MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
MAY 25, 2020 3:00 AM PDT

How to use dependency injection in action filters in ASP.NET Core 3.1

Take advantage of dependency injection to make your action methods in ASP.NET Core lean, clean, and maintainable

Filters enable you to execute code at certain stages of the request processing pipeline. An action filter is a filter that is executed before or after the execution of an action method. By using action filters, you can make your action methods lean, clean, and maintainable.

There are several filters available for action methods, each of which corresponds to a different stage of the request processing pipeline. You can read more about action filters in my earlier article here. In this article we'll discuss how we can work with dependency injection in action filters in ASP.NET Core 3.1.

[Also on InfoWorld: 9 lies programmers tell themselves]

To work with the code examples provided in this article, you should have Visual Studio 2019 installed in your system. If you don't already have a copy, you can download Visual Studio 2019 here.

Create an ASP.NET Core 3.1 MVC project in Visual Studio 2019

UNITED STATES

First off, let's create an ASP.Net Core project in Visual Studio 2019. Assuming Visual Studio 2019 is installed in your system, follow the steps outlined below to create a new ASP.Net Core project in Visual Studio.

1. Launch the Visual Studio IDE.
2. Click on "Create new project."
3. In the "Create new project" window, select "ASP.NET Core Web Application" from the list of templates displayed.
4. Click Next.
5. In the "Configure your new project" window, specify the name and location for the new project.
6. Optionally check the "Place solution and project in the same directory" check box, depending on your preferences.
7. Click Create.
8. In the "Create a New ASP.NET Core Web Application" window shown next, select .NET Core as the runtime and ASP.NET Core 3.1 (or later) from the drop-down list at the top.
9. Select "Web Application (Model-View-Controller)" as the project template to create a new ASP.NET Core MVC application.

UNITED STATES

10. Ensure that the check boxes “Enable Docker Support” and “Configure for HTTPS” are unchecked as we won’t be using those features here.
11. Ensure that Authentication is set to “No Authentication” as we won’t be using authentication either.
12. Click Create.

Following these steps should create a new ASP.Net Core MVC project in Visual Studio 2019. We’ll use this project in the sections below to illustrate the use of dependency injection in action methods in ASP.NET Core 3.1.

Use the RequestServices.GetService method in ASP.NET Core 3.1

Although it’s not a recommended practice, an easy way to retrieve a service that has been added to the service collection is by using the RequestServices.GetService method. Assuming that you have added a service named CustomService to your services container, you can use the following code to retrieve the service instance.

```
public override void OnActionExecuting(ActionExecutingContext context)
        {
            var customService = context.HttpContext.RequestServices.
            GetService(typeof(CustomService));
            //Write your code here
            _logger.LogWarning("Inside OnActionExecuting method...");
            base.OnActionExecuting(context);
        }
```

Use the ServiceFilter attribute in ASP.NET Core 3.1

A better practice is to take advantage of the ServiceFilter attribute — it can be applied at the controller or the action level. Here is the syntax for using this attribute.

RECOMMENDED WHITEPAPERS

BEI Idaas Buyers' Guide

UNITED STATES
The Problem with Password Managers: Smart Alternatives for IT Help Desk Managers
CYBERA

Evaluating Payment HSMs in Financial Services

```
[ServiceFilter(typeof(typeof YourActionFilter))]
```

Below we'll learn how to leverage the ServiceFilter attribute for dependency injection in action filters.

Create a custom ActionFilterAttribute class in ASP.NET Core 3.1

Let's create a custom ActionFilterAttribute class and use it as a ServiceFilter. To create a custom action filter in ASP.NET Core 3.1, you should create a class that extends the ActionFilterAttribute class as shown in the code snippet given below. Note that the ActionFilterAttribute class implements the IActionFilter, IAsyncActionFilter, IResultFilter, IAsyncResultFilter, and IOrderedFilter interfaces.

```
public class CustomActionFilter : ActionFilterAttribute
    {
        private readonly ILogger _logger;
    public CustomActionFilter(ILoggerFactory loggerFactory)
        {
            _logger = loggerFactory.CreateLogger("CustomActionFilter");
        }
     public override void OnActionExecuting(ActionExecutingContext context)
        {
            _logger.LogWarning("Inside OnActionExecuting method...");
            base.OnActionExecuting(context);
        }
    public override void OnActionExecuted(ActionExecutedContext context)
        {
            _logger.LogWarning("Inside OnActionExecuted method...");
            base.OnActionExecuted(context);
        }
    public override void OnResultExecuting(ResultExecutingContext context)
        {
            _logger.LogWarning("Inside OnResultExecuting method...");
            base.OnResultExecuting(context);
        }
    public override void OnResultExecuted(ResultExecutedContext context)
        {
            _logger.LogWarning("Inside OnResultExecuted method...");
            base.OnResultExecuted(context);
        }
    }
```

Register a custom action filter in ASP.NET Core 3.1

In ASP.NET Core you can add services to the services container easily. Dependency injection is not only a first-class citizen in ASP.NET Core, but support for dependency injection is built-in. Hence you can inject both framework services and application services easily. You can use any of the following methods to add your services using dependency injection in ASP.NET Core.

- AddTransient
- AddScoped
- AddSingleton

UNITED STATES

You can learn more about dependency injection and what each of these methods does from my earlier article here.

Next, you must register the custom filter so that it can be used as a ServiceType. The following code snippet illustrates how this can be achieved.

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddScoped<CustomActionFilter>();
  services.AddControllersWithViews();
}
```

Use a custom action filter in the controller in ASP.NET Core 3.1

You can take advantage of the ServiceFilter attribute to inject dependencies in your controller or your controller's action methods. Here is the syntax for using the ServiceFilter attribute.

```
[ServiceFilter(typeof(MyCustomFilter))]
```

The code snippet below shows how you can apply the custom action filter to an action method in a controller class named HomeController. Note how the ServiceFilter attribute has been used.

```
public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }
        [ServiceFilter(typeof(CustomActionFilter))]
        public IEnumerable<string> Get()
        {
            _logger.LogInformation("Inside HttpGet action method...");
            return new string[] { "Joydip", "Michael", "Anand", "Steve" };
        }
    }
```

Set the execution order of the action filters in ASP.NET Core 3.1

You can even set the execution order of the action filters. The action filters that have a higher execution order will be executed later in the execution cycle. The usual execution order of action filters is as follows:

1. OnActionExecuting method of Global action filter
2. OnActionExecuting method of Controller action filter
3. OnActionExecuting method of Action filter
4. Action method
5. OnActionExecuted method of Action filter
6. OnActionExecuted method of Controller action filter
7. OnActionExecuted method of Global action filter

You can set the execution order of an action filter individually. The code snippet below shows how you can achieve this.

```
[ServiceFilter(typeof(CustomActionFilter), Order = 2)]
public IEnumerable<string> Get()
{
   _logger.LogInformation("Inside HttpGet action method...");
    return new string[] { "Joydip", "Michael", "Anand", “Steve” };
}
```

When you execute your application, you’ll observe that the CustomActionFilter is executed after the Global action filter is executed.

Use the TypeFilter attribute in ASP.NET Core 3.1

You can also use type filters to inject dependencies into action filters. The ServiceFilter attribute works nicely for dependencies that are resolved via the IoC (inversion of control) container. TypeFilter works much the same way as ServiceFilter but doesn’t need to be registered with the services container. Moreover, type filters can also be used to pass additional parameters to the action filter.

The following code snippet illustrates how TypeFilter can be used.

```
[TypeFilter(typeof(CustomFilter), Arguments = new object[] {"This is a string parameter"
[HttpGet("")]
public IEnumerable<string> Get()
{
  _logger.LogInformation("Inside HttpGet action method...");
  return new string[] { "Joydip", "Michael", "Anand" };
}
```

Use global action filters in ASP.NET Core 3.1

You can also apply filters globally by adding your action filter to the Filters collection of the IServiceCollection instance as shown in the code snippet given below.

```
{                    UNITED STATES
    services.AddControllers(config =>
    {
        config.Filters.Add<CustomFilter>();
    }).SetCompatibilityVersion(CompatibilityVersion.Version_3_0);
}
```

This enables you to apply the filter to every action method of every controller in your project.

One of the biggest benefits of dependency injection is that it allows you to build applications that are loosely coupled. When working with your controllers and their action methods you might often need to inject filters that modify the way those actions are executed. ASP.NET Core 3.1 allows you to do this easily.

How to do more in .NET Core:

- How to use in-memory caching in ASP.NET Core
- Advanced versioning in ASP.NET Core Web API
- How to use dependency injection in ASP.NET Core
- How to use async and await in .NET Core
- How to consume an ASP.NET Core Web API using RestSharp
- How to consume a WCF SOAP service in ASP.NET Core
- How to improve the performance of ASP.NET Core applications
- Testing with the InMemory provider in Entity Framework Core
- How to work with logging in ASP.NET Core
- How to use MediatR in ASP.NET Core
- How to use System.Threading.Channels in .NET Core
- More advanced AutoMapper examples in .NET Core
- How to work with session state in ASP.NET Core
- How to use Nancy in ASP.NET Core
- How to implement rate limiting in ASP.NET Core

UNITED STATES

- How to implement global exception handling in ASP.NET Core Web API
- How to use IHostedService in ASP.NET Core
- How to work with cookies in ASP.NET Core
- How to use Autofac in ASP.NET Core
- How to upload files in ASP.NET Core MVC
- How to use URL Rewriting Middleware in ASP.NET Core
- How to handle null values in ASP.NET Core MVC
- How to work with worker services in ASP.NET Core
- How to use Elmah for error logging in ASP.NET Core
- How to implement health checks in ASP.NET Core
- How to use Glimpse in ASP.NET Core
- How to use Swagger in ASP.NET Core
- How to implement global exception handling in ASP.NET Core MVC
- How to use conditional middleware in ASP.NET Core
- How to use Azure Application Insights in ASP.NET Core
- How to use NCache in ASP.NET Core
- How to return data from ASP.NET Core Web API
- How to use response caching middleware in ASP.NET Core
- How to schedule jobs using Quartz.NET in ASP.NET Core
- How to enable CORS in ASP.NET Core
- How to use session storage in ASP.NET Core

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

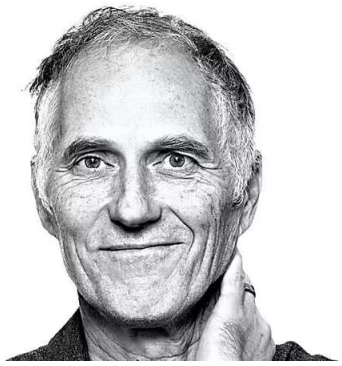
Copyright © 2020 IDG Communications, Inc.

UNITED STATES

- Stay up to date with InfoWorld's newsletters for software developers, analysts, database programmers, and data scientists.
- Get expert insights from our member-only Insider articles.

YOU MAY ALSO LIKE

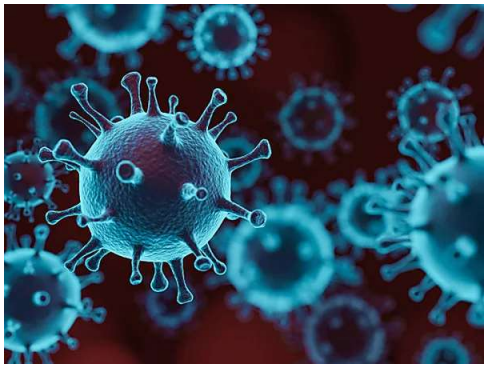
Recommended by



Tim O'Reilly: the golden age of the programmer is over

10 top-notch libraries for C++ programming

Red Hat Enterprise Linux takes aim at edge computing



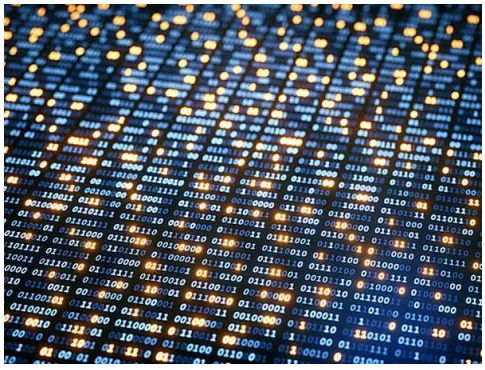
The COVID pandemic's lasting impact on cloud usage

Deno Company forms to back Node.js rival

Google's Logica language addresses SQL's flaws

3 enterprise AI success stories

Oracle Database 21c review: The old RDBMS is



What is a computational storage drive? Much-

How IT priorities are shifting during the COVID-19 crisis

The best open source software of 2020

SPONSORED LINKS

dtSearch® instantly searches terabytes of files, emails, databases, web data. See site for hundreds of reviews; enterprise & developer evaluations

Truly modern web app and API security thinking. It's a thing. See how.

Want lightning fast analytics? See why the Incorta data analytics platform is changing enterprise data forever.

2020 was a year of rapid progression of digital transformation for businesses. The following is a snapshot of the digital transformation advancements made across all facets of business.

DDoS extortion attacks are real. Don’t Negotiate. Mitigate with NETSCOUT. Learn more.

IDG COMMUNICATIONS Copyright © 2021 IDG Communications, Inc.