عنوان: **تزریق وابستگی‌ها در فیلترهای ASP.NET MVC**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۱۸ ۱:۷
آدرس: www.dotnettips.info
گروه‌ها: Design patterns, Dependency Injection, IoC

فرض کنید فیلتر سفارشی لاگ کردن را که از سرویس ILogActionService استفاده می‌کند، به نحو ذیل تعریف کرده‌اید:

```
public interface ILogActionService
{
    void Log(string data);
}

public class LogAttribute : ActionFilterAttribute
{
    public ILogActionService LogActionService { get; set; }

    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        LogActionService.Log("......data......");
        base.OnActionExecuted(filterContext);
    }
}
```

با استفاده‌ای مانند:

```
[Log]
public ActionResult Index()
{}
```

روش متداول تنظیمات تزریق وابستگی‌ها در ASP.NET MVC، بیشتر به بحث کنترلرها مرتبط است و سایر قسمت‌ها را پوشش نمی‌دهد. برای این مورد خاص ابتدا نیاز است یک FilterProvider سفارشی را به نحو ذیل تدارک دید:

```
using StructureMap;
using System.Collections.Generic;
using System.Web.Mvc;

namespace DI06.CustomFilters
{
    public class StructureMapFilterProvider : FilterAttributeFilterProvider
    {
        private readonly IContainer _container;
        public StructureMapFilterProvider(IContainer container)
        {
            _container = container;
        }

        public override IEnumerable<Filter> GetFilters(ControllerContext controllerContext,
ActionDescriptor actionDescriptor)
        {
            var filters = base.GetFilters(controllerContext, actionDescriptor);
            foreach (var filter in filters)
            {
                _container.BuildUp(filter.Instance);
                yield return filter;
            }
        }
    }
}
```

نکته‌ی مهم آن، استفاده از متد BuildUp استراکچرمپ است. نمونه‌ی آن‌را در تنظیمات تزریق وابستگی‌ها در وب فرم‌ها پیشتر ملاحظه کرده‌اید . در این مثال کار آن وهله سازی وابستگی‌های فیلترهای تعریف شده در برنامه است.
پس از اینکه FilterProvider سفارشی مخصوص کار با استراکچرمپ را تهیه کردیم، اکنون نوبت به جایگزین کردن آن با FilterProvider پیش فرض ASP.NET MVC در فایل global.asax.cs به نحو ذیل است:

```
//Using the custom StructureMapFilterProvider
var filterProvider = FilterProviders.Providers.Single(provider => provider is
```

```
FilterAttributeFilterProvider);
FilterProviders.Providers.Remove(filterProvider);
FilterProviders.Providers.Add(SmObjectFactory.Container.GetInstance<StructureMapFilterProvider>());
```

استفاده از SmObjectFactory.Container.GetInstance سبب خواهد شد تا به صورت خودکار، وابستگی تزریق شده‌ی در سازنده‌ی کلاس StructureMapFilterProvider وهله سازی و تامین شود.

همچنین در این مثال چون تزریق وابستگی در کلاس LogAttribute از نوع setter injection است، نیاز است در تنظیمات ابتدایی Container مورد استفاده، Policies.SetAllProperties نیز قید شود:

```
namespace DI06.IocConfig
{
    public static class SmObjectFactory
    {
        private static readonly Lazy<Container> _containerBuilder =
            new Lazy<Container>(defaultContainer, LazyThreadSafetyMode.ExecutionAndPublication);

        public static IContainer Container
        {
            get { return _containerBuilder.Value; }
        }

        private static Container defaultContainer()
        {
            return new Container(x =>
            {
                x.For<ILogActionService>().Use<LogActionService>();

                x.Policies.SetAllProperties(y =>
                {
                    y.OfType<ILogActionService>();
                });
            });
        }
    }
}
```

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:

DI06