

در این پست قصد دارم روش استفاده از ServiceLocator رو به وسیله یک مثال ساده بهتون نمایش بدم. Microsoft Unity روش توصیه شده Microsoft برای پیاده سازی Dependency Injection و ServiceLocator Pattern است. یک ServiceLocator در واقع وظیفه تهیه Instance‌های مختلف از کلاس‌ها رو برای پیاده سازی Dependency Injection بر عهده داره. برای شروع یک پروژه از نوع Console Application ایجاد کنید و یک ارجاع به Assembly‌های زیر رو در برنامه قرار بدید.

Microsoft.Practices.ServiceLocation

Microsoft.Practices.Unity

Microsoft.Practices.EnterpriseLibrary.Common

اگر Assembly‌های بالا رو در اختیار ندارید می‌تونید اون‌ها رو از [اینجا](#) دانلود کنید. Microsoft Enterprise Library یک کتابخانه تهیه شده توسط شرکت Microsoft است که شامل موارد زیر است و بعد از نصب می‌تونید در قسمت‌های مختلف برنامه از اون‌ها استفاده کنید.

Enterprise Library Caching Application Block : یک CacheManager قدرتمند در اختیار ما قرار می‌ده که می‌تونید از اون برای کش کردن داده‌ها استفاده کنید.

Enterprise Library Exception Handling Application Block : یک کتابخانه مناسب و راحت برای پیاده سازی یک Exception Handler در برنامه‌ها است.

Enterprise Library Loggin Application Block : برای تهیه یک Log Manager در برنامه استفاده می‌شود.

Enterprise Library Validation Application Block : برای اجرای Validation برای Entity‌ها با استفاده از Attribute می‌تونید از این قسمت استفاده کنید.

Enterprise Library DataAccess Application Block : یک کتابخانه قدرتمند برای ایجاد یک DataAccess Layer است با Performance بسیار بالا. Enterprise Library Shared Library: برای استفاده از تمام موارد بالا در پروژه باید این Dll رو هم به پروژه Reference بدید. چون برای همشون مشترک است.

برای اجرای مثال ابتدا کلاس زیر رو به عنوان مدل وارد کنید.

```
public class Book
{
    public string Title { get; set; }
    public string ISBN { get; set; }
}
```

حالا باید Repository مربوطه رو تهیه کنید. ابتدا یک Interface به صورت زیر ایجاد کنید.

```
public interface IBookRepository
{
    List<Book> GetBooks();
}
```

سپس کلاسی ایجاد کنید که این Interface رو پیاده سازی کنه.

```
public class BookRepository : IBookRepository
{
    public List<Book> GetBooks()
    {
        List<Book> listOfBooks = new List<Book>();

        listOfBooks.AddRange( new Book[]
        {
            new Book(){Title="Book1" , ISBN="123"},
            new Book(){Title="Book2" , ISBN="456"},
            new Book(){Title="Book3" , ISBN="789"},
            new Book(){Title="Book4" , ISBN="321"},
            new Book(){Title="Book5" , ISBN="654"},
        } );

        return listOfBooks;
    }
}
```

کلاس BookRepository یک لیست از Book رو ایجاد میکنه و اونو برگشت میده. در مرحله بعد باید Service مربوطه برای استفاده از این Repository ایجاد کنید. ولی باید Repository رو به Constructor این کلاس Service پاس بدید. اما برای انجام این کار باید از ServiceLocator استفاده کنیم.

```
public class BookService
{
    public BookService()
        : this( ServiceLocator.Current.GetInstance<IBookRepository>() )
    {
    }

    public BookService( IBookRepository bookRepository )
    {
        this.BookRepository = bookRepository;
    }

    public IBookRepository BookRepository
    {
        get;
        private set;
    }

    public void PrintAllBooks()
    {
        Console.WriteLine( "List Of All Books" );

        BookRepository.GetBooks().ForEach( ( Book item ) =>
        {
            Console.WriteLine( item.Title );
        } );
    }
}
```

همان طور که می بینید این کلاس دو تا Constructor داره که در حالت اول باید یک IBookRepository رو به کلاس پاس داد و در حالت دوم ServiceLocator این کلاس رو برای استفاده دز اختیار سرویس قرار میده. متد Print هم تمام کتابهای مربوطه رو برامون چاپ می کنه. در مرحله آخر باید ServiceLocator رو تنظیم کنید. برای این کار کدهای زیر رو در کلاس Program قرار بدید.

```
class Program
{
    static void Main( string[] args )
    {
        IUnityContainer unityContainer = new UnityContainer();

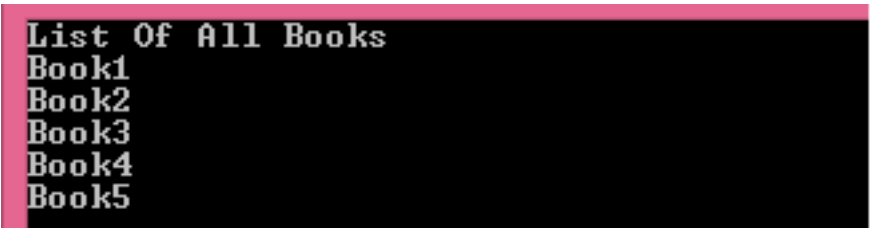
        unityContainer.RegisterType<IBookRepository, BookRepository>();

        ServiceLocator.SetLocatorProvider( () => new UnityServiceLocator( unityContainer ) );

        BookService service = new BookService();
    }
}
```

```
        service.PrintAllBooks();  
        Console.ReadLine();  
    }  
}
```

در این کلاس ابتدا یک `UnityContainer` ایجاد کردم و اینترفیس `IBookRepository` رو به کلاس `BookRepository` Register کردم تا هر جا که به `IBookRepository` نیاز داشتم یک Instance از کلاس `BookRepository` ایجاد بشه. در خط بعدی `ServiceLocator` برنامه رو ست کردم و برای این کار از کلاس `UnityServiceLocator` استفاده کردم. بعد از اجرای برنامه خروجی زیر قابل مشاهده است.



```
List Of All Books  
Book1  
Book2  
Book3  
Book4  
Book5
```

نظرات خوانندگان

نویسنده: sunn

تاریخ: ۱۱:۱۶ ۱۳۹۳/۰۳/۲۰

سلام اول از همه ممنون بابت این همه تلاش، دوم چرا بدون این همه کد نویسی نمیاییم از یه دستور linq ساده استفاده کنیم، نامها رو بگیریم و با یک Foreach ساده پاس بدیم و این همه راه رفتیم و الان MVC از این روشها و استفاده از اینترفیسها و تزریقات وابستگی و ... که من نمیدونم این تزریقات وابستگی چیه استفاده میکنیم ، ممنون میشم توضیح بدین یا به جایی ارجاء بدین منو

نویسنده: وحید نصیری

تاریخ: ۱۱:۴۲ ۱۳۹۳/۰۳/۲۰

جهت مطالعه مباحث مقدماتی تزریق وابستگی‌ها، مراجعه کنید به دوره « [بررسی مفاهیم معکوس سازی وابستگی‌ها و ابزارهای مرتبط با آن](#) ».

عنوان: ایجاد ServiceLocator با استفاده از Ninject

نویسنده: مسعود پاکدل

تاریخ: ۲۰:۵ ۱۳۹۱/۱۲/۰۴

آدرس: www.dotnettips.info

برچسب‌ها: Dependency Injection, ServiceLocator, Ninject

در [پست قبلی](#) روش استفاده از ServiceLocator رو با استفاده از Microsoft Unity بررسی کردیم. در این پست قصد داریم همون مثال رو با استفاده از Ninject پیاده سازی کنیم. Ninject ابزاری برای پیاده سازی Dependency Injection در پروژه‌های دات نت است که کار کردن با اون واقعا راحت. برای شروع کلاس‌های Book و BookRepository و BookService و اینترفیس IBookRepository از این [پست](#) دریافت کنید.

حالا با استفاده از NuGet باید ServiceLocator رو برای Ninject دریافت کنید. برای این کار در Package Manager Console دستور زیر رو وارد کنید.

```
PM> Install-Package CommonServiceLocator.NinjectAdapter
```

بعد از دانلود و نصب Reference‌های زیر به پروژه اضافه می‌شوند.

Ninject

NinjectAdapter

Microsoft.Practices.ServiceLocation

اگر دقت کنید برای ایجاد ServiceLocator داریم از Enterprise Library:ServiceLocator استفاده می‌کنیم. ولی برای این کار به جای استفاده از UnityServiceLocator باید از NinjectServiceLocator استفاده کنیم.

ابتدا

برای پیاده سازی مثال قبل در کلاس Program کدهای زیر رو وارد کنید.

```
using System;
using Ninject;
using NinjectAdapter;
using Microsoft.Practices.ServiceLocation;

namespace ServiceLocatorPattern
{
    class Program
    {
        static void Main( string[] args )
        {
            IKernel kernel = new StandardKernel();

            kernel.Bind<IBookRepository>().To<BookRepository>();

            ServiceLocator.SetLocatorProvider( () => new NinjectServiceLocator( kernel ) );

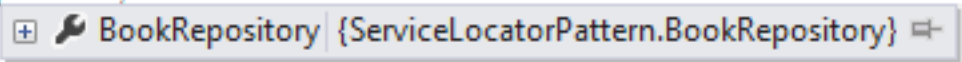
            BookService service = new BookService();

            service.PrintAllBooks();

            Console.ReadLine();
        }
    }
}
```

همان طور که می‌بینید روال انجام کار دقیقا مثل قبل هست فقط Syntax کمی متفاوت شده. برای مثال به جای استفاده از IUnityContainer از IKernel استفاده کردم و به جای دستور RegisterType از دستور Bind استفاده شده. در نهایت هم ServiceLocator به یک NinjectServiceLocator ای که IKernel رو دریافت کرده ست شد. به تصویر زیر دقت کنید.

```
public BookService()
{
    BookRepository = ServiceLocator.Current.GetInstance<IBookRepository>();
}
```

A tooltip is shown for the `BookRepository` property access. It contains a plus icon, a wrench icon, the text `BookRepository`, and a list of available types: `{ServiceLocatorPattern.BookRepository}`.

همان طور که می‌بینید در هر جای پروژه که نیاز به یک Instance از یک کلاس داشته باشید می‌تونید با استفاده از ServiceLocator این کار خیلی راحت انجام بدید.

بعد از اجرای پروژه خروجی دقیقا مانند مثال قبل خواهد بود.