عنوان: **آموزش LightInject IoC Container - قسمت 1**
نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۳/۰۲/۰۸ ۱۶:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, Ioc, Dependency Inversion, LightInject

LightInject در حال حاضر یکی از قدرتمندترین IoC Containerها است که از لحاظ سرعت و کارآیی در بالاترین جایگاه در میان IoC Containerهای موجود قرار دارد. جهت بررسی کارایی IoC Containerها می‌توانید به این لینک مراجعه کنید . LightInject یک IoC Container فوق العاده سبک وزن می‌باشد که تمامی قابلیت‌های متداولی که از یک Service Container انتظار می‌رود را شامل می‌شود. تنها شامل یک فایل .cs می‌باشد که تمامی کدهای آن در همین یک فایل نوشته شده‌اند. در پروژه‌های کوچک تا بزرگ بدون از دست دادن کارآیی، با بالاترین سرعت ممکن عمل تزریق وابستگی را انجام می‌دهد. در این مجموعه مقالات به بررسی کامل این IoC Container می‌پردازیم و تمامی قابلیت‌های آن را آموزش می‌دهیم.

**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 میدهد.

```
PM> Install-Package LightInject
```

همچنین می‌توانید 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 Domain Model ایجاد می‌گردد. **توجه:** جهت سهولت تست و تسریع کدنویسی از لایه بندی و از کلاس‌های ViewModel استفاده نکردیم. **IRepository:** یک Interface عمومی برای تمامی Interfaceهای مربوط به Repository که عملیات مربوط به پایگاه داده مثل بروزرسانی و واکشی اطلاعات را انجام می‌دهند. **IPersonRepository:** واسط بین لایه Service و لایه Repository می‌باشد. **PersonRepository:** پیاده سازی واقعی عملیات مربوط به پایگاه داده برای PersonModel می‌باشد. به کلاسهایی که حاوی پیاده سازی واقعی کد می‌باشند Concrete Class می‌گویند. **IPersonService:** واسط بین رابط کاربری و لایه سرویس می‌باشد. رابط کاربری به جای دسترسی مستقیم به PersonService از IPersonService 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 UI می‌باشد.

پس 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();
        }
    }
```

**توضیحات**

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>()));
        }
```

**توضیحات**

کلاس ServiceContainer وظیفه‌ی Register کردن یک کلاس را برای یک Interface دارد. زمانی که می‌خواهیم Form1 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 می‌شود که چه کلاسی را برای IPersonService نمونه سازی نماید؟
در خطوط قبلی توسط متد Register، کلاس PersonService را برای IPersonService ثبت نمودیم. container نیز برای نمونه سازی به کلاس هایی که برایش Register نمودیم مراجعه می‌نماید و نمونه سازی را انجام می‌دهد. جهت استفاده از PersonService به پارامتر ورودی IPersonRepository برای سازنده‌ی آن نیاز داریم که کلاس PersonRepository 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 می‌کند که امکان نمونه سازی برای IPersonService را ندارد. چون قبلا هیچ کلاسی را برای آن Register نکرده ایم. **Named Services**
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 ServiceContainer بگوییم که کدام کلاس را 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();
        }
    }
```

**توضیحات**

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

حال کد کلاس Program را به صورت زیر تغییر می‌دهیم

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

**توضیحات**

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. **<IEnumerable<T>**

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.

```
container.Register<IPersonService, PersonService>();
            container.Register<IPersonService, WorkerService>("WorkerService");
            var personList = container.GetInstance<IEnumerable<IPersonService>>();
```

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:

```
container.Register<IPersonService, PersonService>();
            container.Register<IPersonService, WorkerService>("WorkerService");
            var personList = container.GetAllInstances<IPersonService>();
```

به جای متد GetInstance از متد GetAllInstances استفاده شده است.

LightInject از Collectionهای زیر نیز پشتیبانی می‌نماید:

Array

<ICollection<T

<IList<T

<IReadOnlyCollection<T

<IReadOnlyList<T

**Values** توسط LightInject می‌توانید مقادیر ثابت را نیز تعریف کنید

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

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

```
container.RegisterInstance<string>("SomeValue","String1");
            container.RegisterInstance<string>("OtherValue","String2");
            var value = container.GetInstance<string>("String2");
```

متغیر value با رشته "OtherValue" مقداردهی می‌گردد.

## نظرات خوانندگان

نویسنده: احمد زاده
تاریخ: ۱۳۹۳/۰۲/۲۱ ۱:۲۷

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 نصیری
تاریخ: ۱۳۹۳/۰۲/۲۱ ۱۰:۳۲

از حالت طول عمر PerRequestLifetime پشتیبانی می‌کند.

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

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