NopCommerce

是

17. NopCommerce

（1）概述

很多人都说通过阅读、学习大神们高质量的代码是提高自己技术能力最快的方式之一。我觉得通过阅读[NopCommerce](http://www.lanhusoft.com/Article/349.html)的源码，可以从中学习很多企业系统、[软件开发](http://www.lanhusoft.com/)的规范和一些新的技术、技巧，可以快速地提高我们的技术能力。所以我最近决定写一个“[nop](http://www.lanhusoft.com/Article/349.html)Commerce源码架构详解”的系列，来详细剖析NopCommerce的架构和原理。

**Nopcommerce主要用到的技术及特点：**

1、Entity Framework

2、ASP.NET [mvc](http://www.lanhusoft.com/Article/169.html)

3、IoC容器+[依赖注入](http://www.lanhusoft.com/Article/105.html)(Autofac)

4、使用EF中的EntityTypeConfiguration+Repository模式+领域驱动开发

5、插件技术

6、Themes主题技术

7、Ajax

8、Validator验证技术

9、面向接口编程

10、事件通知、日志机制

11、缓存（System.Runtime.Caching.MemoryCache）

12、网站计划任务

13、消息队列

14、多语言支持

15、Jquery UI+kendo UI

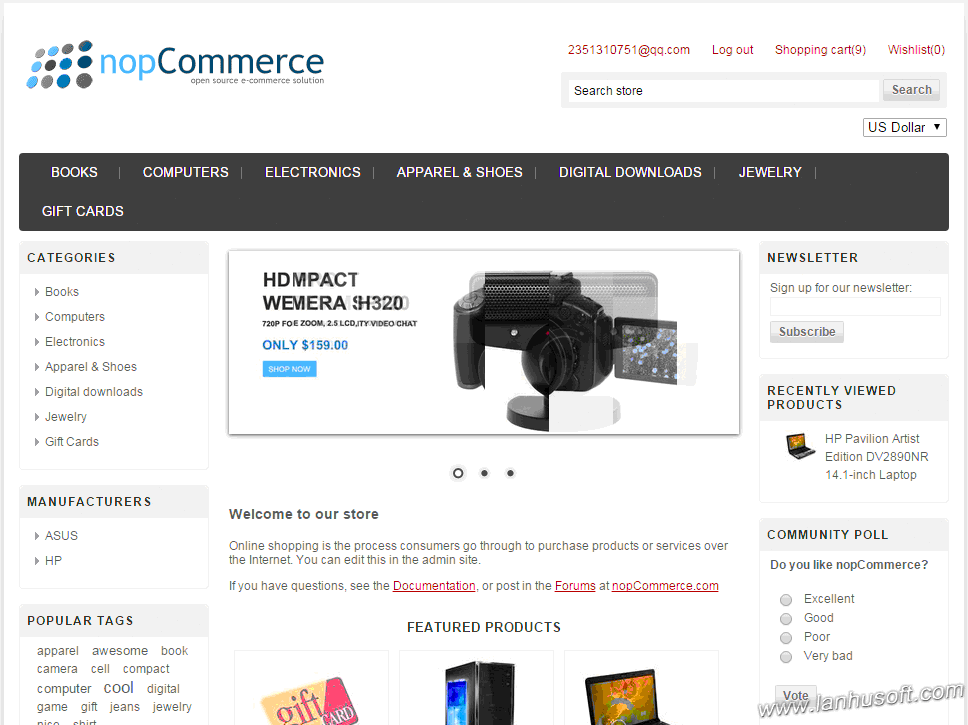
16、多网店支持、促销推广、在线支付

17、[seo](http://www.lanhuseo.com/)友好支持

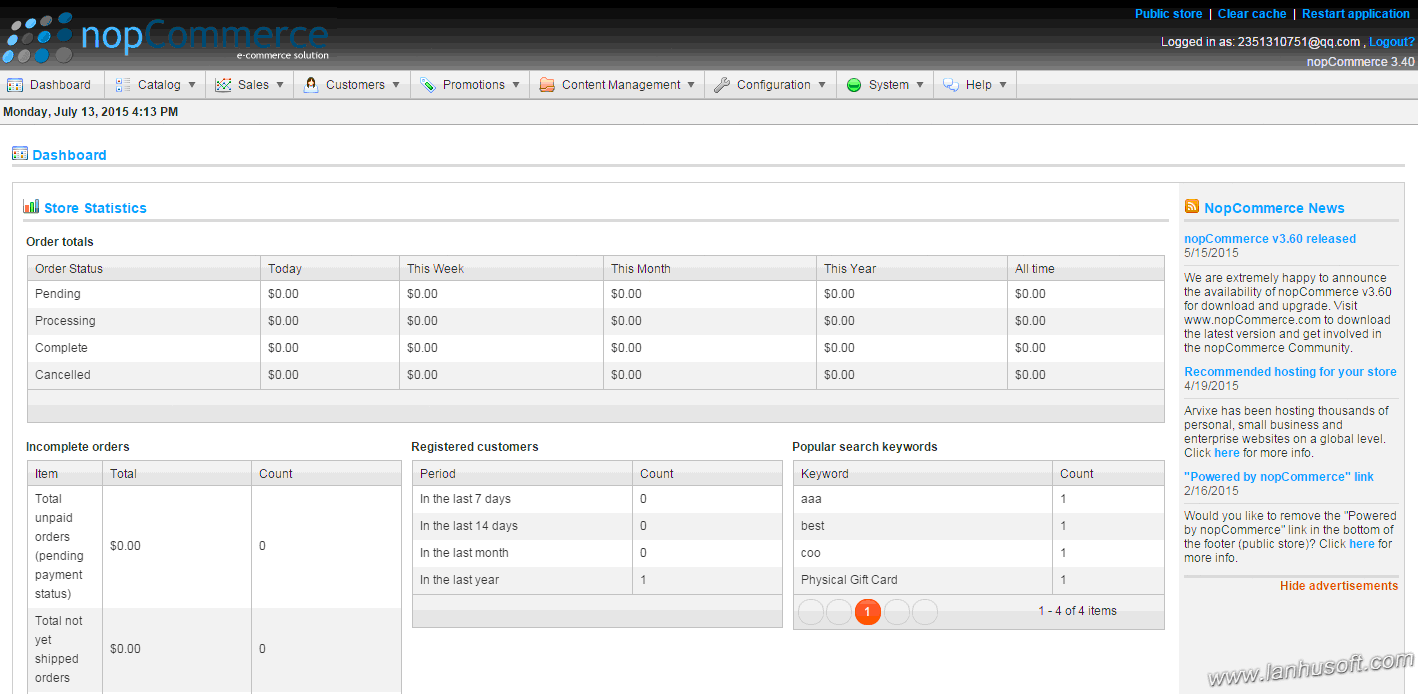
18、其它asp.net MVC和c#最新核心技术

Nopcommerce是国外的一个高质量的开源b2c网站系统，最新版是基于Entity Framework6.0和MVC5.0，使用[razor](http://www.lanhusoft.com/Article/198.html)模板引擎，有很强的插件机制，包括支付配送功能都是通过插件来实现的，基于xml的多语言版本，非常灵活的语言切换功能，包括在后台都能同时编辑产品的中英文属性，非常适合做外贸，优秀超前的程序架构，性能也非常强大，自定义的产品名称和分类又有很好的seo优化。综合能力远远高于国内的一些程序架构糟糕的.net商城程序，是二次开发和大型b2c架构的首选。3.0开始支持多店。

**前台页面效果：**



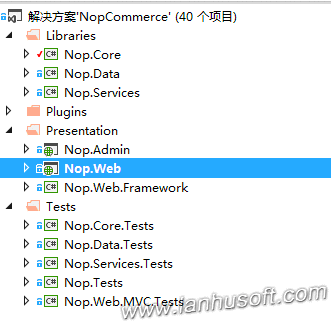
**后台管理页面：**



NopCommerce最新版的在CodePlex的[源码下载](http://nopcommerce.codeplex.com/)。

nopcommerce主要从上往下Nop.Web、Nop.Admin、Nop.Web.Framework、Nop插件、Nop.Services、  
Nop.Core、Nop.Data。引用的第三方模块EntityFramework，Autofac(控制反转，即依赖注入)，telerik.extern.mvc(后台管理用的界面，2.0后开始使用)。

**下图是nopcommerce版本3.4的源码结构：**



**1、Libraries**

Libaries文件夹下项目主要是一些公共库代码。

Nop.Core：封装了项目要用的基础核心类，接口。比如领域对象类，缓存类、接口，[扩展方法](http://www.lanhusoft.com/Article/27.html)等等。

Nop.Data：EF相关的数据访问相关的类封装和扩展。里面最关键的就是Mapping，Nop采用代码API的形式来建立Model和数据库表之间的映射，命名都是以“表名+Map”的形式。比如：

1. using System.Data.Entity.ModelConfiguration;
2. using Nop.Core.Domain.Blogs;
4. namespace Nop.Data.Mapping.Blogs
5. {
6. public partial class BlogCommentMap : EntityTypeConfiguration<BlogComment>
7. {
8. public BlogCommentMap()
9. {
10. this.ToTable("BlogComment");
11. this.HasKey(pr => pr.Id);
13. this.HasRequired(bc => bc.BlogPost)
14. .WithMany(bp => bp.BlogComments)
15. .HasForeignKey(bc => bc.BlogPostId);
17. this.HasRequired(cc => cc.Customer)
18. .WithMany()
19. .HasForeignKey(cc => cc.CustomerId);
20. }
21. }
22. }

以后我会详细的说明这个Mapping的怎么实现的以级这样做的好处。

Nop.Services：真正的处理数据的业务层，都是通过面向接口编程，减少对具体实现的依赖。

**2、Plugins**

Plugins文件夹下是放的插件项目，你也可以按照规则开发属于自己的插件。

**3、Presentation**

Presentation中文意思是呈现、表现的意思。也就是这文件夹下的项目都是解决方案的表示层。

Nop.Admin：后台管理

Nop.Web：前台Web项目

Nop.Web.Framework：Web及MVC相关扩展和公共类的封装，比如：BaseController，Seo相关，主题Themes，autofac依赖注入DependencyRegistrart等等。

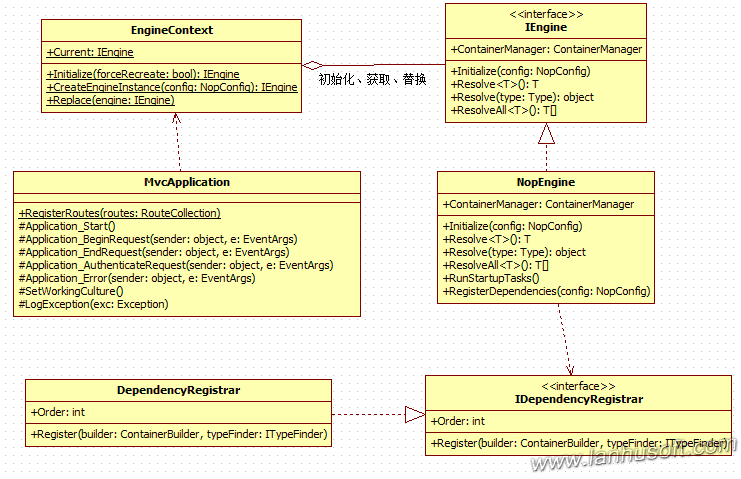
**4、Tests**

Tests下面放的都是对应项目的单元测试。

（2）Autofac依赖注入分析

[NopCommerce](http://www.lanhusoft.com/Article/349.html)整个代码里面都贯彻了一个面向接口编程的思想。[nop](http://www.lanhusoft.com/Article/349.html)Commerce商城系统是通过Autofac对接口依赖的实现进行[依赖注入](http://www.lanhusoft.com/Article/105.html)的。今天我们就来分析一下里面的源码，学习一下Nop在依赖注入方面设计思想，希望在以后的架构设计中对你有启发和帮助。

下面是Nop依赖注入相关类图。



可以看到这里面相关的类为

[mvc](http://www.lanhusoft.com/Article/169.html)Application

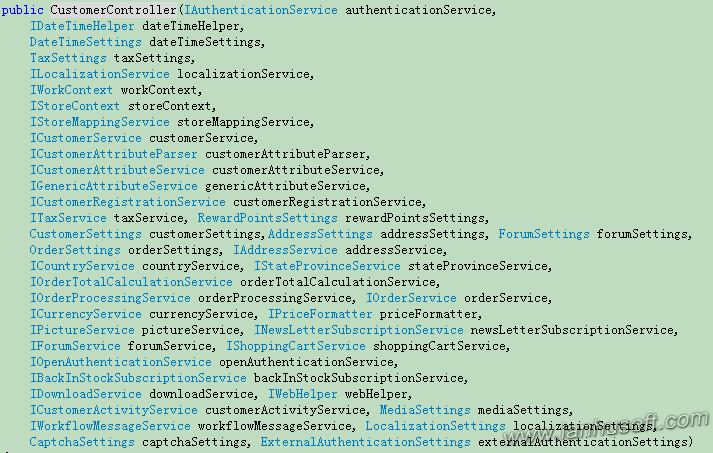
IEngine，NopEngine

EngineContext

IDependencyRegistrar，DependencyRegistrar

下面我们再来看看这些类的核心代码。

我们打开Nop.Web项目的所有的Controller源码，可以看到基本上都是通过构造函数引用抽象的接口。如CustomerController的构造函数的参数类型都是某一个接口，如下图：



那我们要想看这些接口到底依赖的是哪一个真正的实现类呢？

在项目Nop.Web.Framework中有一个类DependencyRegistrar，其作用就是调用Autofac相关的API进行依赖注入，我们只要打开这个类文件，搜索一个指定的接口，就可以找到它依赖的具体类。例如，我们想看到ICategoryService接口是依赖哪一个实现，在这个类中搜索ICategoryService，会找到下面的代码：

1. builder.RegisterType<CategoryService>().As<ICategoryService>().InstancePerLifetimeScope();

可以看到这个接口对应的实现为为CategoryService（Nop.Services.Catalog命名空间下的类CategoryService）。

下面我们来看看Nop在网站启动是怎么调用这个DependencyRegistrar类呢？

在项目Nop.Web也就是前台的起始项目有一个文件Global.asax里有一个函数Application\_Start，这函数的意义这里就不用说了，我相信学过ASP.NET同学都知道它的作用，下面是函数Application\_Start的完整代码：

1. protected void Application\_Start()
2. {
3. //初始化上下文引擎
4. EngineContext.Initialize(false);
6. //model binders
7. ModelBinders.Binders.Add(typeof(BaseNopModel), new NopModelBinder());
9. bool databaseInstalled = DataSettingsHelper.DatabaseIsInstalled();
10. if (databaseInstalled)
11. {
12. //remove all view engines
13. ViewEngines.Engines.Clear();
14. //except the themeable [razor](http://www.lanhusoft.com/Article/198.html) view engine we use
15. ViewEngines.Engines.Add(new ThemeableRazorViewEngine());
16. }
18. //Add some functionality on top of the default ModelMetadataProvider
19. ModelMetadataProviders.Current = new NopMetadataProvider();
21. //Registering some regular mvc stuff
22. AreaRegistration.RegisterAllAreas();
23. RegisterRoutes(RouteTable.Routes);
25. //fluent validation
26. DataAnnotationsModelValidatorProvider.AddImplicitRequiredAttributeForValueTypes = false;
27. ModelValidatorProviders.Providers.Add(new FluentValidationModelValidatorProvider(new NopValidatorFactory()));
29. //start scheduled tasks
30. if (databaseInstalled)
31. {
32. TaskManager.Instance.Initialize();
33. TaskManager.Instance.Start();
34. }
36. //log application start
37. if (databaseInstalled)
38. {
39. try
40. {
41. //log
42. var logger = EngineContext.Current.Resolve<ILogger>();
43. logger.Information("Application started", null, null);
44. }
45. catch (Exception)
46. {
47. //don't throw new exception if occurs
48. }
49. }
50. }

里面有第一行代码就调用了上下文引擎初始化。

1. EngineContext.Initialize(false);

下面我们来看看这个EngineContext的静态方法Initialize：

1. [MethodImpl(MethodImplOptions.Synchronized)]
2. public static IEngine Initialize(bool forceRecreate)
3. {
4. if (Singleton<IEngine>.Instance == null || forceRecreate)
5. {
6. var config = ConfigurationManager.GetSection("NopConfig") as NopConfig;
7. Singleton<IEngine>.Instance = CreateEngineInstance(config);
8. Singleton<IEngine>.Instance.Initialize(config);
9. }
10. return Singleton<IEngine>.Instance;
11. }
12. protected static IEngine CreateEngineInstance(NopConfig config)
13. {
14. if (config != null && !string.IsNullOrEmpty(config.EngineType))
15. {
16. var engineType = Type.GetType(config.EngineType);
17. if (engineType == null)
18. throw new ConfigurationErrorsException("The type '" + config.EngineType + "' could not be found. Please check the configuration at /configuration/nop/engine[@engineType] or check for missing assemblies.");
19. if (!typeof(IEngine).IsAssignableFrom(engineType))
20. throw new ConfigurationErrorsException("The type '" + engineType + "' doesn't implement 'Nop.Core.Infrastructure.IEngine' and cannot be configured in /configuration/nop/engine[@engineType] for that purpose.");
21. return Activator.CreateInstance(engineType) as IEngine;
22. }
24. return new NopEngine();
25. }

从上面可以看到Initialize返回一个IEngine的实例，并调用其方法Initialize。创建IEngine的实例是根据配置文件来获取哪一个IEngine的具体类，如果没有配置这个结点信息或者为空，则返回默认的IEngine的实现NopEngine。

下面我们来看看Nop.Core.Infrastructure.NopEngine里面的核心代码：

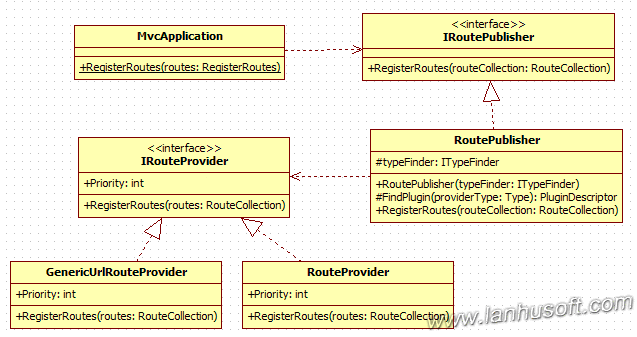
1. public void Initialize(NopConfig config)
2. {
3. //注册依赖
4. RegisterDependencies(config);
6. //startup tasks
7. if (!config.IgnoreStartupTasks)
8. {
9. RunStartupTasks();
10. }
12. }
14. /// <summary>
15. /// 注册依赖
16. /// </summary>
17. /// <param name="config">Config</param>
18. protected virtual void RegisterDependencies(NopConfig config)
19. {
20. var builder = new ContainerBuilder();
21. var container = builder.Build();
23. //we create new instance of ContainerBuilder
24. //because Build() or Update() method can only be called once on a ContainerBuilder.

27. //dependencies
28. var typeFinder = new WebAppTypeFinder(config);
29. builder = new ContainerBuilder();
30. builder.RegisterInstance(config).As<NopConfig>().SingleInstance();
31. builder.RegisterInstance(this).As<IEngine>().SingleInstance();
32. builder.RegisterInstance(typeFinder).As<ITypeFinder>().SingleInstance();
33. builder.Update(container);
35. //register dependencies provided by other assemblies
36. builder = new ContainerBuilder();
37. //查找所有实现了接口IDependencyRegistrar的类
38. var drTypes = typeFinder.FindClassesOfType<IDependencyRegistrar>();
39. var drInstances = new List<IDependencyRegistrar>();
40. foreach (var drType in drTypes)
41. drInstances.Add((IDependencyRegistrar) Activator.CreateInstance(drType));
42. //排序
43. drInstances = drInstances.AsQueryable().OrderBy(t => t.Order).ToList();
44. //依次调用实现IDependencyRegistrar接口的类的方法Register
45. foreach (var dependencyRegistrar in drInstances)
46. dependencyRegistrar.Register(builder, typeFinder);
47. builder.Update(container);

50. this.\_containerManager = new ContainerManager(container);
52. //set dependency resolver
53. DependencyResolver.SetResolver(new AutofacDependencyResolver(container));
54. }

（3）NopCommerce源码架构详解--路由相关源码分析

刚开始研究[nop](http://www.lanhusoft.com/Article/349.html)的同学要找到里面一个Url对应Controller相关代码，可能会有点晕。因为[NopCommerce](http://www.lanhusoft.com/Article/349.html)为了对[seo](http://www.lanhuseo.com/)友好，对其Url做了一些处理，自定义了路由规则，同时为了支持插件机制，加了一些自己的类进行扩展。本文就来分析一个NopCommerce路由相关源码设计思路，同样我们也先来看看相关的类图：



上面就是NopCommerce路由相关功能主要的类、接口及关系。有以下类：

1、[mvc](http://www.lanhusoft.com/Article/169.html)Application

2、IRoutePublisher、RoutePublisher

3、IRouteProvider、RouteProvider、GenericUrlRouteProvider

其中RoutePublisher是用来发布RouteProvider、GenericUrlRouteProvider里面配置的路由规则的，二者都有一个抽象的接口。接下来我们就来看看这些类或接口中代码是如何实现的：

**Nop.Web.MvcApplication**

1. public static void RegisterRoutes(RouteCollection routes)
2. {
3. routes.IgnoreRoute("favicon.ico");
4. routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");
6. //注册自定义的路由规则及插件相关路由
7. var routePublisher = EngineContext.Current.Resolve<IRoutePublisher>();
8. routePublisher.RegisterRoutes(routes);
10. routes.MapRoute(
11. "Default", // Route name
12. "{controller}/{action}/{id}", // URL with parameters
13. new { controller = "Home", action = "Index", id = UrlParameter.Optional },
14. new[] { "Nop.Web.Controllers" }
15. );
16. }

EngineContext.Current.Resolve<IRoutePublisher>()这名代码表示，从Ioc容器Autofac中获取接口IRoutePublisher所依赖的具体实现类。如果对Nop的[依赖注入](http://www.lanhusoft.com/Article/105.html)原理还是很了解，可以参考我之前的文章[NopCommerce源码架构详解--Autofac依赖注入分析](http://www.lanhusoft.com/Article/327.html)。

在Nop.Web.Framework.DependencyRegistrar中有以下代码：

1. builder.RegisterType<RoutePublisher>().As<IRoutePublisher>().SingleInstance();

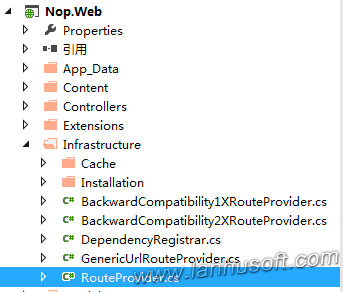
**Nop.Web.Framework.Mvc.Routes.RoutePublisher**

我们找到这个RoutePublisher类（Nop.Web.Framework.Mvc.Routes.RoutePublisher）其中关键的代码如下：

1. public virtual void RegisterRoutes(RouteCollection routes)
2. {
3. //通过typeFinder找出所有（包括插件）实现了接口IRouteProvider相关的类型
4. var routeProviderTypes = typeFinder.FindClassesOfType<IRouteProvider>();
5. var routeProviders = new List<IRouteProvider>();
6. foreach (var providerType in routeProviderTypes)
7. {
8. //Ignore not installed plugins
9. var plugin = FindPlugin(providerType);
10. if (plugin != null && !plugin.Installed)
11. continue;
12. //采用[反射](http://www.lanhusoft.com/Article/156.html)动态创建IRouteProvider的具体类的实例
13. var provider = Activator.CreateInstance(providerType) as IRouteProvider;
14. routeProviders.Add(provider);
15. }
16. routeProviders = routeProviders.OrderByDescending(rp => rp.Priority).ToList();
17. //依次调用RouteProvider的RegisterRoutes方法，注册路由规则
18. routeProviders.ForEach(rp => rp.RegisterRoutes(routes));
19. }

**Nop.Web.Infrastructure.RouteProvider**

现在我们来看看一个具体的RouteProvider里面都有些什么东东。在项目Nop.Web根目录下面有一个文件夹Infrastructure，里面有一个RouteProvider类，如下图：



1. public void RegisterRoutes(RouteCollection routes)
2. {
3. //We reordered our routes so the most used ones are on top. It can improve performance.
5. //home page
6. routes.MapLocalizedRoute("HomePage",
7. "",
8. new { controller = "Home", action = "Index" },
9. new[] { "Nop.Web.Controllers" });
11. //widgets
12. //we have this route for performance optimization because named routes are MUCH faster than usual Html.Action(...)
13. //and this route is highly used
14. routes.MapRoute("WidgetsByZone",
15. "widgetsbyzone/",
16. new { controller = "Widget", action = "WidgetsByZone" },
17. new[] { "Nop.Web.Controllers" });
19. //login
20. routes.MapLocalizedRoute("Login",
21. "login/",
22. new { controller = "Customer", action = "Login" },
23. new[] { "Nop.Web.Controllers" });
24. //register
25. routes.MapLocalizedRoute("Register",
26. "register/",
27. new { controller = "Customer", action = "Register" },
28. new[] { "Nop.Web.Controllers" });
29. //logout
30. routes.MapLocalizedRoute("Logout",
31. "logout/",
32. new { controller = "Customer", action = "Logout" },
33. new[] { "Nop.Web.Controllers" });
35. //shopping cart
36. routes.MapLocalizedRoute("ShoppingCart",
37. "cart/",
38. new { controller = "ShoppingCart", action = "Cart" },
39. new[] { "Nop.Web.Controllers" });
40. //wishlist
41. routes.MapLocalizedRoute("Wishlist",
42. "wishlist/{customerGuid}",
43. new { controller = "ShoppingCart", action = "Wishlist", customerGuid = UrlParameter.Optional },
44. new[] { "Nop.Web.Controllers" });
46. //customer
47. routes.MapLocalizedRoute("CustomerInfo",
48. "customer/info",
49. new { controller = "Customer", action = "Info" },
50. new[] { "Nop.Web.Controllers" });
51. //....省略剩余代码
52. }

RouteProvider的方法RegisterRoutes就是真正自定义路由规则。我们如果要找一个Url对应的Controller就要先在这里面查找一下，才好定位到是哪一个Controller。至于插件的路由我接下来会用专门一篇文章来介绍Nop的插件机制。

Nop.Web.Infrastructure.GenericUrlRouteProvider

GenericUrlRouteProvider和Nop.Web.Infrastructure.RouteProvider是相同级别的都是实现了接口IRouteProvider，区别GenericUrlRouteProvider定义的一般公用的Url规则。如下代码：

1. public partial class GenericUrlRouteProvider : IRouteProvider
2. {
3. public void RegisterRoutes(RouteCollection routes)
4. {
5. //generic URLs
6. routes.MapGenericPathRoute("GenericUrl",
7. "{generic\_se\_name}",
8. new {controller = "Common", action = "GenericUrl"},
9. new[] {"Nop.Web.Controllers"});
11. //define this routes to use in UI views (in case if you want to customize some of them later)
12. routes.MapLocalizedRoute("Product",
13. "{SeName}",
14. new { controller = "Product", action = "ProductDetails" },
15. new[] {"Nop.Web.Controllers"});
17. routes.MapLocalizedRoute("Category",
18. "{SeName}",
19. new { controller = "Catalog", action = "Category" },
20. new[] { "Nop.Web.Controllers" });
22. routes.MapLocalizedRoute("Manufacturer",
23. "{SeName}",
24. new { controller = "Catalog", action = "Manufacturer" },
25. new[] { "Nop.Web.Controllers" });
27. routes.MapLocalizedRoute("Vendor",
28. "{SeName}",
29. new { controller = "Catalog", action = "Vendor" },
30. new[] { "Nop.Web.Controllers" });
32. routes.MapLocalizedRoute("NewsItem",
33. "{SeName}",
34. new { controller = "News", action = "NewsItem" },
35. new[] { "Nop.Web.Controllers" });
37. routes.MapLocalizedRoute("BlogPost",
38. "{SeName}",
39. new { controller = "Blog", action = "BlogPost" },
40. new[] { "Nop.Web.Controllers" });
42. routes.MapLocalizedRoute("Topic",
43. "{SeName}",
44. new { controller = "Topic", action = "TopicDetails" },
45. new[] { "Nop.Web.Controllers" });
46. }
48. public int Priority
49. {
50. get
51. {
52. //it should be the last route
53. //we do not set it to -int.MaxValue so it could be overriden (if required)
54. return -1000000;
55. }
56. }
57. }

可以看到上面定义了商品列表、商品详情及新闻等相关的Url规则。

（4）NopCommerce源码架构详解--单例模式实现源码分析

单例模式是是常用经典十几种[设计模式](http://www.lanhusoft.com/Article/413.html)中最简单的。.NET中单例模式的实现也有很多种方式。下面我来介绍一下[NopCommerce](http://www.lanhusoft.com/Article/349.html)中单例模式实现。

我之前的文章就分析了一下[nop](http://www.lanhusoft.com/Article/349.html)中EngineContext的实现。EngineContext是把一个Web请求用Nop的EngineContext引擎上下文封装。里面提供了一个IEngine的单例对象的访问方式。

下面就是EngineContext的源码：

## 一、EngineContext

1. using System;
2. using System.Configuration;
3. using System.Runtime.CompilerServices;
4. using Nop.Core.Configuration;
6. namespace Nop.Core.Infrastructure
7. {
8. public class EngineContext
9. {
10. #region Utilities
12. /// <summary>
13. /// 通过配置信息创建一个相应IEngine实例
14. /// </summary>
15. /// <param name="config">Config</param>
16. /// <returns>New engine instance</returns>
17. protected static IEngine CreateEngineInstance(NopConfig config)
18. {
19. if (config != null && !string.IsNullOrEmpty(config.EngineType))
20. {
21. var engineType = Type.GetType(config.EngineType);
22. if (engineType == null)
23. throw new ConfigurationErrorsException("The type '" + config.EngineType + "' could not be found. Please check the configuration at /configuration/nop/engine[@engineType] or check for missing assemblies.");
24. if (!typeof(IEngine).IsAssignableFrom(engineType))
25. throw new ConfigurationErrorsException("The type '" + engineType + "' doesn't implement 'Nop.Core.Infrastructure.IEngine' and cannot be configured in /configuration/nop/engine[@engineType] for that purpose.");
26. return Activator.CreateInstance(engineType) as IEngine;
27. }
29. return new NopEngine();
30. }
32. #endregion
34. #region Methods
36. /// <summary>
37. /// Initializes a static instance of the Nop factory.初始化创建一个IEngine的实例
38. /// </summary>
39. /// <param name="forceRecreate">Creates a new factory instance even though the factory has been previously initialized.</param>
40. [MethodImpl(MethodImplOptions.Synchronized)]
41. public static IEngine Initialize(bool forceRecreate)
42. {
43. if (Singleton<IEngine>.Instance == null || forceRecreate)
44. {
45. var config = ConfigurationManager.GetSection("NopConfig") as NopConfig;
46. Singleton<IEngine>.Instance = CreateEngineInstance(config);
47. Singleton<IEngine>.Instance.Initialize(config);
48. }
49. return Singleton<IEngine>.Instance;
50. }
52. /// <summary>
53. /// 替换单例中的实例对象
54. /// </summary>
55. /// <param name="engine">The engine to use.</param>
56. /// <remarks>Only use this method if you know what you're doing.</remarks>
57. public static void Replace(IEngine engine)
58. {
59. Singleton<IEngine>.Instance = engine;
60. }
62. #endregion
64. /// <summary>
65. /// 获取IEngine单例实例对象
66. /// </summary>
67. public static IEngine Current
68. {
69. get
70. {
71. if (Singleton<IEngine>.Instance == null)
72. {
73. Initialize(false);
74. }
75. return Singleton<IEngine>.Instance;
76. }
77. }
79. }
80. }

上面Initialize方法使用[MethodImpl(MethodImplOptions.Synchronized)]声明，就保证只能有一个线程访问，因为.NET的Web程序无论是WebForm还是[mvc](http://www.lanhusoft.com/Article/169.html)都在服务端都是[多线程](http://www.lanhusoft.com/Article/33.html)的。这样就标记只能有一个线程调用Initialize方法，也就保证了实例对象IEngine的在内存中只有一份。然后把单例实例对象的存储到类Singleton中。Singleton就像是一个对象容器，可以把许多单例实例对象存储在里面。

下面我们来看看实例Singleton的实现思路。

## 二、Singleton

1. using System;
2. using System.Collections.Generic;
4. namespace Nop.Core.Infrastructure
5. {
7. public class Singleton<T> : Singleton
8. {
9. static T instance;
11. public static T Instance
12. {
13. get { return instance; }
14. set
15. {
16. instance = value;
17. AllSingletons[typeof(T)] = value;
18. }
19. }
20. }

23. public class SingletonList<T> : Singleton<IList<T>>
24. {
25. static SingletonList()
26. {
27. Singleton<IList<T>>.Instance = new List<T>();
28. }
30. public new static IList<T> Instance
31. {
32. get { return Singleton<IList<T>>.Instance; }
33. }
34. }

37. public class SingletonDictionary<TKey, TValue> : Singleton<IDictionary<TKey, TValue>>
38. {
39. static SingletonDictionary()
40. {
41. Singleton<Dictionary<TKey, TValue>>.Instance = new Dictionary<TKey, TValue>();
42. }
44. public new static IDictionary<TKey, TValue> Instance
45. {
46. get { return Singleton<Dictionary<TKey, TValue>>.Instance; }
47. }
48. }

51. public class Singleton
52. {
53. static Singleton()
54. {
55. allSingletons = new Dictionary<Type, object>();
56. }
58. static readonly IDictionary<Type, object> allSingletons;
60. public static IDictionary<Type, object> AllSingletons
61. {
62. get { return allSingletons; }
63. }
64. }
65. }

Singleton类里面用一个Dictionary<Type, object>()集合来存储所有的单例对象。基于Singleton类创建一些泛型类Singleton<T>，Singleton<IList<T>>，SingletonList<T>，Singleton<IDictionary<TKey, TValue>>和SingletonDictionary<TKey, TValue>。

（5）NopCommerce源码架构详解--EF相关Fluent API实现源码分析一

今天继续来写[NopCommerce](http://www.lanhusoft.com/Article/349.html)源码架构详解相关的文章，我们都知道[nop](http://www.lanhusoft.com/Article/349.html)中使用的是ORM框架是EF，并且是EF中的Fluent API形式。下面我就来分析一下里面的核心类及其实现原理。

## 一、实体相关的基础类

项目：Nop.Core

**实体的抽象类BaseEntity**

1. using System;
2. namespace Nop.Core
3. {
4. /// <summary>
5. /// 所有entities的基类
6. /// </summary>
7. public abstract partial class BaseEntity
8. {
9. /// <summary>
10. /// Id为实体的主键
11. /// </summary>
12. public int Id { get; set; }
13. public override bool Equals(object obj)
14. {
15. return Equals(obj as BaseEntity);
16. }
17. private static bool IsTransient(BaseEntity obj)
18. {
19. return obj != null && Equals(obj.Id, default(int));
20. }
21. private Type GetUnproxiedType()
22. {
23. return GetType();
24. }
25. public virtual bool Equals(BaseEntity other)
26. {
27. if (other == null)
28. return false;
29. if (ReferenceEquals(this, other))
30. return true;
31. if (!IsTransient(this) &&
32. !IsTransient(other) &&
33. Equals(Id, other.Id))
34. {
35. var otherType = other.GetUnproxiedType();
36. var thisType = GetUnproxiedType();
37. return thisType.IsAssignableFrom(otherType) ||
38. otherType.IsAssignableFrom(thisType);
39. }
40. return false;
41. }
42. public override int GetHashCode()
43. {
44. if (Equals(Id, default(int)))
45. return base.GetHashCode();
46. return Id.GetHashCode();
47. }
48. public static bool operator ==(BaseEntity x, BaseEntity y)
49. {
50. return Equals(x, y);//运算符的重载
51. }
52. public static bool operator !=(BaseEntity x, BaseEntity y)
53. {
54. return !(x == y);
55. }
56. }
57. }

项目Nop.Data

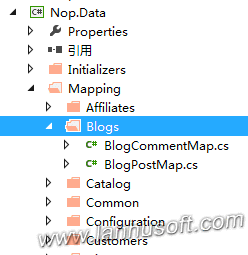
**使用EF的泛型Repository类EfRepository<T>，里面封装了一些公共的数据操作方法。**

1. using System;
2. using System.Data.Entity;
3. using System.Data.Entity.Validation;
4. using System.[linq](http://www.lanhusoft.com/Article/163.html);
5. using Nop.Core;
6. using Nop.Core.Data;
8. namespace Nop.Data
9. {
10. /// <summary>
11. /// Entity Framework repository
12. /// </summary>
13. public partial class EfRepository<T> : IRepository<T> where T : BaseEntity
14. {
15. private readonly IDbContext \_context;
16. private IDbSet<T> \_entities;
18. /// <summary>
19. /// Ctor
20. /// </summary>
21. /// <param name="context">Object context</param>
22. public EfRepository(IDbContext context)
23. {
24. this.\_context = context;
25. }
27. /// <summary>
28. /// Get entity by identifier
29. /// </summary>
30. /// <param name="id">Identifier</param>
31. /// <returns>Entity</returns>
32. public virtual T GetById(object id)
33. {
34. return this.Entities.Find(id);
35. }
37. /// <summary>
38. /// 插入一个entity
39. /// </summary>
40. /// <param name="entity">Entity</param>
41. public virtual void Insert(T entity)
42. {
43. try
44. {
45. if (entity == null)
46. throw new ArgumentNullException("entity");
48. this.Entities.Add(entity);
50. this.\_context.SaveChanges();
51. }
52. catch (DbEntityValidationException dbEx)
53. {
54. var msg = string.Empty;
56. foreach (var validationErrors in dbEx.EntityValidationErrors)
57. foreach (var validationError in validationErrors.ValidationErrors)
58. msg += string.Format("Property: {0} Error: {1}", validationError.PropertyName, validationError.ErrorMessage) + Environment.NewLine;
60. var fail = new Exception(msg, dbEx);
61. //Debug.WriteLine(fail.Message, fail);
62. throw fail;
63. }
64. }
66. /// <summary>
67. /// 更新一个entity
68. /// </summary>
69. /// <param name="entity">Entity</param>
70. public virtual void Update(T entity)
71. {
72. try
73. {
74. if (entity == null)
75. throw new ArgumentNullException("entity");
77. this.\_context.SaveChanges();
78. }
79. catch (DbEntityValidationException dbEx)
80. {
81. var msg = string.Empty;
83. foreach (var validationErrors in dbEx.EntityValidationErrors)
84. foreach (var validationError in validationErrors.ValidationErrors)
85. msg += Environment.NewLine + string.Format("Property: {0} Error: {1}", validationError.PropertyName, validationError.ErrorMessage);
87. var fail = new Exception(msg, dbEx);
88. //Debug.WriteLine(fail.Message, fail);
89. throw fail;
90. }
91. }
93. /// <summary>
94. /// 删除一个 entity
95. /// </summary>
96. /// <param name="entity">Entity</param>
97. public virtual void Delete(T entity)
98. {
99. try
100. {
101. if (entity == null)
102. throw new ArgumentNullException("entity");
104. this.Entities.Remove(entity);
106. this.\_context.SaveChanges();
107. }
108. catch (DbEntityValidationException dbEx)
109. {
110. var msg = string.Empty;
112. foreach (var validationErrors in dbEx.EntityValidationErrors)
113. foreach (var validationError in validationErrors.ValidationErrors)
114. msg += Environment.NewLine + string.Format("Property: {0} Error: {1}", validationError.PropertyName, validationError.ErrorMessage);
116. var fail = new Exception(msg, dbEx);
117. //Debug.WriteLine(fail.Message, fail);
118. throw fail;
119. }
120. }
122. /// <summary>
123. /// 返回实体的集合
124. /// </summary>
125. public virtual IQueryable<T> Table
126. {
127. get
128. {
129. return this.Entities;
130. }
131. }

134. /// <summary>
135. /// 采用no tracking的方式返回实体的集合，在只读情况下性能更好
136. /// </summary>
137. public virtual IQueryable<T> TableNoTracking
138. {
139. get
140. {
141. return this.Entities.AsNoTracking();
142. }
143. }

146. /// <summary>
147. /// Entities
148. /// </summary>
149. protected virtual IDbSet<T> Entities
150. {
151. get
152. {
153. if (\_entities == null)
154. \_entities = \_context.Set<T>();
155. return \_entities;
156. }
157. }
158. }
159. }

## 二、映射实现Mapping



Nop的实体映射到数据库是通过继承基类EntityTypeConfiguration<T>，手动配置映射关系。博客类BlogPost和评论BlogComment是通过下面方式实现的：

**BlogPost.cs**

1. using System;
2. using System.Collections.Generic;
3. using Nop.Core.Domain.Localization;
4. using Nop.Core.Domain.[seo](http://www.lanhuseo.com/);
5. using Nop.Core.Domain.Stores;
7. namespace Nop.Core.Domain.Blogs
8. {
9. /// <summary>
10. /// Represents a blog post
11. /// </summary>
12. public partial class BlogPost : BaseEntity, ISlugSupported, IStoreMappingSupported
13. {
14. private ICollection<BlogComment> \_blogComments;
16. /// <summary>
17. /// Gets or sets the language identifier
18. /// </summary>
19. public int LanguageId { get; set; }
21. /// <summary>
22. /// Gets or sets the blog post title
23. /// </summary>
24. public string Title { get; set; }
26. /// <summary>
27. /// Gets or sets the blog post title
28. /// </summary>
29. public string Body { get; set; }
31. /// <summary>
32. /// Gets or sets a value indicating whether the blog post comments are allowed
33. /// </summary>
34. public bool AllowComments { get; set; }
36. /// <summary>
37. /// Gets or sets the total number of comments
38. /// <remarks>
39. /// We use this property for performance optimization (no SQL command executed)
40. /// </remarks>
41. /// </summary>
42. public int CommentCount { get; set; }
44. /// <summary>
45. /// Gets or sets the blog tags
46. /// </summary>
47. public string Tags { get; set; }
49. /// <summary>
50. /// Gets or sets the blog post start date and time
51. /// </summary>
52. public DateTime? StartDateUtc { get; set; }
54. /// <summary>
55. /// Gets or sets the blog post end date and time
56. /// </summary>
57. public DateTime? EndDateUtc { get; set; }
59. /// <summary>
60. /// Gets or sets the meta keywords
61. /// </summary>
62. public string MetaKeywords { get; set; }
64. /// <summary>
65. /// Gets or sets the meta description
66. /// </summary>
67. public string MetaDescription { get; set; }
69. /// <summary>
70. /// Gets or sets the meta title
71. /// </summary>
72. public string MetaTitle { get; set; }
74. /// <summary>
75. /// Gets or sets a value indicating whether the entity is limited/restricted to certain stores
76. /// </summary>
77. public virtual bool LimitedToStores { get; set; }
79. /// <summary>
80. /// Gets or sets the date and time of entity creation
81. /// </summary>
82. public DateTime CreatedOnUtc { get; set; }
84. /// <summary>
85. /// Gets or sets the blog comments
86. /// </summary>
87. public virtual ICollection<BlogComment> BlogComments
88. {
89. get { return \_blogComments ?? (\_blogComments = new List<BlogComment>()); }
90. protected set { \_blogComments = value; }
91. }
93. /// <summary>
94. /// Gets or sets the language
95. /// </summary>
96. public virtual Language Language { get; set; }
97. }
98. }

**BlogComment.cs**

1. using System;
2. using Nop.Core.Domain.Customers;
4. namespace Nop.Core.Domain.Blogs
5. {
6. /// <summary>
7. /// Represents a blog comment
8. /// </summary>
9. public partial class BlogComment : BaseEntity
10. {
11. /// <summary>
12. /// Gets or sets the customer identifier
13. /// </summary>
14. public int CustomerId { get; set; }
16. /// <summary>
17. /// Gets or sets the comment text
18. /// </summary>
19. public string CommentText { get; set; }
21. /// <summary>
22. /// Gets or sets the blog post identifier
23. /// </summary>
24. public int BlogPostId { get; set; }
26. /// <summary>
27. /// Gets or sets the date and time of instance creation
28. /// </summary>
29. public DateTime CreatedOnUtc { get; set; }
31. /// <summary>
32. /// Gets or sets the customer
33. /// </summary>
34. public virtual Customer Customer { get; set; }
36. /// <summary>
37. /// Gets or sets the blog post
38. /// </summary>
39. public virtual BlogPost BlogPost { get; set; }
40. }
41. }

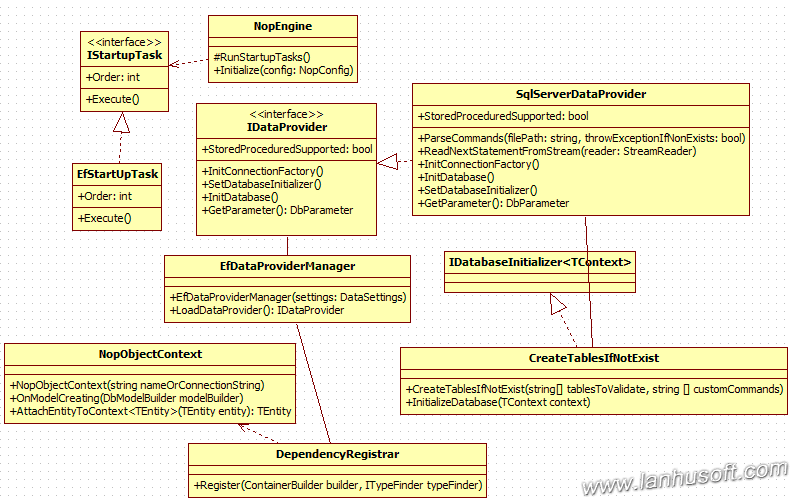
BlogPost和BlogComment一对多的关系配置。

**BlogCommentMap.cs**

1. using System.Data.Entity.ModelConfiguration;
2. using Nop.Core.Domain.Blogs;
4. namespace Nop.Data.Mapping.Blogs
5. {
6. public partial class BlogCommentMap : EntityTypeConfiguration<BlogComment>
7. {
8. public BlogCommentMap()
9. {
10. this.ToTable("BlogComment");
11. this.HasKey(pr => pr.Id);
13. this.HasRequired(bc => bc.BlogPost)
14. .WithMany(bp => bp.BlogComments)
15. .HasForeignKey(bc => bc.BlogPostId);
17. this.HasRequired(cc => cc.Customer)
18. .WithMany()
19. .HasForeignKey(cc => cc.CustomerId);
20. }
21. }
22. }

（6）NopCommerce源码架构详解--EF相关Fluent API实现源码分析二

前面一篇文章介绍了[nop](http://www.lanhusoft.com/Article/349.html)中EF的Fluent API实现的一些基本类，其中最关键的就是Map映射下面我来介绍EF的Fluent API中的基本类和Map实体映射这些是在Nop中怎么调用和生效的。同样的我们先来看看核心的类图。



## 主要类及作用：

1、NopEngine：Nop引擎注册所有的依赖（RegisterDependencies）及启动任务（RunStartupTasks），其中包括执行所有实现接口IStartupTask的类的Execute方法。

1. protected virtual void RunStartupTasks()
2. {
3. var typeFinder = \_containerManager.Resolve<ITypeFinder>();
4. var startUpTaskTypes = typeFinder.FindClassesOfType<IStartupTask>();
5. var startUpTasks = new List<IStartupTask>();
6. foreach (var startUpTaskType in startUpTaskTypes)
7. startUpTasks.Add((IStartupTask)Activator.CreateInstance(startUpTaskType));
8. //sort
9. startUpTasks = startUpTasks.AsQueryable().OrderBy(st => st.Order).ToList();
10. foreach (var startUpTask in startUpTasks)
11. startUpTask.Execute();
12. }
14. public void Initialize(NopConfig config)
15. {
16. //register dependencies
17. RegisterDependencies(config);
19. //startup tasks
20. if (!config.IgnoreStartupTasks)
21. {
22. RunStartupTasks();
23. }
24. }

2、SqlServerDataProvider：初始化数据库连接工厂及设置DatabaseInitializer。

1. using System;
2. using System.Collections.Generic;
3. using System.Data.Common;
4. using System.Data.Entity;
5. using System.Data.Entity.Infrastructure;
6. using System.Data.SqlClient;
7. using System.IO;
8. using System.Text;
9. using System.Web.Hosting;
10. using Nop.Core.Data;
11. using Nop.Data.Initializers;
13. namespace Nop.Data
14. {
15. public class SqlServerDataProvider : IDataProvider
16. {
17. #region Utilities
19. protected virtual string[] ParseCommands(string filePath, bool throwExceptionIfNonExists)
20. {
21. if (!File.Exists(filePath))
22. {
23. if (throwExceptionIfNonExists)
24. throw new ArgumentException(string.Format("Specified file doesn't exist - {0}", filePath));
25. else
26. return new string[0];
27. }

30. var statements = new List<string>();
31. using (var stream = File.OpenRead(filePath))
32. using (var reader = new StreamReader(stream))
33. {
34. var statement = "";
35. while ((statement = ReadNextStatementFromStream(reader)) != null)
36. {
37. statements.Add(statement);
38. }
39. }
41. return statements.ToArray();
42. }
44. protected virtual string ReadNextStatementFromStream(StreamReader reader)
45. {
46. var sb = new StringBuilder();
48. string lineOfText;
50. while (true)
51. {
52. lineOfText = reader.ReadLine();
53. if (lineOfText == null)
54. {
55. if (sb.Length > 0)
56. return sb.ToString();
57. else
58. return null;
59. }
61. if (lineOfText.TrimEnd().ToUpper() == "GO")
62. break;
64. sb.Append(lineOfText + Environment.NewLine);
65. }
67. return sb.ToString();
68. }
70. #endregion
72. #region Methods
74. /// <summary>
75. /// Initialize connection factory
76. /// </summary>
77. public virtual void InitConnectionFactory()
78. {
79. var connectionFactory = new SqlConnectionFactory();
80. //TODO fix compilation warning (below)
81. #pragma warning disable 0618
82. Database.DefaultConnectionFactory = connectionFactory;
83. }
85. /// <summary>
86. /// Initialize database
87. /// </summary>
88. public virtual void InitDatabase()
89. {
90. InitConnectionFactory();
91. SetDatabaseInitializer();
92. }
94. /// <summary>
95. /// Set database initializer
96. /// </summary>
97. public virtual void SetDatabaseInitializer()
98. {
99. //pass some table names to ensure that we have [NopCommerce](http://www.lanhusoft.com/Article/349.html) 2.X installed
100. var tablesToValidate = new[] { "Customer", "Discount", "Order", "Product", "ShoppingCartItem" };
102. //custom commands (stored proedures, indexes)
104. var customCommands = new List<string>();
105. //use webHelper.MapPath instead of HostingEnvironment.MapPath which is not available in unit tests
106. customCommands.AddRange(ParseCommands(HostingEnvironment.MapPath("~/App\_Data/SqlServer.Indexes.sql"), false));
107. //use webHelper.MapPath instead of HostingEnvironment.MapPath which is not available in unit tests
108. customCommands.AddRange(ParseCommands(HostingEnvironment.MapPath("~/App\_Data/SqlServer.StoredProcedures.sql"), false));
110. var initializer = new CreateTablesIfNotExist<NopObjectContext>(tablesToValidate, customCommands.ToArray());
111. Database.SetInitializer(initializer);
112. }
114. /// <summary>
115. /// A value indicating whether this data provider supports stored procedures
116. /// </summary>
117. public virtual bool StoredProceduredSupported
118. {
119. get { return true; }
120. }
122. /// <summary>
123. /// Gets a support database parameter object (used by stored procedures)
124. /// </summary>
125. /// <returns>Parameter</returns>
126. public virtual DbParameter GetParameter()
127. {
128. return new SqlParameter();
129. }
131. #endregion
132. }
133. }

3、EfStartUpTask：执行所依赖对象IDataProvider的方法SetDatabaseInitializer，EF对应数据库Provider的初始化。

1. using Nop.Core;
2. using Nop.Core.Data;
3. using Nop.Core.Infrastructure;
5. namespace Nop.Data
6. {
7. public class EfStartUpTask : IStartupTask
8. {
9. public void Execute()
10. {
11. var settings = EngineContext.Current.Resolve<DataSettings>();
12. if (settings != null && settings.IsValid())
13. {
14. var provider = EngineContext.Current.Resolve<IDataProvider>();
15. if (provider == null)
16. throw new NopException("No IDataProvider found");
17. provider.SetDatabaseInitializer();
18. }
19. }
21. public int Order
22. {
23. //ensure that this task is run first
24. get { return -1000; }
25. }
26. }
27. }

4、EfDataProviderManager：一个IDataProvider的工厂类，可以根据配置返回接口IDataProvider的不同实现的具体类。如：SqlServerDataProvider，SqlCeDataProvider等等。

1. using System;
2. using Nop.Core;
3. using Nop.Core.Data;
5. namespace Nop.Data
6. {
7. public partial class EfDataProviderManager : BaseDataProviderManager
8. {
9. public EfDataProviderManager(DataSettings settings):base(settings)
10. {
11. }
13. public override IDataProvider LoadDataProvider()
14. {
16. var providerName = Settings.DataProvider;
17. if (String.IsNullOrWhiteSpace(providerName))
18. throw new NopException("Data Settings doesn't contain a providerName");
20. switch (providerName.ToLowerInvariant())
21. {
22. case "sqlserver":
23. return new SqlServerDataProvider();
24. case "sqlce":
25. return new SqlCeDataProvider();
26. default:
27. throw new NopException(string.Format("Not supported dataprovider name: {0}", providerName));
28. }
29. }
31. }
32. }

5、NopObjectContext：这相类继承于DbContext, IDbContext通过重写OnModelCreating方法通过[反射](http://www.lanhusoft.com/Article/156.html)获取AppDomain下面所有的程序集，然后动态创建EntityTypeConfiguration的实例，通过EF提供的API把我们配置的所有实体Map映射加入到modelBuilder的配置中去。

1. using System;
2. using System.Collections.Generic;
3. using System.Data;
4. using System.Data.Common;
5. using System.Data.Entity;
6. using System.Data.Entity.Infrastructure;
7. using System.Data.Entity.ModelConfiguration;
8. using System.[linq](http://www.lanhusoft.com/Article/163.html);
9. using System.Reflection;
10. using Nop.Core;
12. namespace Nop.Data
13. {
14. public class NopObjectContext : DbContext, IDbContext
15. {
16. #region Ctor
18. public NopObjectContext(string nameOrConnectionString)
19. : base(nameOrConnectionString)
20. {
21. }
23. #endregion
25. #region Utilities
27. protected override void OnModelCreating(DbModelBuilder modelBuilder)
28. {
29. //获取AppDomain下面所有的程序集
30. var typesToRegister = Assembly.GetExecutingAssembly().GetTypes()
31. .Where(type => !String.IsNullOrEmpty(type.Namespace))
32. .Where(type => type.BaseType != null && type.BaseType.IsGenericType && type.BaseType.GetGenericTypeDefinition() == typeof(EntityTypeConfiguration<>));
33. foreach (var type in typesToRegister)
34. {
35. //动态创建EntityTypeConfiguration的实例，并把我们配置的实体Map映射加入到modelBuilder的配置中去。
36. [dynamic](http://www.lanhusoft.com/Article/67.html) configurationInstance = Activator.CreateInstance(type);
37. modelBuilder.Configurations.Add(configurationInstance);
38. }
39. //...or do it manually below. For example,
40. //modelBuilder.Configurations.Add(new LanguageMap());
42. base.OnModelCreating(modelBuilder);
43. }

46. /// <summary>
47. /// Attach an entity to the context or return an already attached entity (if it was already attached)
48. /// </summary>
49. /// <typeparam name="TEntity">TEntity</typeparam>
50. /// <param name="entity">Entity</param>
51. /// <returns>Attached entity</returns>
52. protected virtual TEntity AttachEntityToContext<TEntity>(TEntity entity) where TEntity : BaseEntity, new()
53. {
54. //little hack here until Entity Framework really supports stored procedures
55. //otherwise, navigation properties of loaded entities are not loaded until an entity is attached to the context
56. var alreadyAttached = Set<TEntity>().Local.FirstOrDefault(x => x.Id == entity.Id);
57. if (alreadyAttached == null)
58. {
59. //attach new entity
60. Set<TEntity>().Attach(entity);
61. return entity;
62. }
63. else
64. {
65. //entity is already loaded.
66. return alreadyAttached;
67. }
68. }
70. #endregion
72. //...其实的一些共用方法
73. }
74. }

6、DependencyRegistrar：[依赖注入](http://www.lanhusoft.com/Article/105.html)相关类的注册，其中就包括了IDbContext用类NopObjectContext来注册。

1. Nop.Web.Framework.DependencyRegistrar : IDependencyRegistrar
2. public virtual void Register(ContainerBuilder builder, ITypeFinder typeFinder)
3. {
4. /....
5. builder.Register(x => x.Resolve<BaseDataProviderManager>().LoadDataProvider()).As<IDataProvider>().InstancePerDependency();
7. if (dataProviderSettings != null && dataProviderSettings.IsValid())
8. {
9. var efDataProviderManager = new EfDataProviderManager(dataSettingsManager.LoadSettings());
10. var dataProvider = efDataProviderManager.LoadDataProvider();
11. dataProvider.InitConnectionFactory();
13. builder.Register<IDbContext>(c => new NopObjectContext(dataProviderSettings.DataConnectionString)).InstancePerLifetimeScope();
14. }
15. else
16. {
17. builder.Register<IDbContext>(c => new NopObjectContext(dataSettingsManager.LoadSettings().DataConnectionString)).InstancePerLifetimeScope();
18. }
19. }

（7）NopCommerce源码架构详解--使用FluentValidation创建Model自定义Validator验证源码分析（公司似乎用的另一套参数验证，说是彪哥写的）

一个健全的系统少不了大量的输入合法性验证，我之前写[mvc](http://www.lanhusoft.com/Article/169.html)教程专题就专门用了两篇文章来介绍MVC中的Model验证。分别是：

[跟蓝狐学MVC教程--详解ASP.NET MVC Model验证](http://www.lanhusoft.com/Article/254.html)

[跟蓝狐学MVC教程--深入ASP.NET MVC Model验证-使用.NET验证特性类指定Mode验证规则](http://www.lanhusoft.com/Article/255.html)

但是[nop](http://www.lanhusoft.com/Article/349.html)中的验证使用的是一个优秀的开源的验证框架。它的好处是，它可以采用[lambda](http://www.lanhusoft.com/Article/168.html)表达式的形式配置要验证的字段和错误信息。今天我们就来看看Nop中是怎么使用FluentValidation创建Model自定义Validator验证的。

FluentValidation的官方地址：<https://github.com/JeremySkinner/FluentValidation>

当然你也可以在VS中用[nuget](http://www.lanhusoft.com/Article/185.html)快速获取安装到你的项目中。

我们可以在Nop的解决方案中的三个项目中都能找到一个叫做Validators的文件夹。



里面就放的使用FluentValidation的API配置的验证器。这三个项目分别为Nop.Admin，Nop.Web，Nop.Web.Framework。

下面我们就找一个项目中Nop.Admin的一个Validator看看具体FluentValidation是怎么使用的。

1. using FluentValidation;
2. using Nop.Admin.Models.Blogs;
3. using Nop.Services.Localization;
5. namespace Nop.Admin.Validators.Blogs
6. {
7. public class BlogPostValidator : AbstractValidator<BlogPostModel>
8. {
9. public BlogPostValidator(ILocalizationService localizationService)
10. {
11. RuleFor(x => x.Title)
12. .NotEmpty()
13. .WithMessage(localizationService.GetResource("Admin.ContentManagement.Blog.BlogPosts.Fields.Title.Required"));
15. RuleFor(x => x.Body)
16. .NotEmpty()
17. .WithMessage(localizationService.GetResource("Admin.ContentManagement.Blog.BlogPosts.Fields.Body.Required"));
18. }
19. }
20. }

上面这个BlogPostValidator是管理后台项目的博客文章验证，我们可以看到里面是通过RuleFor来定义验证规则的，里面一共定义有两条验证规则，分别是标题Title和内容Body不能为空。当验证不通过时就会输出用.WithMessage包含的错误提示信息。

要使上面配置的验证Validator生效，我们还需要更改Global.asax：

1. protected void Application\_Start()
2. {
3. //省略其它代码...
4. //fluent validation
5. DataAnnotationsModelValidatorProvider.AddImplicitRequiredAttributeForValueTypes = false;
6. ModelValidatorProviders.Providers.Add(new FluentValidationModelValidatorProvider(new NopValidatorFactory()));
7. //省略其它代码...
8. }

上面FluentValidationModelValidatorProvider构造函数参数传的是一个重写了GetValidator方法的NopValidatorFactory。

1. using System;
2. using FluentValidation;
3. using FluentValidation.Attributes;
4. using Nop.Core.Infrastructure;
6. namespace Nop.Web.Framework
7. {
8. public class NopValidatorFactory : AttributedValidatorFactory
9. {
10. //private readonly InstanceCache \_cache = new InstanceCache();
11. public override IValidator GetValidator(Type type)
12. {
13. if (type != null)
14. {
15. var attribute = (ValidatorAttribute)Attribute.GetCustomAttribute(type, typeof(ValidatorAttribute));
16. if ((attribute != null) && (attribute.ValidatorType != null))
17. {
18. //validators can depend on some customer specific settings (such as working language)
19. //that's why we do not cache validators
20. //var instance = \_cache.GetOrCreateInstance(attribute.ValidatorType,
21. // x => EngineContext.Current.ContainerManager.ResolveUnregistered(x));
22. var instance = EngineContext.Current.ContainerManager.ResolveUnregistered(attribute.ValidatorType);
23. return instance as IValidator;
24. }
25. }
26. return null;
28. }
29. }
30. }

Nop除了写了很多Model的Validator，它还有一些属性验证。如验证信用卡号的CreditCardPropertyValidator：

1. using System;
2. using System.[linq](http://www.lanhusoft.com/Article/163.html);
3. using FluentValidation.Validators;
5. namespace Nop.Web.Framework.Validators
6. {
7. public class CreditCardPropertyValidator : PropertyValidator
8. {
9. public CreditCardPropertyValidator()
10. : base("Credit card number is not valid")
11. {
13. }
15. protected override bool IsValid(PropertyValidatorContext context)
16. {
17. var ccValue = context.PropertyValue as string;
18. if (String.IsNullOrWhiteSpace(ccValue))
19. return false;
21. ccValue = ccValue.Replace(" ", "");
22. ccValue = ccValue.Replace("-", "");
24. int checksum = 0;
25. bool evenDigit = false;
27. //http://www.beachnet.com/~hstiles/cardtype.html
28. foreach (char digit in ccValue.Reverse())
29. {
30. if (!Char.IsDigit(digit))
31. return false;
33. int digitValue = (digit - '0') \* (evenDigit ? 2 : 1);
34. evenDigit = !evenDigit;
36. while (digitValue > 0)
37. {
38. checksum += digitValue % 10;
39. digitValue /= 10;
40. }
41. }
43. return (checksum % 10) == 0;
44. }
45. }
46. }

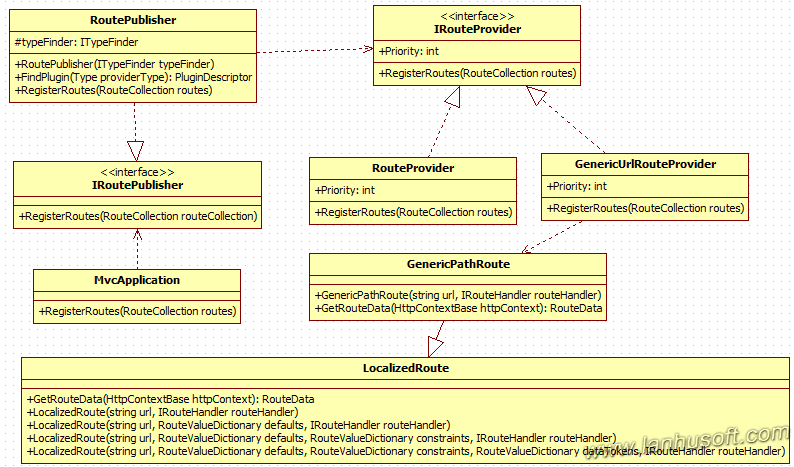
属性验证器继承于PropertyValidator，把验证逻辑写在方法IsValid中。接下来Nop写了一个IRuleBuilder的[扩展方法](http://www.lanhusoft.com/Article/27.html)IsCreditCard来使用这个验证信用卡号的Validator：

1. using FluentValidation;
3. namespace Nop.Web.Framework.Validators
4. {
5. public static class MyValidatorExtensions
6. {
7. public static IRuleBuilderOptions<T, string> IsCreditCard<T>(this IRuleBuilder<T, string> ruleBuilder)
8. {
9. return ruleBuilder.SetValidator(new CreditCardPropertyValidator());
10. }
11. }
12. }

（8）NopCommerce源码架构详解--对seo友好Url的路由机制实现源码分析

可能你刚开始看[nop](http://www.lanhusoft.com/Article/349.html)源码不太清楚一个Url对应具体的Controller是哪一个，因为Nop自身用了对[seo](http://www.lanhuseo.com/)友好的Url，它对路由进行了一些重写。我希望同学们通过我的这个文章对Nop路由有更深入的了解，以后也可以通过借鉴Nop的思路自己实现一个对SEO友好的Url路由。

相关类的类图如下：



下面是相关功能主要类：

1、Nop.Web.Framework.[mvc](http://www.lanhusoft.com/Article/169.html).Routes.IRoutePublisher和IRouteProvider，注册路由的共用接口。

IRoutePublisher：

1. public interface IRoutePublisher
2. {
3. /// <summary>
4. /// Register routes
5. /// </summary>
6. /// <param name="routes">Routes</param>
7. void RegisterRoutes(RouteCollection routeCollection);
8. }

IRouteProvider：

1. public interface IRouteProvider
2. {
3. void RegisterRoutes(RouteCollection routes);
5. int Priority { get; }
6. }

2、RouteProvider，实现接口IRouteProvider，注册一些核心路由规则，如首页、登录、注册、购物车等等。

1. using System.Web.Mvc;
2. using System.Web.Routing;
3. using Nop.Web.Framework.Localization;
4. using Nop.Web.Framework.Mvc.Routes;
6. namespace Nop.Web.Infrastructure
7. {
8. public partial class RouteProvider : IRouteProvider
9. {
10. public void RegisterRoutes(RouteCollection routes)
11. {
12. //We reordered our routes so the most used ones are on top. It can improve performance.
14. //home page
15. routes.MapLocalizedRoute("HomePage",
16. "",
17. new { controller = "Home", action = "Index" },
18. new[] { "Nop.Web.Controllers" });
20. //widgets
21. //we have this route for performance optimization because named routes are MUCH faster than usual Html.Action(...)
22. //and this route is highly used
23. routes.MapRoute("WidgetsByZone",
24. "widgetsbyzone/",
25. new { controller = "Widget", action = "WidgetsByZone" },
26. new[] { "Nop.Web.Controllers" });
28. //login
29. routes.MapLocalizedRoute("Login",
30. "login/",
31. new { controller = "Customer", action = "Login" },
32. new[] { "Nop.Web.Controllers" });
33. //register
34. routes.MapLocalizedRoute("Register",
35. "register/",
36. new { controller = "Customer", action = "Register" },
37. new[] { "Nop.Web.Controllers" });
38. //logout
39. routes.MapLocalizedRoute("Logout",
40. "logout/",
41. new { controller = "Customer", action = "Logout" },
42. new[] { "Nop.Web.Controllers" });
44. //shopping cart
45. routes.MapLocalizedRoute("ShoppingCart",
46. "cart/",
47. new { controller = "ShoppingCart", action = "Cart" },
48. new[] { "Nop.Web.Controllers" });
49. //wishlist
50. routes.MapLocalizedRoute("Wishlist",
51. "wishlist/{customerGuid}",
52. new { controller = "ShoppingCart", action = "Wishlist", customerGuid = UrlParameter.Optional },
53. new[] { "Nop.Web.Controllers" });
55. //customer
56. routes.MapLocalizedRoute("CustomerInfo",
57. "customer/info",
58. new { controller = "Customer", action = "Info" },
59. new[] { "Nop.Web.Controllers" });
60. routes.MapLocalizedRoute("CustomerAddresses",
61. "customer/addresses",
62. new { controller = "Customer", action = "Addresses" },
63. new[] { "Nop.Web.Controllers" });
64. routes.MapLocalizedRoute("CustomerOrders",
65. "customer/orders",
66. new { controller = "Customer", action = "Orders" },
67. new[] { "Nop.Web.Controllers" });
68. routes.MapLocalizedRoute("CustomerReturnRequests",
69. "customer/returnrequests",
70. new { controller = "Customer", action = "ReturnRequests" },
71. new[] { "Nop.Web.Controllers" });
72. routes.MapLocalizedRoute("CustomerDownloadableProducts",
73. "customer/downloadableproducts",
74. new { controller = "Customer", action = "DownloadableProducts" },
75. new[] { "Nop.Web.Controllers" });
77. //省略其它路由注册....
79. //page not found
80. routes.MapLocalizedRoute("PageNotFound",
81. "page-not-found",
82. new { controller = "Common", action = "PageNotFound" },
83. new[] { "Nop.Web.Controllers" });
84. }
86. public int Priority
87. {
88. get
89. {
90. return 0;
91. }
92. }
93. }
94. }

3、Nop.Web.Infrastructure.GenericUrlRouteProvider，同样的实现接口IRoutePublisher。这个RouteProvider定义了一些对SEO友好的路由。

1. using System.Web.Routing;
2. using Nop.Web.Framework.Localization;
3. using Nop.Web.Framework.Mvc.Routes;
4. using Nop.Web.Framework.Seo;
6. namespace Nop.Web.Infrastructure
7. {
8. public partial class GenericUrlRouteProvider : IRouteProvider
9. {
10. public void RegisterRoutes(RouteCollection routes)
11. {
12. //generic URLs
13. routes.MapGenericPathRoute("GenericUrl",
14. "{generic\_se\_name}",
15. new {controller = "Common", action = "GenericUrl"},
16. new[] {"Nop.Web.Controllers"});
18. //define this routes to use in UI views (in case if you want to customize some of them later)
19. routes.MapLocalizedRoute("Product",
20. "{SeName}",
21. new { controller = "Product", action = "ProductDetails" },
22. new[] {"Nop.Web.Controllers"});
24. routes.MapLocalizedRoute("Category",
25. "{SeName}",
26. new { controller = "Catalog", action = "Category" },
27. new[] { "Nop.Web.Controllers" });
29. routes.MapLocalizedRoute("Manufacturer",
30. "{SeName}",
31. new { controller = "Catalog", action = "Manufacturer" },
32. new[] { "Nop.Web.Controllers" });
34. routes.MapLocalizedRoute("Vendor",
35. "{SeName}",
36. new { controller = "Catalog", action = "Vendor" },
37. new[] { "Nop.Web.Controllers" });
39. routes.MapLocalizedRoute("NewsItem",
40. "{SeName}",
41. new { controller = "News", action = "NewsItem" },
42. new[] { "Nop.Web.Controllers" });
44. routes.MapLocalizedRoute("BlogPost",
45. "{SeName}",
46. new { controller = "Blog", action = "BlogPost" },
47. new[] { "Nop.Web.Controllers" });
49. routes.MapLocalizedRoute("Topic",
50. "{SeName}",
51. new { controller = "Topic", action = "TopicDetails" },
52. new[] { "Nop.Web.Controllers" });
53. }
55. public int Priority
56. {
57. get
58. {
59. //it should be the last route
60. //we do not set it to -int.MaxValue so it could be overriden (if required)
61. return -1000000;
62. }
63. }
64. }
65. }

4、Nop.Web.Framework.Localization.LocalizedRoute，它采用基类System.Web.Routing.Route，为了实现路由本地化。它提供一些属性和方法为获取到真正的路由做准备。它重写了基类Route两方法，GetRouteData和GetVirtualPath。

1. public override RouteData GetRouteData(HttpContextBase httpContext)
2. {
3. if (DataSettingsHelper.DatabaseIsInstalled() && this.SeoFriendlyUrlsForLanguagesEnabled)
4. {
5. string virtualPath = httpContext.Request.AppRelativeCurrentExecutionFilePath;
6. string applicationPath = httpContext.Request.ApplicationPath;
7. if (virtualPath.IsLocalizedUrl(applicationPath, false))
8. {
9. string rawUrl = httpContext.Request.RawUrl;
10. var newVirtualPath = rawUrl.RemoveLanguageSeoCodeFromRawUrl(applicationPath);
11. if (string.IsNullOrEmpty(newVirtualPath))
12. newVirtualPath = "/";
13. newVirtualPath = newVirtualPath.RemoveApplicationPathFromRawUrl(applicationPath);
14. newVirtualPath = "~" + newVirtualPath;
15. httpContext.RewritePath(newVirtualPath, true);
16. }
17. }
18. RouteData data = base.GetRouteData(httpContext);
19. return data;
20. }

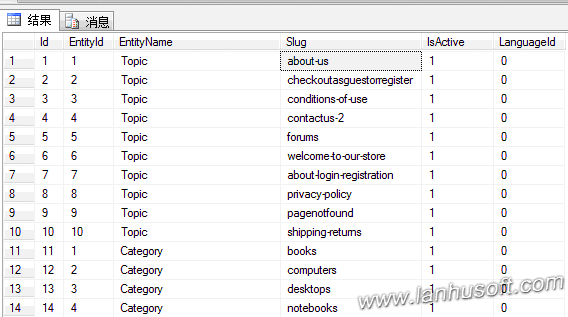
23. public override VirtualPathData GetVirtualPath(RequestContext requestContext, RouteValueDictionary values)
24. {
25. VirtualPathData data = base.GetVirtualPath(requestContext, values);
27. if (data != null && DataSettingsHelper.DatabaseIsInstalled() && this.SeoFriendlyUrlsForLanguagesEnabled)
28. {
29. string rawUrl = requestContext.HttpContext.Request.RawUrl;
30. string applicationPath = requestContext.HttpContext.Request.ApplicationPath;
31. if (rawUrl.IsLocalizedUrl(applicationPath, true))
32. {
33. data.VirtualPath = string.Concat(rawUrl.GetLanguageSeoCodeFromUrl(applicationPath, true), "/",
34. data.VirtualPath);
35. }
36. }
37. return data;
38. }

5、Nop.Web.Framework.Seo.GenericPathRoute，这个类是真正把友好的Url解析到我们在RouteProvider配置好的友好的路由规则。它继承了类Nop.Web.Framework.Localization.LocalizedRoute。

类GenericPathRoute核心代码如下：

1. public override RouteData GetRouteData(HttpContextBase httpContext)
2. {
3. RouteData data = base.GetRouteData(httpContext);
4. if (data != null && DataSettingsHelper.DatabaseIsInstalled())
5. {
6. var urlRecordService = EngineContext.Current.Resolve<IUrlRecordService>();
7. var slug = data.Values["generic\_se\_name"] as string;
8. //performance optimization.
9. //we load a cached verion here. it reduces number of SQL requests for each page load
10. var urlRecord = urlRecordService.GetBySlugCached(slug);//查询url对应的路由规则
11. //comment the line above and uncomment the line below in order to disable this performance "workaround"
12. //var urlRecord = urlRecordService.GetBySlug(slug);
13. if (urlRecord == null)
14. {
15. data.Values["controller"] = "Common";
16. data.Values["action"] = "PageNotFound";
17. return data;
18. }
19. //ensre that URL record is active
20. if (!urlRecord.IsActive)
21. {
22. //URL record is not active. let's find the latest one
23. var activeSlug = urlRecordService.GetActiveSlug(urlRecord.EntityId, urlRecord.EntityName, urlRecord.LanguageId);
24. if (!string.IsNullOrWhiteSpace(activeSlug))
25. {
26. //the active one is found
27. var webHelper = EngineContext.Current.Resolve<IWebHelper>();
28. var response = httpContext.Response;
29. response.Status = "301 Moved Permanently";
30. response.RedirectLocation = string.Format("{0}{1}", webHelper.GetStoreLocation(false), activeSlug);
31. response.End();
32. return null;
33. }
34. else
35. {
36. data.Values["controller"] = "Common";
37. data.Values["action"] = "PageNotFound";
38. return data;
39. }
40. }
42. //ensure that the slug is the same for the current language
43. //otherwise, it can cause some issues when customers choose a new language but a slug stays the same
44. var workContext = EngineContext.Current.Resolve<IWorkContext>();
45. var slugForCurrentLanguage = SeoExtensions.GetSeName(urlRecord.EntityId, urlRecord.EntityName, workContext.WorkingLanguage.Id);
46. if (!String.IsNullOrEmpty(slugForCurrentLanguage) &&
47. !slugForCurrentLanguage.Equals(slug, StringComparison.InvariantCultureIgnoreCase))
48. {
49. //we should make not null or "" validation above because some entities does not have SeName for standard (ID=0) language (e.g. news, blog posts)
50. var webHelper = EngineContext.Current.Resolve<IWebHelper>();
51. var response = httpContext.Response;
52. //response.Status = "302 Found";
53. response.Status = "302 Moved Temporarily";
54. response.RedirectLocation = string.Format("{0}{1}", webHelper.GetStoreLocation(false), slugForCurrentLanguage);
55. response.End();
56. return null;
57. }
59. //处理URL并动态赋值真正的Controller的相关信息
60. switch (urlRecord.EntityName.ToLowerInvariant())
61. {
62. case "product":
63. {
64. data.Values["controller"] = "Product";
65. data.Values["action"] = "ProductDetails";
66. data.Values["productid"] = urlRecord.EntityId;
67. data.Values["SeName"] = urlRecord.Slug;
68. }
69. break;
70. case "category":
71. {
72. data.Values["controller"] = "Catalog";
73. data.Values["action"] = "Category";
74. data.Values["categoryid"] = urlRecord.EntityId;
75. data.Values["SeName"] = urlRecord.Slug;
76. }
77. break;
78. case "manufacturer":
79. {
80. data.Values["controller"] = "Catalog";
81. data.Values["action"] = "Manufacturer";
82. data.Values["manufacturerid"] = urlRecord.EntityId;
83. data.Values["SeName"] = urlRecord.Slug;
84. }
85. break;
86. case "vendor":
87. {
88. data.Values["controller"] = "Catalog";
89. data.Values["action"] = "Vendor";
90. data.Values["vendorid"] = urlRecord.EntityId;
91. data.Values["SeName"] = urlRecord.Slug;
92. }
93. break;
94. case "newsitem":
95. {
96. data.Values["controller"] = "News";
97. data.Values["action"] = "NewsItem";
98. data.Values["newsItemId"] = urlRecord.EntityId;
99. data.Values["SeName"] = urlRecord.Slug;
100. }
101. break;
102. case "blogpost":
103. {
104. data.Values["controller"] = "Blog";
105. data.Values["action"] = "BlogPost";
106. data.Values["blogPostId"] = urlRecord.EntityId;
107. data.Values["SeName"] = urlRecord.Slug;
108. }
109. break;
110. case "topic":
111. {
112. data.Values["controller"] = "Topic";
113. data.Values["action"] = "TopicDetails";
114. data.Values["topicId"] = urlRecord.EntityId;
115. data.Values["SeName"] = urlRecord.Slug;
116. }
117. break;
118. default:
119. {
120. //no record found
122. //generate an event this way developers could insert their own types
123. EngineContext.Current.Resolve<IEventPublisher>()
124. .Publish(new CustomUrlRecordEntityNameRequested(data, urlRecord));
125. }
126. break;
127. }
128. }
129. return data;
130. }

可以看到上面通过获取路由中变量generic\_se\_name的值，然后通过这个值查询这个url对应的路由规则。Nop把这个对应信息存在表UrlRecord里面，如下图：



比如，我们在前台访问：http://localhost:15536/books，其实generic\_se\_name的值就为books，然后会找到字段Slug的值为books的记录。接着进行处理Url的Switch语句：

1. switch (urlRecord.EntityName.ToLowerInvariant())
2. {
3. //....省略其它代码
4. case "category":
5. {
6. data.Values["controller"] = "Catalog";
7. data.Values["action"] = "Category";
8. data.Values["categoryid"] = urlRecord.EntityId;
9. data.Values["SeName"] = urlRecord.Slug;
10. }
11. break;
12. //....省略其它代码
13. }

可以看到请求url:http://localhost:15536/books，真正执行的是Catalog中的Category方法。

6、RoutePublisher，实现接口IRoutePublisher，通过typeFinder.FindClassesOfType查找项目中所有实现了接口IRouteProvider的类，并依次注册其里面的路由。

1. public virtual void RegisterRoutes(RouteCollection routes)
2. {
3. var routeProviderTypes = typeFinder.FindClassesOfType<IRouteProvider>();
4. var routeProviders = new List<IRouteProvider>();
5. foreach (var providerType in routeProviderTypes)
6. {
7. //Ignore not installed plugins
8. var plugin = FindPlugin(providerType);
9. if (plugin != null && !plugin.Installed)
10. continue;
12. var provider = Activator.CreateInstance(providerType) as IRouteProvider;
13. routeProviders.Add(provider);
14. }
15. routeProviders = routeProviders.OrderByDescending(rp => rp.Priority).ToList();
16. routeProviders.ForEach(rp => rp.RegisterRoutes(routes));
17. }

在程序启动的时候就会注册路由，[依赖注入](http://www.lanhusoft.com/Article/105.html)在Nop.Web.Framework.DependencyRegistrar类中有下面的代码把接口IRoutePublisher用类RoutePublisher来注册：

1. builder.RegisterType<RoutePublisher>().As<IRoutePublisher>().SingleInstance();

最后在类MvcApplication中的会调用routePublisher注册所有路由规则到MVC框架中：

1. //register custom routes (plugins, etc)
2. var routePublisher = EngineContext.Current.Resolve<IRoutePublisher>();
3. routePublisher.RegisterRoutes(routes);

（9）NopCommerce源码架构详解--EF数据访问实例详解

今天我来分析一下[nop](http://www.lanhusoft.com/Article/349.html)里面怎么访问数据库，我们知道Nop里面的ORM是使用的EF并采用的仓储Repository模式。

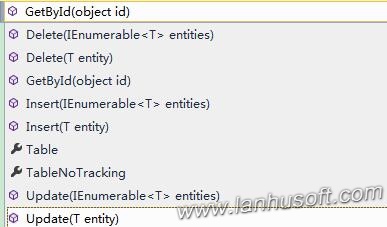
对Repository模式还不是很清楚的可以先学习一下我之前的文章：

[NopCommerce源码架构详解-MVC5使用EF6实现Repository模式相关源码分析](http://www.lanhusoft.com/Article/407.html)

## 一、IRepository<T>

首先，Nop定义了一个通用的泛型接口：

Nop.Core.Data.IRepository<T>



里面定义了常用的增、删、查、改等操作方法，但是由于是接口，只是方法定义没有实现。

声明成泛型的好处只用定义一些通用的方法，这样可以适用于所有的表。而不用每一个表单独定义一个类重复这些增、删、查、改方法。

## 二、EfRepository<T>

Nop.Data.EfRepository<T>就是上面接口Nop.Core.Data.IRepository<T>的EF具体实现。



可以看到使用了EF相关的API，比如：DbContext,DbSet等，下面我具体来看一下插入、添加一条记录的方法：



## 三、[依赖注入](http://www.lanhusoft.com/Article/105.html)

Nop里面代码到处都充分体现了面向接口编程，所有代码你F12跟踪进去都是只定位到对应的接口。

查看Controller代码看到Controller引用是都是Service的接口，如图：



Nop里是通过Autofac对[mvc](http://www.lanhusoft.com/Article/169.html)的Controller进行依赖注入的。MVC的依赖注入可以参考：

[什么是ASP.NET MVC的依赖注入控制反转](http://www.lanhusoft.com/Article/105.html)

[IoC实践--用**Autofac**实现MVC5.0的IoC控制反转方法](http://www.lanhusoft.com/Article/76.html)

这个时候我们要看这些Service接口的具体实现才能找到其“庐山真面目”。

我们随便找一个Service具体实现

UrlRecordService的一些方法：

1. public virtual UrlRecord GetBySlug(string slug)
2. {
3. if (String.IsNullOrEmpty(slug))
4. return null;
6. var query = from ur in \_urlRecordRepository.Table
7. where ur.Slug == slug
8. //first, try to find an active record
9. orderby ur.IsActive descending, ur.Id
10. select ur;
11. var urlRecord = query.FirstOrDefault();
12. return urlRecord;
13. }
14. public virtual UrlRecordForCaching GetBySlugCached(string slug)
15. {
16. if (String.IsNullOrEmpty(slug))
17. return null;
19. if (\_localizationSettings.LoadAllUrlRecordsOnStartup)
20. {
21. //获取已缓存全部Url记录数据
22. var source = GetAllUrlRecordsCached();
23. var query = from ur in source
24. where ur.Slug.Equals(slug, StringComparison.InvariantCultureIgnoreCase)
25. //first, try to find an active record
26. orderby ur.IsActive descending, ur.Id
27. select ur;
28. var urlRecordForCaching = query.FirstOrDefault();
29. return urlRecordForCaching;
30. }
32. //gradual loading
33. string key = string.Format(URLRECORD\_BY\_SLUG\_KEY, slug);
34. return \_cacheManager.Get(key, () =>
35. {
36. var urlRecord = GetBySlug(slug);
37. if (urlRecord == null)
38. return null;
40. var urlRecordForCaching = Map(urlRecord);
41. return urlRecordForCaching;
42. });
43. }
44. /// <summary>
45. /// Gets all cached URL records
46. /// </summary>
47. /// <returns>cached URL records</returns>
48. protected virtual IList<UrlRecordForCaching> GetAllUrlRecordsCached()
49. {
50. //cache
51. string key = string.Format(URLRECORD\_ALL\_KEY);
52. return \_cacheManager.Get(key, () =>
53. {
54. var query = from ur in \_urlRecordRepository.Table
55. select ur;
56. var urlRecords = query.ToList();
57. var list = new List<UrlRecordForCaching>();
58. foreach (var ur in urlRecords)
59. {
60. var urlRecordForCaching = Map(ur);
61. list.Add(urlRecordForCaching);
62. }
63. return list;
64. });
65. }

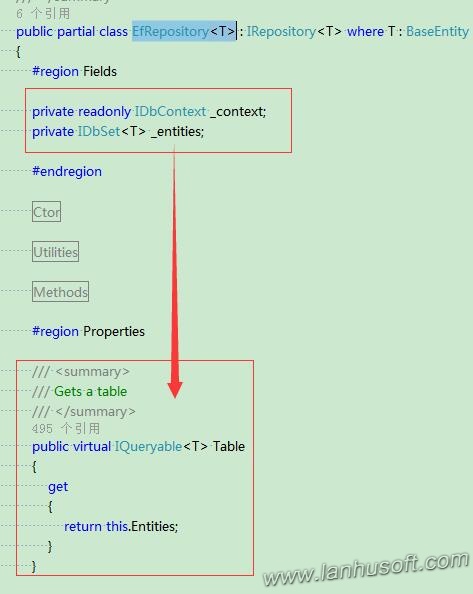
可以从上面代码看到上面通过Repository.Table和[linq](http://www.lanhusoft.com/Article/163.html)来获取数据的。上面有用到Nop的缓存管理cacheManager，对这个部分还不太懂的同学可以参考我之前写的文章：[NopCommerce源码架构详解-Cache缓存相关源码分析](http://www.lanhusoft.com/Article/394.html)

我们再回过头来看看.Table是什么？

在接口Nop.Core.Data.IRepository<T>里面



然后我们再来看接口的实现Nop.Data.EfRepository<T>对应的实现：



可以看到采用的EF的的DbSet返回表的数据，对使用过EF的同学都不用我再解释了吧？

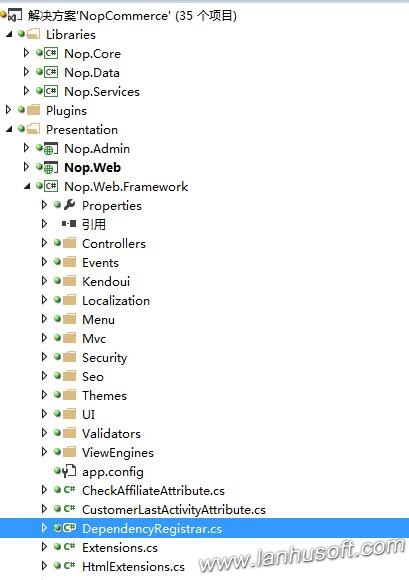
上面我们有说Controller引用的IXXXService接口，而IXXXService接口的具体实现XXXXXXService又是引用的IXXXRepository接口，IXXXRepository<T>接口具体实现是XXXRepository<T>。

不同表，T就相应不同，比如UrlRecordService里面Repository。

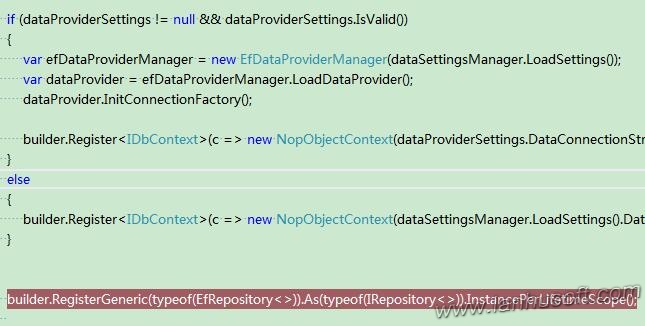


最后，Nop里面又怎么设置Nop.Core.Data.IRepository<T>使用是Nop.Data.EfRepository<T>呢？

在项目Nop.Web.Framework里面有一个类DependencyRegistrar专门告诉Autofac依赖注册的。



在DependencyRegistrar类里面我们可以找到以下代码：



通过builder.RegisterGeneric可以注册泛型依赖。同理你要可以在这个类里面找到IXXXService的依赖注入。

到此，Nop的使用EF访问数据库就来龙去脉已经清楚，现在你可以打开代码自己DIY。

参考：

[NopCommerce源码架构详解--Autofac依赖注入分析](http://www.lanhusoft.com/Article/327.html)[IoC实践--用Autofac实现MVC5.0的IoC控制反转方法](http://www.lanhusoft.com/Article/76.html)

（10）缓存：

为了提高一个系统或网站的性能和IO吞吐量，我们一般都会采用缓存技术。当然[NopCommerce](http://www.lanhusoft.com/Article/349.html)也不例外，本文我们就来给大家分析一下[nop](http://www.lanhusoft.com/Article/349.html)中Cache缓存相关类设计、核心源码及实现原理。

## 一、Nop.Core.Caching.ICacheManager

Nop首先抽象出了一个缓存存储和读取相关管理接口Nop.Core.Caching.ICacheManager。

1. namespace Nop.Core.Caching
2. {
3. /// <summary>
4. /// Cache manager interface
5. /// </summary>
6. public interface ICacheManager
7. {
8. /// <summary>
9. /// Gets or sets the value associated with the specified key.
10. /// </summary>
11. /// <typeparam name="T">Type</typeparam>
12. /// <param name="key">The key of the value to get.</param>
13. /// <returns>The value associated with the specified key.</returns>
14. T Get<T>(string key);
16. /// <summary>
17. /// Adds the specified key and object to the cache.
18. /// </summary>
19. /// <param name="key">key</param>
20. /// <param name="data">Data</param>
21. /// <param name="cacheTime">Cache time</param>
22. void Set(string key, object data, int cacheTime);
24. /// <summary>
25. /// Gets a value indicating whether the value associated with the specified key is cached
26. /// </summary>
27. /// <param name="key">key</param>
28. /// <returns>Result</returns>
29. bool IsSet(string key);
31. /// <summary>
32. /// Removes the value with the specified key from the cache
33. /// </summary>
34. /// <param name="key">/key</param>
35. void Remove(string key);
37. /// <summary>
38. /// Removes items by pattern
39. /// </summary>
40. /// <param name="pattern">pattern</param>
41. void RemoveByPattern(string pattern);
43. /// <summary>
44. /// Clear all cache data
45. /// </summary>
46. void Clear();
47. }
48. }

## 二、Nop.Core.Caching.MemoryCacheManager

接口ICacheManager具体实现是在类Nop.Core.Caching.MemoryCacheManager：

1. using System;
2. using System.Collections.Generic;
3. using System.Runtime.Caching;
4. using System.Text.RegularExpressions;
6. namespace Nop.Core.Caching
7. {
8. /// <summary>
9. /// Represents a manager for caching between HTTP requests (long term caching)
10. /// </summary>
11. public partial class MemoryCacheManager : ICacheManager
12. {
13. protected ObjectCache Cache
14. {
15. get
16. {
17. return MemoryCache.Default;
18. }
19. }
21. /// <summary>
22. /// Gets or sets the value associated with the specified key.
23. /// </summary>
24. /// <typeparam name="T">Type</typeparam>
25. /// <param name="key">The key of the value to get.</param>
26. /// <returns>The value associated with the specified key.</returns>
27. public virtual T Get<T>(string key)
28. {
29. return (T)Cache[key];
30. }
32. /// <summary>
33. /// Adds the specified key and object to the cache.
34. /// </summary>
35. /// <param name="key">key</param>
36. /// <param name="data">Data</param>
37. /// <param name="cacheTime">Cache time</param>
38. public virtual void Set(string key, object data, int cacheTime)
39. {
40. if (data == null)
41. return;
43. var policy = new CacheItemPolicy();
44. policy.AbsoluteExpiration = DateTime.Now + TimeSpan.FromMinutes(cacheTime);
45. Cache.Add(new CacheItem(key, data), policy);
46. }
48. /// <summary>
49. /// Gets a value indicating whether the value associated with the specified key is cached
50. /// </summary>
51. /// <param name="key">key</param>
52. /// <returns>Result</returns>
53. public virtual bool IsSet(string key)
54. {
55. return (Cache.Contains(key));
56. }
58. /// <summary>
59. /// Removes the value with the specified key from the cache
60. /// </summary>
61. /// <param name="key">/key</param>
62. public virtual void Remove(string key)
63. {
64. Cache.Remove(key);
65. }
67. /// <summary>
68. /// Removes items by pattern
69. /// </summary>
70. /// <param name="pattern">pattern</param>
71. public virtual void RemoveByPattern(string pattern)
72. {
73. var regex = new Regex(pattern, RegexOptions.Singleline | RegexOptions.Compiled | RegexOptions.IgnoreCase);
74. var keysToRemove = new List<String>();
76. foreach (var item in Cache)
77. if (regex.IsMatch(item.Key))
78. keysToRemove.Add(item.Key);
80. foreach (string key in keysToRemove)
81. {
82. Remove(key);
83. }
84. }
86. /// <summary>
87. /// Clear all cache data
88. /// </summary>
89. public virtual void Clear()
90. {
91. foreach (var item in Cache)
92. Remove(item.Key);
93. }
94. }
95. }

可以看到上面Nop的缓存数据是使用的的MemoryCache.Default来存储的，MemoryCache.Default是获取对默认 System.Runtime.Caching.MemoryCache 实例的引用，缓存的默认实例，也就是程序运行的内存中。

Nop除了提供了一个MemoryCacheManager，还有一个Nop.Core.Caching.PerRequestCacheManager类，它提供的是MemoryCacheManager相同的功能，不过它是把数据存在HttpContextBase.Items中，如下：

1. using System;
2. using System.Collections;
3. using System.Collections.Generic;
4. using System.Text.RegularExpressions;
5. using System.Web;
7. namespace Nop.Core.Caching
8. {
9. /// <summary>
10. /// Represents a manager for caching during an HTTP request (short term caching)
11. /// </summary>
12. public partial class PerRequestCacheManager : ICacheManager
13. {
14. private readonly HttpContextBase \_context;
16. /// <summary>
17. /// Ctor
18. /// </summary>
19. /// <param name="context">Context</param>
20. public PerRequestCacheManager(HttpContextBase context)
21. {
22. this.\_context = context;
23. }
25. /// <summary>
26. /// Creates a new instance of the NopRequestCache class
27. /// </summary>
28. protected virtual IDictionary GetItems()
29. {
30. if (\_context != null)
31. return \_context.Items;
33. return null;
34. }
36. //省略其它代码....
37. }
38. }

## 三、缓存接口ICacheManager[依赖注入](http://www.lanhusoft.com/Article/105.html)

缓存接口ICacheManager使用了依赖注入，我们在Nop.Web.Framework.DependencyRegistrar类中就能找到对ICacheManager的注册代码：

1. //cache manager
2. builder.RegisterType<MemoryCacheManager>().As<ICacheManager>().Named<ICacheManager>("nop\_cache\_static").SingleInstance();
3. builder.RegisterType<PerRequestCacheManager>().As<ICacheManager>().Named<ICacheManager>("nop\_cache\_per\_request").InstancePerLifetimeScope();

6. //pass MemoryCacheManager as cacheManager (cache settings between requests)
7. builder.RegisterType<ProductTagService>().As<IProductTagService>()
8. .WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_static"))
9. .InstancePerLifetimeScope();

12. //pass MemoryCacheManager as cacheManager (cache settings between requests)
13. builder.RegisterType<PermissionService>().As<IPermissionService>()
14. .WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_static"))
15. .InstancePerLifetimeScope();
16. //pass MemoryCacheManager as cacheManager (cache settings between requests)
17. builder.RegisterType<AclService>().As<IAclService>()
18. .WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_static"))
19. .InstancePerLifetimeScope();
20. //pass MemoryCacheManager as cacheManager (cache settings between requests)
21. builder.RegisterType<PriceCalculationService>().As<IPriceCalculationService>()
22. .WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_static"))
23. .InstancePerLifetimeScope();


27. //pass MemoryCacheManager as cacheManager (cache settings between requests)
28. builder.RegisterType<CustomerActivityService>().As<ICustomerActivityService>()
29. .WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_static"))
30. .InstancePerLifetimeScope();

上面最开始对接口ICacheManager两实现分别是MemoryCacheManager和PerRequestCacheManager并通过.Named来区分。[Autofac高级特性--注册Named命名和Key Service服务](http://www.lanhusoft.com/Article/389.html)

接下来可以配置不同的Service依赖不同的ICacheManager的实现：.WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_static"))或者.WithParameter(ResolvedParameter.ForNamed<ICacheManager>("nop\_cache\_per\_request"))。

## 四、具体实例BlogController

下面我们来举例看一下怎么使用这个缓存的。我们就以Nop.Web.Controllers.BlogController的方法BlogTags为例：

1. [ChildActionOnly]
2. public ActionResult BlogTags()
3. {
4. if (!\_blogSettings.Enabled)
5. return Content("");
7. var cacheKey = string.Format(ModelCacheEventConsumer.BLOG\_TAGS\_MODEL\_KEY, \_workContext.WorkingLanguage.Id, \_storeContext.CurrentStore.Id);
8. var cachedModel = \_cacheManager.Get(cacheKey, () =>
9. {
10. var model = new BlogPostTagListModel();
12. //get tags
13. var tags = \_blogService.GetAllBlogPostTags(\_storeContext.CurrentStore.Id, \_workContext.WorkingLanguage.Id)
14. .OrderByDescending(x => x.BlogPostCount)
15. .Take(\_blogSettings.NumberOfTags)
16. .ToList();
17. //sorting
18. tags = tags.OrderBy(x => x.Name).ToList();
20. foreach (var tag in tags)
21. model.Tags.Add(new BlogPostTagModel()
22. {
23. Name = tag.Name,
24. BlogPostCount = tag.BlogPostCount
25. });
26. return model;
27. });
29. return PartialView(cachedModel);
30. }

上面var cachedModel = \_cacheManager.Get就是从缓存中读取数据，\_cacheManager的Get方法第二个参数是一个[lambda](http://www.lanhusoft.com/Article/168.html)表达式，可以传一个方法，这时我们就可以把数据的从数据库中的逻辑放在里面，注意：当第二次请求数据时，如果缓存中有数据，这个Lambda方法是不会执行的。为什么呢？我们可以选中\_cacheManager的Get方法按F12进去看这个方法的实现就知道了：

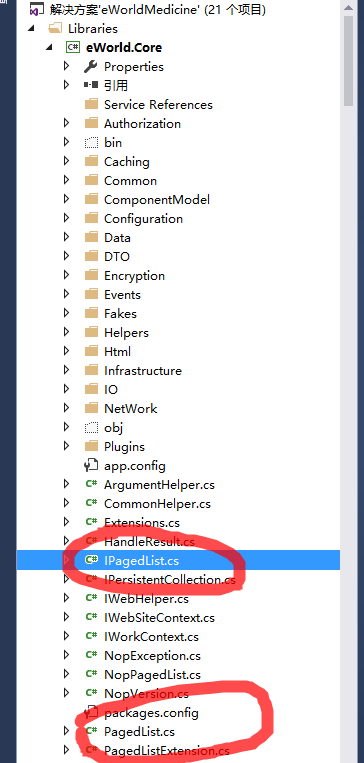
1. using System;
3. namespace Nop.Core.Caching
4. {
5. /// <summary>
6. /// Extensions
7. /// </summary>
8. public static class CacheExtensions
9. {
10. /// <summary>
11. /// Get a cached item. If it's not in the cache yet, then load and cache it
12. /// </summary>
13. /// <typeparam name="T">Type</typeparam>
14. /// <param name="cacheManager">Cache manager</param>
15. /// <param name="key">Cache key</param>
16. /// <param name="acquire">Function to load item if it's not in the cache yet</param>
17. /// <returns>Cached item</returns>
18. public static T Get<T>(this ICacheManager cacheManager, string key, Func<T> acquire)
19. {
20. return Get(cacheManager, key, 60, acquire);
21. }
23. /// <summary>
24. /// Get a cached item. If it's not in the cache yet, then load and cache it
25. /// </summary>
26. /// <typeparam name="T">Type</typeparam>
27. /// <param name="cacheManager">Cache manager</param>
28. /// <param name="key">Cache key</param>
29. /// <param name="cacheTime">Cache time in minutes (0 - do not cache)</param>
30. /// <param name="acquire">Function to load item if it's not in the cache yet</param>
31. /// <returns>Cached item</returns>
32. public static T Get<T>(this ICacheManager cacheManager, string key, int cacheTime, Func<T> acquire)
33. {
34. if (cacheManager.IsSet(key))
35. {
36. return cacheManager.Get<T>(key);
37. }
38. else
39. {
40. var result = acquire();
41. if (cacheTime > 0)
42. cacheManager.Set(key, result, cacheTime);
43. return result;
44. }
45. }
46. }
47. }

可以看到其实上面\_cacheManager.Get调用的是类型ICacheManager的一个[扩展方法](http://www.lanhusoft.com/Article/27.html)。第二个方法就可以知道，当缓存中有数据直接返回cacheManager.Get<T>(key)，如果没有才进入else分支，执行参数的Lambda表达方式acquire()。

（11）分页

a.

位置：



b.

代码：



namespace eWorld.Core

{

/// <summary>

/// Paged list interface

/// </summary>

public interface IPagedList<T> : IList<T>

{

/// <summary>

///

/// </summary>

int PageIndex { get; set; }

/// <summary>

///

/// </summary>

int PageSize { get; set; }

/// <summary>

///

/// </summary>

int TotalCount { get; set; }

/// <summary>

///

/// </summary>

int TotalPages { get; set; }

/// <summary>

///

/// </summary>

bool HasPreviousPage { get; set; }

/// <summary>

///

/// </summary>

bool HasNextPage { get; set; }

}

}



namespace eWorld.Core

{

/// <summary>

/// Paged list

/// </summary>

/// <typeparam name="T">T</typeparam>

public class PagedList<T> : List<T>, IPagedList<T>

{

/// <summary>

/// Ctor

/// </summary>

/// <param name="source">source</param>

/// <param name="pageIndex">Page index</param>

/// <param name="pageSize">Page size</param>

public PagedList(IQueryable<T> source, int pageIndex, int pageSize)

{

int total = source.Count();

this.TotalCount = total;

this.TotalPages = total / pageSize;

if (total % pageSize > 0)

TotalPages++;

this.PageSize = pageSize;

this.PageIndex = pageIndex;

this.AddRange(source.Skip(pageIndex \* pageSize).Take(pageSize).ToList());

}

/// <summary>

/// Ctor

/// </summary>

/// <param name="source">source</param>

/// <param name="pageIndex">Page index</param>

/// <param name="pageSize">Page size</param>

public PagedList(IList<T> source, int pageIndex, int pageSize)

{

TotalCount = source.Count();

TotalPages = TotalCount / pageSize;

if (TotalCount % pageSize > 0)

TotalPages++;

this.PageSize = pageSize;

this.PageIndex = pageIndex;

this.AddRange(source.Skip(pageIndex \* pageSize).Take(pageSize).ToList());

}

/// <summary>

///

/// </summary>

/// <param name="source"></param>

/// <param name="pageIndex"></param>

/// <param name="pageSize"></param>

public PagedList(IEnumerable<T> source, int pageIndex, int pageSize)

{

TotalCount = source.Count();

TotalPages = TotalCount / pageSize;

if (TotalCount % pageSize > 0)

TotalPages++;

this.PageSize = pageSize;

this.PageIndex = pageIndex;

this.AddRange(source.Skip(pageIndex \* pageSize).Take(pageSize).ToList());

}

/// <summary>

/// Ctor

/// </summary>

/// <param name="source">source</param>

/// <param name="pageIndex">Page index</param>

/// <param name="pageSize">Page size</param>

/// <param name="totalCount">Total count</param>

public PagedList(IEnumerable<T> source, int pageIndex, int pageSize, int totalCount)

{

TotalCount = totalCount;

TotalPages = TotalCount / pageSize;

if (TotalCount % pageSize > 0)

TotalPages++;

this.PageSize = pageSize;

this.PageIndex = pageIndex;

this.AddRange(source);

}

/// <summary>

///

/// </summary>

/// <param name="\_TotalCount"></param>

public PagedList(int \_TotalCount)

{

this.TotalCount = \_TotalCount;

}

/// <summary>

///

/// </summary>

/// <param name="totalCount"></param>

/// <param name="pageIndex"></param>

/// <param name="pageSize"></param>

public PagedList(int totalCount, int pageIndex, int pageSize)

{

TotalCount = totalCount;

TotalPages = TotalCount / pageSize;

if (TotalCount % pageSize > 0)

TotalPages++;

this.PageSize = pageSize;

this.PageIndex = pageIndex;

}

/// <summary>

///

/// </summary>

public PagedList()

{

}

/// <summary>

///

/// </summary>

public int PageIndex { get; set; }

/// <summary>

///

/// </summary>

public int PageSize { get; set; }

/// <summary>

///

/// </summary>

public int TotalCount { get; set; }

/// <summary>

///

/// </summary>

public int TotalPages { get; set; }

/// <summary>

///

/// </summary>

public bool HasPreviousPage { get; set; }

/// <summary>

///

/// </summary>

public bool HasNextPage { get; set; }

}

}

原来我在用分页的时候，用的就是这个框架里面的分页数据。

query = query.OrderByDescending(q => q.LastReplyTime).DistinctBy(q => q.Id).AsQueryable();

PagedList<DingQuestionsListModel> page = new PagedList<DingQuestionsListModel>(query, param.PageIndex, param.PageSize);

return page;

然后在bll层加一个data：

result.data = new ApiPageListModel<DingQuestionsListModel>

{

DataList = questionsList,

PageInfo = new PageInfo

{

HasNextPage = questionsList.PageIndex + 1 < questionsList.TotalPages,

HasPreviousPage = questionsList.PageIndex > 0,

PageIndex = questionsList.PageIndex,

PageSize = questionsList.PageSize,

TotalCount = questionsList.TotalCount,

TotalPages = questionsList.TotalPages

}

};

（12）NopCommerce源码架构详解-AutoMapper对象关联映射相关源码分析

前一篇文章介绍了一下[AutoMapper](http://www.lanhusoft.com/Article/408.html)，一个非常好用，强大的对象映射转换工具。有了前面AutoMapper的基础，我们就来分析一下[NopCommerce](http://www.lanhusoft.com/Article/349.html)源码架构中使用AutoMapper的地方。

**主要用到的类或接口：**

1、IStartupTask

2、AutoMapperStartupTask

3、MappingExtensions

4、[nop](http://www.lanhusoft.com/Article/349.html)Engine

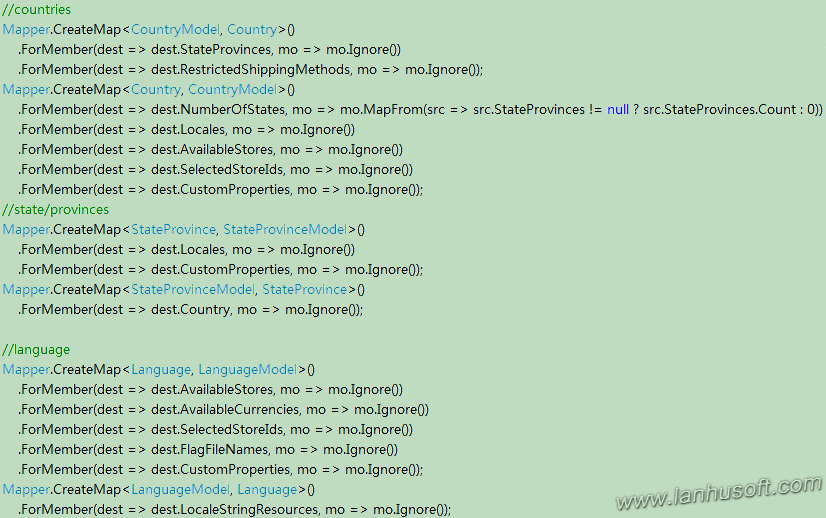
## 一、IStartupTask接口

IStartupTask接口在程序启动的时候要运行的任务，里面只有一个方法Execute，实现个接口就是实现这个Execute方法，里面可以做一些程序初始化或者配置相关的事情。

1. namespace Nop.Core.Infrastructure
2. {
3. /// <summary>
4. /// Interface which should be implemented by tasks run on startup
5. /// </summary>
6. public interface IStartupTask
7. {
8. /// <summary>
9. /// Execute task
10. /// </summary>
11. void Execute();
13. /// <summary>
14. /// Order
15. /// </summary>
16. int Order { get; }
17. }
18. }

## 二、AutoMapperStartupTask

AutoMapperStartupTask就是实现了接口IStartupTask。在里面的Execute()方法中配置了所有的AutoMapper对象映射。



AutoMapper调用Mapper.CreateMap来创建对象之间的映射关系，如下代码：

Mapper.CreateMap<Address, AddressModel>()

Mapper.CreateMap的第一个参数是源类型，第二参数是目标类型。

## 三、MappingExtensions

MappingExtensions是一个静态类，里面封装了许许多多的[扩展方法](http://www.lanhusoft.com/Article/27.html)，其中就包括了调用AutoMapper的Mapper.Map方法来根据前面我们配置的对象映射关系返回转换完了后的指定对象。

1. public static StoreModel ToModel(this Store entity)
2. {
3. return Mapper.Map<Store, StoreModel>(entity);
4. }
6. public static Store ToEntity(this StoreModel model)
7. {
8. return Mapper.Map<StoreModel, Store>(model);
9. }
11. public static CategoryTemplateModel ToModel(this CategoryTemplate entity)
12. {
13. return Mapper.Map<CategoryTemplate, CategoryTemplateModel>(entity);
14. }
16. public static CategoryTemplate ToEntity(this CategoryTemplateModel model)
17. {
18. return Mapper.Map<CategoryTemplateModel, CategoryTemplate>(model);
19. }

这样相应的对象直接就可以调用扩展方法ToEntity或者ToModel转换成相应的对象了。

## 四、NopEngine

在项目中Nop.Core命名空间下Nop.Core.Infrastructure的类NopEngine里面有一个初始化方法RunStartupStasks，里面找到实现了接口IStartupTask所有实现，并动态的创建一个相应的对象，最后调用其方法Execute，也就完成了之前配置好的所有AutoMapper的映射关系初始化操作。

1. protected virtual void RunStartupTasks()
2. {
3. var typeFinder = \_containerManager.Resolve<ITypeFinder>();
4. var startUpTaskTypes = typeFinder.FindClassesOfType<IStartupTask>();
5. var startUpTasks = new List<IStartupTask>();
6. foreach (var startUpTaskType in startUpTaskTypes)
7. startUpTasks.Add((IStartupTask)Activator.CreateInstance(startUpTaskType));
8. //sort
9. startUpTasks = startUpTasks.AsQueryable().OrderBy(st => st.Order).ToList();
10. foreach (var startUpTask in startUpTasks)
11. startUpTask.Execute();
12. }

公司用的是这个：

namespace eWorld.Framework.Mapper

{

/// <summary>

/// AutoMapper 扩展类

/// </summary>

public static class AutoMapperHelper

{

/// <summary>

/// 多对象映射

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="sources"></param>

/// <returns></returns>

public static T MapTo<T>(params object[] sources) where T : class

{

if (!sources.Any())

{

return default(T);

}

var initialSource = sources[0];

var mappingResult = Map<T>(initialSource);

if (sources.Count() > 1)

{

Map(mappingResult, sources.Skip(1).ToArray());

}

return mappingResult;

}

private static void Map(object destination, params object[] sources)

{

if (!sources.Any())

{

return;

}

var destinationType = destination.GetType();

foreach (var source in sources)

{

var sourceType = source.GetType();

AutoMapper.Mapper.DynamicMap(source, destination, sourceType, destinationType);

}

}

private static T Map<T>(object source) where T : class

{

var destinationType = typeof(T);

var sourceType = source.GetType();

var mappingResult = AutoMapper.Mapper.DynamicMap(source, sourceType, destinationType);

return mappingResult as T;

}

/// <summary>

/// 为已经存在的对象进行automapper

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="self"></param>

/// <param name="result"></param>

/// <returns></returns>

public static T MapTo<T>(this object self, T result)

{

if (self == null)

return default(T);

AutoMapper.Mapper.CreateMap(self.GetType().UnderlyingSystemType, typeof(T));

AutoMapper.Mapper.DynamicMap(self, result, self.GetType(), typeof(T));

return result;

}

/// <summary>

/// 类型映射

/// </summary>

public static T MapTo<T>(this object obj)

{

if (obj == null)

return default(T);

AutoMapper.Mapper.CreateMap(obj.GetType(), typeof(T));

return AutoMapper.Mapper.DynamicMap<T>(obj);

}

/// <summary>

/// 集合列表类型映射

/// </summary>

public static List<T> MapToList<T>(this IEnumerable source)

{

if (source == null) return default(List<T>);

foreach (var first in source)

{

var type = first.GetType();

AutoMapper.Mapper.CreateMap(type, typeof(T));

break;

}

return AutoMapper.Mapper.DynamicMap<List<T>>(source);

}

/// <summary>

/// 集合列表类型映射

/// </summary>

public static List<TDestination> MapToList<TSource, TDestination>(this IEnumerable<TSource> source)

{

AutoMapper.Mapper.CreateMap<TSource, TDestination>();

return AutoMapper.Mapper.Map<List<TDestination>>(source);

}

/// <summary>

/// 类型映射

/// </summary>

public static TDestination MapTo<TSource, TDestination>(this TSource source, TDestination destination)

where TSource : class

where TDestination : class

{

if (source == null) return destination;

AutoMapper.Mapper.CreateMap<TSource, TDestination>();

return AutoMapper.Mapper.Map(source, destination);

}

/// <summary>

/// DataReader映射

/// </summary>

public static IEnumerable<T> DataReaderMapTo<T>(this IDataReader reader)

{

AutoMapper.Mapper.Reset();

AutoMapper.Mapper.CreateMap<IDataReader, IEnumerable<T>>();

return AutoMapper.Mapper.Map<IDataReader, IEnumerable<T>>(reader);

}

}

}

（13）NopCommerce源码架构详解-MVC5使用EF6实现Repository模式相关源码分析

Repository模式是属于领域模型范畴，是架构模式。[nop](http://www.lanhusoft.com/Article/349.html)中使用了这个种模板来读取、操作数据。今天我们通过Nop来看看在ASP.NET [mvc](http://www.lanhusoft.com/Article/169.html)5使用EF6实现Repository模式的方案。

## 一、理解Repository模式

首先我们来看看Repository模式相关的概念：

**Repository（资源库）**

               协调领域和数据映射层，利用类似于集合的接口来访问领域对象

定义（来自Martin Fowler的《企业应用架构模式》）：

               Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.

              个人理解：Repository是一个独立的层，介于领域层与数据映射层（数据访问层）之间。它的存在让领域层感觉不到数据访问层的存在，它提供一个类似集合的接口提供给领域层进行领域对象的访问。Repository是仓库管理员，领域层需要什么东西只需告诉仓库管理员，由仓库管理员把东西拿给它，并不需要知道东西实际放在哪。

1. Repository模式是架构模式，在设计架构时，才有参考价值；

2. Repository模式主要是封装数据查询和存储逻辑；

3. Repository模式实际用途：更换、升级ORM引擎，不影响业务逻辑；

4. Repository模式能提高测试效率，单元测试时，用Mock对象代替实际的数据库存取，可以成倍地提高测试用例运行速度。

评估：应用Repository模式所带来的好处，远高于实现这个模式所增加的代码。只要项目分层，都应当使用这个模式。

**关于泛型Repository接口：**

仅使用泛型Repository接口并不太合适，因为Repository接口是提供给Domain层的操作契约，不同的entity对于Domain来说可能有不同的操作约束。因此Repository接口还是应该单独针对每个Eneity类来定义。

泛型的Repository<T>类仍然用来减少重复代码，只是不能被UserRepository类直接继承，因为这样Delete方法将侵入User类，所以改为在UserRepository中组合一个Repository<T>，将开放给domain可见且又能使用泛型重用的功能委托给这个Repository<T>

**Repository与Dal的区别：**

Repository是DDD中的概念，强调Repository是受Domain驱动的，Repository中定义的功能要体现Domain的意图和约束，而Dal更纯粹的就是提供数据访问的功能,并不严格受限于Business层。

使用Repository，隐含着一种意图倾向，就是 Domain需要什么我才提供什么，不该提供的功能就不要提供，一切都是以Domain的需求为核心；而使用Dal，其意图倾向在于我Dal层能使用的数据库访问操作提供给Business层，你Business要用哪个自己选。换一个Business也可以用我这个Dal，一切是以我Dal能提供什么操作为核心。

## 二、Nop中的Repository模式架构

### 2.1、主要类及用途

Nop使用的ORM构架是EF，并且采用Repository模式架构。主要用到的类和接口：

1、Nop.Core.Data.IRepository<T>

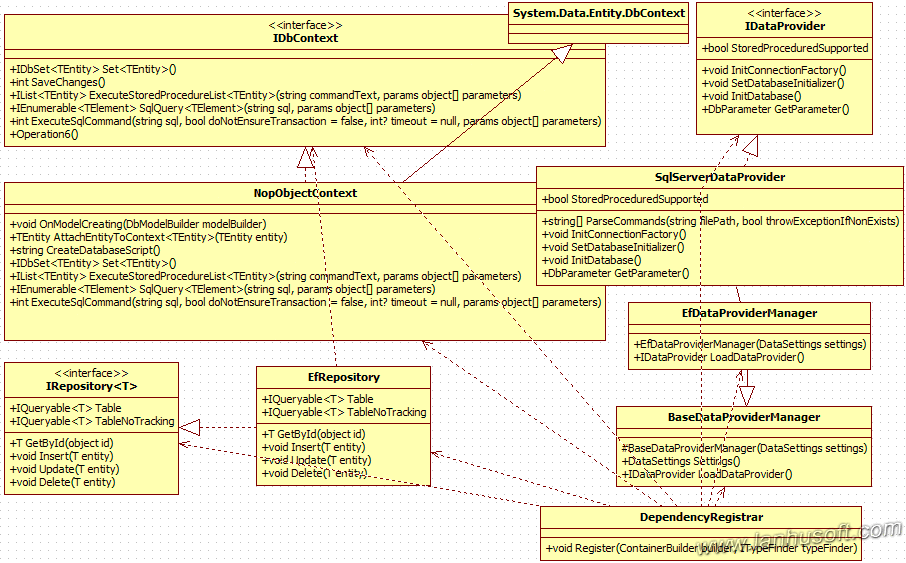
2、Nop.Data.EfRepository<T>

4、Nop.Data.BaseDataProviderManager，Nop.Data.EfDataProviderManager

5、Nop.Data.IDbContext

6、Nop.Data.NopObjectContext

**主要相关类图**



### 2.2、核心代码

**1、泛型的接口IRepository<T>：**

Nop抽象出一个实体常用的一些操作，据此定义一个泛型的接口IRepository<T>，T限制为是必须继承于基类实体BaseEntity。这样接口就可能适用于所有的实体，接口中定义了实体的一些常用的公用方法，比如：增、删、查、改。

1. using System.[linq](http://www.lanhusoft.com/Article/163.html);
3. namespace Nop.Core.Data
4. {
5. /// <summary>
6. /// Repository
7. /// </summary>
8. public partial interface IRepository<T> where T : BaseEntity
9. {
10. /// <summary>
11. /// 通过ID获取实体
12. /// </summary>
13. /// <param name="id">Identifier</param>
14. /// <returns>Entity</returns>
15. T GetById(object id);
17. /// <summary>
18. /// Insert entity
19. /// </summary>
20. /// <param name="entity">Entity</param>
21. void Insert(T entity);
23. /// <summary>
24. /// Update entity
25. /// </summary>
26. /// <param name="entity">Entity</param>
27. void Update(T entity);
29. /// <summary>
30. /// Delete entity
31. /// </summary>
32. /// <param name="entity">Entity</param>
33. void Delete(T entity);
35. /// <summary>
36. /// Gets a table
37. /// </summary>
38. IQueryable<T> Table { get; }
40. /// <summary>
41. /// Gets a table with "no tracking" enabled (EF feature) Use it only when you load record(s) only for read-only operations
42. /// </summary>
43. IQueryable<T> TableNoTracking { get; }
44. }
45. }

我们可以看到Nop的Service中都引用这个接口，如：BlogService的构造函数就用IRepository<BlogPost> blogPostRepository。

**2、IDbContext**

Nop定义了下数据库上下文接口，定义了针对数据库最基本的一些操作，比如设置实体，保存实体，执行sql或存储过程等等。

1. using System.Collections.Generic;
2. using System.Data.Entity;
3. using Nop.Core;
5. namespace Nop.Data
6. {
7. public interface IDbContext
8. {
9. /// <summary>
10. /// Get DbSet
11. /// </summary>
12. /// <typeparam name="TEntity">Entity type</typeparam>
13. /// <returns>DbSet</returns>
14. IDbSet<TEntity> Set<TEntity>() where TEntity : BaseEntity;
16. /// <summary>
17. /// Save changes
18. /// </summary>
19. /// <returns></returns>
20. int SaveChanges();
22. /// <summary>
23. /// 执行存储过程并返回实体的List
24. /// </summary>
25. /// <typeparam name="TEntity">Entity type</typeparam>
26. /// <param name="commandText">Command text</param>
27. /// <param name="parameters">Parameters</param>
28. /// <returns>Entities</returns>
29. IList<TEntity> ExecuteStoredProcedureList<TEntity>(string commandText, params object[] parameters)
30. where TEntity : BaseEntity, new();
32. /// <summary>
33. /// 执行sql语句并返回一个指定类型实体的集合
34. /// </summary>
35. /// <typeparam name="TElement">The type of object returned by the query.</typeparam>
36. /// <param name="sql">The SQL query string.</param>
37. /// <param name="parameters">The parameters to apply to the SQL query string.</param>
38. /// <returns>Result</returns>
39. IEnumerable<TElement> SqlQuery<TElement>(string sql, params object[] parameters);
41. /// <summary>
42. /// 执行一个指定 DDL/DML 命令
43. /// </summary>
44. /// <param name="sql">The command string</param>
45. /// <param name="doNotEnsureTransaction">false - the transaction creation is not ensured; true - the transaction creation is ensured.</param>
46. /// <param name="timeout">Timeout value, in seconds. A null value indicates that the default value of the underlying provider will be used</param>
47. /// <param name="parameters">The parameters to apply to the command string.</param>
48. /// <returns>The result returned by the database after executing the command.</returns>
49. int ExecuteSqlCommand(string sql, bool doNotEnsureTransaction = false, int? timeout = null, params object[] parameters);
50. }
51. }

**3、EfRepository**

EfRepository类就是Nop实现RePository最核心的部分了，它真正的实现了接口IRepositoryj里面定义的所有方法，并且里面引用了接口IDbContext，而不是IDbContext的实现，这样面向接口编程大大的降低了系统模块的耦合性，其实细心的同学可以随处看到Nop中这种面向接口编程的思路。

下面我们来看看EfRepository类的实现：

1. using System;
2. using System.Data.Entity;
3. using System.Data.Entity.Validation;
4. using System.Linq;
5. using Nop.Core;
6. using Nop.Core.Data;
8. namespace Nop.Data
9. {
10. /// <summary>
11. /// Entity Framework repository
12. /// </summary>
13. public partial class EfRepository<T> : IRepository<T> where T : BaseEntity
14. {
15. private readonly IDbContext \_context;
16. private IDbSet<T> \_entities;
18. /// <summary>
19. /// Ctor
20. /// </summary>
21. /// <param name="context">Object context</param>
22. public EfRepository(IDbContext context)
23. {
24. this.\_context = context;
25. }
27. /// <summary>
28. /// Get entity by identifier
29. /// </summary>
30. /// <param name="id">Identifier</param>
31. /// <returns>Entity</returns>
32. public virtual T GetById(object id)
33. {
34. //see some suggested performance optimization (not tested)
35. //http://stackoverflow.com/questions/11686225/dbset-find-method-ridiculously-slow-compared-to-singleordefault-on-id/11688189#comment34876113\_11688189
36. return this.Entities.Find(id);
37. }
39. /// <summary>
40. /// Insert entity
41. /// </summary>
42. /// <param name="entity">Entity</param>
43. public virtual void Insert(T entity)
44. {
45. try
46. {
47. if (entity == null)
48. throw new ArgumentNullException("entity");
50. this.Entities.Add(entity);
52. this.\_context.SaveChanges();
53. }
54. catch (DbEntityValidationException dbEx)
55. {
56. var msg = string.Empty;
58. foreach (var validationErrors in dbEx.EntityValidationErrors)
59. foreach (var validationError in validationErrors.ValidationErrors)
60. msg += string.Format("Property: {0} Error: {1}", validationError.PropertyName, validationError.ErrorMessage) + Environment.NewLine;
62. var fail = new Exception(msg, dbEx);
63. //Debug.WriteLine(fail.Message, fail);
64. throw fail;
65. }
66. }
68. /// <summary>
69. /// Update entity
70. /// </summary>
71. /// <param name="entity">Entity</param>
72. public virtual void Update(T entity)
73. {
74. try
75. {
76. if (entity == null)
77. throw new ArgumentNullException("entity");
79. this.\_context.SaveChanges();
80. }
81. catch (DbEntityValidationException dbEx)
82. {
83. var msg = string.Empty;
85. foreach (var validationErrors in dbEx.EntityValidationErrors)
86. foreach (var validationError in validationErrors.ValidationErrors)
87. msg += Environment.NewLine + string.Format("Property: {0} Error: {1}", validationError.PropertyName, validationError.ErrorMessage);
89. var fail = new Exception(msg, dbEx);
90. //Debug.WriteLine(fail.Message, fail);
91. throw fail;
92. }
93. }
95. /// <summary>
96. /// Delete entity
97. /// </summary>
98. /// <param name="entity">Entity</param>
99. public virtual void Delete(T entity)
100. {
101. try
102. {
103. if (entity == null)
104. throw new ArgumentNullException("entity");
106. this.Entities.Remove(entity);
108. this.\_context.SaveChanges();
109. }
110. catch (DbEntityValidationException dbEx)
111. {
112. var msg = string.Empty;
114. foreach (var validationErrors in dbEx.EntityValidationErrors)
115. foreach (var validationError in validationErrors.ValidationErrors)
116. msg += Environment.NewLine + string.Format("Property: {0} Error: {1}", validationError.PropertyName, validationError.ErrorMessage);
118. var fail = new Exception(msg, dbEx);
119. //Debug.WriteLine(fail.Message, fail);
120. throw fail;
121. }
122. }
124. /// <summary>
125. /// Gets a table
126. /// </summary>
127. public virtual IQueryable<T> Table
128. {
129. get
130. {
131. return this.Entities;
132. }
133. }

136. /// <summary>
137. /// Gets a table with "no tracking" enabled (EF feature) Use it only when you load record(s) only for read-only operations
138. /// </summary>
139. public virtual IQueryable<T> TableNoTracking
140. {
141. get
142. {
143. return this.Entities.AsNoTracking();
144. }
145. }

148. /// <summary>
149. /// Entities
150. /// </summary>
151. protected virtual IDbSet<T> Entities
152. {
153. get
154. {
155. if (\_entities == null)
156. \_entities = \_context.Set<T>();
157. return \_entities;
158. }
159. }
160. }
161. }

**4、NopObjectContext**

NopObjectContext是EfRepository类中引用的接口IDbContext真正实现。

1. using System;
2. using System.Collections.Generic;
3. using System.Data;
4. using System.Data.Common;
5. using System.Data.Entity;
6. using System.Data.Entity.Infrastructure;
7. using System.Data.Entity.ModelConfiguration;
8. using System.Linq;
9. using System.Reflection;
10. using Nop.Core;
12. namespace Nop.Data
13. {
14. /// <summary>
15. /// Object context
16. /// </summary>
17. public class NopObjectContext : DbContext, IDbContext
18. {
19. #region Ctor
21. public NopObjectContext(string nameOrConnectionString)
22. : base(nameOrConnectionString)
23. {
24. //((IObjectContextAdapter) this).ObjectContext.ContextOptions.LazyLoadingEnabled = true;
25. }
27. #endregion
29. #region Utilities
31. protected override void OnModelCreating(DbModelBuilder modelBuilder)
32. {
33. //dynamically load all configuration
34. //System.Type configType = typeof(LanguageMap); //any of your configuration classes here
35. //var typesToRegister = Assembly.GetAssembly(configType).GetTypes()
37. var typesToRegister = Assembly.GetExecutingAssembly().GetTypes()
38. .Where(type => !String.IsNullOrEmpty(type.Namespace))
39. .Where(type => type.BaseType != null && type.BaseType.IsGenericType && type.BaseType.GetGenericTypeDefinition() == typeof(EntityTypeConfiguration<>));
40. foreach (var type in typesToRegister)
41. {
42. [dynamic](http://www.lanhusoft.com/Article/67.html) configurationInstance = Activator.CreateInstance(type);
43. modelBuilder.Configurations.Add(configurationInstance);
44. }
45. //...or do it manually below. For example,
46. //modelBuilder.Configurations.Add(new LanguageMap());


50. base.OnModelCreating(modelBuilder);
51. }

54. /// <summary>
55. /// 将一个实体Attach到一个数据库上下文，如果已经存在就直接返回这个实体
56. /// </summary>
57. /// <typeparam name="TEntity">TEntity</typeparam>
58. /// <param name="entity">Entity</param>
59. /// <returns>Attached entity</returns>
60. protected virtual TEntity AttachEntityToContext<TEntity>(TEntity entity) where TEntity : BaseEntity, new()
61. {
62. //little hack here until Entity Framework really supports stored procedures
63. //otherwise, navigation properties of loaded entities are not loaded until an entity is attached to the context
64. var alreadyAttached = Set<TEntity>().Local.FirstOrDefault(x => x.Id == entity.Id);
65. if (alreadyAttached == null)
66. {
67. //attach new entity
68. Set<TEntity>().Attach(entity);
69. return entity;
70. }
71. else
72. {
73. //entity is already loaded.
74. return alreadyAttached;
75. }
76. }
78. #endregion
80. #region Methods
82. /// <summary>
83. /// 创建一个数据库sql脚本
84. /// </summary>
85. /// <returns>SQL to generate database</returns>
86. public string CreateDatabaseScript()
87. {
88. return ((IObjectContextAdapter)this).ObjectContext.CreateDatabaseScript();
89. }
91. /// <summary>
92. /// Get DbSet
93. /// </summary>
94. /// <typeparam name="TEntity">Entity type</typeparam>
95. /// <returns>DbSet</returns>
96. public new IDbSet<TEntity> Set<TEntity>() where TEntity : BaseEntity
97. {
98. return base.Set<TEntity>();
99. }
101. /// <summary>
102. /// 执行存储过程并返回实体的List
103. /// </summary>
104. /// <typeparam name="TEntity">Entity type</typeparam>
105. /// <param name="commandText">Command text</param>
106. /// <param name="parameters">Parameters</param>
107. /// <returns>Entities</returns>
108. public IList<TEntity> ExecuteStoredProcedureList<TEntity>(string commandText, params object[] parameters) where TEntity : BaseEntity, new()
109. {
110. //add parameters to command
111. if (parameters != null && parameters.Length > 0)
112. {
113. for (int i = 0; i <= parameters.Length - 1; i++)
114. {
115. var p = parameters[i] as DbParameter;
116. if (p == null)
117. throw new Exception("Not support parameter type");
119. commandText += i == 0 ? " " : ", ";
121. commandText += "@" + p.ParameterName;
122. if (p.Direction == ParameterDirection.InputOutput || p.Direction == ParameterDirection.Output)
123. {
124. //output parameter
125. commandText += " output";
126. }
127. }
128. }
130. var result = this.Database.SqlQuery<TEntity>(commandText, parameters).ToList();
132. //performance hack applied as described here - http://www.[NopCommerce](http://www.lanhusoft.com/Article/349.html).com/boards/t/25483/fix-very-important-speed-improvement.aspx
133. bool acd = this.Configuration.AutoDetectChangesEnabled;
134. try
135. {
136. this.Configuration.AutoDetectChangesEnabled = false;
138. for (int i = 0; i < result.Count; i++)
139. result[i] = AttachEntityToContext(result[i]);
140. }
141. finally
142. {
143. this.Configuration.AutoDetectChangesEnabled = acd;
144. }
146. return result;
147. }
149. /// <summary>
150. /// 执行sql语句并返回一个指定类型实体的集合
151. /// <typeparam name="TElement">The type of object returned by the query.</typeparam>
152. /// <param name="sql">The SQL query string.</param>
153. /// <param name="parameters">The parameters to apply to the SQL query string.</param>
154. /// <returns>Result</returns>
155. public IEnumerable<TElement> SqlQuery<TElement>(string sql, params object[] parameters)
156. {
157. return this.Database.SqlQuery<TElement>(sql, parameters);
158. }
160. /// <summary>
161. /// 执行一个指定 DDL/DML 命令
162. /// </summary>
163. /// <param name="sql">The command string</param>
164. /// <param name="doNotEnsureTransaction">false - the transaction creation is not ensured; true - the transaction creation is ensured.</param>
165. /// <param name="timeout">Timeout value, in seconds. A null value indicates that the default value of the underlying provider will be used</param>
166. /// <param name="parameters">The parameters to apply to the command string.</param>
167. /// <returns>The result returned by the database after executing the command.</returns>
168. public int ExecuteSqlCommand(string sql, bool doNotEnsureTransaction = false, int? timeout = null, params object[] parameters)
169. {
170. int? previousTimeout = null;
171. if (timeout.HasValue)
172. {
173. //store previous timeout
174. previousTimeout = ((IObjectContextAdapter) this).ObjectContext.CommandTimeout;
175. ((IObjectContextAdapter) this).ObjectContext.CommandTimeout = timeout;
176. }
178. var transactionalBehavior = doNotEnsureTransaction
179. ? TransactionalBehavior.DoNotEnsureTransaction
180. : TransactionalBehavior.EnsureTransaction;
181. var result = this.Database.ExecuteSqlCommand(transactionalBehavior, sql, parameters);
183. if (timeout.HasValue)
184. {
185. //Set previous timeout back
186. ((IObjectContextAdapter) this).ObjectContext.CommandTimeout = previousTimeout;
187. }
189. //return result
190. return result;
191. }
193. #endregion
194. }
195. }

可以看到这个类NopObjectContext里面使用了EF的一些API，对实体进行操作。比如：

1. base.OnModelCreating(modelBuilder);
3. var result = this.Database.SqlQuery<TEntity>(commandText, parameters).ToList();
4. return this.Database.SqlQuery<TElement>(sql, parameters);
5. var result = this.Database.ExecuteSqlCommand(transactionalBehavior, sql, parameters);

**5、Nop对Repository调用**

Nop中对业务都操作都是抽象为某个服务（XXService），也就是对Repository架构用Service对其进行了进一步的隔离。在Nop的Controller中引用都是引用的Service的接口。如后台项目Nop.Admin中的BlogController：

1. public partial class BlogController : BaseAdminController
2. {
3. #region Fields
5. private readonly IBlogService \_blogService;
6. private readonly ILanguageService \_languageService;
7. private readonly IDateTimeHelper \_dateTimeHelper;
8. private readonly ILocalizationService \_localizationService;
9. private readonly IPermissionService \_permissionService;
10. private readonly IUrlRecordService \_urlRecordService;
11. private readonly IStoreService \_storeService;
12. private readonly IStoreMappingService \_storeMappingService;
14. #endregion

17. public BlogController(IBlogService blogService, ILanguageService languageService,
18. IDateTimeHelper dateTimeHelper,
19. ILocalizationService localizationService, IPermissionService permissionService,
20. IUrlRecordService urlRecordService,
21. IStoreService storeService, IStoreMappingService storeMappingService)
22. {
23. this.\_blogService = blogService;
24. this.\_languageService = languageService;
25. this.\_dateTimeHelper = dateTimeHelper;
26. this.\_localizationService = localizationService;
27. this.\_permissionService = permissionService;
28. this.\_urlRecordService = urlRecordService;
29. this.\_storeService = storeService;
30. this.\_storeMappingService = storeMappingService;
31. }
33. //省略其它代码...
34. }

我们来看其中一个Service的实现，Nop.Services.Blogs.BlogService.cs：

1. namespace Nop.Services.Blogs
2. {
3. /// <summary>
4. /// Blog service
5. /// </summary>
6. public partial class BlogService : IBlogService
7. {
8. #region Fields
10. private readonly IRepository<BlogPost> \_blogPostRepository;
11. private readonly IRepository<BlogComment> \_blogCommentRepository;
12. private readonly IRepository<StoreMapping> \_storeMappingRepository;
13. private readonly CatalogSettings \_catalogSettings;
14. private readonly IEventPublisher \_eventPublisher;
16. #endregion
18. #region Ctor
20. public BlogService(IRepository<BlogPost> blogPostRepository,
21. IRepository<BlogComment> blogCommentRepository,
22. IRepository<StoreMapping> storeMappingRepository,
23. CatalogSettings catalogSettings,
24. IEventPublisher eventPublisher)
25. {
26. this.\_blogPostRepository = blogPostRepository;
27. this.\_blogCommentRepository = blogCommentRepository;
28. this.\_storeMappingRepository = storeMappingRepository;
29. this.\_catalogSettings = catalogSettings;
30. this.\_eventPublisher = eventPublisher;
31. }
33. #endregion
35. //省略其它代码...
36. }
37. }

我可以看到在Service里面我们指定了多个IRepository具体的泛型类型（IRepository<BlogPost>，IRepository<BlogComment>，IRepository<StoreMapping>），而IRepository<T>是通过[依赖注入](http://www.lanhusoft.com/Article/105.html)，研究里面代码我们可以知道是EfRepository<T>，也就是IRepository<BlogPost>这些真正的对应是（EfRepository<BlogPost>，EfRepository<BlogComment>，EfRepository<StoreMapping>）,，这样在Service里面就可以通过这些IRepository相应的变量进行操作数据了。

（14）

（15）