عنوان: **استفاده از RavenDB در ASP.NET MVC به همراه تزریق وابستگی‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۲۱ ۱۵:۵۳
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, RavenDB

جهت تکمیل مباحث این دوره می‌توان به نحوه مدیریت سشن‌ها و document store بانک اطلاعاتی RavenDB با استفاده از یک IoC Container مانند StructureMap در ASP.NET MVC پرداخت. اصول کلی آن به تمام فناوری‌های دات نتی دیگر مانند وب فرم‌ها، WPF و غیره نیز قابل بسط است. تنها پیشنیاز آن مطالعه «کامل» دوره « بررسی مفاهیم معکوس سازی وابستگی‌ها و ابزارهای مرتبط با آن » می‌باشد.

هدف از بحث

ارائه راه حلی جهت تزریق یک وهله از واحد کار تشکیل شده (همان شیء سشن در RavenDB) به کلیه کلاس‌های لایه سرویس برنامه و همچنین زنده نگه داشتن شیء document store آن در طول عمر برنامه است. ایجاد شیء document store که کار اتصال به بانک اطلاعاتی را مدیریت می‌کند، بسیار پرهزینه است. به همین جهت این شیء تنها یکبار باید در طول عمر برنامه ایجاد شود.

ابزارها و پیشنیازهای لازم

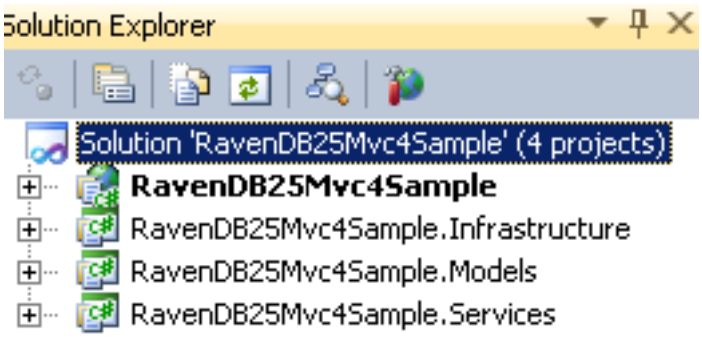
ابتدا یک برنامه جدید ASP.NET MVC را آغاز کنید. سپس ارجاعات لازم را به کلاینت RavenDB، سرور درون پروسه‌ای آن (RavenDB.Embedded) و همچنین StructureMap با استفاده از نیوگت، اضافه نمائید:

```
PM> Install-Package RavenDB.Client
PM> Install-Package RavenDB.Embedded -Pre
PM> Install-Package structuremap
```

دریافت کدهای کامل این مثال

RavenDB25Mvc4Sample.zip

این مثال، به همراه فایل‌های باینری ارجاعات یاد شده، نیست (جهت کاهش حجم 100 مگابایتی آن). برای بازیابی آن‌ها می‌توانید به مطلبی در اینباره در سایت مراجعه کنید.
این پروژه از چهار قسمت مطابق شکل زیر تشکیل شده است:



الف) لایه سرویس‌های برنامه

```
using RavenDB25Mvc4Sample.Models;
using System.Collections.Generic;

namespace RavenDB25Mvc4Sample.Services.Contracts
{
    public interface IUsersService
    {
```

```
        User AddUser(User user);
        IList<User> GetUsers(int page, int count = 20);
    }
}

using System.Collections.Generic;
using System.Linq;
using Raven.Client;
using RavenDB25Mvc4Sample.Models;
using RavenDB25Mvc4Sample.Services.Contracts;

namespace RavenDB25Mvc4Sample.Services
{
    public class UsersService : IUsersService
    {
        private readonly IDocumentStore _documentStore;
        private readonly IDocumentSession _documentSession;
        public UsersService(IDocumentStore documentStore, IDocumentSession documentSession)
        {
            _documentStore = documentStore;
            _documentSession = documentSession;
        }

        public User AddUser(User user)
        {
            _documentSession.Store(user);
            return user;
        }

        public IList<User> GetUsers(int page, int count = 20)
        {
            return _documentSession.Query<User>()
                                   .Skip(page * count)
                                   .Take(count)
                                   .ToList();
        }

        //todo: سایر متدهای مورد نیاز در اینجا

    }
}
```

نکته مهمی که در اینجا وجود دارد، استفاده از اینترفیس‌های خود RavenDB است. به عبارتی IDocumentSession، تشکیل دهنده الگوی واحد کار در RavenDB است و نیازی به تعاریف اضافه‌تری در اینجا وجود ندارد.
هر کلاس لایه سرویس با یک اینترفیس مشخص شده و اعمال آن‌ها از طریق این اینترفیس‌ها در اختیار کنترلرهای برنامه قرار می‌گیرند.

ب) لایه Infrastructure برنامه

در این لایه کدهای اتصالات IoC Container مورد استفاده قرار می‌گیرند. کدهایی که به برنامه جاری وابسته‌اند، اما حالت عمومی و مشترکی ندارند تا در سایر پروژه‌های مشابه استفاده شوند.

```
using Raven.Client;
using Raven.Client.Embedded;
using RavenDB25Mvc4Sample.Services;
using RavenDB25Mvc4Sample.Services.Contracts;
using StructureMap;

namespace RavenDB25Mvc4Sample.Infrastructure
{
    public static class IoCConfig
    {
        public static void ApplicationStart()
        {
            ObjectFactory.Initialize(x =>
            {
                // داکیومنت استور سینگلتون تعریف شده چون باید در طول عمر برنامه زنده نگه داشته شود
                x.ForSingletonOf<IDocumentStore>().Use(() =>
                       {
                           return new EmbeddableDocumentStore
                           {
                               DataDirectory = "App_Data"
                           }.Initialize();
                       });
```

```
                // سشن در برنامه وب هیبرید تعریف شده تا در طول عمر یک درخواست زنده نگه داشته شود
                // در برنامه‌های ویندوزی حالت هیبرید را حذف کنید
                x.For<IDocumentSession>().HybridHttpOrThreadLocalScoped().Use(context =>
                    {
                        return context.GetInstance<IDocumentStore>().OpenSession();
                    });

                // اتصالات لایه سرویس در اینجا
                x.For<IUsersService>().Use<UsersService>();
                // ...
            });
        }

        public static void ApplicationEndRequest()
        {
            ObjectFactory.ReleaseAndDisposeAllHttpScopedObjects();
        }
    }
}
```

تعاریف اتصالات StructureMap را در اینجا ملاحظه می‌کنید.
IDocumentStore و IDocumentSession، دو اینترفیس تعریف شده در کلاینت RavenDB هستند. اولی کار اتصال به بانک اطلاعاتی را مدیریت خواهد کرد و دومی کار مدیریت الگوی واحد کار را انجام می‌دهد. IDocumentStore به صورت Singleton تعریف شده است؛ چون باید در طول عمر برنامه زنده نگه داشته شود. اما IDocumentStore در ابتدای هر درخواست رسیده، وهله سازی شده و سپس در پایان هر درخواست در متد ApplicationEndRequest به صورت خودکار Dispose خواهد شد.
اگر به فایل Global.asax.cs پروژه وب برنامه مراجعه کنید، نحوه استفاده از این کلاس را مشاهده خواهید کرد:

```
using System;
using System.Globalization;
using System.Web.Mvc;
using System.Web.Routing;
using RavenDB25Mvc4Sample.Infrastructure;
using StructureMap;

namespace RavenDB25Mvc4Sample
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            IoCConfig.ApplicationStart();

            AreaRegistration.RegisterAllAreas();

            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);

            //Set current Controller factory as StructureMapControllerFactory
            ControllerBuilder.Current.SetControllerFactory(new StructureMapControllerFactory());
        }

        protected void Application_EndRequest(object sender, EventArgs e)
        {
            IoCConfig.ApplicationEndRequest();
        }
    }

    public class StructureMapControllerFactory : DefaultControllerFactory
    {
        protected override IController GetControllerInstance(RequestContext requestContext, Type
controllerType)
        {
            if (controllerType == null)
                throw new InvalidOperationException(string.Format("Page not found: {0}",

requestContext.HttpContext.Request.Url.AbsoluteUri.ToString(CultureInfo.InvariantCulture)));
            return ObjectFactory.GetInstance(controllerType) as Controller;
        }
    }
}
```

در ابتدای کار برنامه، متد IoCConfig.ApplicationStart جهت برقراری اتصالات، فراخوانی می‌شود. در پایان هر درخواست نیز شیء سشن جاری تخریب خواهد شد. همچنین کلاس StructureMapControllerFactory نیز جهت وهله سازی خودکار کنترلرهای برنامه به همراه تزریق وابستگی‌های مورد نیاز، تعریف گشته است.

ج) استفاده از کلاس‌های لایه سرویس در کنترلرهای برنامه

```
using System.Web.Mvc;
using Raven.Client;
using RavenDB25Mvc4Sample.Models;
using RavenDB25Mvc4Sample.Services.Contracts;

namespace RavenDB25Mvc4Sample.Controllers
{
    public class HomeController : Controller
    {
        private readonly IDocumentSession _documentSession;
        private readonly IUsersService _usersService;
        public HomeController(IDocumentSession documentSession, IUsersService usersService)
        {
            _documentSession = documentSession;
            _usersService = usersService;
        }

        [HttpGet]
        public ActionResult Index()
        {
            return View(); //نمایش صفحه ثبت
        }

        [HttpPost]
        public ActionResult Index(User user)
        {
            if (this.ModelState.IsValid)
            {
                _usersService.AddUser(user);
                _documentSession.SaveChanges();

                return RedirectToAction("Index");
            }
            return View(user);
        }
    }
}
```

پس از این مقدمات و طراحی اولیه، استفاده از کلاس‌های لایه سرویس در کنترل‌ها، ساده خواهند بود. تنها کافی است اینترفیس‌های مورد نیاز را از طریق روش تزریق در سازنده کلاس‌ها تعریف کنیم. سایر مسایل وهله سازی آن خودکار خواهند بود.