**ASP.NET MVC Life Cycle**

**ASP.NET MVC Life Cycle**

In this article, I am going to discuss the **ASP.NET MVC Life Cycle** in detail. In the ASP.NET MVC application, when a client makes a request, then that request goes through various stages before returning the response to the client. The various stages the request goes through are nothing but the **ASP.NET MVC Request Life Cycle** or you can say **ASP.NET MVC Request Pipeline**. So, if you are working as an ASP.NET MVC developer, then you should be aware of the **ASP.NET MVC Request Life Cycle** i.e. from birth to death of a request. As part of this article, we will learn two things. They are as follows:

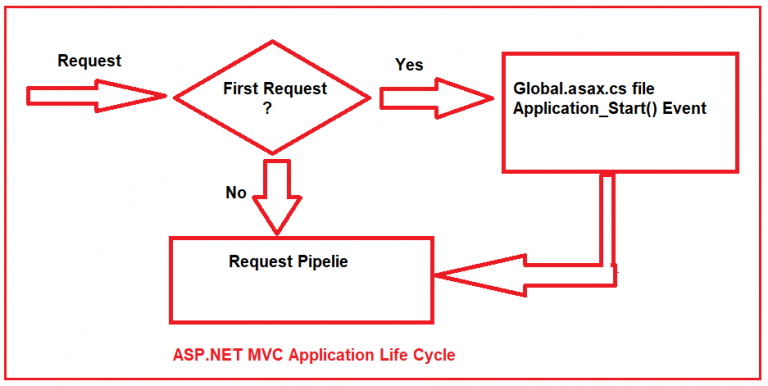
1. **Application Life cycle**
2. **Request Life Cycle**

**Note:** In many articles on the web, you can find what they are saying both are the same. But in reality, both are not the same. Let discuss what exactly they are.

**Application Life Cycle of ASP.NET MVC Application:**

The Application Life Cycle and Request Life Cycle both are not the same. The **Request**

**Life Cycle in ASP.NET MVC** application starts when a request is made for a resource (or you can say for a page) from the client and it ends when the response is sent back to the client and this will happen for each and every request made by a client. On the other hand, the **Application life cycle in ASP.NET MVC** application starts when the first request comes to the Application from a client.

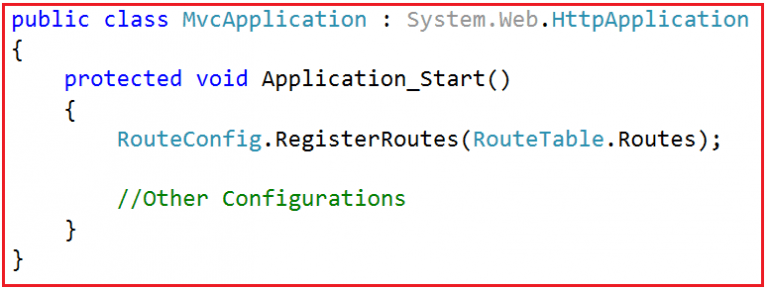


**Explanation of the above Architecture:**

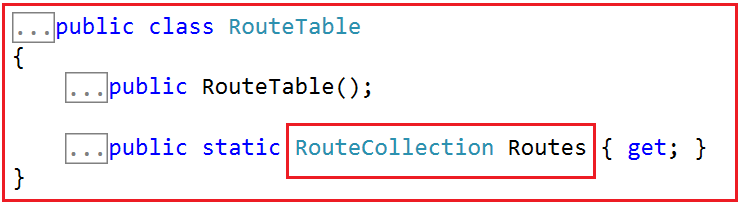
As you can see in the above image, when the client made the first request, then the **Application\_Start()** method of the **Global.asax.cs** class file is going to be fired. The point that you need to remember is, this is the first method of your ASP.NET MVC application that is executed when the application starts.

The **Application\_Start**event is used to set up the initial required configuration needed in order to run the application such as Registering the Routes, Registering the Filters, Areas and Bundle Configurations, etc. But here in this article, we are only interested in the **RegisterRoutes** method.

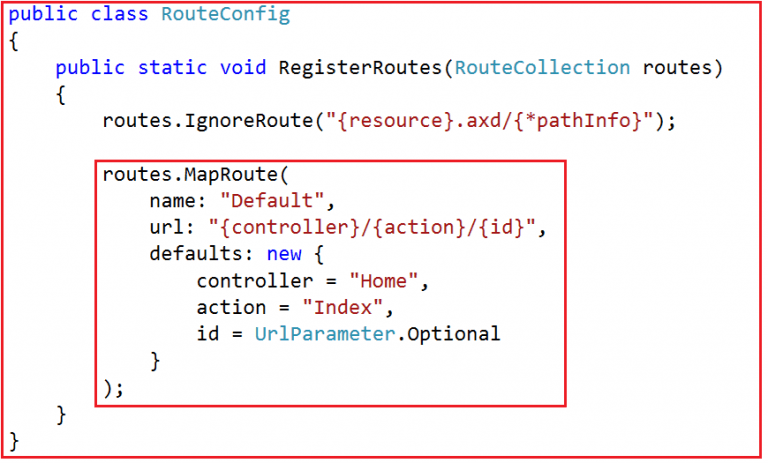
The **Application\_Start()** method calls the static **RegisterRoutes** Method of the **RouteConfig** class by passing the **RouteCollection** (i.e. **RouteTable.Routes**) as a parameter as shown below.



If you go to the definition of **RouteTable** class then you will see that the **Routes** is nothing but a static property of type **RouteCollection** as shown in the below image.



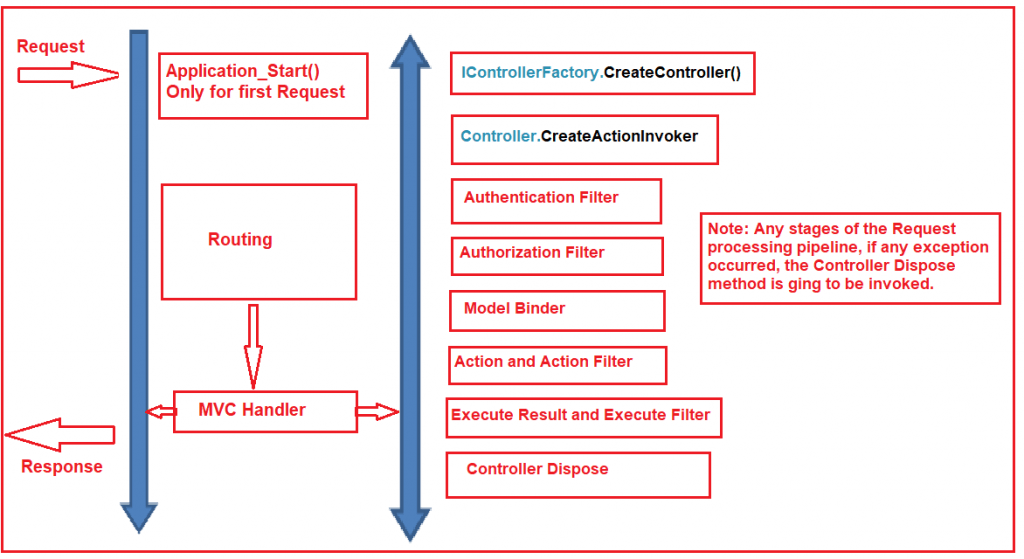
Once the **Application\_Start** event calls the **RegisterRoutes** Method of the **RouteConfig** class, then the **RegisterRoutes** method fills the **Route Table** (i.e. comes as a parameter to the **RegisterRoutes** method) with the routes defined for your application as shown in the below image.



The **Application\_End** event of **Global.asax.cs** file is going to be fired when the Web Server Recycles the application pool on which it is hosted or when the CPU or memory thresholds are exceeded. The point that you need to remember is, once the **Application\_End** event is fired, then the next request coming to the application will be treated as the first request and again the **Application\_Start** event is fired and all the above steps we discussed are going to happen in order.

**Request Life Cycle of ASP.NET MVC Application**

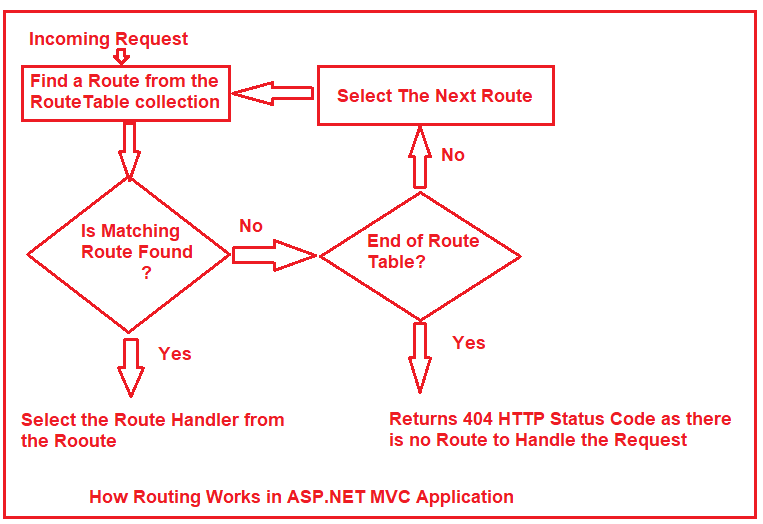
Now let us understand the **Request Life Cycle of the ASP.NET MVC** application. Please have a look at the following diagram which shows a high-level architecture of the **ASP.NET MVC Request Processing Pipeline**.



When the client makes a request, and if it is the first request to the application then the **Application\_Start** event is going to be fired and what these events do internally we already discussed at the start of this article. So, in short, the **Application\_Start** event calls the **RegisterRoutes** Method of the **RouteConfig** class which will fill the **RouteTable** with the routes defined for your application. Then the Request comes to the Routine module.

**Routing in ASP.NET MVC Life Cycle:**

The aim of the [**Routing Module**](https://dotnettutorials.net/lesson/asp-dot-net-mvc-routing/) in **ASP.NET MVC Application** is very simple. It just maps the incoming request i.e. **URL** to a **Route** and then selects the **HttpHandler** which is associated with the Route to handle that Request. Please have a look at the following diagram to understand **How the Routing Module works in ASP.NET MVC** Application.



As shown in the above image, we can say that the **Routing Module is the first module in the ASP.NET MVC Request Processing** Pipeline which receives the incoming request made by the client. Once it receives the incoming request, then it tries to find a **matching URL pattern defined in the Route Table**. If it does not find any matching URL pattern in the Route table then it simply returns a **404 HTTP Status** code to the client.

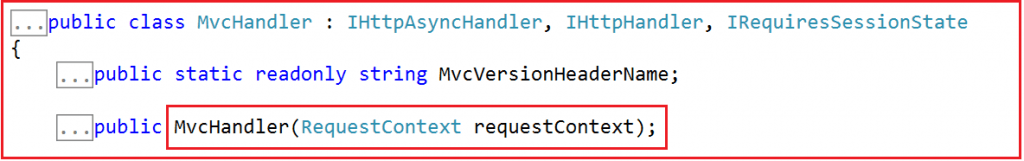
On the other hand, if it found a matching URL Pattern in the Route table, then it selects the **HTTP Handler** (a handler that implements the **IHttpHandler** interface) which is associated with that **Route,** and then forward that request to the corresponding handler.

The **Routeing Module** is also responsible for creating the **RequestContext** object. And also responsible for passing that **RequestContext** object to the corresponding **Handler**. The default handler used by an ASP.NET MVC application is **MvcHandler**. But if you want then you can change this default handler that we will discuss in our upcoming articles.

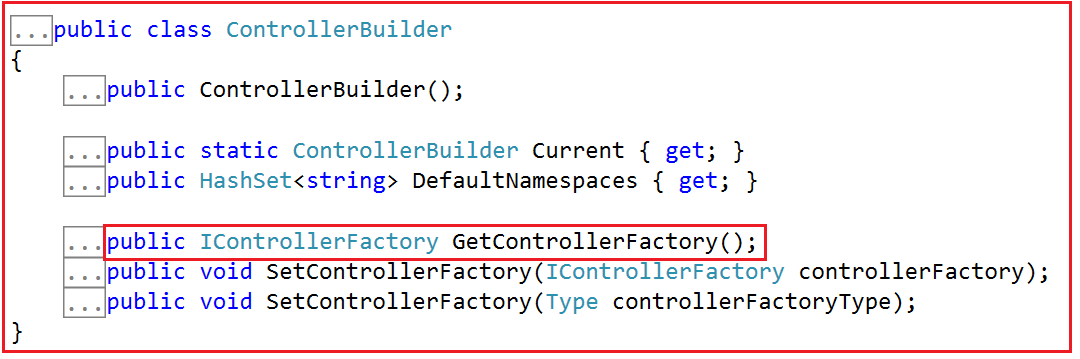
**Note:** The **RequestContext** object created by Route Module is available in each and every stage of the ASP.NET MVC Request processing Pipeline. The Request Context object holds the information about the Controller, action method, route data, etc.

**MvcHandler in ASP.NET MVC Life Cycle:**

The purpose of the **MVC Handler in the ASP.NET MVC** application is to create the **Controller Instance** which is going to process the current request. The definition of **MvcHandler** is given below.



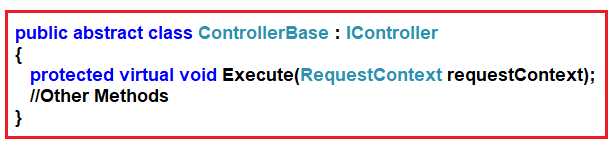
As shown in the above image, the **MvcHandler** is a class that implements the **IHttpHandler** interface. The **MvcHandler** takes the **RequestContext** object as its parameter. From the **URL** it gets the Controller name. Then it calls the **GetControllerFactory()** method of the **ControllerBuilder** which will create the **IControllerFactory** instance as shown in the below image.



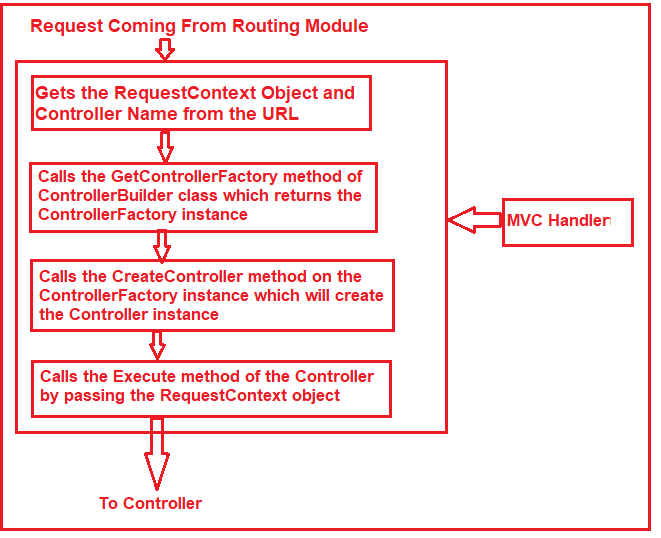
Once it gets the **ControllerFactory** instance (a class that implements the **IControllerFactory** interface) then it calls the **CreateController** method of the **IControllerFactory** interface by providing the **Controller name** and the **RequestContext** object as parameters as shown below in the below image.



The **CreateController** method of the **IControllerFactory** interface is responsible for creating the **controller** instance. Once the Controller instance is created, then it calls the **Execute** method of the Controller class by passing the **RequestContext** object. You can find the **Execute** method in **ControllerBase** class as shown in the below image.



**The steps we discussed here are shown in the following image.**

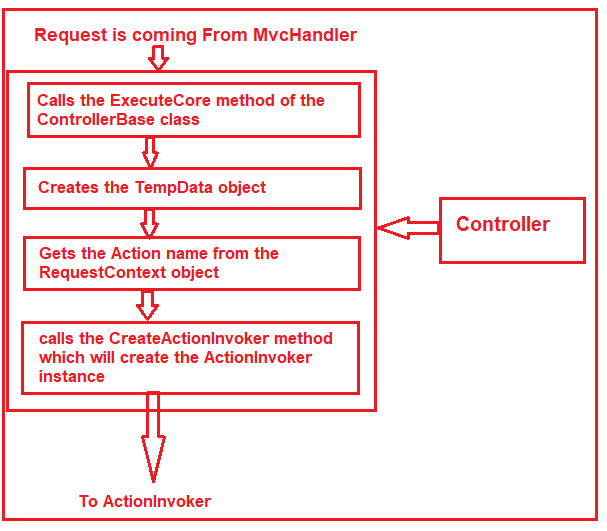


**The controller in ASP.NET MVC Life Cycle:**

The purpose of the Controller in **ASP.NET MVC Request Processing Pipeline** is to call the **ActionInvoker** method of the **Controller**class to deal with the current request.

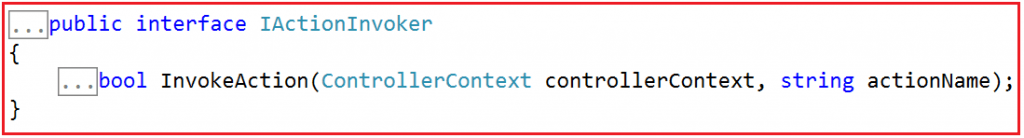
In the ASP.NET MVC application, the **Controller** class implements the **IController** interface as well as it also inherits from the **ControllerBase class**. The **ControllerBase** class exposes the **Execute** method which is called by the **MvcHandler** by providing the **RequestContext** object as a parameter.

Once the Execute method is called, then it calls the **ExecuteCore** method of the **ControllerBase** class. Then it creates the **TempData** object. The **ExecuteCore** method gets the Action name from the **RequestContext** object. Finally, the **ExecuteCore** methodcalls the **CreateActionInvoker method**which will create the **ActionInvoker** instance. The above steps we discussed here are shown in the following image.



**Action Invoker in ASP.NET MVC Request Life Cycle:**

The Role of Action Invoker in ASP.NET MVC Request Life Cycle is to execute the Action Method passed to it by the controller. The **ActionInvoker** class implements the **IActionInvoker** interface. The **IActionInvoker** interface has a single method i.e. **InvokeAction** as shown in the below image.



The Default Implementation is implemented in the **ControllerActionInvoker** class.

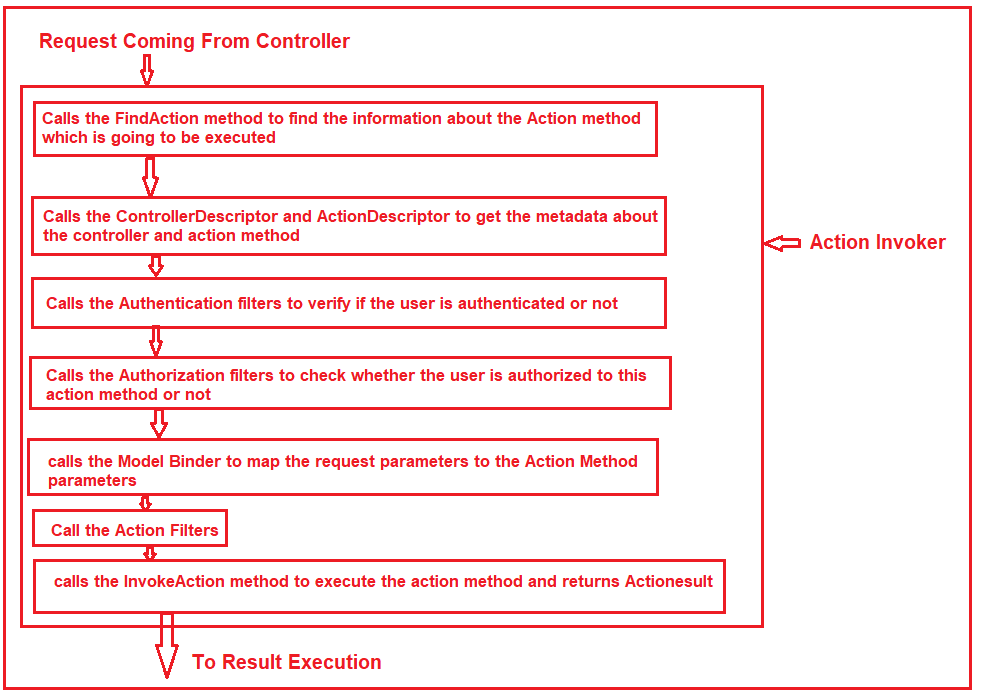
The **ControllerActionInvoker**calls the**FindAction method**to find the information about the Action method which is going to be executed. Then it uses the **ControllerDescriptor** and **ActionDescriptor** to get the metadata about the controller and action method such as **ControllerType**, **ControllerName**, **Filters,** **ActionName**, and **method** **Parameters,**etc.

The **ActionInvoker** then calls the Authentication filters to verify if the user is authenticated or not. If the user is an authenticated user then it calls the authorization filters to check whether the user is authorized to this action method or not.

If the user is an authorized user then the **ActionInvoker** calls the Model Binder object to map the request parameters to the Action Method parameters.  This process is called **Model Binding in the ASP.NET MVC application.**

Then it calls the Action Filters to add the pre and post-processing logic. Finally, the **ControllerActionInvoker** calls the **InvokeAction** method to execute the action method and returns the **ActionResult.**

**The above steps discussed in Action Invoker are shown in the below image for your better understanding.**



**Result Execution in ASP.NET MVC Application**

The Role of Result Execution in ASP.NET MVC application is to render the view as a web page to the client who initially made the request.

ActionInvoker in the previous stage executes the Action Method of the controller and then it returns the ActionResult. The ActionResult is an abstract class and an Action Method in ASP.NET MVC Controller can return many types of ActionResults such as ViewResult, JsonResult, ContentResult, FileResult, EmptyResult, etc. Among all the ActionResults, ViewResult is the one that is going to return an HTML page to the browser.

Once the Result is generated, then the Result Filters are going to be applied before sending the Final Response to the browser.

**Note:** The most important point that you need to remember is you can write your own implementation of all the components discussed here to implement your own logic.

In this article, I try to explain the **ASP.NET MVC Life Cycle** step by step. I hope you understood the **Life Cycle of the ASP.NET MVC Application**. Please give your valuable feedback about this article.