

فرض کنید لایه سرویس برنامه دارای اینترفیس و کلاس‌های زیر است:

```
namespace SignalR02.Services
{
    public interface ITestService
    {
        int GetRecordsCount();
    }
}
```

```
namespace SignalR02.Services
{
    public class TestService : ITestService
    {
        public int GetRecordsCount()
        {
            return 10; // It's just a sample to test IOC's.
        }
    }
}
```

قصد داریم از این لایه، توسط تزریق وابستگی‌ها در Hub برنامه استفاده کنیم:

```
[HubName("chat")]
public class ChatHub : Hub
{
    // جهت آزمایش تزریق خودکار وابستگی‌ها
    private readonly ITestService _testService;
    public ChatHub(ITestService testService)
    {
        _testService = testService;
    }

    public void SendMessage(string message)
    {
        var msg = string.Format("{0}:{1}", Context.ConnectionId, message);
        Clients.All.hello(msg);

        Clients.All.hello(string.Format("RecordsCount: {0}", _testService.GetRecordsCount()));
    }
}
```

برنامه، همان برنامه‌ای است که در دوره جاری تکمیل گردیده است. فقط در اینجا سازنده کلاس اضافه شده و سپس اینترفیس ITestService به عنوان پارامتر آن تعریف گردیده است. در ادامه می‌خواهیم کار وهله سازی و تزریق نمونه مرتبط را توسط [StructureMap](#) به صورت خودکار انجام دهیم. برای این منظور یک کلاس جدید را به نام StructureMapDependencyResolver به برنامه اضافه کنید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNet.SignalR;
using StructureMap;

namespace SignalR02.Utils
{
    public class StructureMapDependencyResolver : DefaultDependencyResolver
    {
        private readonly IContainer _container;
        public StructureMapDependencyResolver(IContainer container)
        {
            if (container == null)
            {
                throw new ArgumentNullException("container");
            }
        }
    }
}
```

```

        }
        _container = container;
    }
    public override object GetService(Type serviceType)
    {
        return !serviceType.IsAbstract && !serviceType.IsInterface && serviceType.IsClass
            ? _container.GetInstance(serviceType)
            : (_container.TryGetInstance(serviceType) ??
base.GetService(serviceType));
    }
    public override IEnumerable<object> GetServices(Type serviceType)
    {
        return
        _container.GetAllInstances(serviceType).Cast<object>().Concat(base.GetServices(serviceType));
    }
}
}

```

کار این کلاس، تعویض DefaultDependencyResolver توکار SignalR با StructureMap است. از این جهت که برای مثال در سراسر برنامه از StructureMap جهت تزریق وابستگی‌ها استفاده شده است و قصد داریم در قسمت Hub آن نیز یکپارچگی کار حفظ گردد.

برای استفاده از این کلاس تعریف شده فقط کافی است Application_Start فایل Global.asax.cs برنامه هاب را به نحو ذیل تغییر دهیم:

```

using System;
using System.Web;
using System.Web.Routing;
using Microsoft.AspNet.SignalR;
using SignalR02.Services;
using SignalR02.Utils;
using StructureMap;

namespace SignalR02
{
    public class Global : HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            ObjectFactory.Initialize(cfg =>
            {
                cfg.For<IDependencyResolver>().Singleton().Add<StructureMapDependencyResolver>();
                // the rest ...
                cfg.For<ITestService>().Use<TestService>();
            });
            GlobalHost.DependencyResolver = ObjectFactory.GetInstance<IDependencyResolver>();

            // Register the default hubs route: ~/signalr
            RouteTable.Routes.MapHubs(new HubConfiguration
            {
                EnableCrossDomain = true
            });
        }
    }
}

```

در اینجا در ابتدای کار IDependencyResolver توکار StructureMap با کلاس StructureMapDependencyResolver وهله سازی می‌گردد. سپس تعاریف متداول تنظیمات کلاس‌ها و اینترفیس‌های لایه سرویس برنامه اضافه می‌شوند. همچنین نیاز است GlobalHost.DependencyResolver توکار SignalR نیز به نحوی که ملاحظه می‌کنید مقدار دهی گردد.

اینبار اگر برنامه را اجرا کنید و سپس یکی از کلاینت‌های آن‌را فراخوانی نمایید، می‌توان مشاهده کرد که کار وهله سازی و تزریق وابستگی سرویس مورد استفاده به صورت خودکار انجام گردیده است:

```
7 [HubName("chat")]
8 public class ChatHub : Hub
9 {
10     // جهت آزمایش تزریق خودکار وابستگی‌ها
11     private readonly ITestService _testService;
12     public ChatHub(ITestService testService)
13     {
14         _testService = testService;
15     }
16
17     public void SendMessage(string message)
18     {
19         var msg = string.Format("{0}:{1}", Context.ConnectionId, message);
20         Clients.All.hello(msg);
21         Clients.All.hello(string.Format("RecordsCount: {0}", _testService.GetRecordsCount()));
22     }
23 }
```

🔍 _testService | {SignalR02.Services.TestService} ⇐

نظرات خوانندگان

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۱۰/۱۰ ۰:۱۶

سلام،

پایه‌سازی این قسمت باید به صورت جدا از پایه‌سازی [DefaultControllerFactory](#) صورت بگیره؟ یعنی برای هر کدام وابستگی‌ها به صورت جداگانه تزریق شوند؟

```
private void InitStructureMap()
{
    ObjectFactory.Initialize(x =>
    {
        x.For<IUnitOfWork>().HybridHttpOrThreadLocalScoped().Use(() => new MyDbContext());
        x.For<IRequestService>().Use<RequestService>();
        x.For<IRequestTypeService>().Use<RequestTypeService>();
        x.For<IUnitService>().Use<UnitService>();
        x.For<IDomainService>().Use<DomainService>();
    });

    ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());

    ObjectFactory.Initialize(cfg =>
    {
        cfg.For<IDependencyResolver>().Singleton().Add<StructureMapDependencyResolver>();
        // the rest ...
        cfg.For<IUnitOfWork>().HybridHttpOrThreadLocalScoped().Use(() => new MyDbContext());
        cfg.For<IRequestService>().Use<RequestService>();
        cfg.For<IRequestTypeService>().Use<RequestTypeService>();
        cfg.For<IUnitService>().Use<UnitService>();
        cfg.For<IDomainService>().Use<DomainService>();
    });
    GlobalHost.DependencyResolver = ObjectFactory.GetInstance<IDependencyResolver>();
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۰ ۰:۲۰

تعاریف اولیه IoC Container یکی هست. فقط این مورد خاص، سطر GlobalHost.DependencyResolver را بیشتر دارد.

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۳/۱۰/۱۱ ۱۲:۳۴

خیلی ممنون، مشکلم حل شد، فقط یک سوال:

در [این](#) حالت تزریق وابستگی دقیقاً باید در کجا صورت بگیره؟ اینکار رو درون متد defaultContainer انجام دادم:

```
private static Container defaultContainer()
{
    return new Container(ioc =>
    {
        //....
        cfg.For<IDependencyResolver>().Singleton().Add<StructureMapDependencyResolver>();
    });
}
```

و نهایتاً در Application_Start کد زیر را برای جایگزینی GlobalHost.DependencyResolver انجام دادم:

```
GlobalHost.DependencyResolver = ObjectFactory.GetInstance<IDependencyResolver>();
```

نویسنده: وحید نصیری

- اگر از [SmObjectFactory](#) استفاده می‌کنید، Container آن با ObjectFactory یکی نیست یا به عبارتی ObjectFactory اطلاعی در مورد تنظیمات کلاس سفارشی Sm ObjectFactory ندارد. بنابراین دیگر نباید از ObjectFactory قدیمی استفاده کنید.

- من Sm ObjectFactory را از این جهت اضافه کردم، چون قرار است ObjectFactory را در نگارش بعدی استراکچرمپ حذف کنند. همین مساله سبب بروز مشکلات زیادی در مدیریت Container های آن می‌شود؛ چون هر Container مشخصی، وهله‌های مجزا و تنظیمات مجزایی را خواهد داشت و در این حالت بسیاری با مدیریت برای مثال یک وهله از IUnitOfWork در طول یک درخواست مشکل پیدا می‌کنند. وهله‌ی a از IUnitOfWork توسط container 1 با وهله مشابهی از آن توسط container 2 یکی نیست. حتی اگر مدیریت طول عمر container 1 به درستی انجام نشود، باز هم به چندین وهله از container 1 خواهیم رسید.

ممنون، اتفاقاً Resharper هم این پیغام رو میداد: **Class 'StructureMap.ObjectFactory' is obsolete: ObjectFactory will be removed in a future 4.0 release of StructureMap. Favor the usage of the Container class for future work**

نهایتاً به این صورت مشکل حل شد:

```
SmObjectFactory.Container.Configure(x =>
{
    x.For<IDependencyResolver>().Singleton().Add<StructureMapDependencyResolver>();
});
GlobalHost.DependencyResolver =
SmObjectFactory.Container.GetInstance<IDependencyResolver>();
```