Articles from Jinal Desai .NET

MVC Interview Questions – Deep Dive

2012-05-26 12:05:38 Jinal Desai

For basic Interview Questions of MVC refer my first post of MVC Interview Questions.

1. Can you describe ASP.NET MVC Request Life Cycle?
Ans. There are two answers for the question.
Short Answer: Following are the steps that are executed in ASP.NET MVC Request Life Cycle.

I. Receive request, look up Route object in RouteTable collection and create RouteData object.
II. Create RequestContext instance.
III. Create MvcHandler and pass RequestContext to handler.
IV. Identify IControllerFactory from RequestContext.
V. Create instance of class that implements ControllerBase.
VI. Call MyController.Execute method.
VII. The ControllerActionInvoker determines which action to invoke on the controller and executes the action on the controller, which results in calling the model and returning a view.

Detailed Answer: There are five main steps occurs in ASP.NET Request Life Cycle.
I. Initialize Application (Route table is created)
Desc. When we request a page from a normal ASP.NET application, there is a physical copy of page on the disk that is corresponds to each page request. In ASP.NET MVC, there is no page on the disk that corresponds to the request. Request is routed to the controller. The controller is responsible for generating page that is sent back to the browser.

ASP.NET MVC has Route Table which stores mapping entries of particular URLs to action methods of contoller. i.e. URL “http://jinaldesai.net/Articles/CSharp” maps to the “CSharp” method of “Articles” controller.

In ASP.NET MVC application like normal ASP.NET application when application starts first time, it calls Application_Start() application event from Global.asax. Route table is registered(created) from Appication_Start() event.

II. Routing Mechanism (UrlRoutingModule intercepts the request)
When a request is coming to ASP.NET MVC application, the request is intercepted by UrlRoutingModule HTTP Module. The first thing the module does is to wrap up the current HttpContext in an HttpContextWrapper2 object. Next, the module passes the wrapped HttpContext to the RouteTable that was setup in the previous step, which includes includes the URL, form parameters, query string parameters, and cookies

associated with the current request.

Next, the module creates a RouteContext object that represents the current HttpContext and RouteData. The module then instantiates a new HttpHandler based on the RouteTable and passes the RouteContext to the new handler’s constructor(In ASP.NET MVC case it is MvcHandler).

Last step taken by module is setting MvcHandler as the current HTTP Handler.

III. MvcHandler Executes (Instantiate and execute controller)
In normal ASP.NET application after second step, it fires set of events including Start, BeginRequest, PostResolveRequestCache, etc.

While in case of ASP.NET MVC application, MvcHandler(which is created in previous step) creates appropriate controller instance to serve the request by passing controller name(found through the route mapping table) and RequestContext to the static method CreateController() method of ControllerFactory class(i.e. ControllerFactory.CreateController()) to get particular controller.

Next is ControllerContext object is created from the RequestContext and the controller. And the last step in Step 3 is the Execute() method is called on the controller class.(Here factory pattern is implemented)

IV. Controller Executes (Locate and invoke controller action)
The Controller.Execute() method takes RouteData and other context information to locate appropriate action method. It also maps incoming request parameters(querystring, form, etc) to the parameter list of the controller action method.

Next is the controller calls it’s InvokeAction() method, passing details of the choosen action method and invokes the action method and our code finally runs.

V. RenderView Method Invoked (Instantiate and render view)
The controller object created in previous step has a property ViewFactory of type IViewFactory. The IViewFactory has a method CreateView(), which is called next by passing view name and other context information. The CreateView() method returns an IView.

Next the controller invokes RenderView() method of returned IView with again supplying necessary context information includes ViewData and IHttpResponse object. The response data is a HTML, an image, or any other binary or text data.

2. What is MVP?
Ans. MVP(Model View Presentor) is a user interface design pattern engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic.
It is derived from MVC(Model View Controller) design pattern. The MVP in the pattern has following description.
The **model** is an interface defining the data to be displayed or otherwise acted upon

in the user interface.
The **view** is an interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.
The presenter acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.
http://msdn.microsoft.com/en-us/magazine/cc188690.aspx

3. What is difference between MVC and MVP?
Ans. The key difference between MVC and MVP is that MVP truly separates the UI from the domain/service layer of the application. In MVP the presenter assumes the functionality of the middle-man(played by the Controller in MVC). MVP is specially geared towards a page event model such as ASP.NET.
For more differences see my article: MVC Vs MVP

4. What is Application Areas in ASP.NET MVC application? Why it is necessary?
Ans. An Area in ASP.NET MVC application is subset of the project structure based on a logical grouping. Each area contains area specific models, views and controllers. In ASP.NET MVC we can use the default project structure for most websites without having any problems. However, there are some sites that are very large; keeping all the models, views and controllers in a single folder set can be difficult to manage. For such cases, we can define different project ares in ASP.NET MVC application.

5. What are the different return types controller action method supports in ASP.NET MVC?
Ans. There are total nine return types we can use to return results from controller to view. The base type of all these result types is ActionResult.
ViewResult (View) : Used to return a webpage from an action method.
PartialviewResult (Partialview) : Used to send a section of a view to be rendered inside another view.
RedirectResult (Redirect) : Used to redirect to another controller and action method
based on a URL.
RedirectToRouteResult (RedirectToAction, RedirectToRoute) : Used to redirect to
another action method.
ContentResult (Content) : Used to return a custom content type as the result of the
action method. This is an HTTP content type, such as text/plain.
jsonResult (json) : Used to return a message formatted as JSON.
javascriptResult (javascript) : Used to return JavaScript code that will be executed
in the user's browser.
FileResult (File) : Used to send binary output as the response.
EmptyResult : Used to return nothing (void) as the result.

6. What is action filters in ASP.NET MVC?

Ans. In ASP.NET MVC we can run our code before controller's action method is executed or after the action method run. To achieve this we can use action filters. We can apply action filters by applying attributes to action methods.

7. Can you describe different types of action filters in brief?
Ans. There are mainly three types of action filters provided in ASP.NET MVC.
Authorization Filter : It makes security decisions about whether to execute an action method, such as performing authentication or validating properties of the request. The **AuthorizeAttribute** class is one example of an authorization filter.
Result Filter : It wraps execution of the ActionResult object. This filter can perform additional processing of the result, such as modifying the HTTP response. The **OutputCacheAttribute** class is one example of a result filter.
Execution Filter : It executes if there is an unhandled exception thrown somewhere in action method, starting with the authorization filters and ending with the execution of the result. Exception filters can be used for tasks such as logging or displaying an error page. The **HandleErrorAttribute** class is one example of an exception filter.
http://msdn.microsoft.com/en-us/library/dd410209.aspx

Apart from the readymade action filters provided by ASP.NET MVC, you can also implement your own action filter by inheriting ActionFilterAttribute abstract class. It has four virtual methods that you can override: OnActionExecuting, OnActionExecuted, OnResultExecuting and OnResultExecuted. To implement an action filter, you must override at least one of these methods.
http://msdn.microsoft.com/en-us/library/dd381609

8. What are the enhancements ASP.NET MVC 3 provided compared to ASP.NET MVC 2?
Ans. There are many enhancements come with ASP.NET MVC 3. Following are list of some enhancements in ASP.NET MVC 3.
Razor View Engine
Partial Page Output Caching
Unobtrusive Javascript and Validation
New Dynamic ViewModel Property
Global Filters
New ActionResult Types (HttpNotFoundResult, HttpStatusCodeResult, HttpRedirectResult)
Built in JSON binding support

9. What is Data Annotations?
Ans. Data Annotations are validation mechanism we can use to validate our data stored in form of Entity Data Model, LINQ to SQL or any other data models. We can use it in form of attributes to our properties of mode class. These attributes provide common validation patterns, such as range checking and required fields. Once we apply these attributes to our model class, ASP.NET MVC will provide both client and server side validation checks with no additional coding required. You can also implement your custom Data Annotation Validator by inheriting ValidationAttribute class and overriding IsValid method. To name a few there is RangeAttribute, RequiredAttribute, StringLengthAttribute, RegularExpressionAttribute, etc.

http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.aspx (Complete List of Data Annotation attributes)

10. What are HTML Helpers in ASP.NET MVC?
Ans. In MVC, HTML Helpers are much like traditional ASP.NET web form controls. Just like web form controls in ASP.NET, HTML helpers are used to modify HTML. But HTML helpers are more lightweight. Unlike Web Form controls, an HTML helper does not have an event model and a view state. In most cases, an HTML helper is just a method that returns a string. With MVC, you can create your own helpers, or use the built in HTML helpers. To name a few there is BeginForm(), EndForm(), TextBox(), TextArea(), ListBox(), RadioButton(), etc.
http://msdn.microsoft.com/en-us/library/system.web.mvc.htmlhelper.aspx
(Complete List of HTML Helpers)

References:
http://stephenwalther.com/blog/archive/2008/03/18/asp-net-mvc-in-depth-the-life-of-an-asp-net-mvc-request.aspx
http://blog.codeville.net/2007/11/20/aspnet-mvc-pipeline-lifecycle/
http://weblogs.asp.net/scottgu/archive/2010/07/27/introducing-asp-net-mvc-3-preview-1.aspx
http://weblogs.asp.net/imranbaloch/archive/2011/07/26/using-the-features-of-asp-net-mvc-3-futures.aspx
http://msdn.microsoft.com/en-us/library/ee256141(VS.100).aspx
http://www.w3schools.com/aspnet/mvc_htmlhelpers.asp