DEPENDENCY INJECTION IN ACTION FILTERS IN ASP.NET CORE

iFour Team - 12 Feb 2021

It is quite common to decorate the ASP.NET MVC controller actions with filter attributes to differentiate cross-cutting concerns from the main concern of the action. Sometimes these filters require the use of other components but the attributes are very

Table of Content

DI and action filter attributes

The Attributes in C # are very simple. You can pass static values to the constructor and/or set public properties directly.

Because attribute parameters are evaluated at compile time, they should be compiled time constant. So injecting dependencies from an IoC container is not an option. In situation where we want an attribute to have access to another component, we mu use a workaround.

Let's look at some options:

The RequestServices.GetService Service Locator

If we cannot inject a component into our attribute, it seems that the next best option is to request the component from our IoC container (either directly or by wrapper).

In the .NET core, we can use the service location to resolve components from a built-in IoC container using the RequestServices.GetService:

```
1.      public class ThrottleFilterAttribute : Attribute, IActionFilter
2.  {
```

```
    ...
}
</idistributedcache>
```

Note that in order to use the generic version of the GetService you need to add the following statements:

```
using Microsoft.Extensions.DependencyInjection;
```

This will work but is not recommended. Unlike constructor injection, it can be much harder to work out your dependence with the service locator (anti)pattern. The beauty of constructor injection is simplicity. Just by looking at the definition of a constructor, we immediately know all the classes on which the class depends. The service location makes the unit test harder which requires you to have more knowledge about the internals of the subject under test.

Read More: Hashing, Encryption And Random In Asp.net Core

The ServiceFilter attribute

The servicefilter attribute may be used at the action or controller level. Usage is ve straightforward:

```
[ServiceFilter(typeof(ThrottleFilter))]
```

The servicefilter attribute allows us to specify the type of our action filter and can automatically resolve the class from the built-in IoC container. This means we can change our action filter to accept dependencies directly from the constructor:

```
public class ThrottleFilter :IActionFilter
{
    private readonlyIDistributedCache _cache;

    public ThrottleFilter(IDistributedCache cache)
```

Naturally, as we are resolving our filters from the IoC container, we need to register it:

```
1.  public void ConfigureServices(IServiceCollection services)
2.  {
3.      ...
4.  services.AddScoped<throttlefilter>();
5.      ...
6.  }
7.  </throttlefilter>
```

The TypeFilter attribute

The servicefilter is very useful for attributes that have dependencies that need to be resolved from the IoC container but the lack of property support is a major limitation. If we modify our ThrottleFilter example to add configuration properties (while retaining IDistributedCache dependencies), then the servicefilter is no longer useful to us. We can however use typefilter.

```
1.  public class ThrottleFilter :IActionFilter
2.  {
3.      private readonlyIDistributedCache _cache;
4.
5.      public intMaxRequestPerSecond{ get; set; }
6.
7.      public ThrottleFilter(IDistributedCache cache, intmaxRequestPerSecond)
8.      {
9.          _cache = cache ?? throw new ArgumentNullException(nameof(cache));
10. MaxRequestPerSecond = maxRequestPerSecond;
11.     }
12. }
```

The typefilter is similar to the servicefilter, but there are two notable differences:

- The type being resolved doesn't need to be registered with the IoC container.
- Arguments can be provided that are used while constructing the filter.

Global action filters

```
5.  options.Filters.Add<throttlefilter>();
6.      });
7.  }
8.  </throttlefilter>
```

This allows us to resolve the filter from the IoC container in the same way as using a servicefilter attribute on an action or controller but instead, it will be applied to every action of each controller. You cannot use global filter isolation even if you need to be able to configure individual actions.

Passive Attributes

If we think about it, it does not make sense for action filter attributes to house complex logic and interactions with other components. Attributes are best for identification and by choosing which actions or controllers we belong to. They are also best for individual actions or configurations of controllers. They probably shouldn't be full of complex code.

The presence of a ThrottleAttribute on the action indicates that it should be throttled. This is an opt-in approach but we can easily use an opt-out approach where an attribute indicates that we do not want to throttle the action.

We also want to be able to alternately refer to maximum requests per second. Implementation is trivial:

```
1.  public class ThrottleAttribute : Attribute, IFilterMetadata
2.  {
3.      public intMaxRequestPerSecond{ get; set; }
```

beginning of the section. All dependencies are resolved automatically by the built-in IoC container.

```
1.  public class ThrottleFilter :IActionFilter
2.  {
3.      private readonlyIDistributedCache _cache;
4.
5.      public ThrottleFilter(IDistributedCache cache)
6.      {
7.          _cache = cache ?? throw new ArgumentNullException(nameof(cache));
8.      }
9.
10.     public void OnActionExecuted(ActionExecutedContext context)
11.     {
12.         //noop
13.     }
14.
15.     public void OnActionExecuting(ActionExecutingContext context)
16.     {
17. varthrottleAttribute = context.ActionDescriptor.FilterDescriptors
18. .Select(x =>x.Filter).OfType<throttleattribute>().FirstOrDefault();
19.
20.         if (throttleAttribute != null)
21.         {
22.
23.         }
24.     }
25. }
26. </throttleattribute>
```

Conclusion

It is not possible to inject components directly into action filter attributes but there are various workarounds to allow the same thing to be accomplished effectively. Using a servicefilter is a relatively clean way to allow dependency injection into individual action filters. Specifying the type for the filter in this way does mean that invalid types can be entered and will not be discovered until runtime, however.

iFour Technolab Pvt. Ltd.
Sustainable Solution

Select Country

recent post

20 Sep 2021
A comprehensive guide on advanced React Component Patterns [more]

17 Sep 2021
An in-depth guide on View Result and Partial View Result in MVC [more]

17 Sep 2021
The advent of E-commerce during the COVID-19 pandemic [more]

category

- Project Management (24)
- Business verticals (1)
  - Aviation (2)
  - Retail (2)
  - Healthcare Development (4)
  - Finance Development (2)
  - Transportation & Logistics (5)
  - Legal development (1)
  - Education Development (3)
  - Hospitality Development (1)
  - Fleet development (1)

Get Quote

iFour Technolab Pvt. Ltd.
Sustainable Solution

Select Country

- Content Management System (7)
- Digital India (4)
- eCommerce solutions (25)
- eGovernance (6)
- Enterprise Solutions (11)
  - BDaaS (2)
  - CRM (5)
  - ERP (4)
  - SaaS (5)
  - Supply chain management (2)
- Indian Government Initiatives (4)
- Mobile Application Development (22)
  - Android app development (4)
  - Hybrid Mobile App Development (4)
  - App Store Optimitation (1)
  - Mobile App Development Ideas (2)
- Security (20)
- Software Outsourcing (9)
- Software Quality (5)
- Software Technologies (50)

Get Quote

- Charting tools (2)
- Digital Currency (3)
- JavaScript frameworks (5)
- NopCommerce (2)
- Telerik AppBuilder (2)
- VSTO Addin development (1)
- Xamarin (16)
- IoT - Internet of Things (2)
- nodejs Development (15)
- VueJs (2)
- Angular Development (45)
- ASP.NET Development (91)
- Javascript (3)
- Ionic (2)
- WPF Development (10)
- C# development (7)
- MVC development (21)
- Java Development (4)

Technology News (9)

Top ten (28)

iFour Technolab Pvt. Ltd.
Sustainable Solution

Select Country

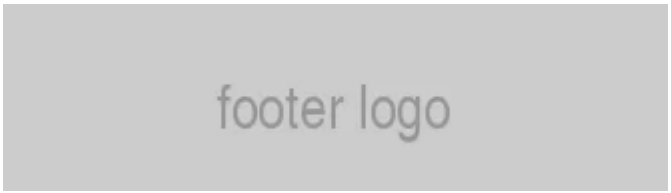
Cloud (12)

Web Development (117)

Technology Trends (18)

Database Management (2)

React JS (7)



FOLLOW US ON

COMPANY

- Company
- Events & News
- Career
- Contact Us

OUR EXPERTISE

Get Quote

iFour Technolab Pvt. Ltd.
Sustainable Solution

Select Country

- Insights

BUSINESS WITH US

- IP Protection
- Business Affiliates
- Process
- Send RFP

WEBSITE USE

- Home
- Privacy
- Terms Of Use
- Sitemap