ASP.NET Core is designed from scratch to support Dependency Injection. ASP.NET Core injects objects of dependency classes through constructor or method by using built-in IoC container.

ASP.NET Core framework contains simple out-of-the-box IoC container which does not have as many features as other third party IoC containers. If you want more features such as auto-registration, scanning, interceptors, or decorators then you may replace built-in IoC container with a third party container.

The built-in container is represented by IServiceProvider implementation that supports constructor injection by default. **The types (classes) managed by built-in IoC container is called** **services**.

There are basically two types of services in ASP.NET Core:

1.Framework Services: Services which are a part of ASP.NET Core framework such as IApplicationBuilder, IHostingEnvironment, ILoggerFactory etc.

2.Application Services: The services (custom types or classes) which you as a programmer create for your application.

**In order to let the IoC container automatically inject our application services, we first need to register them with IoC container.**

Registering Application Service:

Consider the following example of simple ILog interface and its implementation class. We will see how to register it with built-in IoC container and use it in our application.

1. public interface ILog
2. {
3. void info(string str);
4. }
6. class MyConsoleLogger: ILog
7. {
8. public void info(string str)
9. {
10. Console.WriteLine(str);
11. }
12. }

**ASP.NET Core allows us to register our application services with IoC container, in the** **ConfigureServices method of the Startup class**. The ConfigureServices method includes a parameter of **IServiceCollection type which is used to register application services.**

Let's register above ILog with IoC container in ConfigureServices() method as shown below.

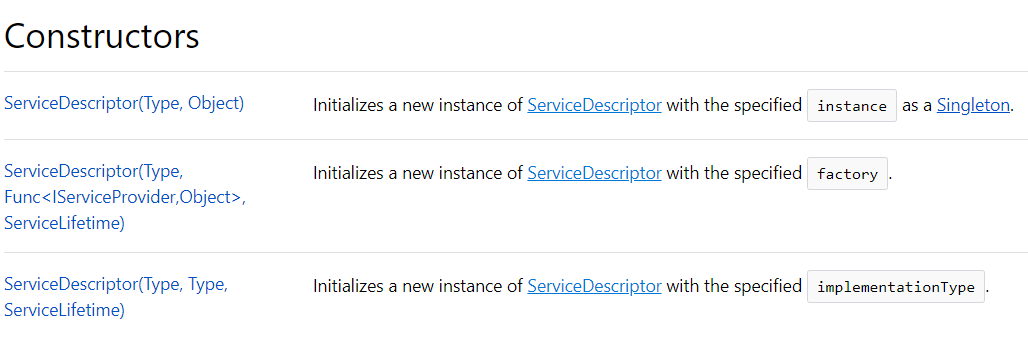
1. public class Startup
2. {
3. public void ConfigureServices(IServiceCollection services)
4. {
5. services.Add(new ServiceDescriptor(typeof(ILog), new MyConsoleLogger()));
6. }
7. // other code removed for clarity..
8. }

As you can see above, Add() method of IServiceCollection instance is used to register a service with an IoC container.

**The ServiceDescriptor is used to specify a service type and its instance. We have specified ILog as service type and MyConsoleLogger as its instance.**

**This will register ILog service as a singleton by default.** **Now, an IoC container will create a singleton object of MyConsoleLogger class and inject it in the constructor of classes wherever we include ILog as a constructor or method parameter throughout the application.**

Thus, we can register our custom application services with an IoC container in ASP.NET Core application. There are other extension methods available for quick and easy registration of services which we will see later in this chapter.



Understanding Service Lifetime:

**Built-in IoC container manages the lifetime of a registered service type**. It automatically disposes a service instance based on the specified lifetime.

**The built-in IoC container supports three kinds of lifetimes:**

**- Singleton:** IoC container will create and share a single instance of a service throughout the application's lifetime.

**- Transient:** The IoC container will create a new instance of the specified service type every time you ask for it.

**- Scoped:** IoC container will create an instance of the specified service type once per request and will be shared in a single request.

The following example shows how to register a **service with different lifetimes.**

1. public void ConfigureServices(IServiceCollection services)
2. {
3. services.Add(new ServiceDescriptor(typeof(ILog), new MyConsoleLogger())); // singleton
4. services.Add(new ServiceDescriptor(typeof(ILog), typeof(MyConsoleLogger), ServiceLifetime.Transient)); // Transient
6. services.Add(new ServiceDescriptor(typeof(ILog), typeof(MyConsoleLogger), ServiceLifetime.Scoped)); // Scoped
7. }

**Extension Methods for Registration:**

ASP.NET Core **framework includes extension methods for each types of lifetime;**

AddSingleton(), AddTransient()and AddScoped()

methods for singleton, transient and scoped lifetime respectively.

The following example shows the ways of registering types (service) using extension methods.

1. public void ConfigureServices(IServiceCollection services)
2. {
3. services.AddSingleton < ILog, MyConsoleLogger > ();
4. services.AddSingleton(typeof(ILog), typeof(MyConsoleLogger));
5. services.AddTransient < ILog, MyConsoleLogger > ();
6. services.AddTransient(typeof(ILog), typeof(MyConsoleLogger));
8. services.AddScoped < ILog, MyConsoleLogger > ();
9. services.AddScoped(typeof(ILog), typeof(MyConsoleLogger));
10. }

## Constructor Injection:

Once we register a service, the IoC container automatically performs constructor injection if a service type is included as a parameter in a constructor. For example, we can use ILog service type in any MVC controller. Consider the following example.

1. public class HomeController: Controller
2. {
3. ILog \_log;
4. public HomeController(ILog log)
5. {
6. \_log = log;
7. }
8. public IActionResult Index()
9. {
10. \_log.info("Executing /home/index");
11. return View();
12. }
13. }

**In the above example, an IoC container will automatically pass an instance of MyConsoleLogger** to the constructor of HomeController. **We don't need to do anything else.** **An IoC container will create and dispose an instance of ILog** based on the registered lifetime.

Action Method Injection:

Sometimes we may only need dependency service **type in a single action method.** For this, use **[FromServices] attribute** with the service type parameter in the method.

1. Example: Action Method Injection using Microsoft.AspNetCore.Mvc;
2. public class HomeController: Controller
3. {
4. public HomeController() {}
5. public IActionResult Index([FromServices] ILog log)
6. {
7. log.info("Index method executing");
8. return View();
9. }
10. }

Property Injection:

Built-in IoC container does not support property injection. You will have to use third party IoC container.

Get Services Manually:

It is not required to include dependency services in the constructor. We can access dependent services configured with built-in IoC container manually using RequestServices property of HttpContext as shown below.

1. public class HomeController: Controller
2. {
3. public HomeController() {}
5. public IActionResult Index()
6. {
7. var services = this.HttpContext.RequestServices;
8. var log = (ILog) services.GetService(typeof(ILog));
9. log.info("Index method executing");
10. return View();
11. }
12. }

It is recommended to use constructor injection instead of getting it using RequestServices.

Sources: 12.05.2018, 13:02 UTC+1

tutorialsteacher.com/core/dependency-injection-in-aspnet-core

docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.servicedescriptor?view=aspnetcore-2.1