Dependency Injection in Service Stack Application

Author: David Huang

Date: 16, Feb., 2015

1, Introduction

Visual Studio 2015 uses DI everywhere. I want to see how DI works in VS2015 Service Stack application development.

1, Get Started

Create an empty asp.net MVC web application in this project. Service Stack web service is Http based, so we need to build it in web application. Via configuration in Web.Config, we can convert IIS web site as a Service Stack web service. Now what I need to find out is how the DI will be used in here.

2, The Request class

```
[Route ("/contacts", "GET")]
public class Getallcontacts : IReturn<List<Contact>>
        {
            public int? Id { get; set; }

        }
```

This class is similar with the service contract class in WCF web service. Contact is simply an entity class as below

```
public class Contact
       {

        [AutoIncrement]
        public int Id { get; set; }
        public string Name { get; set; }
        public string email { get; set; }
        public int Age { get; set; }

       }
```

3, The Response class

```
public class ContactSTKInterface : Service
    {
        IRepos db { get; set; }//autowred =new repository();

        public ContactSTKInterface(IRepos _db)
        {
            db = _db;
        }

       public List<Contact> Any(Getallcontacts request){
            return request.Id != null?null : db.Getall();
       }
    }
```

Here we can find out that IRepos interface is Ioced here. Somehow we need to enable this in AppHost.cs file that is the rootstrap class file used to boot the application.

Interface:

```
public interface IRepos
    {
        void Dispose();
        List<Contact> Getall();
    }
```

Interface Implementation:

```
public class Repository : IDisposable, IRepos
    {
        protected OrmLiteConnectionFactory dbFactory = new
           OrmLiteConnectionFactory(ConfigurationManager.ConnectionStrings["AppDb"].
           ConnectionString, SqlServerOrmLiteDialectProvider.Instance);

        protected IDbConnection dbConn;

public Repository()
        {
            dbConn = dbFactory.OpenDbConnection();
        }

public List<Contact> Getall()
        {
            return dbConn.Select<Contact>();
        }

public void Dispose()
        {
            if (dbConn != null)
                dbConn.Dispose();
        }
    }
```

Interface implements a repository class that takes responsibility for database connection as DBContext class in MVC application.

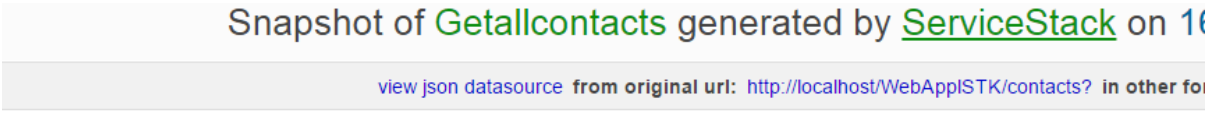
4, Resolve Repository in DI

In Application_Start() method, we initiate an object from a concert class inherited from AppHostBase abstract class as below

```
protected void Application_Start()
        {
            //AreaRegistration.RegisterAllAreas();
            //RouteConfig.RegisterRoutes(RouteTable.Routes);
            new AppHost().Init();

         }

public class AppHost :AppHostBase
    {
        public AppHost(): base("Contact Web Service",
typeof(Models.ContactSTKInterface).Assembly)
        {
        }
```

```
public override void Configure(Container container)
        {
            //client such as mvcclient etc will be configure here.
            container.Register<IDbConnectionFactory>(c => new
OrmLiteConnectionFactory(ConfigurationManager.ConnectionStrings["AppDb"].ConnectionStr
ing, SqlServerDialect.Provider));
            container.Register<IRepos>(c=>new Repository());
        }

    }
```

After AppHost is initiated, Configure () method is called, DI for database connection component and third party data services such as Repository class are built here. Fung.container is a DI container that can register Database connection Repository class with its interface IRepos in DI container. Now IRepos Interface is ready for Repository class and will represent it as a real object in data service as previous mentioned. The final result is as below



Id	Name	Email	Age
1	davi	as@ff.com	12
2	steven	arw@fgr.com	34

When we call link http://localhost/STK/contacts, ServiceStack will really think we are passing Http request in, ServiceStack then handle this request and returns a http Response (search result) back to caller. Http response class will inject Repository data class interface such as IRepos<> in, after initiation of this interface, it become an object. This object then can do CRUD operations for you in front end.

3, SUMMARY

Similar to WPF application, AppHost.cs boots the application when URL is called. AppHost.cs runs configure method to register dependency and its interface in DI container. Those dependencies then can be called by its interface anywhere in application. That is when we call "~/contacts", web initiates a request, service stack proceeds this request and return a http response, http response may do something before returning result back to client, It may inject a data access component such as Irepos interface into service class. Service class then use this interface to deal with database and do CRUD operations via database connection that has been built via DI container when application starts. This service class now can return results back to client as JSON or XML data format or other format such as CSV JSV etc. Front end client application such as Angular JS PA and JQuery/HTML 5 etc.