

TechnoCastle by Arasu

.Net Technologies

Monday, June 29, 2015

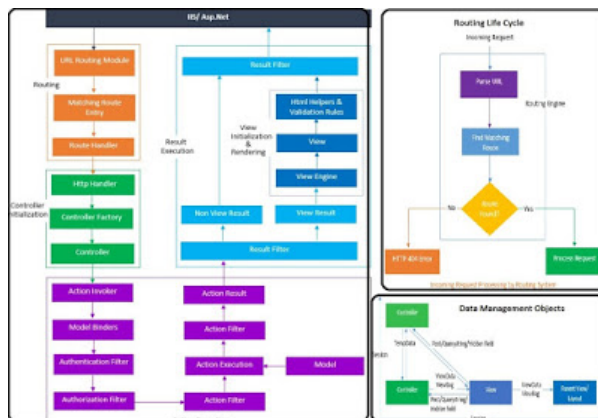
Asp.Net MVC Life Cycle

MVC Life Cycle:

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)`.



2. Controller Initialization -

The `MvcHandler` initiates the real processing inside ASP.NET MVC pipeline by using `ProcessRequest` method. This method uses the `ControllerFactory` instance (default is `System.Web.Mvc.DefaultControllerFactory`) to create corresponding controller.

3. 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 `IAuthorizationFilter`.

□ 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

Blog Archive

- 2016 (5)
- ▼ 2015 (10)
 - August (2)
 - July (1)
 - ▼ June (7)

[Asp.Net MVC Life Cycle](#)

Steps to attach DataBase to .Net Application - App...

Reference Links

Books To Refer

OOPS Concepts with Real Time Example

.Net Questions

C# IEnumerable and IQueryable

Search

- Design Patterns
- C#
- IOC & DI
- Important Question for .Net Professionals.
- OOPS
- Reference Books
- SOLID Principle
- challenging Bugs

About Me



TamilArasu S

[Follow](#) 84

[View my complete profile](#)

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.

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.

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

ViewData:

□ ViewData is a dictionary object that is derived from ViewDataDictionary class.

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

□ ViewData is used to pass data from controller to corresponding view.

□ Its life lies only during the current request.

□ If redirection occurs then its value becomes null.

□ It's required typecasting for getting data and check for null values to avoid error.

ViewBag:

□ ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.

```
public Object ViewBag { get; }
```

□ Basically it is a wrapper around the ViewData and also used to pass data from controller to corresponding view.

□ Its life also lies only during the current request.

□ If redirection occurs then its value becomes null.

□ It doesn't required typecasting for getting data.

TempData:

□ TempData is a dictionary object that is derived from TempDataDictionary class and stored in short lives session.

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

TempData is used to pass data from current request to subsequent request (means redirecting from one page to another).

□ Its life is very short and lies only till the target view is fully loaded.

□ It's required typecasting for getting data and check for null values to avoid error.

□ It's used to store only one time messages like error messages, validation messages.

Session:

□ In ASP.NET MVC, Session is a property of Controller class whose type is HttpSessionStateBase.

```
public HttpSessionStateBase Session { get; }
```

- Session is also used to pass data within the ASP.NET MVC application and Unlike TempData, it persist data for a user session until it is time out (by default session timeout is 20 minutes).
- Session is valid for all requests, not for a single redirect.
- It's also required typecasting for getting data and check for null values to avoid error.

How to persist data in TempData?

The life of TempData is very short and lies only till the target view is fully loaded. But you can persist data in TempData by calling Keep() method after request completion

- **void Keep()** - Calling this method with in the current action ensures that all the items in TempData are not removed at the end of the current request.

```
public ActionResult Index()
{
    ViewBag.Message = TempData["Message"];
    Employee emp = TempData["emp"] as Employee; //need type casting
    TempData.Keep();//persist all strings values
    return View();
}
```

- **void Keep(string key)** - Calling this method with in the current action ensures that specific item in TempData is not removed at the end of the current request.

```
public ActionResult Index()
{
    ViewBag.Message = TempData["Message"];
    Employee emp = TempData["emp"] as Employee; //need type casting
    //persist only data for emp key    and Message key will be destroy
    TempData.Keep("emp");
    return View();
}
```

How to control Session behavior in ASP.NET MVC?

By default, ASP.NET MVC support session state. Session is used to store data values across requests. Whether you store some data values with in the session or not ASP.NET MVC must manage the session state for all the controllers in your application that is time consuming. Since, session is stored on server side and consumes server memory, hence it also affect 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 controller and can gain slight performance improvement of your application. You can simplify session state for your application by using available options for session state.

In ASP.NET MVC4, SessionState attribute provides you more control over the behavior of session-state by specifying the value of SessionStateBehavior enumeration as shown below:

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.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6 using System.Web.SessionState; // Required for controlling session state
7 namespace Mvc4_SessionState.Controllers
8 {
9     [SessionState(SessionStateBehavior.Disabled)]
10    public class HomeController : Controller
11    {
12        public ActionResult Index()
13        {
14            TempData["Message"] = "Dot";
15            return View();
16        }
17    }
18 }
```

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

In ASP.NET MVC, TempData use session state for storing the data values across requests.

Hence, when you will disabled the session state for the controller, it will throw the exception as shown below:

Server Error in '/' Application.

The SessionStateTempDataProvider class requires session state to be enabled.

Exception Details: System.InvalidOperationException: The SessionStateTempDataProvider class requires session state to be enabled.


Posted by [TamilArasu S](#) at [11:13 PM](#)



No comments:

Post a Comment

Enter your comment...

 Comment as: hadt0786@gmail.com ▼

Sign out

Publish

Preview

☐ Notify me

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

Travel theme. Powered by [Blogger](#).