



MICROSOFT ARCHITECT

By Joydip Kanjilal, Columnist, InfoWorld
OCT 16, 2017 3:00 AM PDT

How to use dependency injection in ASP.NET Core

Take advantage of dependency injection in ASP.NET Core to plug in components and improve code maintenance and testability

Support for dependency injection is built into ASP.NET Core, Microsoft's open source, cross platform, lean, and modular framework for building high performance, scalable web applications. In ASP.NET Core, both framework services and application services can be injected into your classes, rather than being tightly coupled. In this article we'll look at how we can work with dependency injection in ASP.NET Core.

Dependency injection (also known as DI) is a design pattern in which a class or object has its dependent classes injected (passed to it by another class or object) rather than create them directly. Dependency injection facilitates loose coupling and promotes testability and maintenance. Moreover, dependency injection allows you to change your implementations without having to change the classes or interfaces that leverage those implementations.

[[Also on InfoWorld: The most valuable software developer skills to get hired now](#)]

Making a service available via dependency injection in ASP.NET

We will now build a simple service in Visual Studio using ASP.NET Core and illustrate how we can add it to the dependency injection container, register it with the pipeline, and then consume it in our application. Follow these steps to create a new ASP.NET Core project in Visual Studio 2017 or Visual Studio 2015. If you're using Visual Studio 2015, make sure you have .NET Core installed.

1. Open Visual Studio UNITED STATES ▼

2. Click File -> New -> Project

3. In the New Project Dialog Window, select the “ASP.NET Core Web Application” project template

4. Specify the name and location for your project and click OK to save

Now, create the following POCO (plain old CLI object) class. This class contains just one property – it represents all of the topic areas covered by the authors of a particular publishing company.

```
public class TopicArea
{
    public string Name { get; set; }
}
```

Consider the following interface named `ITopicAreaService` that represents the contract for the `TopicAreaService`.

```
public interface ITopicAreaService
{
    IEnumerable<TopicArea> GetAllTopicAreas();
}
```


The `ITopicAreaService` interface contains the declaration of one method called `GetAllTopicAreas()`. The `TopicAreaService` class implements the `ITopicAreaService` as shown below.


```
public class TopicAreaService : ITopicAreaService
{
    public IEnumerable<TopicArea> GetAllTopicAreas()
    {
        return new List<TopicArea>
        {
            new TopicArea {Name = ".NET Core" },
            new TopicArea {Name = "Docker" },
            new TopicArea { Name = "C#" }
        };
    }
}
```


Registering services for dependency injection in ASP.NET

The next step is to register the `TopicAreaService` with the dependency injection container available as part of ASP.NET Code. To do this, write the following piece of code in the `ConfigureServices` method in the `Startup.cs` file. The `ConfigureServices` method adds services to the services container, which makes them available in your app via dependency injection. It is called by the runtime automatically.

RECOMMENDED WHITEPAPERS

 Advantages of using a Managed Service Provider

 How ComplyScore Checks Vendors for AWS Best Practices Compliance

 Top 50 Security Threats

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<ITopicAreaService, TopicAreaService>();
    // Add framework services.
    services.AddMvc();
}
```

If you have multiple services that need to be registered, you can use an extension method as shown below.

```
public static class ServiceExtensions
{
    public static IServiceCollection RegisterServices(
        this IServiceCollection services)
    {
        services.AddTransient<ITopicAreaService, TopicAreaService>();
        // Add all other services here.
        return services;
    }
}
```

Using the `RegisterServices` method allows you to keep your `ConfigureServices` method lean and maintainable. Instead of specifying each service in `ConfigureServices`, all you need to do is call the `RegisterServices` extension method once in your `ConfigureServices` method as shown in the code snippet below.

```
public void ConfigureServices(IServiceCollection services)
{
    services.RegisterServices();
    // Add framework services.
    services.AddMvc();
}
```

Dependency injection lifetimes in ASP.NET

The dependency injection lifetime is used to specify when the dependent objects are created and re-created. As far as lifetimes for dependency injection instances in ASP.NET Core applications is concerned, there are three possibilities:

1. Singleton: This implies only a single instance will be created and shared by all consumers.
2. Scoped: This implies that one instance per scope (i.e., one instance per request to the application) will be created.
3. Transient: This implies that the components will not be shared but will be created each time they are requested.

Note that in this example we have used the `Transient` type. The following code snippet illustrates how you can use the other types of lifetime when registering your service.

```
services.AddScoped<ISecurityService, SecurityService>();
services.AddSingleton<ICachingService, CachingService>();
```

Using a service via dependency injection in ASP.NET

Now that the service we implemented has been added to the pipeline, you can use it in any of the controllers in your ASP.NET Core project. The following code snippet illustrates how you can request an instance of the `TopicAreaService` in your controller.

```
private readonly ITopicAreaService _topicAreaService;
public DefaultController(ITopicAreaService topicAreaService)
{
    _topicAreaService = topicAreaService;
}
```

Here's how the GetAllTopicAreas method of the TopicAreaService is called from your controller's action method.

```
[HttpGet]
public IEnumerable<TopicArea> GetAllTopicAreas()
{
    return _topicAreaService.GetAllTopicAreas();
}
```

Given below is the complete code listing of the controller class for your reference.

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
namespace IDGASPNETCoreDI.Controllers
{
    [Produces("application/json")]
    [Route("api/Default")]
    public class DefaultController : Controller
    {
        private readonly ITopicAreaService _topicAreaService;
        public DefaultController(ITopicAreaService topicAreaService)
        {
            _topicAreaService = topicAreaService;
        }
        [HttpGet]
        public IEnumerable<TopicArea> GetAllTopicAreas()
        {
            return _topicAreaService.GetAllTopicAreas();
        }
    }
}
```

You can leverage the in-built support for dependency injection in ASP.NET Core to build applications that are modular, lean, and clean, easy to maintain and test. The built-in dependency injection provider in ASP.NET Core is not as feature-rich as containers like StructureMap and Ninject, but it is quite fast and, as we've seen, easy to configure and use.

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.

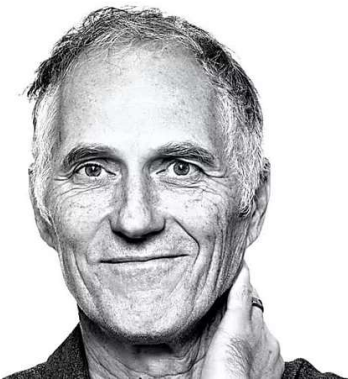
Follow     

Copyright © 2017 IDG Communications, Inc.

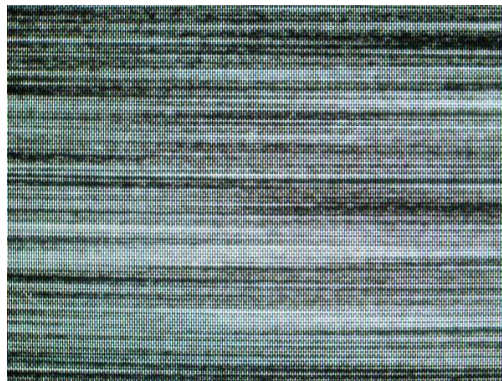
- Stay up to date with InfoWorld's newsletters for software developers, analysts, database programmers, and data scientists.
- Get expert insights from our member-only Insider articles.

YOU MAY ALSO LIKE

Recommended by



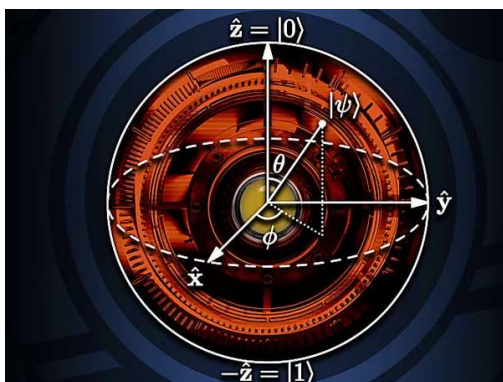
Tim O'Reilly: the golden age of the programmer is over



Python developers want static typing



Edge computing archetypes are emerging, and they are not pretty



Amazon Braket: Get started with quantum computing



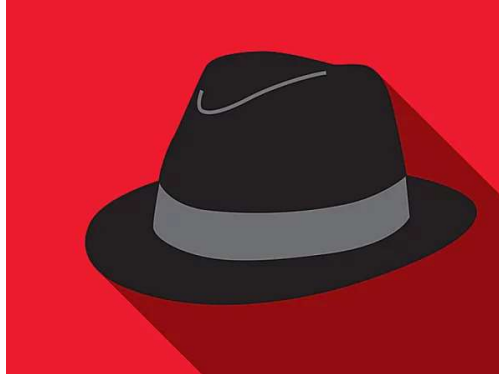
What's new in Kubernetes 1.20



2 common cloud computing predictions for 2021 are wrong



Using Pulumi 3.0 to manage Azure infrastructure



Red Hat OpenShift ramps up security and manageability



What is a computational storage drive? Much-needed

How IT priorities are shifting during the COVID-19 crisis

When Kubernetes is not the solution

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.