Types of Filters

ASP.NET MVC framework supports the following action filters −

* **Action Filters** − Action filters are used to implement logic that gets executed before and after a controller action executes. We will look at Action Filters in detail in this chapter.
* **Authorization Filters** − Authorization filters are used to implement authentication and authorization for controller actions.
* **Result Filters** − Result filters contain logic that is executed before and after a view result is executed. For example, you might want to modify a view result right before the view is rendered to the browser.
* **Exception Filters** − Exception filters are the last type of filter to run. You can use an exception filter to handle errors raised by either your controller actions or controller action results. You also can use exception filters to log errors.

Action filters are one of the most commonly used filters to perform additional data processing, or manipulating the return values or cancelling the execution of action or modifying the view structure at run time.

| **Filter Type** | **Description** | **Built-in Filter** | **Interface** |
| --- | --- | --- | --- |
| Authorization filters | Performs authentication and authorizes before executing action method. | [Authorize], [RequireHttps] | IAuthorizationFilter |
| Action filters | Performs some operation before and after an action method executes. |  | IActionFilter |
| Result filters | Performs some operation before or after the execution of view result. | [OutputCache] | IResultFilter |
| Exception filters | Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline. | [HandleError] | IExceptionFilter |

**Filter Attributes**

MVC provides abstract base classes that you can inherit from to create custom filters. These abstract classes inherit from *Attribute* class and therefore can be used to decorate Controllers and action methods.

* *ActionFilterAttribute*
* *ResultFilterAttribute*
* *ExceptionFilterAttribute*
* *ServiceFilterAttribute*
* *TypeFilterAttribute*

**Filter Types**

There are various types of filters that run at different stages of the filter pipeline. Below, a figure from official documentation illustrates the sequence:

**Authorization**

This is the first filter to run and short circuit the request for unauthorized users. They only have one method (unlike most other filters that have *Executing* and *Executed* methods). Normally, you won’t write your own Authorization filters, the built-in filter calls into framework’s authorization mechanism.

**Resource**

They run before model binding and can be used for changing how it behaves. Also, they run after the result has been generated and can be used for caching etc.

**Action**

They run before and after the action method, thus are very useful to manipulate the action parameters or its result. The context supplied to these filters lets you manipulate the action parameters, controller and result.

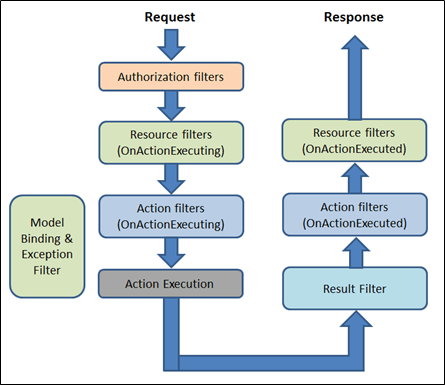
**Exception**

They can be used for unhandled exception before they’re written to the response. Exception handling middleware works for most scenarios however this filter can be used if you want to handle errors differently based on the invoked action.

**Result**

They run before and after the execution of action method’s result, if the result was successful. They can be used to manipulate the formatting of the result.

Following diagram shows how these filters interact in filter pipeline during request and response life cycle.



What is SignalR

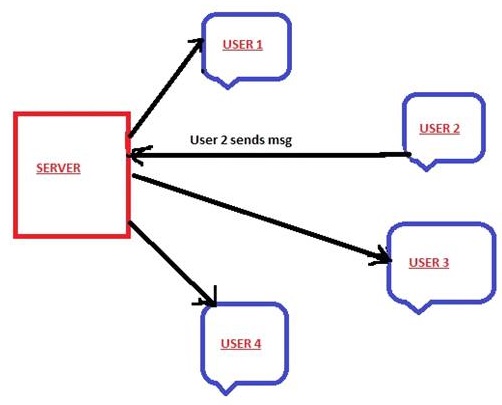
SignalR is a new developer's API provided for ASP.NET web applications, used to add "real time" web functionality to ASP.NET applications. "Real Time" web functionality is the ability to have server code to push contents to connected clients.

SignalR supports "server push" or "broadcasting" functionality. It handles connection management automatically. In classic HTTP connections for client-server communication connection is re-established for each request, but SignalR provides persistent connection between the client and the server. In SignalR the server code calls out to a client code in the browser using Remote Procedure Calls (RPC), rather than request-response model today. SignalR is an open-source API, and is accessible through GitHub.

Where to use:

1. Chat room applications
2. Real-time monitoring applications
3. Job progress updates
4. Real time forms

You can see the use of SignalR for a chat room application in the following image:



In the above example as soon as user2 sends some message, it will be received by all other users.

API Details

SignalR provides two models for communication:

1. Persistent Connections

The Persistent Connection API gives developer direct access to the low level communication protocol that SignalR exposes. This API uses the format of the actual message sent that needs to be specified and if the developer prefers to work with messaging and dispatching model rather than a remote invocation.

1. Hubs:

It's a High Level API written over PersistentConnection. This API allows the client and server to call methods on each other directly. Hubs also allow you to pass strongly typed parameters to methods, enabling model binding