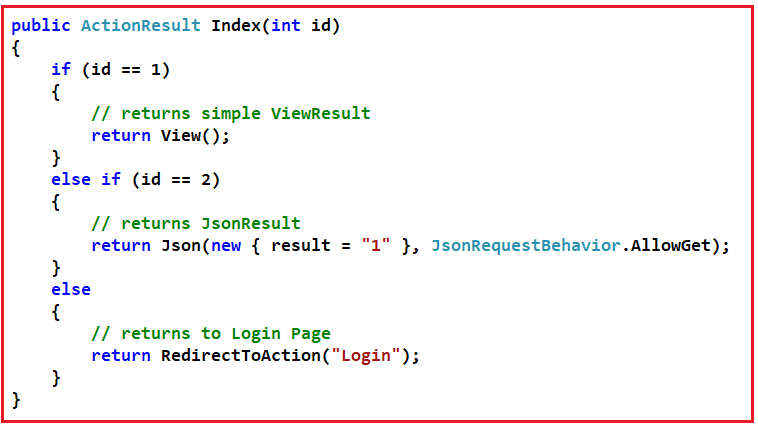
Controller actions are methods defined in the controller class and responsible to perform required operations on the user’s inputs like form values, query strings values, etc. with the help of the Model and passing the results back to the View. Asp.net MVC has the following built-in ActionResults Type and Helper methods:

1. **ViewResult** – Returns a ViewResult which renders the specified or default view by using the controller View() helper method.
2. **PartialViewResult** – Returns a PartialViewResult which renders the specified or default partial view (means a view without its layout) by using the controller PartialView() helper method.
3. **RedirectResult** – Returns a RedirectResult which Issues an HTTP 301 or 302 redirection to a specific URL by using controller Redirect() helper method.
4. **RedirectToRouteResult** – Returns a RedirectToRouteResult which Issues an HTTP 301 or 302 redirection to an action method or specific route entry by using controller RedirectToAction(), RedirectToActionPermanent(), RedirectToRoute(), RedirectToRoutePermanent() helper methods.
5. **ContentResult** – Returns a ContentResult which renders raw text like “Hello, DotNet Tutorials!” by using the controller Content() helper method.
6. **JsonResult** – Returns a JsonResult which serializes an object in JSON format ( like as “{ “Message”: Hello, World! }”) and renders it by using controller Json() helper method.
7. **JavaScriptResult** – Returns a JavaScriptResult which renders a snippet of JavaScript code like as “function hello() { alert(Hello, World!); }” by using controller JavaScript() helper method. This is used only in AJAX scenarios.
8. **FileResult** – Returns a FileResult which renders the contents of a file like PDF, DOC, Excel, etc. by using the controller File() helper method.
9. **EmptyResult** – Returns no result returned by an action. This has no controller helper method.
10. **HttpNotFoundResult** – Returns an HttpNotFoundResult which renders a 404 HTTP Status Code response by using controller HttpNotFound() helper method.
11. **HttpUnauthorizedResult** – Returns an HttpUnauthorizedResult which renders a 401 HTTP Status Code (means “not authorized”) response. This has no controller helper method. This is used for authentication (forms authentication or Windows authentication) to ask the user to log in.
12. **HttpStatusCodeResult** – Returns an HttpStatusCodeResult which renders a specified HTTP code response. This has no controller helper method.

**Note:** All public methods of a Controller in an ASP.NET MVC application are considered as Action Methods by default. If we want our controller to have a Non-Action Method, we need to explicitly mark it with the NonAction attribute as follows:

**What is ActionResult and how is it different from others?**

The ActionResult class is the base class for all action results. An action result can be of type ViewResult, JsonResult, RedirectResult and so on. Hence, when your action method returns multiple results based on different conditions, ActionResult is the best choice. Since it can return any type of result.



##### ****How to make a Non-Action method in ASP.NET MVC?****

By default, the ASP.NET MVC framework treats all public methods of a controller class as action methods. If you do not want a public method to be an action method, you must mark that method with the NonActionAttribute attribute.

##### ****Can you change the action method name?****

We can also change the action method name by using the ActionName attribute. Now action method will be called by the name defined by the ActionName attribute.

**[**ActionName**(**"DoAction"**)]**

**public** ActionResult DoSomething**()**

**{**

//TODO: return View();

**}**

Now, DoSomething action will be identified and called by the name DoAction.

##### ****How to restrict an action method to be invoked only by HTTP GET, POST, PUT or DELETE?****

By default, each and every action method can be invoked by an HTTP request (i.e. GET, PUT, POST, and DELETE). But you can restrict an action to be invoked only by a specific HTTP request by applying HttpGet or HttpPost or HttpPut or HttpDelete attribute.

If you want to restrict an action method for HTTP Get request only then decorate it with HttpGet action method selector attribute as given below:

**[**HttpGet**]**

**public** ActionResult Index**()**

**{**

//TODO:

**return** View**()**;

**}**

##### ****How to determine an action method is invoked by HTTP GET or POST?****

By using the HttpMethod property of HttpRequestBase class, you can find out whether an action is invoked by HTTP GET or POST.

**public** ActionResult Index**(int**? id**)**

**{**

**if** **(**Request.HttpMethod == "GET"**)**

**{**

//TODO:

**}**

**else** **if** **(**Request.HttpMethod == "POST"**)**

**{**

//TODO:

**}**

**else**

**{**

//TODO:

**}**

**return** View**()**;

**}**

##### ****How to determine an AJAX request?****

We can determine an AJAX request by using Request.IsAjaxRequest() method. It will return true if the request is an AJAX request else returns false.

**public** ActionResult DoSomething**()**

**{**

**if** **(**Request.IsAjaxRequest**())**

**{**

//TODO:

**}**

**return** View**()**;

**}**

##### ****What is Child action and how to invoke it?****

Suppose we have a scenario where we have one action method and we don’t want that action method to be invoked via URL rather we want that action method to be invoked by other actions (from parent views) of our application. Then in such scenarios, **ChildActionOnly**Attribute can be handy. So, when we decorate an action method with the **ChildActionOnly** attribute, then it is called child action in ASP.NET MVC Application and child action methods are only accessible by a child request. That means once an action method becomes a child action, then it will not respond to the URL requests rather it will be invoked by other action methods of your application.

So, an action method that is decorated with the **ChildActionOnly** attribute cannot be called independently. It always will be called within a parent view otherwise it would give an error.

**[**ChildActionOnly**]**

**public** ActionResult MenuBar**()**

**{**

//TODO:

**return** PartialView**()**;

**}**

A child action is invoked by using @Html.RenderAction or @Html.Action helper methods from inside of a view.

##### ****What are Partial Views in ASP.NET MVC and it’s needed?****

A partial view is like user control in ASP.NET Webforms that are used for code re-usability. Partial views help us to reduce code duplication. Hence partial views are reusable views like Header and Footer views. We can use the partial view to display blog comments, product category, social bookmarks buttons, a dynamic ticker, calendar, etc.

It is best practice to create a partial view in the shared folder and the partial view name is preceded by “\_”, but it is not mandatory. The “\_” before view name specifies that it is a reusable component i.e. partial view.

##### ****What are the Layouts in ASP.NET MVC?****

Layouts are used to maintain a consistent look and feel across multiple views within the ASP.NET MVC application. As compared to Web Forms, layouts serve the same purpose as master pages but offer a simple syntax and greater flexibility. A basic structure of layout is given below:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8" **/>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**@ViewBag.Title**</title>**

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

**</head>**

**<body>**

@RenderBody() @Scripts.Render("~/bundles/jquery")

@RenderSection("scripts", required: false)

**</body>**

**</html>**

**You can use a layout to define a common template for your site. A layout can be declared at the top of view as:**

@{

Layout = "~/Views/Shared/SiteLayout.cshtml";

}

##### ****What are Sections in ASP.NET MVC?****

A section allows us to specify a region of content within a layout. It expects one parameter which is the name of the section. If you don’t provide that, an exception will be thrown. A section on a layout page can be defined by using the following code.  
**@section header{ <h1>Header Content</h1> }**

You can render the above-defined section header on the content page as given below:  
**@RenderSection(“header”)**

By default, sections are mandatory. To make sections optional just provide the second parameter value as false, which is a Boolean value.  
**@RenderSection(“header”,false)**

**Note:** A view can define only those sections that are referred to in the layout page otherwise an exception will be thrown.

##### ****What are RenderBody and RenderPage in ASP.NET MVC?****

This is one of the Frequently asked ASP.NET MVC interview questions and answers. RenderBody method exists in the Layout page to render child page/view. It is just like the ContentPlaceHolder on the master page. A layout page can have only one RenderBody method.

**<body>**

@RenderBody()

@RenderPage("~/Views/Shared/\_Header.cshtml")

@RenderPage("~/Views/Shared/\_Footer.cshtml")

@RenderSection("scripts", false)

@section scripts{

**<script** src="~/Scripts/jquery-1.7.1.min.js"**></script>**

}

**</body>**

The reader page method also exists in the Layout page to render another page that exists in your application. A layout page can have multiple RenderPage methods.

**@RenderPage(“~/Views/Shared/\_Header.cshtml”)**

##### ****What are the Styles.Render and Scripts.Render?****

Style.Render is used to render a bundle of CSS files defined within BundleConfig.cs files. Styles.Render create style tag(s) for the CSS bundle. Like Style.Render, Scripts.Render is also used to render a bundle of Script files by rendering script tag(s) for the Script bundle.

**public** **class** BundleConfig

**{**

// For more information on bundling, visit http://go.microsoft.com/fwlink/?LinkId=301862

**public** **static** **void** RegisterBundles**(**BundleCollection bundles**)**

**{**

bundles.Add**(**new ScriptBundle**(**"~/bundles/jquery"**)**.Include**(**

"~/Scripts/jquery-{version}.js"**))**;

// Use the development version of Modernizr to develop with and learn from. Then, when you're

// ready for production, use the build tool at http://modernizr.com to pick only the tests you need.

bundles.Add**(**new ScriptBundle**(**"~/bundles/modernizr"**)**.Include**(**

"~/Scripts/modernizr-\*"**))**;

bundles.Add**(**new ScriptBundle**(**"~/bundles/bootstrap"**)**.Include**(**

"~/Scripts/bootstrap.js",

"~/Scripts/respond.js"**))**;

bundles.Add**(**new StyleBundle**(**"~/Content/css"**)**.Include**(**

"~/Content/bootstrap.css",

"~/Content/site.css"**))**;

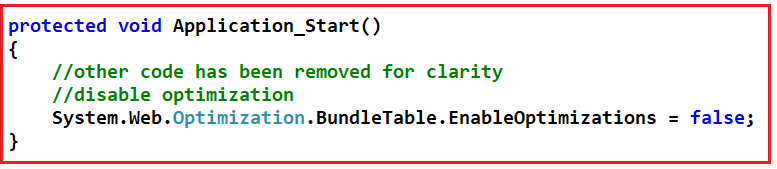
**}**

**}**

The Styles.Render and Scripts.Render generate multiple styles and script tags for each item in the CSS bundle and Script bundle when optimizations are disabled. When optimizations are enabled, Styles.Render and Scripts.Render generate a single style and script tag to a version-stamped URL which represents the entire bundle for CSS and Scripts.

##### ****How to enable and disable optimizations in ASP.NET MVC?****

We can enable and disable optimizations by setting the EnableOptimizations property of BundleTable class to true or false within Global.asax.cs file as shown below.



**What is \_ViewStart?**

The \_ViewStart.cshml page is used to serve a common layout page(s) for a group of views. 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.

**What are the different ways of rendering layout in ASP.NET MVC?**

There are following four different ways of rendering layout in ASP.NET MVC:

**Using \_ViewStart file in the root directory of the Views folder:**

The \_ViewStart file within the Views folder is used to serve the default Layout page for your ASP.NET MVC application. You can also change the default rendering of layouts within \_ViewStart file based on the controller as shown below:

@**{**

var controller = HttpContext.Current.Request.RequestContext.RouteData.Values**[**"Controller"**]**.ToSt ring**()**;

**string** layout = "";

**if** **(**controller == "Admin"**)**

**{**

layout = "~/Views/Shared/\_AdminLayout.cshtml";

**}**

**else**

**{**

layout = "~/Views/Shared/\_Layout.cshtml";

**}**

Layout = layout;

**}**

**Adding the \_ViewStart file in each of the directories**

You can also set the default layout for a particular directory by putting the \_ViewStart file in each of the directories with the required Layout information as shown below:

**Defining Layout within each view on the top**

@{

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

**Returning Layout from ActionResult**

**public** ActionResult Index**()**

**{**

RegisterModel model = new RegisterModel**()**;

//TODO:

**return** View**(**"Index", "\_AdminLayout", model**)**;

**}**

**What is the App\_Start folder in ASP.NET MVC?**

App\_Start folder has been introduced in MVC4. It contains various configurations files like as

BundleConfig.cs, FilterConfig.cs, RouteConfig.cs, WebApiConfig.cs for your application. All these settings are registered within the Application\_Start method of Global.asax.cs file.

**BundleConfig.cs** – This is used to create and register bundles for CSS and JS files. By default, various bundles are added in these files including jQuery, jQueryUI, jQuery Validation, Modernizr, and Site CSS.

**FIlterConfig.cs** – This is used to register global MVC filters like error filters, action filters, etc. By default, it contains the HandleErrorAttribute filter.

**RouteConfig.cs** – This is used to register various route patterns for your ASP.NET MVC application. By default, one route is registered here named as Default Route.

**WebApiConfig.cs** – This is used to register various WEB API routes like ASP.NET MVC, as well as set any additional WEB API configuration settings.

**What are the different ways of returning/rendering a view in ASP.NET MVC?**

1. **Return View() –** This tells MVC to generate HTML to be displayed for the specified view and sends it to the browser. This acts as a Server.Transfer() in ASP.NET WebForm.
2. **Return RedirectToAction()** – This tells MVC to redirect to specified action instead of rendering HTML. In this case, the browser receives the redirect notification and make a new request for the specified action. This acts like Response.Redirect() in ASP.NET WebForm.

Moreover, RedirectToAction constructs a redirect URL to a specific action/controller in your application and use the routing table to generate the correct URL. RedirectToAction causes the browser to receive a 302 redirect within your application and gives you an easier way to work with your router table.

1. **Return Redirect()** – This tells MVC to redirect to a specified URL instead of rendering HTML. In this case, the browser receives the redirect notification and make a new request for the specified URL. This also acts like a Response.Redirect() in ASP.NET WebForm. In this case, you have to specify the full URL to redirect.

Moreover, Redirect also causes the browser to receive a 302 redirect within your application, but you have to construct the URLs yourself.

1. **Return RedirectToRoute()** – This tells MVC to look up the specified route into the Route table that is defined in global.asax and then redirect to that controller/action defined in that route. This also make a new request like RedirectToAction().

**Points to Remember:**

1. Return View doesn’t make a new request, it just renders the view without changing URLs in the browser’s address bar.
2. The Return RedirectToAction makes a new request and the URL in the browser’s address bar is updated with the generated URL by MVC.
3. Return Redirect also makes a new request and the URL in the browser’s address bar is updated, but you have to specify the full URL to redirect
4. Between RedirectToAction and Redirect, the best practice is to use RedirectToAction for anything dealing with your application actions/controllers. If you use Redirect and provide the URL, you’ll need to modify those URLs manually when you change the routing table.
5. RedirectToRoute redirects to a specific route defined in the Route table.

**What is Area in ASP.NET MVC?**

Areas were introduced in Asp.net MVC2 which allows us to organize models, views, and controllers into separate functional sections of the application, such as administration, billing, customer support, and so on. This is very helpful in a large web application, where all the controllers, views, and models have a single set of folders and that becomes difficult to manage.

Each MVC area has its own folder structure which allows us to keep separate controllers, views, and models. This also helps the multiple developers to work on the same web application without interfering with one another.

**How to register Area in ASP.NET MVC?**

Before working with the area, make sure you have registered your area within the Application\_Start method in Global.asax as shown below.

**protected** **void** Application\_Start**()**

**{**

//Register all application Areas

AreaRegistration.RegisterAllAreas**()**;

**}**

Always remember the order of registering the Areas must be on top so that all of the settings, filters, and routes registered for the applications will also apply to the Areas.

**What is Scaffolding?**

We (developers) spent most of our time writing code for CRUD operations that are connecting to a database and performing operations like Create, Retrieve, Update, and Delete. Microsoft introduces a very powerful feature called Scaffolding that does the job of writing CRUD operations code for us.

Scaffolding is basically a Code Generation framework. Scaffolding Engine generates basic controllers as well as views for the models using Microsoft’s T4 template. Scaffolding blends with Entity Framework and creates the instance for the mapped entity model and generates code of all CRUD Operations. Further, we can edit or customize this auto-generated code according to our needs. As a result, we get the basic structure for a tedious and repetitive task.

Following are a few advantages of Scaffolding:

1. RAD approach for data-driven web applications.
2. Minimal effort to improve the Views.
3. Data Validation based on the database schema.
4. Easily created filters for the foreign key or boolean fields.

**What are the differences between ViewData, ViewBag, TempData, and Session?**

In ASP.NET MVC there are three ways – ViewData, ViewBag, and TempData to pass data from the controller to view and in the next request. Like WebForm, we can also use Session to persist data during a user session.

**ViewData in ASP.NET MVC**

**public ViewDataDictionary ViewData { get; set; }**

1. ViewData is a dictionary object that is derived from ViewDataDictionary class.
2. The ViewData is used to pass data from the controller to the corresponding view.
3. Its life lies only during the current request.
4. If redirection occurs then its value becomes null.
5. It’s required typecasting for getting data and check for null values to avoid the error.

**ViewBag in ASP.NET MVC**

**public Object ViewBag { get; set;}**

1. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
2. Basically, it is a wrapper around the ViewData and also used to pass data from the controller to the corresponding view.
3. Its life also lies only during the current request.
4. If redirection occurs then its value becomes null.
5. It doesn’t require typecasting for getting data.

**TempData in ASP.NET MVC**

**public TempDataDictionary TempData { get; set; }**

1. TempData is a dictionary object that is derived from the TempDataDictionary class and stored in a short life session.
2. TempData is used to pass data from current request to subsequent request (means redirecting from one page to another).
3. Its life is very short and lies only until the target view is fully loaded.
4. It’s required typecasting for getting data and check for null values to avoid the error.
5. It’s used to store only one time messages like error messages, validation messages.

**Session in ASP.NET MVC**

**public HttpSessionStateBase Session { get;set; }**

1. In ASP.NET MVC, Session is a property of Controller class whose type is HttpSessionStateBase.
2. The session is also used to pass data within the ASP.NET MVC application and Unlike TempData, it persists data for a user session until it is timeout (by default session timeout is 20 minutes).
3. A session is valid for all requests, not for a single redirect.
4. It’s also required typecasting for getting data and check for null values to avoid the error.

**How to control Session behavior in ASP.NET MVC?**

By default, ASP.NET MVC support session state. The session is used to store data values across requests. Whether you store some data values within the session or not ASP.NET MVC must manage the session state for all the controllers in your application that is time-consuming. Since, the session is stored in the server-side and consumes server memory, hence it also affects your application performance.

If some of the controllers of your ASP.NET MVC application are not using session state features, you can disable session for those controllers and can gain slight performance improvement of your application. You can simplify the session state for your application by using available options for the session state.

SessionState attribute provides you more control over the behavior of session-state by specifying the value of SessionStateBehavior enumeration as shown below:

**Value          Description**  
**Default** => The default ASP.NET behavior is used to determine the session state behavior.  
**Disabled** => Session state is disabled entirely.  
**ReadOnly** => Read-only session state behavior is enabled.  
**Required**=> Full read-write session state behavior is enabled.

**How is TempData related to Session in ASP.NET MVC?**

In ASP.NET MVC, TempData uses a session state for storing the data values across requests. Hence, when you will disable the session state for the controller, it will throw the exception.

**What is a View Engine in ASP.NET MVC application?**

A View Engine in ASP.NET MVC application is used to translate the views to HTML and then render the HTML in a browser.

The point that you need to remember is, the View Engine in ASP.NET MVC application having its own markup syntax. As discussed, the View Engine in the ASP.NET MVC application is responsible for converting the Views into HTML markup and then rendering the HTML in a browser. Initially, the ASP.NET MVC framework comes with one view engine i.e. web forms (ASPX) view engine and from the ASP.NET MVC3 framework a new view engine i.e. Razor view engine comes. Now, it is also possible to use other third-party view engines such as Spark, NHaml, etc.

**How does the View Engine work in the ASP.NET MVC application?**

Each and every view engine in ASP.NET MVC application has the following three major components:

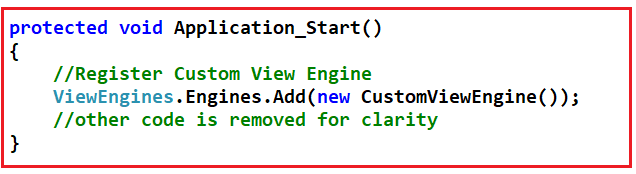
1. **ViewEngine Class** – The **ViewEngine** class implements the **IViewEngine** interface and the responsibility of this class is locating the view templates.
2. **View Class** – The View class implements the **IView** interface and the responsibility of this class is to combine the template with data from the current context and then convert it to HTML markup.
3. **Template Parsing Engine** – This parses the template and compiles the view into executable code.

**What is Razor View Engine?**

The Razor Engine was introduced in ASP.NET MVC3. This is an advanced View Engine that provides a new way to write markup syntax which will reduce the typing of writing the complete HTML tag. the Razor syntax is easy to learn and much cleaner than Web Form syntax. Razor uses @ symbol to write markup.

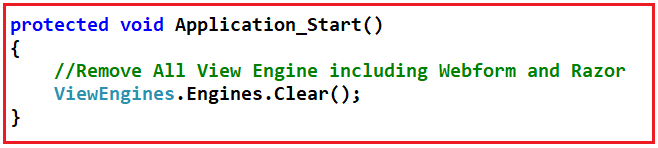
**How to register Custom View Engine in ASP.NET MVC?**

In order to use the custom View Engine, first, we need to register it in the **Application\_Start()** method of the global.asax.cs file so that the framework will use our custom View Engine instead of the default one.



##### ****Can you remove the default View Engine in ASP.NET MVC?****

Yes, we can remove default view engines (i.e. Razor and WebForm) provided by ASP.NET MVC.



**What is the difference between the Razor and WebForm engine?**

The main differences between ASP.NET Web Form and ASP.NET MVC are given below:

1. Razor Engine was introduced with ASP.NET MVC3 Framework. It is a new markup syntax whereas Web Form View Engine is the default View Engine from the beginning of ASP.NET MVC Framework.
2. The Razor Engine belongs to **System.Web.Razor** namespace whereas the Webform Engine belongs to **System.Web.Mvc.WebFormViewEngine**namespace.
3. Razor View Engine has new and advances syntax that is compact, expressive, and reduces typing whereas Web Form View Engine has the same syntax as WebForms.
4. The Razor View Engine syntaxes are easy to learn as well as much cleaner than the Web Form View Engine syntax. It uses @ symbol to switch between the C# code and HTML code whereas the Web Form View Engine syntaxes are borrowed from ASP.NET WebForms. The Webform uses <% and %> delimiters to switch between the C# and HTML code.
5. By default, the Razor View Engine prevents Cross-Site Scripting Attacks (XSS attacks). That means it encodes the script or HTML tags before rendering to the view. On the other hand, the Web Form View Engine does not prevent Cross-Site Scripting Attacks. That means any script saved in the database will be fired while rendering the page.
6. The Razor View Engine doesn’t support the design mode in the visual studio. That means we cannot see how our page looks at the time developing. Whereas the Web Form View Engine supports the design mode in the visual studio. That means we can see how our page looks at the time of development without running the application.

**What are HTML Helpers in ASP.NET MVC?**

The HTML Helpers in ASP.NET MVC application are nothing but methods returning an HTML string that can render an HTML tag. For example, a link, a textbox, a label, or other form elements.

Developers who have worked with ASP.NET Web Forms can map the HTML helper methods to Web Form Controls because both serve the same purpose. But HTML helpers are comparatively lightweight because they don’t have view state and event like Web Form Controls. Along with the built-in HTML helpers, it is also possible to create our own custom helper methods to fulfill our specific purpose. For example, we can create custom HTML Helpers to render more complex content such as a menu strip or an HTML table for displaying database data.

**Note:** The HTML helpers are implemented as extension methods.

**Is it mandatory to use HTML helpers?**

No, we can type the required HTML but using HTML helpers will greatly reduce the amount of HTML that we have to write in a view. Views should be as simple as possible. All the complicated logic to generate a control can be encapsulated into the helper to keep views simple.

**What is the Difference between Html.TextBox and Html.TextBoxFor?**

**Html.TextBox** is not a strongly typed helper method and hence it doesn’t require a strongly typed view. That means we can hardcode whatever name we want. On the other hand Html.TextBoxFor is a strongly typed HTML Helper method and hence it requires a strongly typed view and the name is given by using the lambda expression.

The Strongly typed HTML helper methods provide compile-time error checking. In most of the real-time application, we use strongly typed views, so we prefer to use Html.TextBoxFor over their counterpart HTML.TextBox.

The most important point that we need to remember is, whether we use the Html.TextBox or Html.TextBoxFor helper method the end result is going to be the same that is they produce the same HTML. Strongly typed HTML helpers are introduced in ASP.NET MVC MVC2.

**What is the Validation Summary in the ASP.NET MVC Application?**

The ValidationSummary HTML Helper method displays all the validation errors of the ModelState dictionary in an unordered list. This ValidationSummary HTML Helper method accepts a boolean value (i.e. true or false) and based on the boolean value it displays the errors. When the boolean parameter value is set to true, then it shows only the model-level errors and excludes model property-level errors (i.e. any errors that are associated with a specific model property). When the Boolean value is set to false, then it is going to shows both model-level and property-level errors.

Suppose, you have the following lines of code somewhere in the controller action rendering a view:

**ModelState.AddModelError(“”, “This is Model-level error!”);**  
**ModelState.AddModelError(“Title”, “This Model property-level error!”);**

In the first statement, there is no key is to associate with the error. In the second statement, there is a key with the name “Title” that is associated with the error for the model property Name.

**@Html.ValidationSummary(true) @\*//shows model-level errors\*@**  
**@Html.ValidationSummary(false) @\*//shows model-level and property-level errors\*@**

Hence, when the boolean type value is set to true then the ValidationSummary HTML Helper method will display only the model-level errors and exclude property-level errors. If we set the value to false, then it will display both Model-level and property-level errors.

**What is unobtrusive AJAX?**

ASP.NET MVC Framework supports unobtrusive Ajax. The unobtrusive Ajax means, we can use the HTML Helper methods to define our Ajax features, rather than adding blocks of code throughout our views.

**What is Cross-Domain AJAX?**

By default, a web browser allows AJAX calls only to the same domain i.e. site hosted server. This restriction helps us to prevent various security issues like **cross-site scripting (XSS)** attacks. But, sometimes we need to interact with externally hosted API(s) like Twitter or Google. Hence to interact with these external API(s) or services our web application must support **JSONP** requests or **Cross-Origin Resource Sharing (CORS)**. By default, ASP.NET MVC does not support **JSONP or Cross-Origin Resource Sharing.** For this, you need to do a little bit of coding and configuration.

**What is Caching?**

Caching is the most important aspect of a high-performance web application. Caching provides a way of storing frequently accessed data and reusing that data. Practically, this is an effective way of improving the web application’s performance.

**When to use Caching in ASP.NET MVC Application?**

1. Use caching for contents that are accessed frequently.
2. Avoid caching for contents that are unique per user.
3. Avoid caching for contents that are accessed infrequently/rarely.
4. Use the VaryByCustom function to cache multiple versions of a page based on customization aspects of the request such as cookies, role, theme, browser, and so on.
5. For efficient caching use a 64-bit version of Windows Server and SQL Server.
6. For database caching make sure your database server has sufficient RAM otherwise, it may degrade the performance.
7. For caching of dynamic contents that change frequently, define a short cache–expiration time rather than disabling caching.

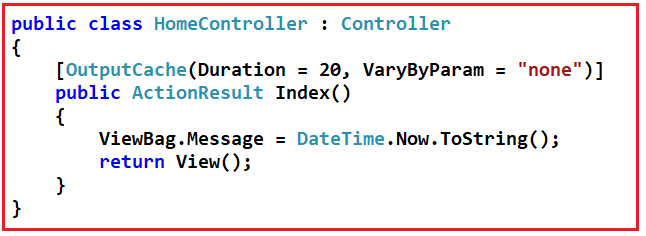
**What are the advantages of caching in ASP.NET MVC Application?**

We are getting the following advantages of using caching:

1. Reduce hosting server round-trips
2. When content is cached at the client or in proxies, it causes the minimum request to the server.
3. Reduce database server round-trips
4. When content is cached at the web server, it can eliminate the database request.
5. Reduce network traffic
6. When content is cached at the client-side, it also reduces the network traffic.
7. Avoid time-consumption for regenerating reusable content
8. When reusable content is cached, it avoids the time consumption for regenerating reusable content.
9. Improve performance
10. Since cached content reduces round-trips, network traffic and avoids time consumption for regenerating reusable content which causes a boost in the performance.

**What is the Output Caching in ASP.NET MVC Application?**

The OutputCache filter allows you to cache the data that is the output of an action method. By default, this attribute filter caches the data until 60 seconds. After 60 sec, ASP.NET MVC will execute the action method again and cache the output again.



The output of the Index() action method will be cached for 20 seconds. If you will not define the duration, it will cache it for by default cache duration 60 sec.

**Output Caching Location**

By default, content is cached in three locations: the web server, any proxy servers, and the user’s browser. You can control the content’s cached location by changing the location parameter of the OutputCache attribute to any of the following values: Any, Client, Downstream, Server, None, or ServerAndClient.

By default, the location parameter has the value and which is appropriate for most of the scenarios. But sometimes there are scenarios when you required more control over the cached data.

**How to allow users to submit HTML tags in ASP.NET MVC?**

By default, ASP.NET MVC doesn’t allow a user to submit HTML for avoiding the Cross-Site Scripting attack on your application. You can achieve it by using the ValidateInput attribute and AllowHtml attribute. The ValidateInput attribute can enable or disable input validation at the controller level or at any action method.

**[**ValidateInput**(false)]**

**public** **class** HomeController : Controller

**{**

**public** ActionResult AddArticle**()**

**{**

**return** View**()**;

**}**

**}**

The ValidateInput attribute allows the Html input for all the properties and that is unsafe. Since you have enabled Html input for only one-two properties then how to do this. To allow Html input for a single property, you should use the AllowHtml attribute.

**public** **class** BlogModel

**{**

**[**Required**]**

**[**Display**(**Name = "Title"**)]**

**public** **string** Title **{** **get**; **set**; **}**

**[**AllowHtml**]**

**[**Required**]**

**[**Display**(**Name = "Description"**)]**

**public** **string** Description **{** **get**; **set**; **}**

**}**

**What is loose coupling and how is it possible in MVC Design Pattern?**

One of the most important features of the MVC design pattern is that it enables the separation of concerns. Hence you can make your application’s components independent as much as possible. This is known as loose coupling and it makes testing and maintenance of our application easier. Using Dependency Injection you can make your application’s components more loosely coupled.

**What is Test-Driven Development (TDD)?**

TDD is a methodology that says; write your tests first before you write your code. In TDD, tests drive your application design and development cycles. You do not do the check-in of your code into source control until your entire unit tests pass.

**What are the commonly used tool for Unit Testing in ASP.NET MVC?**

ASP.NET MVC has been designed for testability without dependencies on the IIS server, on a database, or on external classes. There are the following popular tools for ASP.NET MVC testing:

1. **NUnit** – This is the most popular unit testing framework for Microsoft.NET. Its syntax is relatively simple and easy to use. It comes with a test runner GUI and a command-line utility. NUnit is also available as a NuGet package for download.
2. **xUnit.NET** – This provides a way to run automated unit tests. It is simple, easily extended, and has a very clean syntax.
3. **Ninject** – This provides a way to wire up classes in your application.
4. **Moq** – This provides a framework for mocking interfaces and classes during testing.

### **What do you understand by filters in MVC?**

### Controllers in MVC define action methods, and these methods have a one-to-one relationship with UI controls. Example: UserController class contains UserAdd and UserDelete methods. However, many times, we wish to perform some action after or before a particular operation. With the feature provided by ASP.NET MVC, pre and post-action behaviors can be added to the controller's action methods.

### **What is the function of beforeRender() in the controller?**

### This function is required when we call render() manually before the end of a given action. This function is called before the view is rendered and after controller action logic. It is not used often.

**What do you know about beforeFilter() and afterFilter() functions in controllers?**

beforeFilter() is run every action in the controller. It is the right situation to inspect permissions or check for an active session. At the same time, afterFilter() is called after every rendering is done and after every controller action. This is the last controller method to be run.

**Define DispatcherServerlet.**

DispatcherServerlet is a class that receives incoming requests and maps them to the most appropriate resources, including Views, Models, and Controllers.

**What is attribute-based routing in MVC?**

We have a new attribute route in ASP.NET MVC. By using the ‘route’ attribute, URL structure can be defined. If we decorate the ‘GotoAbout' action with the route attribute, the route attribute says that 'GotoAbout’ can be invoked using the ‘Users/about’ URL structure.

**How is routing carried out in the MVC pattern?**

A group of routes known as RouteCollection consists of registered routes in the application. The RegisterRoutes method records the collection routes. The route and a handler define the URL pattern if the request matches the pattern. The route's name is the first parameter to the MapRoute; the second parameter is the pattern to which URL matches, and the third parameter is default values for the placeholders.

**What is the difference between ‘ViewResult’ and ‘ActionResult’?**

ViewResult is derived from the 'AbstractResult' class, and 'ActionResult’ is an abstract class. ActionResult is good when you are dynamically deriving different types of views. The derived classes of ActionResult are FileStreamResult, ViewResult, and JsonResult.

**Define ORM and give its use.**

ORM (object-relational mapping) framework is a framework that helps in reducing the amount of handwritten code in a web app. ORM is used when there are no extreme performance requirements, but frameworks like Dapper can be used in high-load systems.

**Define POST and GET action types.**

POST action type submits data to be processed to a specified resource. We pass the essential URL and data with all the POST requests. It can take up overloads.

GET action type requests data from a specified resource. We pass the compulsory URLs with all the GET requests. It can take up overloads.

**If we place the log.txt file in the ~/bin/folder of an asp.net MVC application, will it affect the app?**

It is not good to put log files in the bin folder because change will cause pool restart. This can cause problems that would be difficult to track down.

**How will you implement validation in MVC?**

We can implement validation in the MVC application with the help of validators defined in the System.ComponentModel.DataAnnotations namespace. The different validators are DataType, Required, Range, and StringLength.

**What do you understand by WebAPI?**

WebAPI is a technology with which you can expose data over HTTP using the REST principles. This approach was proposed to satisfy a broad range of clients requiring to consume data from JavaScript, mobile, Windows, etc.

**Tell us the benefit of using an IoC container in an application.**

The benefits include external management of life of every object, change of implementation of the contract in future, change to dependency list not affecting an object using the service, and sharing of an instance by several unrelated customers.

**Can you tell two instances where routing is not required or implemented?**

We do not require routing when routing is disabled for a URL pattern and when a physical file matching the URL pattern is found.

**What is the valuable lifetime for an ORM context in an ASP.NET MVC application?**

The lifetime needs to be the same as the requests do not live long. If it is possible to wrap the whole request in one transaction, it can help comply with ACID principles.

**How is form authentication implemented in MVC?**

Authentication means giving users access to a specific service on verification of identity using username and password. It assures that the correct user is logged in for a specific service, and exemplary service is provided to the user based on their role.

**What do you understand by ViewState in MVC?**

One of the most common asp net interview questions is based on ViewState. Unlike WebForms, MVC does not have ViewState. This is because ViewState is stored in a hidden field on the page, which increases its size significantly and impacts the page loading time.

**How is exception handling carried out in MVC?**

We can override the ‘OnException' event in the controller and set the result to the view name needed to be invoked when an error occurs.

Example:

public ActionResult Index()

{

int l = 1;

int m = 0;

int n = 0;

m = l / m; //it would cause exception.

return View();

}

protected override void OnException(ExceptionContext filterContext)

{

string action = filterContext.RouteData.Values["action"].ToString();

Exception e = filterContext.Exception;

filterContext.ExceptionHandled = true;

filterContext.Result = new VR()

{

ViewName = "err"

};

}

**What is the need to use Html.Partial in MVC?**

This method renders the specified partial view as an HTML string. It does not depend on any action methods. It is used like this: @Html.Partial(“TestPartialView”)

**How is JSON response returned from the Action method?**

To return the JSON response Action method, we need to return the JsonResult object. This object calls the Json helper method. The given an example returns a response in JSON format:

public JsonResult sample()

{

string send = "hello";

return Json(send, JsonRequestBehavior.AllowGet);

}

**Tell us the benefit of the area in MVC.**

The Area in MVC helps to integrate with other areas generated by other apps, helps organize views, controllers, and models in different functional sets, and is suitable for unit testing.

**What do you understand by Model Binding?**

In action methods, we need to retrieve data from requests to be used by the data. In MVC, model binding maps data from HTTP request to action method parameters. This task of retrieving data from HTTPRequest is repetitive and is removed from the action method.

**Explain some of the rules of Razor syntax.**

Razor is a markup syntax that programming professionals can use for embedding code. It has a default view engine that doesn’t interrupt coding or block servers in HTML links. The markup is simple to use, making it especially good for beginners. It also enables programmers to mix server-side code and HTML in C#.

**Here are some of the rules of Razor syntax:**

Razor code blocks are enclosed in @{ }

Arranged expressions (variables and functions) start with @

Code statements end with a semicolon

Variables are declared with the var keyword

Strings are enclosed with quotation marks

C# code is case sensitive

**What are Action Filters in MVC?**

Action Filters are additional attributes that can be applied to either a controller section or the entire controller to modify the way in which action is executed. These attributes are special .NET classes derived from System.Attribute which can be attached to classes, methods, properties, and fields.

ASP.NET MVC provides the following action filters,

**Output Cache**

This action filter caches the output of a controller action for a specified amount of time.

**Handle Error**

This action filter handles errors raised when a controller action executes.

**Authorize**

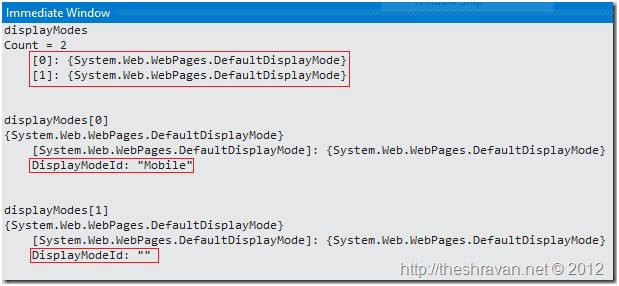
This action filter enables you to restrict access to a particular user or role.

**Explain the need of display mode in MVC?**

DisplayModes give you another level of flexibility on top of the default capabilities we saw in the last section. DisplayModes can also be used along with the previous feature so we will simply build off of the site we just created.

**Using display modes involves in 2 steps**

1. We should register Display Mode with a suffix for particular browser using “DefaultDisplayMode”e class inApplication\_Start() method in the Global.asax file.
2. View name for particular browser should be appended with suffix mentioned in first step.



1. Desktop browsers (without any suffix. e.g.: Index.cshtml, \_Layout.cshtml).
2. Mobile browsers (with a suffix “Mobile”. e.g.: Index.Mobile.cshtml,Layout.Mobile.cshtml)  
   If you want design different pages for different mobile device browsers (any different browsers) and render them depending on the browser requesting. To handle these requests you can register custom display modes. We can do that using DisplayModeProvider.Instance.Modes.Insert(int index, IDisplayMode item) method.

**What is Bundling and Minification in MVC?**

Bundling and minification are two new techniques introduced to improve request load time. It improves load time by reducing the number of requests to the server and reducing the size of requested assets (such as CSS and JavaScript).

**Bundling**

It lets us combine multiple JavaScript (.js) files or multiple cascading style sheet (.css) files so that they can be downloaded as a unit, rather than making individual HTTP requests.

**Minification**

It squeezes out whitespace and performs other types of compression to make the downloaded files as small as possible. At runtime, the process identifies the user agent, for example IE, Mozilla, etc. and then removes whatever is specific to Mozilla when the request comes from IE.

## **How can we do Custom Error Page in MVC?**

The HandleErrorAttribute allows you to use a custom page for this error. First you need to update your web.config file to allow your application to handle custom errors.

<system.web>

<customErrors mode="On">

</system.web>

Then, your action method needs to be marked with the atttribute.

[HandleError]

public class HomeController: Controller

{

[HandleError]

publicActionResultThrowException()

{

throw new ApplicationException();

}

}

By calling the ThrowException action, this would then redirect the user to the default error page. In our case though, we want to use a custom error page and redirect the user there instead.So, let's create our new custom view page.



Next, we simply need to update the HandleErrorAttribute on the action method.

[HandleError]

public class HomeController: Controller

{

[HandleError(View = "CustomErrorView")]

publicActionResultThrowException()

{

throw new ApplicationException();

}

}

**What is the use of remote validation in MVC?**

Remote validation is the process where we validate specific data posting data to a server without posting the entire form data to the server. Let's see an actual scenario, in one of my projects I had a requirement to validate an email address, whetehr it already exists in the database. Remote validation was useful for that; without posting all the data we can validate only the email address supplied by the user.

**Server Side Validation in MVC?**

The ASP.NET MVC Framework validates any data passed to the controller action that is executing, It populates a ModelState object with any validation failures that it finds and passes that object to the controller. Then the controller actions can query the ModelState to discover whether the request is valid and react accordingly.

I will use two approaches in this article to validate a model data. One is to manually add an error to the ModelState object and another uses the Data Annotation API to validate the model data.

## **Explain RenderSection in MVC?**

RenderSection() is a method of the WebPageBase class. Scott wrote at one point, The first parameter to the "RenderSection()" helper method specifies the name of the section we want to render at that location in the layout template. The second parameter is optional, and allows us to define whether the section we are rendering is required or not. If a section is "required", then Razor will throw an error at runtime if that section is not implemented within a view template that is based on the layout file (that can make it easier to track down content errors). It returns the HTML content to render.

<div id="body">

@RenderSection("featured", required: false)

<section class="content-wrapper main-content clear-fix">

@RenderBody()

</section>

</div>

**What are Url Helpers?**

Url helpers allow you to render HTML links and raw URLs. The output of these helpers is dependent on the routing configuration of your ASP.NET MVC application.

## **Explain the difference between Html.RenderPartial() and Html.Partial() methods in ASP.NET MVC.**

|  |  |
| --- | --- |
| **Html.RenderPartial()** | **Html.Partial()** |
| This is used to render the partial view to the output directly. | This is used to store the content which is already rendered in a variable. |
| There is nothing to return. | A string that has the partial view displayed in it is returned. |
| This is mostly used when there is no need for changes to the content and can be directly rendered. | This is mostly used when you want changes to be made to the content before rendering it. |

## **How can you handle multiple submit buttons in a single ASP.NET MVC form?**

To handle multiple submit buttons in a single ASP.NET MVC form, what you can do is for each button used in the form, give it a unique name or another method is using JavaScript or jQuery to see which button was clicked and the correct action can be performed after clicking.

## **What is the purpose of the Web.config file in an ASP.NET MVC application?**

In an ASP.NET MVC application, a Web.config file is used for the purpose of configuring the settings that are in that specific application. They can be database connections, security configurations, application wide settings or other parameters which are required so that the application can run in the correct manner as it should.