.NET Core Dependency Injection - One Interface, Multiple Implementations

 Akshay Patel Updated date Feb 05, 2019 120.8k 8 18

Consider a scenario where you want to get shopping cart object and you have implemented IShoppingCart Interface. Now, you have multiple options to get the shopping cart like from Database, API, or Cache. Here, we need to implement all these three as concrete implementations for an interface, IShoppingCart. Now, the question is how we can decide which instance is to be instantiated as generally we see one implementation for one interface and add it into service collection for dependency injection. So, let's see the implementation step by step.

Open Visual Studio and create a new project.

Select the API as the template and click OK.

Create an IShoppingcart Interface having GetCart method.

```
01. namespace MultipleImplementation
02. {
03.     public interface IShoppingCart
04.     {
05.         object GetCart();
06.     }
07. }
```

Implement the GetCart method of IShoppingCart interface in ShoppingCartCache.

```
01. namespace MultipleImplementation
02. {
03.     public class ShoppingCartCache : IShoppingCart
04.     {
05.         public object GetCart()
06.         {
07.             return "Cart loaded from cache.";
08.         }
09.     }
10. }
```

Implement the same interface in ShoppingCartDB.

```
01. namespace MultipleImplementation
02. {
03.     public class ShoppingCartDB : IShoppingCart
04.     {
05.         public object GetCart()
06.         {
07.             return "Cart loaded from DB";
```

```
}
```

At last, implement the same interface in ShoppingCartAPI.

```
namespace MultipleImplementation
{
    public class ShoppingCartAPI : IShoppingCart
    {
        public object GetCart()
        {
            return "Cart loaded through API.";
        }
    }
}
```

Now, we need a repository which will internally decide which implementation should be instantiated and called to get the GetCart method from it. So, create IShoppingCartRepository having the GetCart method.

```
namespace MultipleImplementation
{
    public interface IShoppingCartRepository
    {
        object GetCart();
    }
}
```

Create an enum to select the concrete implementation type, i.e, Cache, DB, or API.

```
namespace MultipleImplementation
{
    public class Constants
    {
    }

    public enum CartSource
    {
        Cache=1,
        DB=2,
        API=3
    }
}
```

In the implementation of IShoppingCartRepository, we use constructor injection and we take Func delegate as a parameter. Func delegate expects a string as parameter and IShoppingCart as a return value. So in the GetCart method, you can see that we are using an enum to pass the parameter value which indicates the type of implementation we want to instantiate.

```
using System;

namespace MultipleImplementation
{
    public class ShoppingCartRepository : IShoppingCartRepository
    {
        private readonly Func<string, IShoppingCart> shoppingCart;
        public ShoppingCartRepository(Func<string, IShoppingCart> shoppingCart)
        {
            this.shoppingCart = shoppingCart;
        }

        public object GetCart()
        {
            return shoppingCart(CartSource.DB.ToString()).GetCart();
        }
```

In the same way, we need to add Func in startup class. First, we add the concrete implementation in service collection and on each request based on the parameter value, we used to get that concrete class.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System;

namespace MultipleImplementation
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {

            services.AddScoped<IShoppingCartRepository, ShoppingCartRepository>();

            services.AddSingleton<ShoppingCartCache>();
            services.AddSingleton<ShoppingCartDB>();
            services.AddSingleton<ShoppingCartAPI>();

            services.AddTransient<Func<string, IShoppingCart>>(serviceProvider => key =>
            {
                switch (key)
                {
                    case "API":
                        return serviceProvider.GetService<ShoppingCartAPI>();
                    case "DB":
                        return serviceProvider.GetService<ShoppingCartDB>();
                    default:
                        return serviceProvider.GetService<ShoppingCartCache>();
                }
            });

            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseMvc();
        }
    }
}
```

You can download the sample from here.

.Net Core | .Net Core Dependency Injection | Dependency Injection | One Interface Multiple Implementation