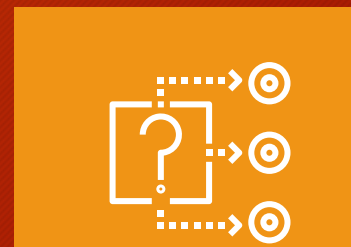
Microsoft DI vs Autofac

Dependency injection

Required background



What is the "dependency injection"?

Comparison of key features

- Add to project
- Registration
- Service lifetimes

Add to project

Microsoft DI

```
No need. Dotnet built-in.
```

Autofac

```
PM> Install-Package Autofac

PM> Install-Package Autofac.Extensions.DependencyInjection
```

Registration

Microsoft DI

```
builder.Services.AddSingleton<IWeatherForecaster, WeatherForecaster>();
```

Autofac

```
builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory())
    .ConfigureContainer<ContainerBuilder>(builder => builder.RegisterModule(new AutofacBusinessModule()));
```

```
public class AutofacBusinessModule:Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterType<WeatherForecaster>().As<IWeatherForecaster>().SingleInstance();
        }
    }
```

Service lifetimes

Microsoft DI

```
builder.Services.AddSingleton<IWeatherForecaster, WeatherForecaster>();
builder.Services.AddScoped<IWeatherForecaster, WeatherForecaster>();
builder.Services.AddTransient<IWeatherForecaster, WeatherForecaster>();
```

Autofac

```
builder.RegisterType<WeatherForecaster>().As<IWeatherForecaster>().SingleInstance();
builder.RegisterType<WeatherForecaster>().As<IWeatherForecaster>().InstancePerLifetimeScope();
builder.RegisterType<WeatherForecaster>().As<IWeatherForecaster>().InstancePerDependency();
```

Autofac awesome features

- Instance per matching lifetime scope
- Thread Scope
- Named and keyed services
- Dynamic Instantiation(Func<B>)

Instance per matching lifetime scope

A component with per-matching-lifetime scope will have at most a single instance per nested lifetime scope that matches a given name.

Thread Scope

Autofac can enforce that objects bound to one thread will not satisfy the dependencies of a component bound to another thread.

Named and keyed services

```
builder.RegisterType<OnlineState>().Named<IDeviceState>("online");
var r = container.ResolveNamed<IDeviceState>("online");
```

```
public enum DeviceState { Online, Offline }

public class OnlineState : IDeviceState { }
var builder = new ContainerBuilder();
builder.RegisterType<OnlineState>().Keyed<IDeviceState>(DeviceState.Online);
builder.RegisterType<OfflineState>().Keyed<IDeviceState>(DeviceState.Offline);
var r = container.ResolveKeyed<IDeviceState>(DeviceState.Online);
```

Dynamic Instantiation (Func<B>)

Create instance when:

- You need to create more than one instance of a given service.
- You want to specifically control when the setup of the service occurs.
- You're not sure if you're going to need a service and want to make the decision at runtime.

Some Other features

- Adding Metadata to a Component Registration
- Delayed Instantiation (Lazy<B>)
- …

Conclusion

Fundamentally the difference is in the additional features and choosing the best framework depends on your project needs but Autofac can offer you some additional features if you need them.

For simple applications, the Microsoft DI can offer adequate functionality.

Read more:

Welcome to Autofac’s documentation! — Autofac 7.0.0 documentation

Dependency injection - .NET | Microsoft Learn

Thank you for your attention

Mohammad Shoorabi

https://www.linkedin.com/in/mohammad-shoorabi