آموزش LightInject IoC Container - قسمت 1

نویسنده: میثم خوشبخت

عنوان:

تاریخ: ۱۶:۳۰ ۱۳۹۳/۰۲/۰۸ تاریخ: ۱۶:۳۰ ۱۳۹۳/۰۲/۰۸

گروهها: Design patterns, Dependency Injection, Ioc, Dependency Inversion, LightInject

نحوه نصب و راه اندازی LightInject

در پنجره Package Manager Console میتوانید با نوشتن دستور ذیل، نسخه باینری آن را نصب کنید که به فایل .dll آن Reference

PM> Install-Package LightInject

همچنین می توانید توسط دستور ذیل فایل .cs آن را به پروژه اضافه نمایید.

PM> Install-Package LightInject.Source

آماده سازی پروژه نمونه

قبل از شروع کار با LightInject، یک پروژه Windows Forms Application را با ساختار کلاسهای ذیل ایجاد نمایید. (در مقالات بعدی و پس از آموزش کامل LightInject نحوه استفاده از آن را در ASP.NET MVC نیز آموزش میدهیم)

```
public class PersonModel
        public int Id { get; set; }
public string Name { get; set; }
public string Family { get; set; }
         public DateTime Birth { get; set; }
    public interface IRepository<T> where T:class
         void Insert(T entity)
         IEnumerable<T> FindAll();
    public interface IPersonRepository:IRepository<PersonModel>
    public class PersonRepository: IPersonRepository
         public void Insert(PersonModel entity)
             throw new NotImplementedException();
         public IEnumerable<PersonModel> FindAll()
             throw new NotImplementedException();
    }
    public interface IPersonService
         void Insert(PersonModel entity);
         IEnumerable \( PersonModel > FindAll();
```

```
public class PersonService:IPersonService
{
    private readonly IPersonRepository _personRepository;
    public PersonService(IPersonRepository personRepository)
    {
        _personRepository = personRepository;
    }
    public void Insert(PersonModel entity)
    {
        _personRepository.Insert(entity);
    }
    public IEnumerable<PersonModel> FindAll()
    {
        return _personRepository.FindAll();
    }
}
```

توضیحات PersonModel: ساختار داده ای جدول Person در سمت Application، که در لایه PersonModel: یک Interface جهت سهولت تست و تسریع کدنویسی از لایه بندی و از کلاسهای ViewModel استفاده نکردیم. (Interface یک Sepository: یک Repository عمومی برای تمامی Interface هربوط به پایگاه داده مثل بروزرسانی و واکشی اطلاعات را انجام میدهند. (PersonRepository: پیاده سازی واقعی Service بیاده سازی واقعی ایده سازی واقعی که حاوی پیاده سازی واقعی که میباشد و PersonModel: واسط بین لایه PersonModel میباشد. به کلاسهایی که حاوی پیاده سازی واقعی کد میباشند Concrete Class میگویند. PersonService: واسط بین رابط کاربری و لایه سرویس میباشد. رابط کاربری به جای دسترسی مستقیم به میگویند. PersonService استفاده میکند. PersonService: دریافت درخواستهای رابط کاربری و بررسی قوانین تجاری، سپس ارسال درخواست به لایه Repository در صورت صحت درخواست، و در نهایت ارسال پاسخ دریافتی به رابط کاربری. در واقع واسطی بین Repository و ID میباشد.

یس از ایجاد ساختار فوق کد مربوط به Form1 را بصورت زیر تغییر دهید.

```
public partial class Form1 : Form
{
    private readonly IPersonService _personService;
    public Form1(IPersonService personService)
    {
        _personService = personService;
        InitializeComponent();
    }
}
```

توضيحات

در کد فوق به منظور ارتباط با سرویس از IPersonService استفاده نمودیم که به عنوان پارامتر ورودی برای سازنده Form1 تعریف شده است. حتما با Dependency Inversion و انواع Dependency Injection آشنا هستید که به سراغ مطالعه این مقاله آمدید و علت این نوع کدنویسی را هم میدانید. بنابراین توضیح بیشتری در این مورد نمیدهم.

حال اگر برنامه را اجرا کنید در Program.cs با خطای عدم وجود سازنده بدون پارامتر برای Form1 مواجه میشوید که کد آن را باید به صورت زیر تغییر میدهیم.

```
static void Main()
{
          Application.EnableVisualStyles();
          Application.SetCompatibleTextRenderingDefault(false);
          var container = new ServiceContainer();
          container.Register<IPersonService, PersonService>();
          container.Register<IPersonRepository, PersonRepository>();
          Application.Run(new Form1(container.GetInstance<IPersonService>()));
}
```

توضيحات

کلاس ServiceContainer وظیفهی Register کردن یک کلاس را برای یک Interface دارد. زمانی که میخواهیم Forml را نمونه سازی نماییم و Application را راه اندازی کنیم، باید نمونه ای را از جنس IPersonService ایجاد نموده و به سازندهی Form1 ارسال نماییم. با رعایت اصل DIP، نمونه سازی واقعی یک کلاس لایه دیگر، نباید در داخل کلاسهای لایه جاری انجام شود. برای این منظور از شیء container استفاده نمودیم و توسط متد GetInstance، نمونهای از جنس IPersonService را ایجاد نموده و به Form1 پاس دادیم. حال container از کجا متوجه می شود که چه کلاسی را برای IPersonService نمونه سازی نماید؟ در خطوط قبلی توسط متد Register، کلاس PersonService را برای IPersonService ثبت نمودیم. Register نیز برای نمونه سازی به کلاس هایی که برایش Register نمودیم مراجعه می نماید و نمونه سازی را انجام می دهد. جهت استفاده از PersonRepository به پارامتر ورودی IPersonRepository برای سازندهی آن نیاز داریم که کلاس PersonRepository را برای IPersonRepository را برای IPersonRepository ثبت کردیم.

حال اگر برنامه را اجرا کنید، به درستی اجرا خواهد شد. برنامه را متوقف کنید و به کد موجود در Program.cs مراجعه نموده و دو خط مربوط به Register را Comment نمایید. سپس برنامه را اجرا کنید و خطای تولید شده را ببینید. این خطا بیان می کند که امکان نمونه سازی برای IPersonService را ندارد. چون قبلا هیچ کلاسی را برای آن Register نکرده ایم. Named Services در زمان در برخی مواقع، بیش از یک کلاس وجود دارند که ممکن است از یک Interface ارث بری نمایند. در این حالت و در زمان Register، باید به کلاسهای زیر را به با باید نمونه سازی نماید. برای بررسی این موضوع، کلاسهای زیر را به ساختار پروژه اضافه نمایید.

```
public class WorkerModel:PersonModel
        public ManagerModel Manager { get; set; }
    public class ManagerModel:PersonModel
        public IEnumerable<WorkerModel> Workers { get; set; }
    public class WorkerRepository: IPersonRepository
        public void Insert(PersonModel entity)
            throw new NotImplementedException();
        public IEnumerable<PersonModel> FindAll()
            throw new NotImplementedException();
    }
    public class ManagerRepository:IPersonRepository
        public void Insert(PersonModel entity)
            throw new NotImplementedException();
        public IEnumerable<PersonModel> FindAll()
            throw new NotImplementedException();
    }
    public class WorkerService:IPersonService
        private readonly IPersonRepository personRepository;
        public WorkerService(IPersonRepository personRepository)
            _personRepository = personRepository;
        public void Insert(PersonModel entity)
            var worker = entity as WorkerModel;
            _personRepository.Insert(worker);
        public IEnumerable<PersonModel> FindAll()
            return personRepository.FindAll();
        }
    }
    public class ManagerService: IPersonService
        private readonly IPersonRepository _personRepository;
```

```
public ManagerService(IPersonRepository personRepository)
{
    _personRepository = personRepository;
}

public void Insert(PersonModel entity)
{
    var manager = entity as ManagerModel;
    _personRepository.Insert(manager);
}

public IEnumerable<PersonModel> FindAll()
{
    return _personRepository.FindAll();
}
```

توضيحات

دو کلاس Manager و Worker به همراه سرویسها و Repository هایشان اضافه شده اند که از IPersonService و IPersonRepository

حال کد کلاس Program را به صورت زیر تغییر میدهیم

توضيحات

در کد فوق، چون WorkerService بعد از PersonService ثبت یا Register شده است، LightInject در زمان ارسال پارامتر به Form1، نمونه ای از کلاس WorkerService را ایجاد میکند. اما اگر بخواهیم از کلاس PersonService نمونه سازی نماید باید کد را به صورت زیر تغییر دهیم.

```
container.Register<IPersonService, PersonService>("PersonService");
container.Register<IPersonService, WorkerService>();
container.Register<IPersonRepository, PersonRepository>();
container.Register<IPersonRepository, WorkerRepository>();
Application.Run(new Form1(container.GetInstance<IPersonService>("PersonService")));
```

همانطور که مشاهده مینمایید، در زمان Register نامی را به آن اختصاص دادیم که در زمان نمونه سازی از این نام استفاده شده است:

اگر در زمان ثبت، نامی را به نمونهی مورد نظر اختصاص داده باشیم، و فقط یک Register برای آن Interface معرفی نموده باشیم، در زمان نمونه سازی، LightInject آن نمونه را به عنوان سرویس پیش فرض در نظر میگیرد.

```
container.Register<IPersonService, PersonService>("PersonService");
  Application.Run(new Form1(container.GetInstance<IPersonService>()));
```

در کد فوق، چون برای IPersonService فقط یک کلاس برای نمونه سازی معرفی شده است، با فراخوانی متد GetInstance، حتی بدون ذکر نام، نمونه ای را از کلاس PersonService ایجاد میکند. Postistance ایجاد میکند. The Desistance این قابلت ایدا دیکه این

زمانی که چند کلاس را که از یک Interface مشتق شده اند، با هم Register مینمایید، LightInject این قابلیت را دارد که این کلاسهای Register شده را در قالب یک لیست شمارشی برگردانید.

در کد فوق لیستی با دو آیتم ایجاد میشود که یک آیتم از نوع PersonService و دیگری از نوع WorkerService میباشد. همچنین از کد زیر نیز میتوانید استفاده کنید:

به جای متد GetInstance از متد GetAllInstances استفاده شده است.

LightInject از Collectionهای زیر نیز یشتیبانی مینماید:

Array

<ICollection<T

<IList<T

<IReadOnlyCollection<T</pre>

<IReadOnlyList<T</pre>

Values توسط LightInject مىتوانيد مقادير ثابت را نيز تعريف كنيد

```
container.RegisterInstance<string>("SomeValue");
    var value = container.GetInstance<string>();
```

متغیر value با رشته "SomeValue" مقداردهی می گردد. اگر چندین ثابت رشته ای داشته باشید می توانید نام جداگانه ای را به هر کدام اختصاص دهید و در زمان فراخوانی مقدار به آن نام اشاره کنید.

متغیر value با رشته "OtherValue" مقداردهی می گردد.

نظرات خوانندگان

نویسنده: احمد زاده تاریخ: ۲/۲۱ ۱۳۹۳/۱۲۲۷

ممنون از مطلب خوبتون

من یه مقایسه دیگه دیدم که اونجا گفته بود Ligth Inject از Instance Per Request پشتیبانی نمیکنه میخواستم جایگزین Unity کنم برای حالتی که unit of work داریم و DBContext for per request اگر راهنمایی کنید، ممنون میشم

> نویسنده: وحید نصی*ری* تاریخ: ۲/۲۱ ۱۳۹۳/۳۲:۳۲

از حالت طول عمر PerRequestLifetime پشتیبانی میکند.

نویسنده: میثم خوشبخت تاریخ: ۱۱:۱۴ ۱۳۹۳/۰۲/۲۱

خواهش مىكنم

همانطور که آقای نصیری نیز عنوان کردند، از PerRequestLifeTime استفاده میشود که در مقاله بعدی در مورد آن صحبت خواهم کرد.