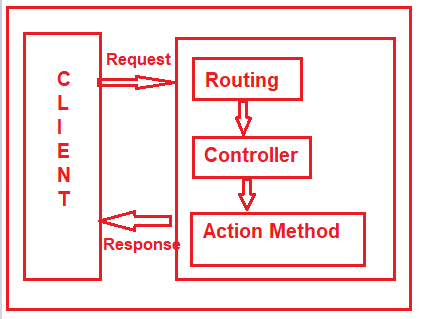
**Filters in ASP.NET MVC Application**

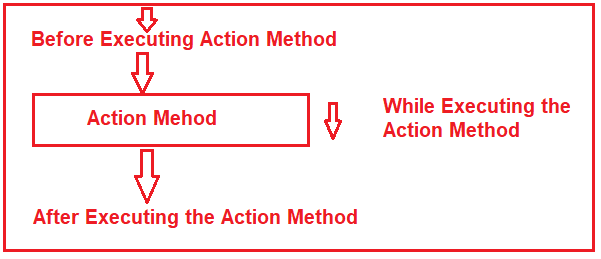
Filters are one of the most important concepts in ASP.NET MVC Application and as a developer, you should be aware of this concept.

##### **What are Filters in ASP.NET MVC Application?**

As of now, we discussed when a client makes a request, then that request comes to the Routing Engine and then the Routing Engine navigates that Request to the Controller. The controller then selects the appropriate action method to execute. So, it is the Controller action method that is going to handle the incoming request and send the response back to the client who initially made the request as shown in the below image.



But what will you do, if you want to execute some code or logic either before or after the action method executed as shown in the below image?



If that is your requirement then you need to use Filters in the ASP.NET MVC application. The Filters in ASP.NET MVC Framework are the attribute that allows us to inject some logic or code which is going to be executed either before or after an action method is invoked.

##### **Why do we need to use Filters in the ASP.NET MVC Applications?**

Basically, ASP.NET MVC Filters are used to perform the following common functionalities in your application.

1. Caching
2. Logging
3. Error Handling
4. Authentication and Authorization, etc.

##### **What are the Different Types of Filters available in ASP.NET MVC Framework?**

The ASP.NET MVC 5 framework provides five different types of Filters. They are as follows

1. **Authentication Filter (Introduced in MVC 5)**
2. **Authorization Filter**
3. **Action Filter**
4. **Result Filter**
5. **Exception Filter**

**Note:**This is also the order of the execution of Filters if more than one filter is applied. But the point that you need to remember is the Exception Filter can be executed at any point in time when there is an unhandled exception occurred in your application.

##### **What are the Predefined Filters?**

Some of the filters are already built by the ASP.NET MVC framework and they are ready to be used. For example

1. **Authorize**
2. **ValidateInput**
3. **HandleError**
4. **RequireHttps**
5. **OutputCache, etc**

##### **Can we Create Custom Filters in MVC?**

Yes, we can create custom filters in MVC. If the built-in filters do not serve our purpose then we can create our own custom filter as per our business requirements. We can create the Custom Filter for all the five categories (Authentication Filter, Authorization Filter, Action Filter, Result Filter, and Exception Filter) of Filters.

##### **Where we can configure filters in ASP.NET MVC?**

We can configure the filters at three different levels of our application. They are as follows

1. Global Level (Applicable to all controllers and all action methods)
2. Controller Level (Applicable to all the action methods of the particular controller)
3. Action Level (Applicable to the specific action methods)

##### **Configuring Filters at Global Level in ASP.NET MVC:**

Here you need to register the Filter within the **Application\_Start()** method of **Global.asax.cs** file as shown below. As we know this is the first method of our application which is going to be executed when the application starts. When you register a filter at the Global level, then it is applicable to all the Action Methods of all the Controllers of your MVC application.

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

**{**

FilterConfig.RegisterGlobalFilters**(**GlobalFilters.Filters**)**;

**}**

##### **Configuring Filters at Controller Level in ASP.NET MVC:**

Here you need to apply the filter at the top of the controller name as shown below. When you apply the filter at the Controller level, then it is applicable to all the action methods of that controller only.

**[**Authorize**(**Roles = "Admin"**)]**

**public** **class** AdminController : Controller

**{**

//Code

}

##### **Configuring Filters at Action Level in ASP.NET MVC:**

Here you need to apply the filter on the top of the action method name as shown below. When you apply the filter to a particular action method, then it is only applicable to that particular action method.

**public** **class** UserController : Controller

**{**

**[**Authorize**(**Users = "User1,User2"**)]**

**public** ActionResult LinkToLogin**(**string provider**)**

**{**

// Code

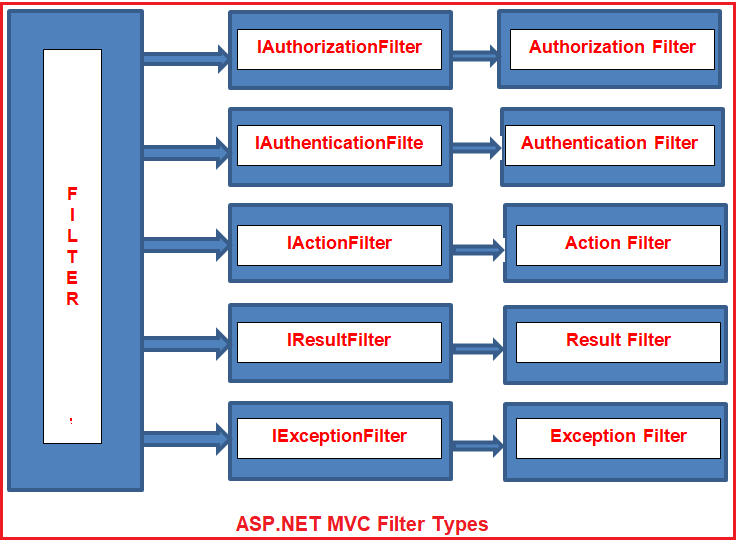
**return** View**()**;

**}**

**}**

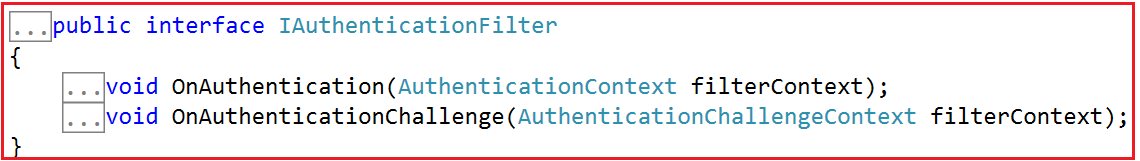
##### **Role and Responsibility of MVC Filters:**

As we already discussed we have five types of Filters (Authentication Filter, Authorization Filter, Action Filter, Result Filter, and Exception Filter) in the ASP.NET MVC application. Let us discuss the overview of each filter. Here we only discuss the purpose and when that filter is going to be executed and from our next article onwards we will discuss each filter in detail.



##### **Authentication Filter in ASP.NET MVC:**

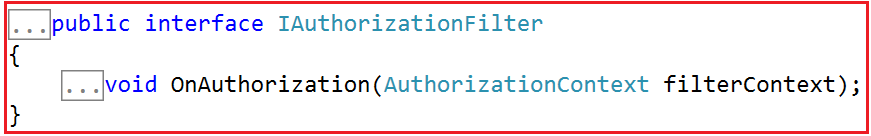
The Authentication filter is the first filter that is going to be executed before executing any other filter or action method. This filter checks that the user from where the request is coming is a valid user or not. The Authentication filters in ASP.NET MVC Framework implements the **IAuthenticationFilter** interface. This filter is introduced with ASP.NET MVC5. The **IAuthenticationFilter** interface is used to create a Custom Authentication filter. The definition of the **IAuthenticationFilter** interface is given below-



As of now, there is no in-built Authentication Filter provided ASP.NET MVC Framework. If you want to create Custom Authentication Filter then you need to implement the **IAuthenticationFilter** interface.

##### **Authorization Filters in ASP.NET MVC**

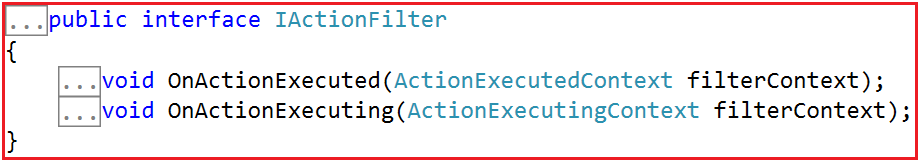
The Authorization Filters are executed after the Authentication Filter. This filter is used to check whether the user has the right to access a particular resource or page. The built-in **AuthorizeAttribute** and **RequireHttpsAttribute** are examples of Authorization Filters. The Authorization Filters in ASP.NET MVC Framework implements the **IAuthorizationFilter** interface. The definition of the **IAuthorizationFilter** interface is given below.



If you want to create a [**Custom Authorization Filter**](https://dotnettutorials.net/lesson/customizing-authorization-filter-mvc/) then you need to implement the **IAuthorizationFilter** interface.

##### **Action Filters in ASP.NET MVC:**

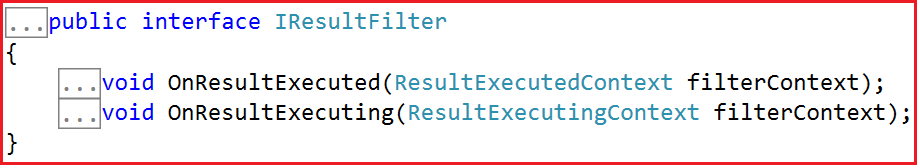
The Action Filters in ASP.NET MVC Application will be executed before the action method starts executing or after the action has been executed. So, if you want to execute some custom logic that is going to be executed before or after an action method is executed, then you need to use the Action Filters in MVC applications. The definition of the **IActionFilter** interface is given below.



The Action filters implement the **IActionFilter** interface that has two methods **OnActionExecuting** and **OnActionExecuted**. If you want to execute the Custom Logic before the action method starts executing, then you need to implement the OnActionExecuting method and if you want to write custom logic after the action method is executed, then you need to implement the OnActionExecuted method.

##### **Result Filters in ASP.NET MVC:**

The Result filters in the ASP.NET MVC application are executed before or after generating the result for an action. Action Result type can be ViewResult, PartialViewResult, RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResult and EmptyResult which derives from the ActionResult abstract class. Result filters are called after the Action filters. The in-built **[OutputCacheAttribute](https://dotnettutorials.net/lesson/outputcache-attribute-mvc/)**is an example of Result Filters. The Result Filters in MVC implements the **IResultFilter** interface. The definition of the IResultFilter interface is given below.



The Result filters implement the **IResultFilter** interface that has two methods **OnResultExecuting** and **OnResultExecuted**. If you want to execute the Custom Logic before generating the result, then you need to implement the **OnResultExecuting** method and if you want to write custom logic after generating the result, then you need to implement the **OnResultExecuted**method. If you want to create a Custom Result Filter then you need to implement the **IResultFilter** interface.

##### **Exception Filters in ASP.NET MVC:**

The Exception filters are executed when there is an unhandled exception occurs during either the execution of actions or filters. The in-built **[HandleErrorAttribute](https://dotnettutorials.net/lesson/exception-filter-mvc/)**is an example of Exception Filters. The **IExceptionFilter** interface is used to create a [**Custom Exception Filter**](https://dotnettutorials.net/lesson/custom-exception-filter-mvc/) which provides the **OnException** method which will be executed when there is an unhandled exception occurs during the actions or filters execution. The definition of **IExceptionFilter** is given below.

