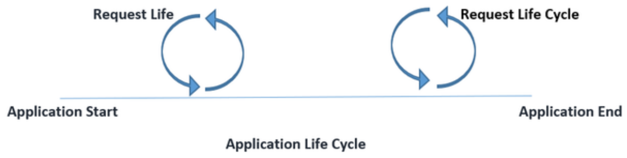
geeksforgeeks.org

ASP.NET MVC Life Cycle

manaligujarathi0manaligujarathi0

10–12 minutes

The goal of this article is to provide a good understanding of the MVC pipeline. The life cycle is basically is set of certain stages which occur at a certain time.



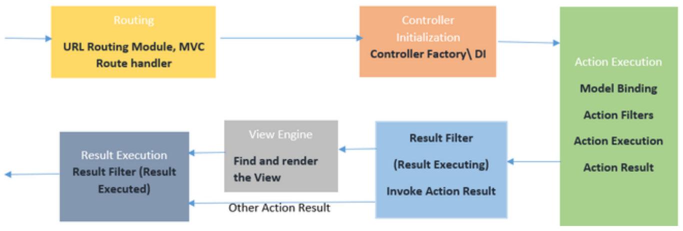
Application Life Cycle

MVC actually defined in two life cycles, the application life cycle, and the request life cycle.

The **application life cycle**, in which the application process starts the running server until the time it stops. and it tagged the two events in the startup file of your application. i.e the application start and end events

This is separate from the **request life cycle**, which is the sequence of events or stages that executed every time an HTTP request is handled by the application

MVC Request Life Cycle:



MVC Request Life Cycle

- The Starting point for every MVC application begins with routing. After that, the received request figures out and finds how it should be handled with the help of the URL Routing Module. The routing module is responsible for matching the incoming URL to routes that we defined in our application.
- Every route has an associated route handler associated with them. If the request is matched to a route defined in our MVC application, the MVC Route Handler retrieves an instance of the MVC HttpHandler.
- The MVC Handler Start the process of initializing and executing a controller. The MVC framework handles converting route data into a specific controller that can handle the requests.
- This is achieved through the MVC components like the Controller Factory and Activators which are responsible for creating an instance of the Controller class.
- After the controller has been created, A component called the action invoker finds and selects an appropriate Action method to invoke on our controller.
- Model binding happens before the method is called, which maps the data from our HTTP request to the parameters of our action methods. Also called action filter before and after action results are generated.
- Now after our action result has been prepared, the next stage triggers, which is Result Execution.
- If the result is a view type, the View Engine will be called and it will find the view and render it.
- If not a view type then the action result will execute on its own. This Result Execution is nothing but generates an actual response to that original HTTP request.

Lets we see now code end,

The MVC application is the main part of every MVC project. This class is inherited from HttpApplication Class. This class exposes a number of Request life cycle events that we can attach event handlers to and run our own code.

The application start and ends. These two events run before and after any other events and allow our application to start or stop receiving the requests.

Application Start :

so every MVC application life starts with an application start event. This event fires when the application receives its first request. This is the entry point file of the MVC application. This event fires when the application receives its first request.it allows us to perform global configuration before anything will happen which we can see default implementation here.

As you can see here,

- C#

C#

```
using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

namespace MVC_Application {

    public class MvcApplication :
System.Web.HttpApplication {

    protected void Application_Start()

        {

            AreaRegistration.RegisterAllAreas();

            FilterConfig.RegisterGlobalFilters(

                GlobalFilters.Filters);

            RouteConfig.RegisterRoutes(RouteTable.Routes);

            BundleConfig.RegisterBundles(BundleTable.Bundle

        }

    }
}
```

- In MVC application, you can use this for various tasks like applying global configuration, register script, and CSS bundles .as you can see here default implements RegisterBundles, RegisterGlobalFilters so on
- The RegisterRoutes method adds any routes that we define to static collection on RouteTable Class. This is the collection of routes that URL Routing tries to match to incoming URLs.
- We need to define this before anything will happen so that the request has something to be matched. and each of these routes also needs to have an associated RoutHandler class that will actually process the incoming requests after it's been matched to the route. RouteHandler is used to retrieve the right Httphandler for request.

Application End:

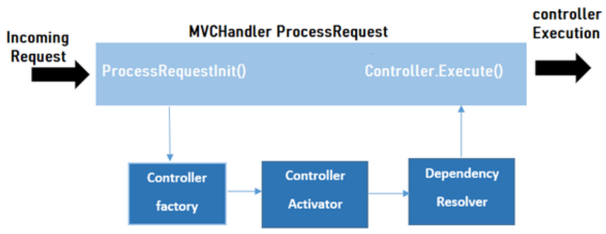
This event is fired when our application stops and then no more requests can be received.it is rare to use this event but if you want to run any task before the application completely shuts down intentionally. This is a good place to do it.

PreApplication Start:

There are some cases in which you want to execute some custom code before the application starts In this situation we have the option of using the PreApplication Start method. Actually, this method defines on assembly level.

Lets we see every phase in detail:

Controller Initialization:

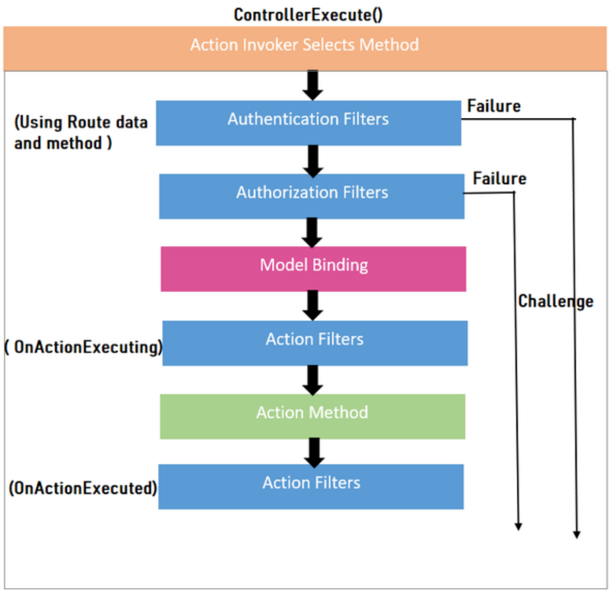


MVC controller Initialization Flow

The MVCHandler's,

- The **ProcessRequest** method is responsible for generating a response to an incoming request. To do this it must create and execute a controller.
- First, it calls a child method named **ProcessRequestInit**, which asks a component called the Controller Factory to provide a Controller for the current request. The Controller Factory selects an appropriate Controller class from the current application using the passed route data.
- Next, the factory uses a component called the Controller Activator to actually create an instance of that class. The Controller Activator uses a Dependency Resolver when instantiating the requested class. it allows patterns like dependency injection to be implemented After the MVCHandler has acquired a Controller from these steps, it calls the Execute method on that Controller. It's from within this method that all of the powerful tools like Action methods are exposed and begin processing.

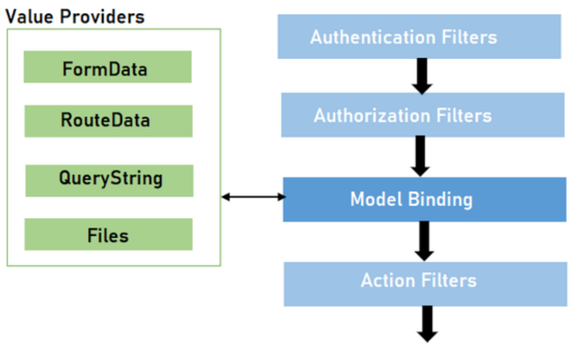
Action Method Execution Process:



Action Method execution Flow

Next Phase,

- The Action Invoker is responsible for selecting and executing the best method to handle the incoming request.it chooses by matching route data to method name along with action selector applied to that method.
- once the method has been selected before, it executed the authentication and authorization process. These security steps are implemented through MVC filters.it is components that allow you to inject logic into the processing pipeline so
- The first **authentication**, this filter verifies the person if the person is who they say is if authentication fails then it is sent back to the requestor if it succeeds. we move on to **authorization** it is checked what the requestor is allowed to do if the requestor is not authorized it is sent back to the browser.
- if the user passes both phases then the pipeline moves forward so the action invoker has been chosen method to execute before the method is executed it need to have parameter populated so that's where **model binding** comes in. so it is taking data from the request and use that data to create requestor object. after all parameters for a given method have been passed then the action invoker is able to call that method. The model binder itself retrieves data to populate the action method parameter from value providers.



Model Binding Process Flow

- The MVC offers four default providers that collect data from common places that are Form Data, Route Data, Query String, Files these classes provide information from various sources related to the current requests then an action filter is executed. it executed two methods i.e **OnActionExecuting** which fire before the action method and **OnActionExecuted** which fire afterward so at this point OnActionExecuting filters are run so after these execution are done finally action method call itself it return action result in short it determine and prepare the type of response that will handle the request after that OnActionExecuted filters run this allows us to add logic into the pipeline before the result executed.

In-build, MVC provides action filters are:

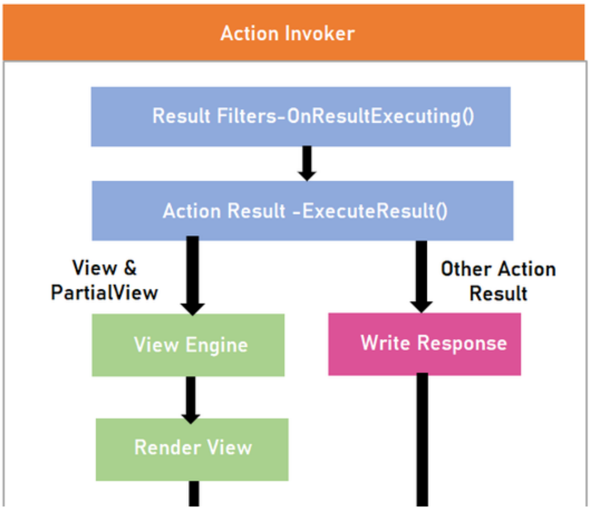
Output Cache: This filter is used to caches the output of action for a certain duration of time.

Handle Error: This filter is used to handles the error caused by an action or controller, if any exception occurs it redirects the action to a custom error page.

Authorize: This filter is used for filtering the authorized user to access the resource.

Note: we can create a custom filter by implementing that particular filter interface class.

View Result Execution Process:



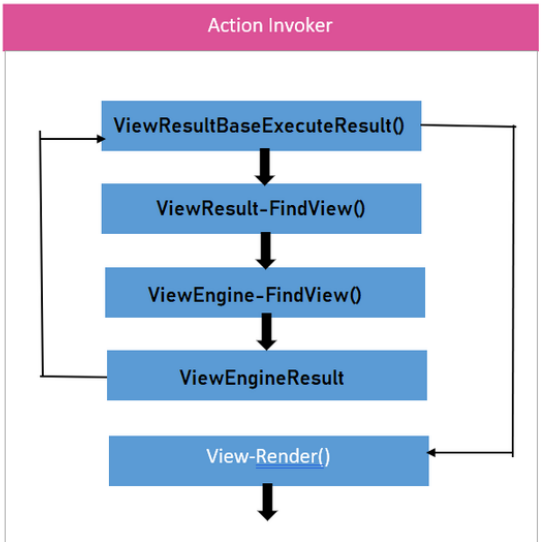
Result Filters- OnResultExecuted

View Result Execution Flow

Next Phase,

- Action result execution is triggered by action invoker before the result execute the associated result filter are run first this filter also expose two methods.
- The first **OnResultExecuting** fires before the action result in itself after this method run the execute result of the action result is called. The MVC provided the different types of action results the execute result provided pipeline branches between two paths if the result is in **view** or **partial** view form then the view engine renders the view and if the result is in another form then this result generally handles writing the responses out itself.After **onResultExecuted** method of result filters are run .this is the last point of MVC to inject your own logic

Action Result Execution Process:



Action Result Execution Flow

Next Phase,

- View rendering process Starts when the Action Invoker calls the **ExecuteResult** method on that object.
- In the case of View Results, the ExecuteResult method is actually defined on a parent class called ViewResultBase. So the invoker calls the base class's ExecuteResult method, which in turn calls an abstract method named FindView. The child ViewResult class overrides this method and uses it to ask the View Engine to find the right View. The View Engine returns a ViewEngineResult, which will contain either the acquired View or a list of locations that it tried to search for one. Views are nothing more than a class that implements the IView interface, which defines only one method that's called Render. Render is the method that finally writes out the contents of our template to the response for the request.

That is all about MVC Life Cycle.

Last Updated : 22 Mar, 2022

Like Article

Save Article