**MVC 3 FILter order and not web-API**

The ASP.NET MVC framework supports the following types of filters:

1. Authorization filters, which implement the [IAuthorizationFilter](http://msdn.microsoft.com/en-us/library/system.web.mvc.iauthorizationfilter(v=vs.98).aspx) interface.
2. Action filters, which implement the [IActionFilter](http://msdn.microsoft.com/en-us/library/system.web.mvc.iactionfilter(v=vs.98).aspx) interface.
3. Response filters, which implement the [IResultFilter](http://msdn.microsoft.com/en-us/library/system.web.mvc.iresultfilter(v=vs.98).aspx) interface.
4. Exception filters, which implement the [IExceptionFilter](http://msdn.microsoft.com/en-us/library/system.web.mvc.iexceptionfilter(v=vs.98).aspx) interface.

Filters run in the order listed — for example, authorization filters run first and exception filters run last. Within each filter type, the [Order](http://msdn.microsoft.com/en-us/library/system.web.mvc.filter.order(v=vs.98).aspx) value specifies the run order. Within each filter type and order, the [Scope](http://msdn.microsoft.com/en-us/library/system.web.mvc.filter.scope(v=vs.98).aspx) value specifies the order for filters. This enumeration defines the following filter scope values (in the order in which they run):

1. First
2. Global
3. Controller
4. Action
5. Last

For example, an action filter that has the [Order](http://msdn.microsoft.com/en-us/library/system.web.mvc.filter.order(v=vs.98).aspx) property set to zero and filter scope set to First runs before an action filter that has the [Order](http://msdn.microsoft.com/en-us/library/system.web.mvc.filter.order(v=vs.98).aspx) property set to zero and filter scope set to Action.

<http://www.asp.net/mvc/tutorials/older-versions/controllers-and-routing/understanding-action-filters-cs>

The UI logic belongs in the view. Input logic belongs in the controller. Business logic belongs in the model

http://mvc-tutorials.com/differences-between-viewdata-viewbag-session-and-tempdata-in-mvc3

**ViewData**

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

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

3.It’s life lies only during the current request.

4.If redirection occurs then it’s value becomes null.

5.It’s required typecasting for complex data type and check for null values to avoid error.

**ViewBag**

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

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

3.It’s life also lies only during the current request.

4.If redirection occurs then it’s value becomes null.

5.It doesn’t required typecasting for complex data type.

**TempData**

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

2.TempData is used to pass data from current request to subsequent request means incase of redirection.

3.It’s life is very short and lies only till the target view is fully loaded.

4.It’s required typecasting for complex data type and check for null values to avoid error.

5.It is used to store only one time messages like error messages, validation messages.

Unity Dependency

You can include only one **<constructor>** element in each **register** section.

You cannot nest containers in the configuration file. All **<container>**elements reside at the same "level" within the **<containers>** element. You configure nested containers by creating the containers in the required hierarchy in your code and then populating them from the appropriate**<container>** elements. If required, you can load more than one container from the same **<container>** element, and you can use more than one container element in a configuration file.

<http://www.asp.net/web-api/overview/web-api-routing-and-actions/exception-handling>

Exception Handling in ASP.NET Web API

* [HttpResponseException](http://www.asp.net/web-api/overview/web-api-routing-and-actions/exception-handling#httpresponserexception)
* [Exception Filters](http://www.asp.net/web-api/overview/web-api-routing-and-actions/exception-handling#exception_filters)
* [Registering Exception Filters](http://www.asp.net/web-api/overview/web-api-routing-and-actions/exception-handling#registering_exception_filters)
* [HttpError](http://www.asp.net/web-api/overview/web-api-routing-and-actions/exception-handling#httperror)

<http://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-and-action-selection>

Selecting a Controller

Controller selection is handled by the **IHttpControllerSelector.SelectController** method. This method takes an**HttpRequestMessage** instance and returns an **HttpControllerDescriptor**. The default implementation is provided by the **DefaultHttpControllerSelector** class. This class uses a straightforward algorithm:

1. Look in the route dictionary for the key "controller".
2. Take the value for this key and append the string "Controller" to get the controller type name.
3. Look for a Web API controller with this type name.

For example, if the route dictionary contains the key-value pair "controller" = "products", then the controller type is "ProductsController". If there is no matching type, or multiple matches, the framework returns an error to the client.

For step 3, **DefaultHttpControllerSelector** uses the **IHttpControllerTypeResolver** interface to get the list of Web API controller types. The default implementation of **IHttpControllerTypeResolver** returns all public classes that (a) implement **IHttpController**, (b) are not abstract, and (c) have a name that ends in "Controller".

| Interface | Description |
| --- | --- |
| **IHttpControllerSelector** | Selects the controller. |
| **IHttpControllerTypeResolver** | Gets the list of controller types. The **DefaultHttpControllerSelector**chooses the controller type from this list. |
| **IAssembliesResolver** | Gets the list of project assemblies. The **IHttpControllerTypeResolver**interface uses this list to find the controller types. |
| **IHttpControllerActivator** | Creates new controller instances. |
| **IHttpActionSelector** | Selects the action. |
| **IHttpActionInvoker** | Invokes the action. |

Can we render user control (ascx) in MVC view.

Yes

<http://www.codeproject.com/Questions/265326/How-to-View-Ascx-page-in-Mvc-View>

<![CDATA[<% Html.RenderPartial("~/Views/Shared/YourUserControl.ascx", {data model object}) %>]]>

<![CDATA[<%= Html.RenderUserControl("~/Views/Shared/YourUserControl.ascx", {data model object}) %>]]>

**?? null-coalescing operator**

The ?? operator is called the null-coalescing operator and is used to define a default value for nullable value types or reference types. It returns the left-hand operand if the operand is not null; otherwise it returns the right operand.

By default the put and delete are not enabled in IIS. You need to uncheck webdav/webdavmodule to enable them.

ModelBinder: <http://blogs.msdn.com/b/jmstall/archive/2012/04/20/how-to-bind-to-custom-objects-in-action-signatures-in-mvc-webapi.aspx>

Web-API Message handlers:

<http://www.asp.net/web-api/overview/working-with-http/http-message-handlers>

WEB-API piple line:

<http://www.dotnetcurry.com/ShowArticle.aspx?ID=888>

<http://www.asp.net/posters/web-api/ASP.NET-Web-API-Poster-grayscale.pdf>

http://www.c-sharpcorner.com/UploadFile/2b481f/lifecycle-of-Asp-Net-web-api-message/

Understanding how the parameterbinding works

<http://www.strathweb.com/2013/04/asp-net-web-api-parameter-binding-part-1-understanding-binding-from-uri/>

**How to Create Asynchronous Controller**

1. Inherits MVC controller with **AsyncController** instead of **Controller**
2. Create two methods for the action.  The method which will initiate the asynchronous process should end with “**Async**” and its return type must be void. The method that will get invoked automatically when the asynchronous process finishes (callback method) must have a name consists of the action and end with “**Completed**” and its return type must be **ActionResult**.

Automatic ETag Management with Web API Message Handlers:

<http://codebetter.com/howarddierking/2011/07/01/automatic-etag-management-with-web-api-message-handlers/>

The XML serializer does not support anonymous types or JObject instances.

About serialization:

<http://www.asp.net/web-api/overview/formats-and-model-binding/json-and-xml-serialization>

An Introduction To RESTful Services With WCF:

<http://msdn.microsoft.com/en-us/magazine/dd315413.aspx>

Why web-api and not wcf?

Web-api Vs MVC3

How is web-api light weight?

Delegate and use of delegate in project

Data Annotations.

What is model state?

What is singleresult in LINQ