NHibernate开发架构开发文档

作者：yangxie (谢炀)

文档最后更新：2008-05-03

文档版本号：1.0.0

Copyright 2006-2008 Telamon Global Services

[1 概述 4](#_Toc230418285)

[1.1 简介 4](#_Toc230418286)

[2 创建一个入门项目 5](#_Toc230418287)

[2.1 创建数据库 5](#_Toc230418288)

[2.1.1 创建示例项目 5](#_Toc230418289)

[2.1.2 配置代码生成工具并生成代码 11](#_Toc230418290)

[2.1.3 配置系统设置并运行示例 23](#_Toc230418291)

[2.1.4 完成一个最简单的CURD（增删改查）功能 26](#_Toc230418292)

[3 代码生成工具的使用 34](#_Toc230418293)

[3.1 MyGeneration简介 34](#_Toc230418294)

[2. NHibernate架构表现层代码模版：用于自动生成系统中常用的增删改查页面。 35](#_Toc230418295)

[3.2 NHibernate架构表代码模版介绍 35](#_Toc230418296)

[3.2.1 使用简介 35](#_Toc230418297)

[3.2.2 配置参数详解 38](#_Toc230418298)

[3.3 NHibernate架构表现层代码模版 45](#_Toc230418299)

[3.3.1 使用简介 45](#_Toc230418300)

[3.3.2 配置参数详解 45](#_Toc230418301)

[4 系统配置文件的设置 45](#_Toc230418302)

[4.1 简介 45](#_Toc230418303)

[4.2 Web.Config 配置 45](#_Toc230418304)

[4.3 数据库配置 45](#_Toc230418305)

[4.4 Spring容器配置 45](#_Toc230418306)

[4.5 日志配置 45](#_Toc230418307)

[5 框架调用说明 45](#_Toc230418308)

[5.1 框架结构简介 45](#_Toc230418309)

[5.2 实体层类 46](#_Toc230418310)

[5.3 数据层类 47](#_Toc230418311)

[5.3.1 数据层类简介 47](#_Toc230418312)

[5.3.2 数据修改操作 47](#_Toc230418313)

[5.3.3 数据查询操作 47](#_Toc230418314)

[5.4 业务层类 47](#_Toc230418315)

[5.4.1 业务层类简介 47](#_Toc230418316)

[5.4.2 事务管理 47](#_Toc230418317)

[5.5 Ado.Net数据操作支持 47](#_Toc230418318)

[5.5.1 执行SQL语句 47](#_Toc230418319)

[5.5.2 执行存储过程 47](#_Toc230418320)

# 概述

## 简介

本开发框架是一个集成了Spring.NET & NHibernate的WEB快速开发框架，包括框架基础库以及与之配套的代码生成工具。

框架使用开源组件NHibernate 2.0.1访问数据，使用Spring.Net 1.2.0 做为容器框架管理数据层、业务层、表现层之间的耦合关系。

框架使用到以下第三方组件：

* antlr.runtime.dll
* Castle.Core.dll
* Castle.DynamicProxy2.dll
* Common.Logging.dll
* Common.Logging.Log4Net.dll
* Iesi.Collections.dll
* log4net.dll
* NHibernate.dll
* Spring.Aop.dll
* Spring.Core.dll
* Spring.Data.dll
* Spring.Data.NHibernate20.dll
* Spring.Web.dll

框架包含两个配套的代码生成工具分别用于辅助生成后台代码以及表现层页面。

框架功能：

1. 支持在类的方法上面打标签控制事务。
2. 支持类似Linq的动态条件构造查询。
3. 配套的代码生成工具可以自动根据数据库表结构生辰所有的业务层、数据层类。

# 创建一个入门项目

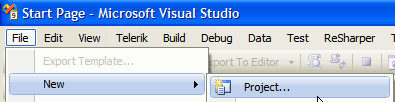
下面创建一个基于微软Northwind的数据库操作例子来简要的介绍一下如何使用框架。

## 创建数据库

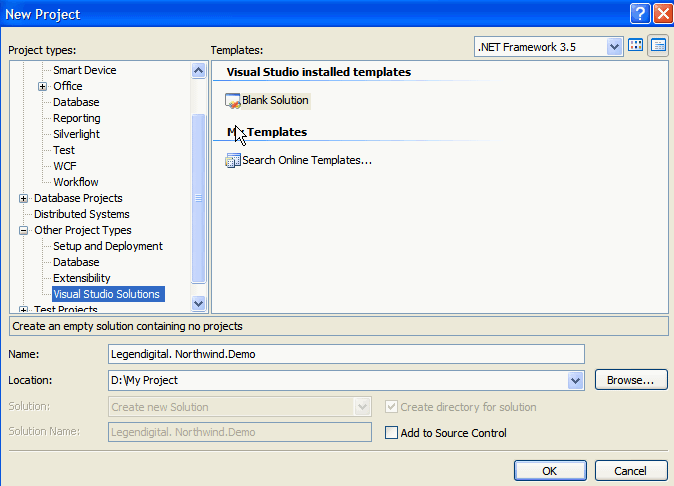
安装Northwind数据库，附加Northwind到数据库

### 创建示例项目

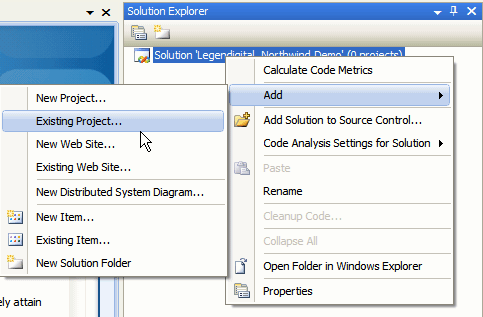
1. 创建一个空的Legendigital. Northwind.Demo解决方案
   * 点击菜单”File->New->Project”

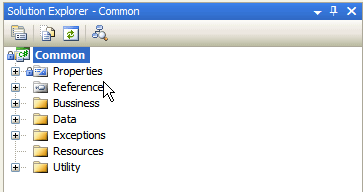


* + 选择”Other Project Type->Visual Stuido Solution-> Blank Solution”



1. 添加Legendigital.Framework.Common项目到解决方案
   * 添加框架基础库项目引用到解决方案

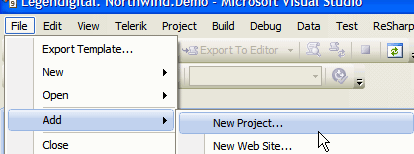




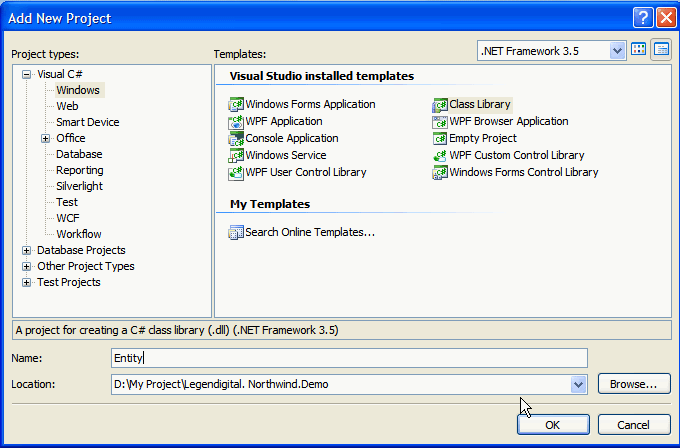
1. 创建实体层项目Legendigital. Northwind.Entity

接下来需要创建实体层项目

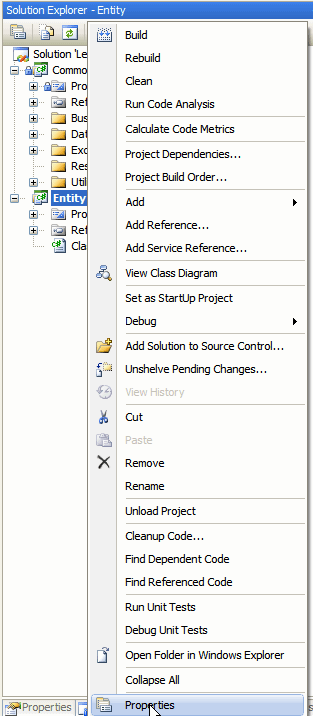
* + 点击菜单”File->Add->New Project”



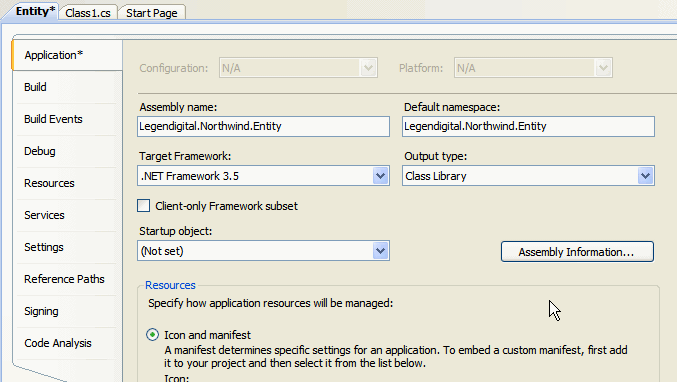
* + 创建名为Entity的Class Library项目



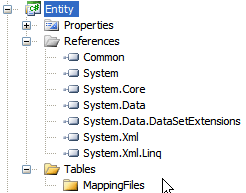
* + 在解决方面窗口里面选择Entity项目右键点击选择属性，打开项目属性设置窗口。



* + 修改默认名称空间名和默认程序集名修改为Legendigital. Northwind.Entity。按Ctrl+S进行保存。



* + Entity项目添加对Common项目的引用。
  + 删除自动生成的class1.cs
  + 在项目里面创建Tables—> MappingFiles目录

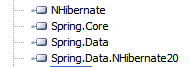


1. 创建数据层项目Legendigital. Northwind.Data

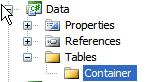
* 点击菜单”File->Add->New Project”
* 创建名为Data的Class Library项目
* 在解决方面窗口里面选择Data项目右键点击选择属性，打开项目属性设置窗口。
* 修改默认名称空间名和默认程序集名修改为Legendigital.Northwind.Data。按Ctrl+S进行保存。
* Data项目添加对Common项目以及Entity项目的引用。

Snap11.gif

* 确保以下第三方Dll被Data项目引用。



* 删除自动生成的class1.cs
* 在项目中创建Table->Container目录

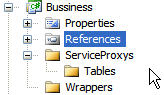


1. 创建业务层项目Legendigital. Northwind. Bussiness

* 点击菜单”File->Add->New Project”
* 创建名为Bussiness的Class Library项目
* 在解决方面窗口里面选择Bussiness项目右键点击选择属性，打开项目属性设置窗口。
* 修改默认名称空间名和默认程序集名修改为Legendigital.Northwind.Bussiness。按Ctrl+S进行保存。
* Bussiness项目添加对Common项目以及Entity项目Data项目的引用。
* Snap13.gif
* 确保以下第三方Dll被Bussiness项目引用。

Snap14.gif

* 删除自动生成的class1.cs
* 在项目里面创建ServiceProxys-> Tables, Wrappers目录

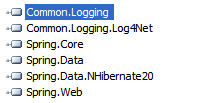


1. 创建Web表现层项目Legendigital. Northwind. Web

* 点击菜单”File->Add->New Project”
* 创建名为Web的Web Application项目
* 在解决方面窗口里面选择Bussiness项目右键点击选择属性，打开项目属性设置窗口。
* 修改默认名称空间名和默认程序集名修改为Legendigital. Northwind. Bussiness。按Ctrl+S进行保存。
* Bussiness项目添加对Common项目以及Entity项目还有Data项目Bussiness项目的引用。

Snap17.gif

* 确保以下第三方Dll被Web项目引用。

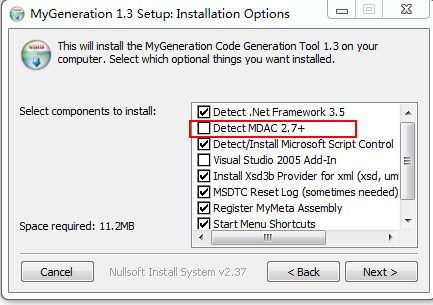


* 删除自动生成的Default.aspx
* 将Web项目设置为启动项目

### 配置代码生成工具并生成代码

1. 安装MyGeneration

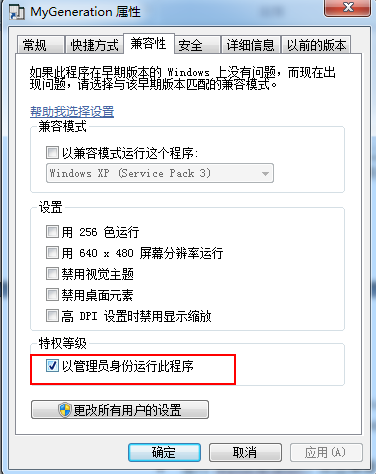
* 运行mygeneration\_1309\_20081006.exe安装MyGeneration。
* 如果是在windows 7 系统下面安装MyGeneration，取消Detect MDAC 2.7+ 选项

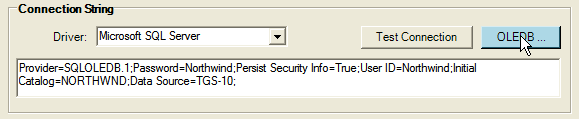


1. 将Legendigital.Code.MyGenAddin组件拷贝到MyGeneration安装目录下面

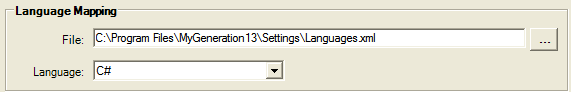
* Legendigital.Code.MyGenAddin.dll拷贝到MyGeneration安装目录(如： C:\Program Files\MyGeneration13) 下面

1. 配置MyGeneration

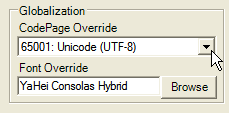
* 运行MyGeneration，点击菜单File->Default Settings
* 如果是Window 7系统，请修改MyGeneration快捷方式设置为管理员身份运行，已确保程序有访问配置文件的权限。
* 
* 设置数据库连接，Drivers选择Microsoft Sql Server，点击OleDb设置数据库连接，链接到本地的Northwind数据库。



* Lanauage mapping 里面设置Lanauage为C#

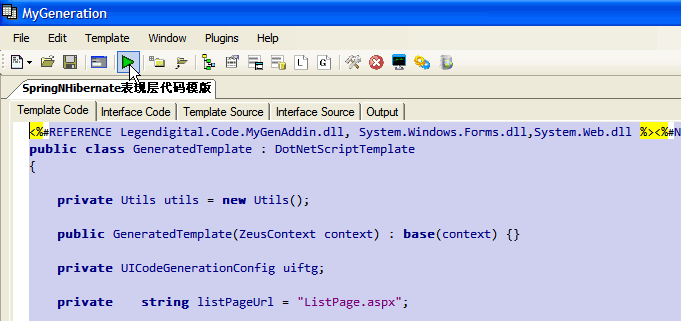


* 点击Templates选项卡设置Globalization里面的codepage override为 UTF-8

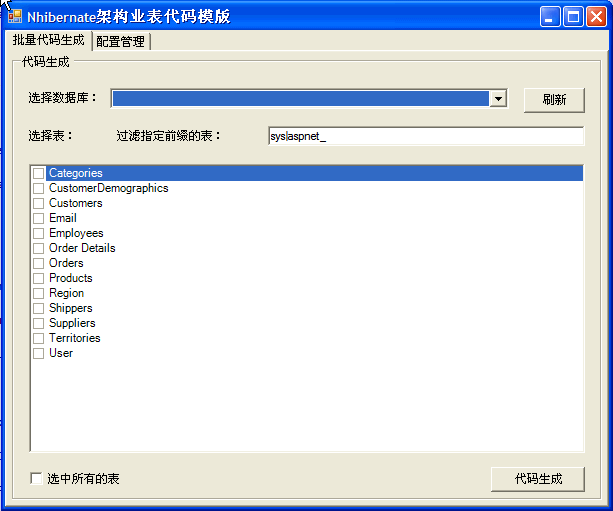


1. 运行代码模板并配置

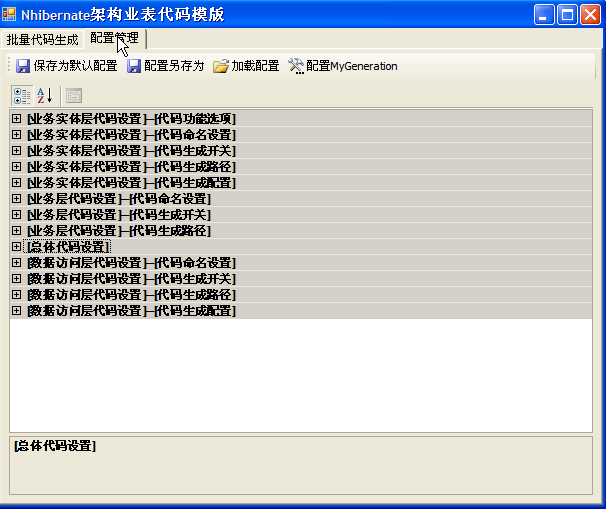
* 点击菜单File->Open打开代码生成模板文件“SpringNHibernate开发架构表代码模板.zeus”
* 点击运行按钮，运行模板。



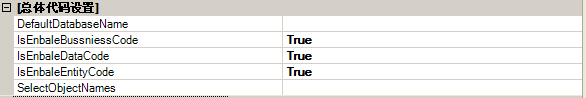
* 弹出代码生成窗口



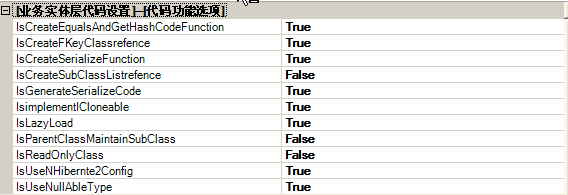
* 点击配置管理选项卡，对代码生成进行设置



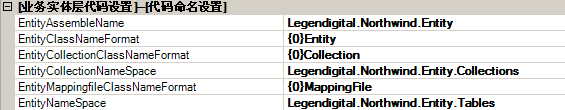
* 总体代码设置



* DefaultDatabaseName 不设置
* IsEnbaleBussniessCode 设置为 **True**
* IsEnbaleDataCode 设置为 **True**
* IsEnbaleEntityCode 设置为 **True**
* SelectObjectNames 不设置
* 业务实体层代码设置-代码功能选项设置



* IsCreateEqualsAndGetHashCodeFunction 设置为 **True**
* IsCreateFKeyClassrefence 设置为 **True**
* IsCreateSerializeFunction 设置为 **True**
* IsCreateSubClassListrefence 设置为 **False**
* IsimplementICloneable 设置为 **True**
* IsGenerateSerializeCode 设置为 **True**
* IsLazyLoad 设置为 **True**
* IsParentClassMaintainSubClass 设置为**False**
* IsReadOnlyClass 设置为**False**
* IsUseNHibernte2Config 设置为 **True**
* IsUseNullAbleType 设置为 **True**
* 业务实体层代码设置-代码命名设置



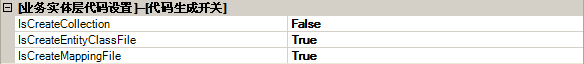
* EntityAssembleName 设置为**Legendigital.Northwind.Entity**
* EntityClassNameFormat 设置为 **{0}Entity**
* EntityCollectionClassNameFormat 设置为 **{0}Collection**
* EntityCollectionNameSpace 设置为

**Legendigital.Northwind.Entity. Collections**

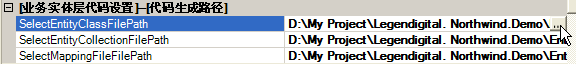
* EntityMappingfileClassNameFormat 设置为**{0}MappingFile**
* EntityNameSpace 设置为

**Legendigital.Northwind.Entity. Tables**

* 业务实体层代码设置-代码生成开关



* IsCreateCollection 设置为 **False**
* IsCreateEntityClassFile 设置为 **True**
* IsCreateMappingFile 设置为 **True**
* 业务实体层代码设置-代码生成路径



* SelectEntityClassFilePath 设置为

**项目中对应实体类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Entity\Tables*）**

* SelectEntityCollectionFilePath 设置为

**项目中对应实体类配置文件夹目录**

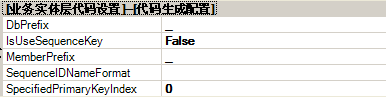
**（如：*D:\My Project\Legendigital. Northwind.Demo\Entity\Tables\Collections*）**

* SelectMappingFileFilePath 设置为

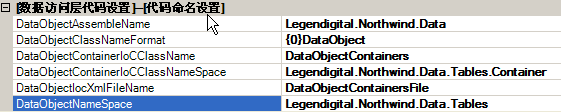
**项目中对应实体类集合类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Entity\Tables\MappingFiles*）**

* 业务实体层代码设置-代码生成配置



* DbPrefix 设置为 **\_**
* IsUseSequenceKey 设置为 **False**
* MemberPrefix 设置为 **\_**
* SequenceIDNameFormat 不设置
* SpecifiedPrimaryKeyIndex 设置为 **0**
* 数据访问层代码设置--代码命名设置



* DataObjectAssembleName 设置为***Legendigital.Northwind.Data***
* DataObjectClassNameFormat 设置为

***{0}DataObject***

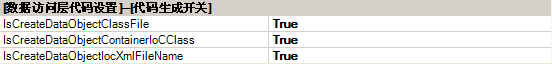
* DataObjectContainerIoCClassName 设置为

***DataObjectContainers***

* DataObjectContainerIoCClassNameSpace 设置为***Legendigital.Northwind.Data.Tables.Container***
* DataObjectIocXmlFileName 设置为

***DataObjectContainersFile***

* DataObjectNameSpace 设置为***Legendigital.Northwind.Data.Tables***
* 数据访问层代码设置--代码生成开关



* IsCreateDataObjectClassFile 设置为**True**
* IsCreateDataObjectContainerIoCClass 设置为**True**
* IsCreateDataObjectIocXmlFileName 设置为**True**
* 数据访问层代码设置--代码生成路径

Snap57.gif

* DataObjectContainerIocClassFilePath 设置为

**项目中对应数据层容器类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Data\Tables\Container*）**

* DataObjectIocXmlFilePath 设置为

**项目中对应数据层容器配置文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Data\Tables\Container*）**

* SelectDataObjectClassFilePath 设置为

**项目中对应数据层类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Data\Tables*）**

* 数据访问层代码设置--代码生成配置

Snap38.gif

* *ImportNameSpace* 设置为

***using System;***

***using System.Collections.Generic;***

***using System.Text;***

***using NHibernate.Criterion;***

***using Legendigital.Framework.Common.Data.NHibernate.DynamicQuery;***

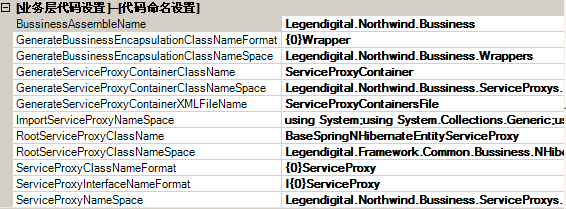
* *RootDataObjectName* 设置为

***BaseNHibernateDataObject***

* *RootDataObjectNameSpace* 设置为

***Legendigital.Framework.Common.Data.NHibernate***

* 业务层代码设置--代码命名设置



参数设置

* *BussinessAssembleName*  设置为

***Legendigital.Northwind.Bussiness***

* *GenerateBussinessEncapsulationClassNameFormat*  设置为

***{0}Wrapper***

* *GenerateBussinessEncapsulationClassNameSpace* 设置为

***Legendigital.Northwind.Bussiness.Wrappers***

* *GenerateServiceProxyContainerClassName* 设置为

***ServiceProxyContainer***

* *GenerateServiceProxyContainerClassNameSpace*  设置为

***Legendigital.Northwind.Bussiness.ServiceProxys.Tables.Container***

* *GenerateServiceProxyContainerXMLFileName*  设置为***ServiceProxyContainersFile***
* ImportServiceProxyNameSpace 设置为

***using System;***

***using System.Collections.Generic;***

***using System.Text;***

***using Legendigital.Framework.Common.Bussiness.Interfaces;***

***using Legendigital.Framework.Common.Data.Interfaces;***

* RootServiceProxyClassName 设置为***BaseSpringNHibernateEntityServiceProxy***
* RootServiceProxyClassNameSpace 设置为

***Legendigital.Framework.Common.Bussiness.NHibernate***

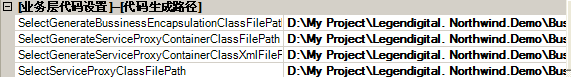
* ServiceProxyClassNameFormat 设置为

***{0}ServiceProxy***

* ServiceProxyInterfaceNameFormat 设置为***I{0}ServiceProxy***
* ServiceProxyClassNameSpace 设置为***Legendigital.Northwind.Bussiness.ServiceProxys.Tables***
* 业务层层代码设置--代码生成开关

Snap59.gif

* IsCreateServiceProxyClassFile 设置为True
* IsGenerateBussinessEncapsulationClass设置为True
* IsGenerateServiceProxyContainerClass 设置为True
* 业务层层代码设置--代码生成路径



* SelectGenerateBussinessEncapsulationClassFilePath 设置为

**项目中对应数据层容器类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\Wrappers*）**

* SelectGenerateServiceProxyContainerClassFilePath 设置为

**项目中对应数据层容器配置文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\ServiceProxys\Tables\Container*）**

* SelectGenerateServiceProxyContainerClassXmlFilePath 设置为

**项目中对应数据层类文件夹目录**

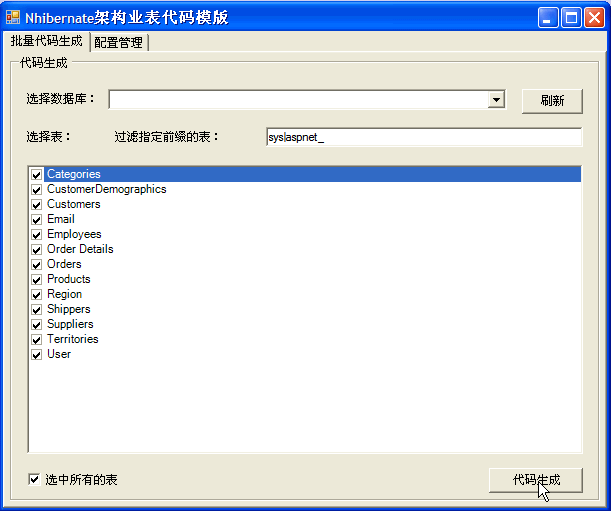
**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\ServiceProxys\Tables\Container*）**

* SelectServiceProxyClassFilePath 设置为

**项目中对应数据层类文件夹目录**

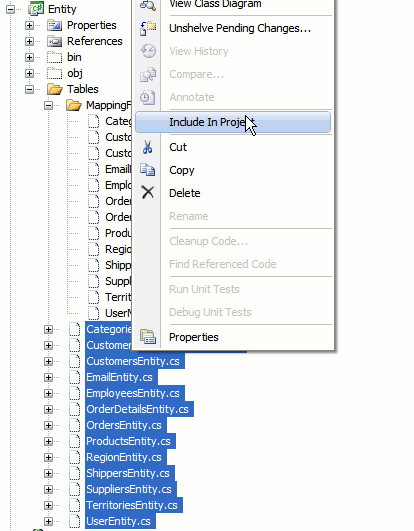
**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\ServiceProxys\Tables*）**

1. 自动生成业务层实体层数据层代码

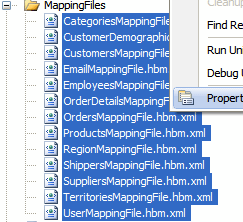


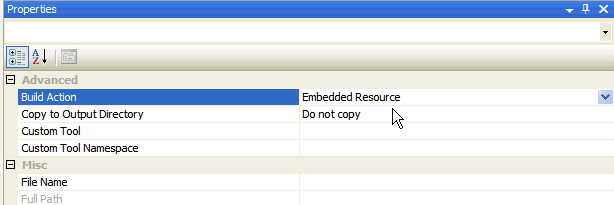
选中需要生成代码的表点击代码生成按钮，生成所有的代码。

将生成的代码包含到项目中



所有的xml文件请设置为嵌入式资源（Embedded Resources）





编译项目，如果无错，代表代码生成成功

### 配置系统设置并运行示例

1. 配置Web.Config
   1. 在<configuration><configSections>配置节中添加以下配置节定义

|  |
| --- |
| <sectionGroup name="common">  <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging"/>  </sectionGroup>  <sectionGroup name="spring">  <section name="context" type="Spring.Context.Support.WebContextHandler, Spring.Web"/>  <section name="parsers" type="Spring.Context.Support.NamespaceParsersSectionHandler, Spring.Core"/>  </sectionGroup> |

* 1. 在<appSettings>中添加以下配置

|  |
| --- |
| <add key="Spring.Data.NHibernate.Support.OpenSessionInViewModule.SessionFactoryObjectName" value="NHibernateSessionFactory"/> |

以上配置节用于定义Spring.Net集成NHibernate调用NHibernate的SessionFactory名字。

* 1. 在<configuration>中添加一下配置

|  |
| --- |
| <common>  <logging>  <factoryAdapter type="Common.Logging.Log4Net.Log4NetLoggerFactoryAdapter, Common.Logging.Log4Net">  <!-- choices are INLINE, FILE, FILE-WATCH, EXTERNAL-->  <!-- otherwise BasicConfigurer.Configure is used -->  <!-- log4net configuration file is specified with key configFile-->  <arg key="configType" value="FILE-WATCH"/>  <arg key="configFile" value="~/Config/Log4Net.xml"/>  </factoryAdapter>  </logging>  </common>  <spring>  <parsers>  <parser type="Spring.Data.Config.DatabaseNamespaceParser, Spring.Data"/>  <parser type="Spring.Transaction.Config.TxNamespaceParser, Spring.Data"/>  </parsers>  <context>  <resource uri="~/Config/DataConfig.xml"/>  <resource uri="assembly://Legendigital.Northwind.Data/Legendigital.Northwind.Data.Tables.Container/DataObjectContainersFile.xml"/>  <resource uri="assembly://Legendigital.Northwind.Bussiness/Legendigital.Northwind.Bussiness.ServiceProxys.Tables.Container/ServiceProxyContainersFile.xml"/>  <resource uri="~/Config/WebPage.xml"/>  </context>  </spring> |

assembly://Legendigital.Northwind.Data/Legendigital.Northwind.Data.Tables.Container/DataObjectContainersFile.xml

格式为assembly://应用程序集名/容器xml文件对应的名称空间名/容器xml文件

* 1. 在<pages><controls>配置里面添加以下配置

|  |
| --- |
| <add tagPrefix="spring" namespace="Spring.Web.UI.Controls" assembly="Spring.Web"/> |

* 1. 在<httpHandlers>配置里面添加以下配置

|  |
| --- |
| <add verb="\*" path="\*.aspx" type="Spring.Web.Support.PageHandlerFactory, Spring.Web"/>  <add verb="\*" path="\*.asmx" type="Spring.Web.Services.WebServiceHandlerFactory, Spring.Web"/>  <add verb="\*" path="ContextMonitor.ashx" type="Spring.Web.Support.ContextMonitor, Spring.Web"/> |

* 1. 在<httpModules>配置里面添加以下配置

|  |
| --- |
| <add name="Spring" type="Spring.Context.Support.WebSupportModule, Spring.Web"/>  <add name="OpenSessionInView" type="Spring.Data.NHibernate.Support.OpenSessionInViewModule, Spring.Data.NHibernate20"/> |

* 1. 在<system.web>配置里面添加以下配置

|  |
| --- |
| <httpRuntime enableKernelOutputCache="false"/> |

1. 配置数据库

在网站根目录创建Config文件夹

拷贝DataConfig.xml文件到目录里

修改以下设置

|  |
| --- |
| <db:provider id="DbProvider"  provider="SqlServer-2.0"  connectionString="Persist Security Info=False;Integrated Security=true;Initial Catalog= Northwind;server=(local)"/> |

改成对应数据库连接

|  |
| --- |
| <object id="NHibernateSessionFactory" type="Spring.Data.NHibernate.LocalSessionFactoryObject, Spring.Data.NHibernate20">  <property name="DbProvider" ref="DbProvider"/>  <property name="MappingAssemblies">  <list>  <value>Legendigital.Northwind.Entity</value>  </list>  </property>  <value>Legendigital.Northwind.Entity</value> |

Legendigital.Northwind.Entity改成对应实体项目名称应用程序集名

1. 配置日志

将Log4Net.xml文件拷贝到Config文件夹

1. 运行系统

创建一个default.Aspx页面

页面里面拖入一个GridView控件

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml" >  <head runat="server">  <title></title>  </head>  <body>  <form id="form1" runat="server">  <div>  <asp:GridView ID="GridView1" runat="server">  </asp:GridView>  </div>  </form>  </body>  </html> |

在页面的page\_load方法中加入以下代码

|  |
| --- |
| protected void Page\_Load(object sender, EventArgs e)  {  this.GridView1.DataSource = ProductsWrapper.FindAll();  this.GridView1.DataBind();  } |

运行页面，页面成功显示数据代表配置例子配置成功

### 完成一个最简单的CURD（增删改查）功能

1. 编写列表页面

创建一个List.aspx页面

List.aspx页面Html代码

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head runat="server">  <title></title>  </head>  <body>  <form id="form1" runat="server">  <div>  <a href=Add.aspx>添加</a>  <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"  DataKeyNames="CategoryID" onrowcommand="GridView1\_RowCommand">  <Columns>  <asp:BoundField HeaderText="CategoryID" DataField="CategoryID" />  <asp:BoundField HeaderText="CategoryName" DataField="CategoryName" />  <asp:BoundField HeaderText="Description" DataField="Description" />  <asp:TemplateField HeaderText="操作">  <ItemTemplate>  <asp:HyperLink ID="linkEdit" runat="server" NavigateUrl='<%# "Edit.aspx?ID=" + Eval("CategoryID").ToString() %>'  Text="更新"></asp:HyperLink>&nbsp;  </ItemTemplate>  <ItemStyle HorizontalAlign="Center" />  </asp:TemplateField>  <asp:TemplateField HeaderText="删除">  <ItemTemplate>  <asp:LinkButton ID="lbtnDelete" runat="server" CausesValidation="False"  CommandName="cmdDelete" Text="删除"></asp:LinkButton>  </ItemTemplate>  </asp:TemplateField>  </Columns>  </asp:GridView>  </div>  </form>  </body>  </html> |

后台代码

|  |
| --- |
| public partial class List : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  if(this.Page.IsPostBack)  return;  BindList();  }  private void BindList()  {  this.GridView1.DataSource = CategoriesWrapper.FindAll();  this.GridView1.DataBind();  }  protected void GridView1\_RowCommand(object sender, GridViewCommandEventArgs e)  {  GridViewRow drv = ((GridViewRow)(((LinkButton)(e.CommandSource)).Parent.Parent)); //此得出的值是表示那行被选中的索引值  int id = Convert.ToInt32(GridView1.DataKeys[drv.RowIndex].Value); //此获取的值为GridView中绑定数据库中的主键值  CategoriesWrapper.DeleteByID(id);  BindList();  }  } |

1. 编写添加页面

创建Add.aspx 页面

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head runat="server">  <title></title>  </head>  <body>  <form id="form1" runat="server">  <table width="100%" cellpadding="1" cellspacing="3" border="0">  <tr align="center" valign="middle">  <td colspan="4" class="table\_body">  Categories  </td>  </tr>  <tr>  <td style="width: 15%;" class="table\_body" align="left">  CategoryName:  </td>  <td style="width: 35%;" class="table\_none">  <asp:TextBox ID="txtCategoryName" ToolTip="输入CategoryName" MaxLength='15' runat="server"  ValidationGroup="vgCategories"></asp:TextBox>  </td>  <td style="width: 15%;" class="table\_body" align="left">  Description:  </td>  <td style="width: 35%;" class="table\_none">  <asp:TextBox ID="txtDescription" ToolTip="输入Description" MaxLength='1073741823' runat="server"  ValidationGroup="vgCategories"></asp:TextBox>  <asp:RequiredFieldValidator ID="rfvtxtDescription" runat="server" ControlToValidate="txtDescription"  Display="None" ErrorMessage="【Description】项不能为空！" SetFocusOnError="True" ValidationGroup="vgCategories"></asp:RequiredFieldValidator>  </td>  </tr>  <tr>  <td align="left" class="table\_body" style="width: 15%">  </td>  <td class="table\_none" colspan="3">  <asp:Label ID="lblMessage" runat="server" EnableViewState="False" ForeColor="Red"></asp:Label>  </td>  </tr>  <tr>  <td align="left" class="table\_body" style="width: 15%">  </td>  <td class="table\_none" colspan="3">  <asp:Button ID="btnAdd" runat="server" OnClick="btnAdd\_Click" Text="创建" ValidationGroup="vgCategories"  Width="57px" />&nbsp;  </td>  </tr>  </table>  </form>  </body>  </html> |

Add.aspx页面后台代码

|  |
| --- |
| public partial class Add : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  }  protected void btnAdd\_Click(object sender, EventArgs e)  {  //验证不通过返回  if (!this.Page.IsValid)  return;  CategoriesWrapper obj = new CategoriesWrapper();  obj.CategoryName = this.txtCategoryName.Text.Trim();  obj.Description = this.txtDescription.Text.Trim();  //添加数据  try  {  CategoriesWrapper.Save(obj);  this.Response.Redirect("List.aspx");  }  catch (Exception e1)  {  this.lblMessage.Text = "添加数据失败，错误原因：" + e1.Message;  }  }  } |

1. 编写修改页面

添加Edit.Aspx页面

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml" >  <head runat="server">  <title></title>  </head>  <body>  <form id="form1" runat="server">  <table width="100%" cellpadding="1" cellspacing="3" border="0">  <tr align="center" valign="middle">  <td colspan="4" class="table\_body">Categories</td>  </tr>  <tr>  <td style="width:15%;" class="table\_body" align="left">CategoryName:</td>  <td style="width:35%;" class="table\_none">  <asp:textbox id="txtCategoryName" ToolTip="输入CategoryName" MaxLength='15' runat="server" ValidationGroup="vgCategories"></asp:textbox>  </td>  <td style="width:15%;" class="table\_body" align="left">Description:</td>  <td style="width:35%;" class="table\_none">  <asp:textbox id="txtDescription" ToolTip="输入Description" MaxLength='1073741823'  runat="server" ValidationGroup="vgCategories" style="margin-top: 0px"></asp:textbox>  <asp:RequiredFieldValidator ID="rfvtxtDescription" runat="server" ControlToValidate="txtDescription" Display="None" ErrorMessage="【Description】项不能为空！" SetFocusOnError="True" ValidationGroup="vgCategories"></asp:RequiredFieldValidator>  </td>  </tr>  <tr>  <td align="left" class="table\_body" style="width: 15%"></td>  <td class="table\_none" colspan="3"><asp:Label ID="lblMessage" runat="server" EnableViewState="False" ForeColor="Red"></asp:Label></td>  </tr>  <tr>  <td align="left" class="table\_body" style="width: 15%">  </td>  <td class="table\_none" colspan="3">  <asp:Button ID="btnEdit" runat="server" OnClick="btnEdit\_Click" Text="更新" ValidationGroup="vgCategories" Width="56px" />&nbsp;  </td>  </tr>  </table>  </form>  </body>  </html> |

Edit页面后台代码

|  |
| --- |
| public partial class Edit : Page  {  private CategoriesWrapper CurrentData  {  get  {  if (Context.Items["CurrentData"] == null)  return null;  else  return (CategoriesWrapper)Context.Items["CurrentData"];  }  set { Context.Items["CurrentData"] = value; }  }  private int GetID  {  get  {  try  {  return int.Parse(Request.QueryString["ID"]);  }  catch  {  return 0;  }  }  }  protected void Page\_Load(object sender, EventArgs e)  {  int id = GetID;  if (id == 0)  {  Response.Write("无法获取ID");  return;  }  CurrentData = CategoriesWrapper.FindById(id);  if (CurrentData == null)  {  Response.Write("该条数据已不存在ID");  return;  }  if (Page.IsPostBack)  return;  txtCategoryName.Text = CurrentData.CategoryName.ToString();  txtDescription.Text = CurrentData.Description.ToString();  }  protected void btnEdit\_Click(object sender, EventArgs e)  {  //验证不通过返回  if (!Page.IsValid)  return;  CategoriesWrapper obj = CurrentData;  obj.CategoryName = txtCategoryName.Text.Trim();  obj.Description = txtDescription.Text.Trim();  //添加数据  try  {  CategoriesWrapper.Update(obj);  this.Response.Redirect("List.aspx");  }  catch (Exception e1)  {  lblMessage.Text = "更新数据失败，错误原因：" + e1.Message;  }  }  } |

1. 运行系统

把List.aspx设置为起始页，运行系统。这样入门项目就成功的创建了。

# 代码生成工具的使用

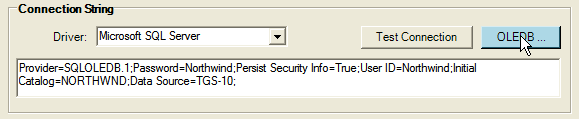
## MyGeneration简介

MyGeneration 是一款优秀的开源代码生成工具引擎，它基于模板（Template）工作，MyGeneration本身提供强大的数据库库结构解析库，支持多种主流的数据库，用户可以再MyGeneration基础上编写代码模板用来生成，代码、脚本、甚至是asp.net页面。

本框架采用的是最新的MyGeneration 1.3.0.9

MyGeneration的使用简介

* 连接数据库



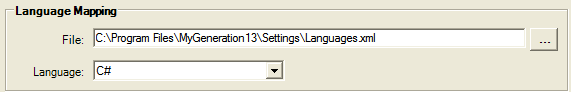
Drivers：是选择数据库类型

点击OleDb设置连接到数据库的数据库连接字符串。自行填写连接字符串的话请注意一定要是Oledb的方式连接数据库。

* 设置其他参数

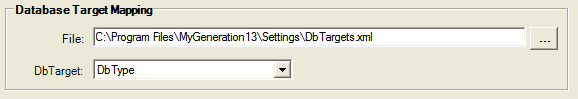
语言映射设置，Lanaguage代表需要生成代码语言类型C#、VB.Net

File代表语言映射包，不要随便修改。

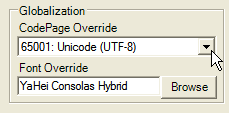


数据库映射设置，DbTarget代表需要Ado.Net的参数数据库类型OleDb,SqlClient,DbType.

File代表数据库映射包，不要随便修改。

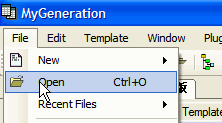


设置模板文字编码以及模板字体。



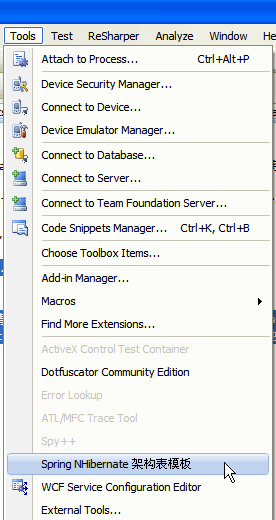
* 打开运行模板

可以通过直接双击模板文件或者在MyGeneration的File->Open菜单中点击打开菜单模板文件。



* 编译模板并将模板集成到Vs.Net菜单里

1. 编译模板，打开模板文件，点击Template->Compile as,点击确定继续，将会弹出代码生成界面，点击代码生成按钮，将会弹出编译模板保存，选择一个新的文件名（xxxCompiled.zeus）保存编译好的模板(最好不要覆盖现有的未编译的模板).
2. 打开VS 的Tools-> External Tools.. 菜单.
3. 点击add.
4. 输入自定义的菜单名.
5. "Command”项: 选择ZeusCmd.exe文件. 默认情况下该文件位于：C:\Program Files\MyGeneration13\ZeusCmd.exe
6. "Arguments”项目: -t "C:\complete\path\to\Template\File.zeus" 路径为编译好的模板地址。
7. 选中"Use Output Window" 单选框.
8. 单击OK 保存.
9. 这样就可以在VS 的Tools看到一个新的自定义的菜单项用于呼出模板了（无需打开MyGeneration生成代码）。



本框架包含两个代码模板：

1. NHibernate架构表代码模版：用于生成自动三层架构代码。

## NHibernate架构表现层代码模版：用于自动生成系统中常用的增删改查页面。

## NHibernate架构表代码模版介绍

### 使用简介

使用该模板前请确保将Legendigital.Code.MyGenAddin.dll文件拷贝到MyGeneration得安装目录（如：C:\Program Files\MyGeneration13）

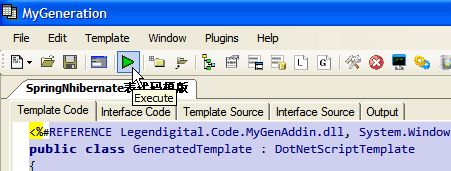
注意事项，系统中所有的自动生成的.Designer.cs文件如果已经存在，代码生成机下次生成代码的时候不会成代码覆盖，如需生成这部分文件，请先手动删除改文件再生成。

**SpringNHibernate开发架构表代码模板.zeus**

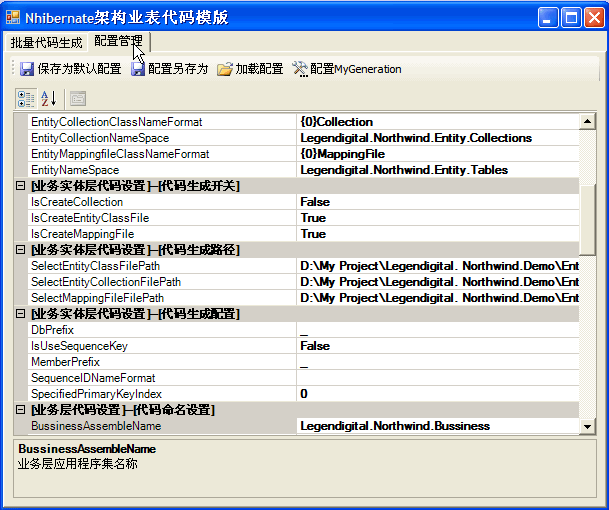
Snap71.gif

双击模板打开模板，或者用MyGeneration打开模板文件。

运行模板



点击配置设置选项卡



设置代码生成配置，设置完以后，点击**保存为默认配置**

Snap68.gif

其他按钮作用：

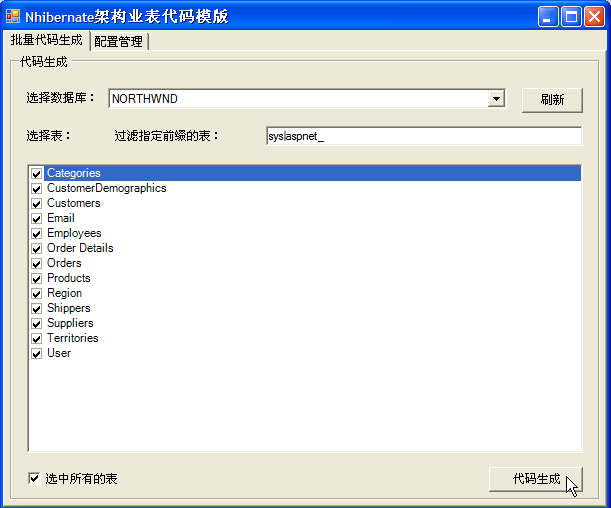
**配置另存为**：将当前配置保存为一个文件，以便以后使用的时候再次加载。

**加载配置**：直接加载以前保存过的配置文件。

**配置MyGeneration**：直接配置MyGeneration软件配置项。

配置完成以后，点击批量代码生成选项卡，选择需要生成代码的表，点击代码生成直接生成选中的表的代码。

过滤指定前缀的表：选择列表中过滤部分特定前缀的表（如系统表），出现选择框中，支持多个前缀名，多个前缀名之间用|分隔。



### 配置参数详解

#### 总体设置

* 总体代码设置
* DefaultDatabaseName

默认选中的数据库名,打开代码生成机默认选中数据名,(连接Oracle数据库使用，连接Sqlserver无需设置)

（例如：Northwind）.

* IsEnbaleBussniessCode

是否生成业务层代码

可用值（true| false）

* IsEnbaleDataCode

是否生成数据层代码

可用值（true| false）

* IsEnbaleEntityCode

是否生成业务实体代码

可用值（true| false）

* SelectObjectNames

默认选中的表，填写多个表名，打开代码生成机的时候将默认选中这些表。

（例如：Products

Categories

Orders

）.每个表之间用换行分隔。

#### 业务实体层代码生成设置

* 业务实体层代码设置-代码功能选项设置
* IsCreateEqualsAndGetHashCodeFunction

是否生成Equals以及GetHashCode方法（NHibernate用判断对象是否相等的代码）。

可用值（true| false）

* IsCreateFKeyClassrefence

是否将外键应用字段映射成外键所对应的类

可用值（true| false）

* IsCreateSerializeFunction

是否生成序列化方法

可用值（true| false）

* IsCreateSubClassListrefence

如果该表属于其他表的外键，为子表生成集合属性

可用值（true| false）

* IsimplementICloneable

是否实现ICloneable方法

可用值（true| false）

* IsGenerateSerializeCode

实体类是否可序列化

可用值（true| false）

* IsLazyLoad

是否可延迟加载

可用值（true| false）

* IsParentClassMaintainSubClass

是否父对象来维持父子关系

可用值（true| false）

* IsReadOnlyClass

是否为只读类

可用值（true| false）

* IsUseNHibernte2Config

是否使用NHibernte 2.0 配置方式

可用值（true| false）

* IsUseNullAbleType

是否支持可空类型。

可用值（true| false）

* 业务实体层代码设置-代码命名设置
* EntityAssembleName

实体层项目应用程序集名（例如：***Legendigital.Northwind.Entity***）

* EntityClassNameFormat

实体类命名格式 （例如：{0}Entity,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范））

* EntityCollectionClassNameFormat

实体集合类命名格式 （例如：{0}Collection,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范））

* EntityCollectionNameSpace

实体集合类所在名称空间名（例如：***Legendigital.Northwind.Entity.Tables.Collection***）

* EntityMappingfileClassNameFormat

实体类映射配置文件格式名 （例如：{0}MappingFile,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范））

* EntityNameSpace

实体类所在名称空间名（例如：***Legendigital.Northwind.Entity.Tables***）

* 业务实体层代码设置-代码生成开关
* IsCreateCollection

是否生成实体集合类。

可用值（true| false）

* IsCreateEntityClassFile

是否生成实体类。

可用值（true| false）

* IsCreateMappingFile

是否生成实体类映射文件。

可用值（true| false）

* 业务实体层代码设置-代码生成路径
* SelectEntityClassFilePath

**项目中对应实体类文件夹目录**

**（例如：*D:\My Project\Legendigital. Northwind.Demo\Entity\Tables*）**

* SelectEntityCollectionFilePath

**项目中对应实体类配置文件夹目录**

**（例如：*D:\My Project\Legendigital. Northwind.Demo\Entity\Tables\Collections*）**

* SelectMappingFileFilePath

**项目中对应实体类集合类文件夹目录**

**（例如：*D:\My Project\Legendigital. Northwind.Demo\Entity\Tables\MappingFiles*）**

* 业务实体层代码设置-代码生成配置
* DbPrefix

过滤表名前缀，需要过滤的数据库表名前缀,生成实体类名时自动去掉这些前缀，例如（tbl\_,sys\_，tb\_）。

* IsUseSequenceKey

是否使用Sequence生成主键（仅对Oracle之类支持Sequence的数据库有效）

* MemberPrefix

实体类成员变量前缀 （建议值:\_）

* SequenceIDNameFormat

如果使用Sequence生成主键，每一个表对应的Sequence命名规范（例如：{0}\_Sequence,{0}代表表名）。

* SpecifiedPrimaryKeyIndex

手动指定生成表的主键列（使用列索引查找，0代表第一列），该配置仅在该数据表不存在主键的时候生效。

#### 数据层代码生成设置

* 数据访问层代码设置--代码命名设置
* DataObjectAssembleName

数据层数据操作类项目应用程序集名（例如：***Legendigital.Northwind.Data***）

* DataObjectClassNameFormat

数据层数据操作类命名格式

（例如：***{0}DataObject***,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范）。

* DataObjectContainerIoCClassName

数据层数据操作类容器类名

（例如：***DataObjectContainers***）

* DataObjectContainerIoCClassNameSpace

数据层数据操作类容器类所在名称空间名

（例如：***Legendigital.Northwind.Data.Tables.Container***）

* DataObjectIocXmlFileName

数据层数据操作类容器配置文件名

（例如：***DataObjectContainersFile***）

* DataObjectNameSpace

数据层数据操作类所在名称空间名

（例如：***Legendigital.Northwind.Data.Tables***）

* 数据访问层代码设置--代码生成开关
* IsCreateDataObjectClassFile

是否生成数据操作类

可用值（true| false）

* IsCreateDataObjectContainerIoCClass

是否生成数据层数据操作类容器类

可用值（true| false）

* IsCreateDataObjectIocXmlFileName

是否生成数据层数据操作类容器配置文件

可用值（true| false）

* 数据访问层代码设置--代码生成路径
* DataObjectContainerIocClassFilePath

**项目中对应数据层容器类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Data\Tables\Container*）**

* DataObjectIocXmlFilePath

**项目中对应数据层容器配置文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Data\Tables\Container*）**

* SelectDataObjectClassFilePath

**项目中对应数据层类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Data\Tables*）**

* 数据访问层代码设置--代码生成配置
* *ImportNameSpace*

数据层数据操作类预导入名称空间，生成数据层数据操作类时，预先加入的名称空间。

例如：

***using System;***

***using System.Collections.Generic;***

***using System.Text;***

***using NHibernate.Criterion;***

***using Legendigital.Framework.Common.Data.NHibernate.DynamicQuery;***

* *RootDataObjectName*

数据层数据操作类基类名

***BaseNHibernateDataObject***

* *RootDataObjectNameSpace*

数据层数据操作类基类所在名称空间

***Legendigital.Framework.Common.Data.NHibernate***

#### 业务层代码生成设置

* 业务层代码设置--代码命名设置

参数设置

* *BussinessAssembleName*

业务层项目应用程序集名

（例如：***Legendigital.Northwind.Business***）

* *GenerateBussinessEncapsulationClassNameFormat*

业务层业务包装类命名规范

（例如：***{0} Wrapper***,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范）。

* *GenerateBussinessEncapsulationClassNameSpace*

业务层业务包装类所在名称空间名

（例如：***Legendigital.Northwind.Bussiness.Wrappers***）

* *GenerateServiceProxyContainerClassName*

业务层事务类容器类名

（例如：***ServiceProxyContainer***）

* *GenerateServiceProxyContainerClassNameSpace*

业务层事务类容器类所在名称空间名

（例如：***Legendigital.Northwind.Bussiness.ServiceProxys.Tables.Container***）

* *GenerateServiceProxyContainerXMLFileName*

业务层事务类容器类配置文件名

（例如：***ServiceProxyContainersFile***）

* ImportServiceProxyNameSpace

业务层事务类预导入名称空间。例如：

***using System;***

***using System.Collections.Generic;***

***using System.Text;***

***using Legendigital.Framework.Common.Bussiness.Interfaces;***

***using Legendigital.Framework.Common.Data.Interfaces;***

* RootServiceProxyClassName

业务层事务类基类名

（例如：***BaseSpringNHibernateEntityServiceProxy***）

* RootServiceProxyClassNameSpace

业务层事务类基类名称空间

（例如：***Legendigital.Framework.Common.Bussiness.NHibernate***）

* ServiceProxyClassNameFormat

业务层事务类命名规范

（例如：***{0}ServiceProxy***,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范）。

* ServiceProxyInterfaceNameFormat

业务层事务类接口命名规范

（例如：***I{0}ServiceProxy***,{0}代表处理过的数据库表名（去掉指定前缀，过滤表名中的空格，下划线，减号，应用PascalCase命名规范）。

* ServiceProxyClassNameSpace

业务层事务类所在名称空间名

（例如：***Legendigital.Northwind.Bussiness.ServiceProxys.Tables***）

* 业务层层代码设置--代码生成开关
* IsCreateServiceProxyClassFile

是否生成业务层事务类

可用值（true| false）

* IsGenerateBussinessEncapsulationClass

是否生成业务层业务包装类

可用值（true| false）

* IsGenerateServiceProxyContainerClass

是否生成业务层事务类容器类

可用值（true| false）

* 业务层层代码设置--代码生成路径
* SelectGenerateBussinessEncapsulationClassFilePath

**项目中对应数据层容器类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\Wrappers*）**

* SelectGenerateServiceProxyContainerClassFilePath

**项目中对应数据层容器配置文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\ServiceProxys\Tables\Container*）**

* SelectGenerateServiceProxyContainerClassXmlFilePath

**项目中对应数据层类文件夹目录**

**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\ServiceProxys\Tables\Container*）**

* SelectServiceProxyClassFilePath

**项目中对应数据层类文件夹目录**

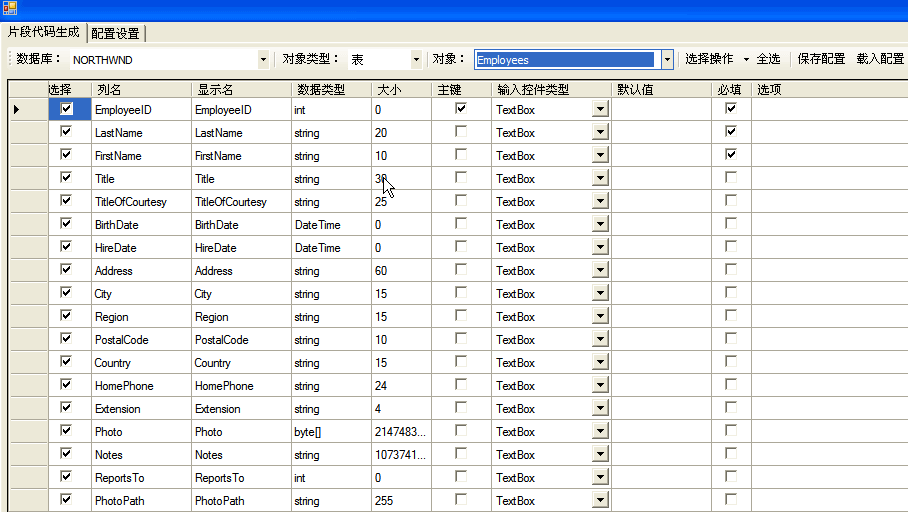
**（如：*D:\My Project\Legendigital. Northwind.Demo\Bussiness\ServiceProxys\Tables*）**

## NHibernate架构表现层代码模版

表现层代码模板用于生成系统中常用的增删改查页面，自动生成对应的页面的页面代码和后台代码。

### 使用简介

运行模板，打开代码生成页面。



选择好数据库和对应的表以后，下面就会列出这个表下面索引的数据库类。

设置以下属性以便生成界面代码。

选择：代表是否为该列生成代码（可修改）。

列名：数据库字段名。

显示名：字段对应的界面显示名称（可修改）。

数据类型：数据库字段对应C#语言的类型。

大小：字段长度值（仅对string有效）。

主键：该字段是否为主键。

输入控件类型：该字段添加编辑页面所使用的控件（可修改）。

默认值：代表在添加页面中这些字段出现在输入控件上的默认值。

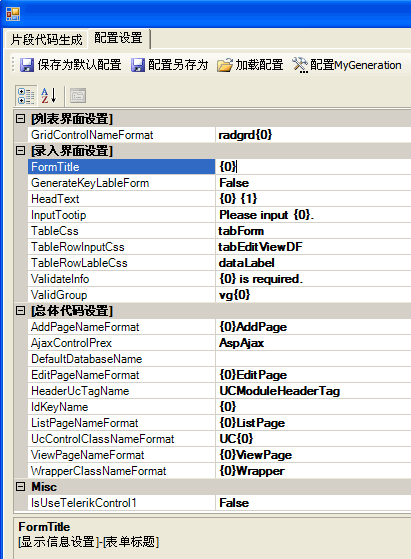
必填：代表是否生成验证控件提示该字段必填。

选项：如果是DropDownList控件的话，输入DropDownList所有的Item值，格式：

界面生成设置配好以后，可以点击保存配置，将界面生成配置保存成一个文件，以便以后加载。

点击加载配置可以直接加载以前保存过的界面生成配置文件。

点击配置设置，可以修改全局代码生成配置。



### 配置参数详解

* 列表界面设置
  + GridControlNameFormat：表格控件变量命名规范

例如：radgrd{0}

* 录入界面设置
  + FormTitle：表单标题名称。

例如：{0}

* + GenerateKeyLableForm：是否显示主键列。

可用值（true| false）

* + HeadText：表头导航文字。

例如：{0} {1}

* + InputTootip：输入提示信息。

例如：Please input {0}.

* + TableCss：录入表格CSS

例如：tabForm

* + TableRowInputCss：录入表格输入列CSS

例如：tabEditViewDF

* + TableRowLableCss：录入表格标签列CSS
  + 例如：dataLabel
  + ValidateInfo：字段必填验证信息。

例如：{0} is required.

* + ValidGroup：验证组命名规范

例如：vg{0}

* 总体代码设置
  + AddPageNameFormat：添加页面命名规范

例如：{0}AddPage

* + AjaxControlPrex：AjaxToolkit控件引用名

例如：AspAjax

* + DefaultDatabaseName：默认选中的数据库名

例如：Northwind

* + EditPageNameFormat：编辑页面命名规范

例如：{0}EditPage

* + HeaderUcTagName：表头用户控件引用名

例如：UCModuleHeaderTag

* + IdKeyName：主键查询字符串key名

例如：{0}

* + ListPageNameFormat：列表页面命名规范

例如：{0}ListPage

* + UcControlClassNameFormat：用户控件命名规范

例如：UC{0}

* + ViewPageNameFormat：查看页面命名规范

例如：{0}ViewPage

* + WrapperClassNameFormat：业务包装类命名规范

例如：{0}Wrapper

* 其他设置
  + isUseTelerikControl：是否使用Telerik控件

可用值（true| false）

# 系统使用开源项目简介以及配置设置

## 使用开源项目简介

框架使用Spring.Net、NHibernate、Log4net等开源项目，下面对这些开源项目做一个简要的介绍。

NHibernate是一个基于.Net,用于关系数据库的对象持久化类库.它是Java著名的Hibernate的.Net版本.

Log4net, 是 [apache.org](http://www.apache.org/) 在 log4j的基础上推出的针对.NET程序的开源的日志组件。支持多种方式记录日志。

Spring.NET是一个应用程序框架，其目的是协助开发人员创建企业级的.NET应用程序。它提供了很多方面的功能，比如依赖注入、面向方面编程（AOP）、数据访问抽象及ASP.NET扩展等等。Spring.NET以Java版的Spring框架为基础，将Spring.Java的核心概念与思想移植到了.NET平台上。

## Web.Config 配置

1. 在<configuration><configSections>配置节中添加以下配置节定义

|  |
| --- |
| <sectionGroup name="common">  <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging"/>  </sectionGroup>  <sectionGroup name="spring">  <section name="context" type="Spring.Context.Support.WebContextHandler, Spring.Web"/>  <section name="parsers" type="Spring.Context.Support.NamespaceParsersSectionHandler, Spring.Core"/>  </sectionGroup> |

上述代码定义web.config中使用的common、spring子配置节名。

1. 在<appSettings>中添加以下配置

|  |
| --- |
| <add key="Spring.Data.NHibernate.Support.OpenSessionInViewModule.SessionFactoryObjectName" value="NHibernateSessionFactory"/> |

以上配置节用于定义Spring.Net集成NHibernate调用NHibernate的SessionFactory名字。

1. 在<configuration>中添加一下配置

|  |
| --- |
| <common>  <logging>  <factoryAdapter type="Common.Logging.Log4Net.Log4NetLoggerFactoryAdapter, Common.Logging.Log4Net">  <!-- choices are INLINE, FILE, FILE-WATCH, EXTERNAL-->  <!-- otherwise BasicConfigurer.Configure is used -->  <!-- log4net configuration file is specified with key configFile-->  <arg key="configType" value="FILE-WATCH"/>  <arg key="configFile" value="~/Config/Log4Net.xml"/>  </factoryAdapter>  </logging>  </common>  <spring>  <parsers>  <parser type="Spring.Data.Config.DatabaseNamespaceParser, Spring.Data"/>  <parser type="Spring.Transaction.Config.TxNamespaceParser, Spring.Data"/>  </parsers>  <context>  <resource uri="~/Config/DataConfig.xml"/>  <resource uri="assembly://Legendigital.Northwind.Data/Legendigital.Northwind.Data.Tables.Container/DataObjectContainersFile.xml"/>  <resource uri="assembly://Legendigital.Northwind.Bussiness/Legendigital.Northwind.Bussiness.ServiceProxys.Tables.Container/ServiceProxyContainersFile.xml"/>  </context>  </spring> |

Logging配置节中定义了日志配置文件路径以及采用什么组件实现日志（Spring.Net支持很多日志组件）。

configFile代表Log4net日志组件配置文件的路径。

spring配置节里面放置容器实例配置文件的加载以及数据库访问配置文件。

***assembly://Legendigital.Northwind.Data/Legendigital.Northwind.Data.Tables.Container/DataObjectContainersFile.xml***

格式为***assembly://应用程序集名/容器xml文件对应的名称空间名/容器xml文件***

1. 在<pages><controls>配置里面添加以下配置

|  |
| --- |
| <add tagPrefix="spring" namespace="Spring.Web.UI.Controls" assembly="Spring.Web"/> |

添加spring.Net控件注册

1. 在<httpHandlers>配置里面添加以下配置

|  |
| --- |
| <add verb="\*" path="\*.aspx" type="Spring.Web.Support.PageHandlerFactory, Spring.Web"/>  <add verb="\*" path="\*.asmx" type="Spring.Web.Services.WebServiceHandlerFactory, Spring.Web"/>  <add verb="\*" path="ContextMonitor.ashx" type="Spring.Web.Support.ContextMonitor, Spring.Web"/> |

添加spring.Net的 httpHandler以便可以管理web页面实例，ContextMonitor.ashx代表一个容器监视程序，可以在页面上打开查看容器里面有哪些实例。

1. 在<httpModules>配置里面添加以下配置

|  |
| --- |
| <add name="Spring" type="Spring.Context.Support.WebSupportModule, Spring.Web"/>  <add name="OpenSessionInView" type="Spring.Data.NHibernate.Support.OpenSessionInViewModule, Spring.Data.NHibernate20"/> |

添加spring.Net的httpModules

1. 在<system.web>配置里面添加以下配置

|  |
| --- |
| <httpRuntime enableKernelOutputCache="false"/> |

## 数据库配置

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <objects xmlns="http://www.springframework.net"  xmlns:db="http://www.springframework.net/database">  <!-- Referenced by main application context configuration file -->  <description>  The Northwind object definitions for the Data Access Objects.  </description>  <!-- Database and NHibernate Configuration -->  <db:provider id="DbProvider"  provider="SqlServer-2.0"  connectionString="Persist Security Info=False;Integrated Security=true;Initial Catalog=Northwind;server=(local)"/>  <!--<db:provider id="DbProvider"  provider="SqlServer-2.0"  connectionString="Initial Catalog=WebSiteAdministratorDb;server=atom\sqlexpress;uid=sa;pwd=1234;"/> -->  <object id="NHibernateSessionFactory" type="Spring.Data.NHibernate.LocalSessionFactoryObject, Spring.Data.NHibernate20">  <property name="DbProvider" ref="DbProvider"/>  <property name="MappingAssemblies">  <list>  <value>Legendigital.Northwind.Entity</value>  </list>  </property>  <property name="HibernateProperties">  <dictionary>  <entry key="connection.provider"  value="NHibernate.Connection.DriverConnectionProvider"/>  <entry key="dialect"  value="NHibernate.Dialect.MsSql2005Dialect"/>  <entry key="connection.driver\_class"  value="NHibernate.Driver.SqlClientDriver"/>  </dictionary>  </property>  <property name="ExposeTransactionAwareSessionFactory" value="true" />  </object>  <object id="transactionManager"  type="Spring.Data.NHibernate.HibernateTransactionManager, Spring.Data.NHibernate20">  <property name="DbProvider" ref="DbProvider"/>  <property name="SessionFactory" ref="NHibernateSessionFactory"/>  </object>  <object id="HibernateTemplate" type="Spring.Data.NHibernate.Generic.HibernateTemplate">  <property name="SessionFactory" ref="NHibernateSessionFactory" />  <property name="TemplateFlushMode" value="Auto" />  <property name="CacheQueries" value="true" />  </object>  </objects> |

## Spring容器配置

数据层容器配置文件

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <objects xmlns="http://www.springframework.net"  xmlns:db="http://www.springframework.net/database">  <!-- 供主程序配置文件引用 -->  <description>  Spring.Net框架数据访问层容器配置文件  </description>  <!-- 数据访问类 -->  <object id="CategoriesDataObjectIocID" type="Legendigital.Northwind.Data.Tables.CategoriesDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="CustomerDemographicsDataObjectIocID" type="Legendigital.Northwind.Data.Tables.CustomerDemographicsDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="CustomersDataObjectIocID" type="Legendigital.Northwind.Data.Tables.CustomersDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="EmailDataObjectIocID" type="Legendigital.Northwind.Data.Tables.EmailDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="EmployeesDataObjectIocID" type="Legendigital.Northwind.Data.Tables.EmployeesDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="OrderDetailsDataObjectIocID" type="Legendigital.Northwind.Data.Tables.OrderDetailsDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="OrdersDataObjectIocID" type="Legendigital.Northwind.Data.Tables.OrdersDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="ProductsDataObjectIocID" type="Legendigital.Northwind.Data.Tables.ProductsDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="RegionDataObjectIocID" type="Legendigital.Northwind.Data.Tables.RegionDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="ShippersDataObjectIocID" type="Legendigital.Northwind.Data.Tables.ShippersDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="SuppliersDataObjectIocID" type="Legendigital.Northwind.Data.Tables.SuppliersDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="TerritoriesDataObjectIocID" type="Legendigital.Northwind.Data.Tables.TerritoriesDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="UserDataObjectIocID" type="Legendigital.Northwind.Data.Tables.UserDataObject, Legendigital.Northwind.Data">  <property name="HibernateTemplate" ref="HibernateTemplate"/>  </object>  <object id="DataObjectContainersIocID" type="Legendigital.Northwind.Data.Tables.Container.DataObjectContainers, Legendigital.Northwind.Data">  <property name="CategoriesDataObjectInstance" ref="CategoriesDataObjectIocID"/>  <property name="CustomerDemographicsDataObjectInstance" ref="CustomerDemographicsDataObjectIocID"/>  <property name="CustomersDataObjectInstance" ref="CustomersDataObjectIocID"/>  <property name="EmailDataObjectInstance" ref="EmailDataObjectIocID"/>  <property name="EmployeesDataObjectInstance" ref="EmployeesDataObjectIocID"/>  <property name="OrderDetailsDataObjectInstance" ref="OrderDetailsDataObjectIocID"/>  <property name="OrdersDataObjectInstance" ref="OrdersDataObjectIocID"/>  <property name="ProductsDataObjectInstance" ref="ProductsDataObjectIocID"/>  <property name="RegionDataObjectInstance" ref="RegionDataObjectIocID"/>  <property name="ShippersDataObjectInstance" ref="ShippersDataObjectIocID"/>  <property name="SuppliersDataObjectInstance" ref="SuppliersDataObjectIocID"/>  <property name="TerritoriesDataObjectInstance" ref="TerritoriesDataObjectIocID"/>  <property name="UserDataObjectInstance" ref="UserDataObjectIocID"/>  </object>  </objects> |

业务层容器配置文件

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <objects xmlns="http://www.springframework.net"  xmlns:tx="http://www.springframework.net/tx">  <!-- 供主程序配置文件引用 -->  <description>  Spring.Net框架服务层容器配置文件  </description>  <!-- 服务层类 -->  <object id="CategoriesServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.CategoriesServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="CategoriesDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="CustomerDemographicsServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.CustomerDemographicsServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="CustomerDemographicsDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="CustomersServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.CustomersServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="CustomersDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="EmailServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.EmailServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="EmailDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="EmployeesServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.EmployeesServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="EmployeesDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="OrderDetailsServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.OrderDetailsServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="OrderDetailsDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="OrdersServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.OrdersServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="OrdersDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="ProductsServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.ProductsServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="ProductsDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="RegionServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.RegionServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="RegionDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="ShippersServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.ShippersServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="ShippersDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="SuppliersServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.SuppliersServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="SuppliersDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="TerritoriesServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.TerritoriesServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="TerritoriesDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="UserServiceProxyIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.UserServiceProxy, Legendigital.Northwind.Bussiness">  <property name="SelfDataObj" ref="UserDataObjectIocID"/>  <property name="DataObjectsContainerIocID" ref="DataObjectContainersIocID"/>  </object>  <object id="ServiceProxyContainerIocID" type="Legendigital.Northwind.Bussiness.ServiceProxys.Tables.Container.ServiceProxyContainer, Legendigital.Northwind.Bussiness">  <property name="CategoriesServiceProxyInstance" ref="CategoriesServiceProxyIocID"/>  <property name="CustomerDemographicsServiceProxyInstance" ref="CustomerDemographicsServiceProxyIocID"/>  <property name="CustomersServiceProxyInstance" ref="CustomersServiceProxyIocID"/>  <property name="EmailServiceProxyInstance" ref="EmailServiceProxyIocID"/>  <property name="EmployeesServiceProxyInstance" ref="EmployeesServiceProxyIocID"/>  <property name="OrderDetailsServiceProxyInstance" ref="OrderDetailsServiceProxyIocID"/>  <property name="OrdersServiceProxyInstance" ref="OrdersServiceProxyIocID"/>  <property name="ProductsServiceProxyInstance" ref="ProductsServiceProxyIocID"/>  <property name="RegionServiceProxyInstance" ref="RegionServiceProxyIocID"/>  <property name="ShippersServiceProxyInstance" ref="ShippersServiceProxyIocID"/>  <property name="SuppliersServiceProxyInstance" ref="SuppliersServiceProxyIocID"/>  <property name="TerritoriesServiceProxyInstance" ref="TerritoriesServiceProxyIocID"/>  <property name="UserServiceProxyInstance" ref="UserServiceProxyIocID"/>  </object>  <tx:attribute-driven />  </objects> |

## 日志配置

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <log4net debug="false">    <appender name="AspNetTraceAppender" type="log4net.Appender.AspNetTraceAppender">  <layout type="log4net.Layout.PatternLayout">  <conversionPattern value="[%thread] %-5level - %message" />  </layout>  </appender>    <appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">  <file value="Logs/log" />  <appendToFile value="true" />  <rollingStyle value="Date" />  <datePattern value="yyyyMMdd" />  <layout type="log4net.Layout.PatternLayout">  <conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />  </layout>  </appender>  <appender name="TraceAppender" type="log4net.Appender.TraceAppender">  <layout type="log4net.Layout.PatternLayout">  <conversionPattern value="%-5level %M - %message%newline" />  </layout>  </appender>  <!-- Set default logging level to DEBUG -->  <root>  <level value="DEBUG" />  <!--<appender-ref ref="AspNetTraceAppender" />-->  <appender-ref ref="TraceAppender" />  <appender-ref ref="RollingFileAppender" />  </root>  <!--  In Spring.NET there is a 1-1 correspondence between the logger name and  the namespace of the class doing the logging...  -->  <logger name="Spring">  <level value="DEBUG" />  </logger>  <logger name="NHibernate">  <level value="INFO" />  </logger>  </log4net> |

# 框架调用说明

## 框架结构简介

本框架采用三层结构，整个解决方案被被分为了Common、Entity、Data、Bussiness、Web等五个项目。

Common项目：所有系统公用代码库

Entity项目：数据库实体层（以Entity结尾的对象），保存数据库与业务实体对象的映射关系的项目，实体类与映射文件都保存在这个项目里，数据库实体对表现层是不可见的，表现层不能直接访问业务实体。

Data项目：数据操作对象层（以DataObject结尾的对象），保存所有的数据操作对象，每一个数据操作对象对应一个数据库表，负责操作这个表的数据（增删改查），数据层数据操作对象不会涉及数据库事务，数据库事务由业务层事务对象处理，原则上来说数据操作对象中的每一个操作方法都应该是不可分割，业务层事务对象组合数据操作对象的操作，在业务层合成一个事务。数据操作对象只对事务对象可见。

Bussiness项目：业务层，业务层包含事务对象（以ServiceProxy结尾的对象）和业务包装对象（以Wrapper结尾的对象），业务包装对象是面向表现层的对象，每一个业务包装对象自身会包含一个业务实体对象，以便操作数据库，业务包装对象不直接操作数据库，业务包装对象通过调用事务对象的接口访问数据库。事务对象是在业务层里能够操作数据库的对象，事务对象通过调用多个数据操作对象来实现对数据库的访问，他会将这些操作组合成一个事务，所有的事物层对象都以一个接口向业务包装对象提供操作方法。

Web表现层：表现层直接调用业务包装对象进行业务操作。

## 实体层类

实体层对象（以Entity结尾的）

实体层对象主要是完全和数据库表对应的对象，每一个表会生成一个实体类，以及这个实体类对应的hbm.xml映射文件，通过这个映射文件将数据库中的表映射到实体类，注意所有的hbm.xml映射文件都必须设置为嵌入式资源，实体层的代码基本都是代码生成器生成的，基本上不需要对其进行修改。

## 数据层类

数据层项目中包括数据访问类（DataObject结尾的），数据访问类容器类，数据访问类容器类配置文件。

### 数据操作类简介

数据操作类是架构中直接操作数据库的类，每一个数据操作类对应一个数据库表，数据操作类负责对应的表的所有数据访问操作。

每一个数据操作类别分为了一个基本类，一个设计类

(如：CategoriesDataObject.cs、CategoriesDataObject.Designer.cs)

设计类（CategoriesDataObject.Designer.cs）中保存代码生成机生成代码的类，我们在开发过程不要修改这个类，所有的代码都应该写到CategoriesDataObject.cs类里面。

CategoriesDataObject.Designer.cs代码

|  |
| --- |
| // Generated by MyGeneration Version # (1.3.0.9)  using System;  using System.Collections.Generic;  using System.Text;  using NHibernate.Criterion;  using Legendigital.Framework.Common.Data.NHibernate.DynamicQuery;  using Legendigital.Northwind.Entity.Tables;  using Legendigital.Framework.Common.Data.NHibernate;  namespace Legendigital.Northwind.Data.Tables  {  public partial class CategoriesDataObject : BaseNHibernateDataObject<CategoriesEntity>  {  #region Expression Query Property (标准查询字段)  public static readonly Property PROPERTY\_CATEGORYID = Property.ForName(CategoriesEntity.PROPERTY\_NAME\_CATEGORYID);  public static readonly Property PROPERTY\_CATEGORYNAME = Property.ForName(CategoriesEntity.PROPERTY\_NAME\_CATEGORYNAME);  public static readonly Property PROPERTY\_DESCRIPTION = Property.ForName(CategoriesEntity.PROPERTY\_NAME\_DESCRIPTION);  public static readonly Property PROPERTY\_PICTURE = Property.ForName(CategoriesEntity.PROPERTY\_NAME\_PICTURE);  #endregion    public override string[] PkPropertyName  {  get { return new string[] { "CategoryID" }; }  }    public override Type GetFieldTypeByFieldName(string fieldName)  {  switch (fieldName)  {  case "CategoryID":  return typeof (int);  break;  case "CategoryName":  return typeof (string);  break;  case "Description":  return typeof (string);  break;  case "Picture":  return typeof (byte[]);  break;  }  return typeof(string);  }  }  } |

可以看到CategoriesDataObject 类会继承一个泛型基类BaseNHibernateDataObject<CategoriesEntity>，CategoriesEntity就代表对应的数据库表。

里面还会自动构建一些查询字段对象。（关于查询字段对象的用法，我们后面再讲）。

### 数据更新操作

#### 添加数据

调用CategoriesDataObject的Save（CategoriesEntity entity）方法用于保存实体。

我们可以在CategoriesDataObject里面写一个addNew方法测试一下。

|  |
| --- |
| public partial class CategoriesDataObject  {  public void AddNew(CategoriesEntity entity)  {  this.Save(entity);  }  } |

#### 修改数据

调用CategoriesDataObject的Update（CategoriesEntity entity）方法用于保存实体。

我们可以在CategoriesDataObject里面写一个UpdateObj方法测试一下。

|  |
| --- |
| public partial class CategoriesDataObject  {  public void UpdateObj(CategoriesEntity entity)  {  this. Update (entity);  }  } |

#### 删除数据

删除数据可以调用DeleteByID(object id)、Delete(CategoriesEntity entity)、DeleteAll()三个方法删除数据。

DeleteByID 方法是传入主键值删除数据。

Delete 方法是传入实体对象删除数据。

DeleteAll 方法是删除所有的数据。

调用代码如下：

|  |
| --- |
| public void DeleteTest(CategoriesEntity entity)  {  this.DeleteByID(10);  this.Delete(entity);  this.DeleteAll();  } |

### 数据查询操作

#### 通过ID加载数据

调用Load(object id) 通过ID加载数据。

调用代码如下：

|  |
| --- |
| public CategoriesEntity FindTest(object id)  {  return this.Load(id);  } |

#### 动态构建查询对象查询

数据层提供动态查询对象（NHibernateDynamicQueryGenerator<T>）构建查询，简化数据查询的工作。

动态查询对象的创建代码如下：

|  |
| --- |
| public void NewQueryObject()  {  NHibernateDynamicQueryGenerator<CategoriesEntity> dynamicQueryGenerator = this.GetNewQueryBuilder();  } |

动态查询对象是基于泛型的，需要查询哪个表，就必须指定哪个表对应的实体类型

每一个数据层操作类都提供一个GetNewQueryBuilder()提供本数据操作类对应表的动态查询对象实例。

查询对象添加where条件

|  |
| --- |
| //添加产品名为apple条件  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq(“apple”)); |

查询对象添加OrderBy条件

|  |
| --- |
| dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Asc（）);  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTID.Desc（）); |

##### 查询属性的使用

每一个数据库操作类的设置类中都会自动生成很多对应数据库字段的查询属性，利用这些属性，我们可以快速的生成查询中所需的where条件或者是排序、条件。

|  |
| --- |
| public partial class ProductsDataObject : BaseNHibernateDataObject<ProductsEntity>  {  #region Expression Query Property (标准查询字段)  public static readonly Property PROPERTY\_PRODUCTID = Property.ForName(ProductsEntity.PROPERTY\_NAME\_PRODUCTID);  public static readonly Property PROPERTY\_PRODUCTNAME = Property.ForName(ProductsEntity.PROPERTY\_NAME\_PRODUCTNAME);  public static readonly Property PROPERTY\_SUPPLIERID = Property.ForName(ProductsEntity.PROPERTY\_NAME\_SUPPLIERID); |

其中的PROPERTY\_PRODUCTNAME是代码生成器自动构建的查询属性，会对应每一个数据表的字段，查询属性可以直接构建等于，大于，小于，Like，In查询，IsEmpty等查询条件。

下面详细介绍一下查询属性的方法

Eq：等于

|  |
| --- |
| //产品名等于Apple  PROPERTY\_PRODUCTNAME.Eq("Apple"); |

IsNull：为Null

|  |
| --- |
| //产品名为Null  PROPERTY\_PRODUCTNAME.IsNull(); |

IsNotNull：不为Null

|  |
| --- |
| //产品名不为Null  PROPERTY\_PRODUCTNAME.IsNotNull(); |

IsEmpty：为空

|  |
| --- |
| //产品名为空  PROPERTY\_PRODUCTNAME.IsEmpty(); |

IsNotEmpty：不为空

|  |
| --- |
| //产品名不为空  PROPERTY\_PRODUCTNAME.IsNotEmpty(); |

Gt：大于

|  |
| --- |
| //产品单价大于10  PROPERTY\_UNITPRICE.Gt(10); |

Ge：大于等于

|  |
| --- |
| //产品单价大于等于10  PROPERTY\_UNITPRICE.Ge(10); |

Lt：小于

|  |
| --- |
| //产品单价小于10  PROPERTY\_UNITPRICE.Lt(10); |

Le：小于等于

|  |
| --- |
| //产品单价小于等于10  PROPERTY\_UNITPRICE.Le(10); |

Between：介于两个值之间

|  |
| --- |
| //产品单价位于到之间  PROPERTY\_UNITPRICE.Between(3, 25); |

In：包含于

|  |
| --- |
| //产品名为"apple"或者"orange"  PROPERTY\_PRODUCTNAME.In(new Object[] { "apple", "orange"}); |

Like：Like查询

|  |
| --- |
| //产品名中包含A，等同于Like ‘%A%’  PROPERTY\_PRODUCTNAME.Like("A", MatchMode.Anywhere);  //产品名中以A开头，等同于Like ‘A%’  PROPERTY\_PRODUCTNAME.Like("A", MatchMode.Start);  //产品名中以A结尾，等同于Like ‘%A’  PROPERTY\_PRODUCTNAME.Like("A", MatchMode.End);  //产品名等于A，等同于Like ‘A’  PROPERTY\_PRODUCTNAME.Like("A", MatchMode.Exact); |

EqProperty：等于另外一个字段

|  |
| --- |
| //产品名和产品ID相等  PROPERTY\_PRODUCTNAME.EqProperty(PROPERTY\_PRODUCTID); |

GtProperty：大于另外一个字段

|  |
| --- |
| //订单日期大于发货日期  PROPERTY\_ORDERDATE.GtProperty(PROPERTY\_SHIPPEDDATE); |

GeProperty：大于等于另外一个字段

|  |
| --- |
| //订单日期大于等于发货日期  PROPERTY\_ORDERDATE. GeProperty(PROPERTY\_SHIPPEDDATE); |

LtProperty：小于另外一个字段

|  |
| --- |
| //订单日期小于发货日期  PROPERTY\_ORDERDATE.LtProperty(PROPERTY\_SHIPPEDDATE); |

LeProperty：小于另外一个字段

|  |
| --- |
| //订单日期小于等于发货日期  PROPERTY\_ORDERDATE.LeProperty(PROPERTY\_SHIPPEDDATE); |

排序条件

Asc:正序

|  |
| --- |
| //订单日期从小到大  PROPERTY\_ORDERDATE. Asc(); |

Desc：反序

|  |
| --- |
| //订单日期从大到小  PROPERTY\_ORDERDATE. Desc(); |

否定条件

Not（）

调用数据操作类的Not方法：

|  |
| --- |
| //产品名不等于等于Apple  Not(PROPERTY\_PRODUCTNAME.Eq("apple")); |

条件与操作

And(,,,,,,,)

|  |
| --- |
| //产品名等于apple1 与产品名等于apple2 与产品名等于apple3  And(PROPERTY\_PRODUCTNAME.Eq("apple1"), PROPERTY\_PRODUCTNAME.Eq("apple2"), PROPERTY\_PRODUCTNAME.Eq("apple3")); |

条件或操作

Or(,,,,,,,)

|  |
| --- |
| //产品名等于apple1 或 产品名等于apple2 或 产品名等于apple3  Or(PROPERTY\_PRODUCTNAME.Eq("apple1"), PROPERTY\_PRODUCTNAME.Eq("apple2"), PROPERTY\_PRODUCTNAME.Eq("apple3")); |

##### 查找数据集合

创建好查询对象以后调用FindListByQueryBuilder（NHibernateDynamicQueryGenerator qg）

就可以直接返回对应类型的泛型列表。

查找所有产品名等于Apple的产品,按照ID反序排序

|  |
| --- |
| public List<ProductsEntity> GetProductsByName(string name)  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = this.GetNewQueryBuilder();  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq(name));  dynamicQueryGenerator.AddOrderBy(PROPERTY\_PRODUCTID.Desc());  return this.FindListByQueryBuilder(dynamicQueryGenerator);  } |

产品名等于Apple **与** 价格大于 10 的产品，按照价格正序，ID反序

|  |
| --- |
| public List<ProductsEntity> GetProducts(string name, decimal price)  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = this.GetNewQueryBuilder();  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq("Apple"));  dynamicQueryGenerator.AddWhereClause(PROPERTY\_UNITPRICE.Gt(10));  dynamicQueryGenerator.AddOrderBy(PROPERTY\_UNITPRICE.Asc());  dynamicQueryGenerator.AddOrderBy(PROPERTY\_PRODUCTID.Desc());  return this.FindListByQueryBuilder(dynamicQueryGenerator);  } |

##### 查找数据集合带分页

如果需要对查询结果进行分页，可以直接对动态查询对象进行设置。

|  |
| --- |
| public List<ProductsEntity> GetProductsByName(string name)  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = this.GetNewQueryBuilder();  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq(name));  dynamicQueryGenerator.AddOrderBy(PROPERTY\_PRODUCTID.Desc());  //设置从第条记录开始读取  dynamicQueryGenerator.SetFirstResult(10);  //设置最多读取条记录  dynamicQueryGenerator.SetMaxResults(12);  return this.FindListByQueryBuilder(dynamicQueryGenerator);  } |

在很多情况下，我们需要获取总记录数，

可以调用FindListByPageByQueryBuilder(NHibernateDynamicQueryGenerator<DomainType> queryBuilder, out int recordCount)

获取总记录数

|  |
| --- |
| public List<ProductsEntity> GetProductsByName(string name, out int recordCount)  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = GetNewQueryBuilder();  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq(name));  dynamicQueryGenerator.AddOrderBy(PROPERTY\_PRODUCTID.Desc());  //设置从第条记录开始读取  dynamicQueryGenerator.SetFirstResult(10);  //设置最多读取条记录  dynamicQueryGenerator.SetMaxResults(12);  return FindListByPageByQueryBuilder(dynamicQueryGenerator, out recordCount);  } |

下面的例子可以实现Top 查询

|  |
| --- |
| public List<ProductsEntity> GetTopProducts(int top)  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = GetNewQueryBuilder();  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq("apple"));  dynamicQueryGenerator.AddOrderBy(PROPERTY\_PRODUCTID.Desc());  //设置最多读取条记录  dynamicQueryGenerator.SetMaxResults(top);  return FindListByQueryBuilder(dynamicQueryGenerator);  } |

##### 查找单个数据实体

当我们的查询结果是单个实体而不是集合的时候，我们可以调用FindSingleEntityByQueryBuilder(NHibernateDynamicQueryGenerator<DomainType> queryBuilder)

方法获取查询结果

|  |
| --- |
| public ProductsEntity GetProductByName(string name)  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = GetNewQueryBuilder();  dynamicQueryGenerator.AddWhereClause(PROPERTY\_PRODUCTNAME.Eq(name));  dynamicQueryGenerator.AddOrderBy(PROPERTY\_PRODUCTID.Desc());  return this.FindSingleEntityByQueryBuilder(dynamicQueryGenerator);  } |

##### 投影查询

上面的查询都是查出一个表的所有字段，有的时候，我们只需要查出一个或者多个字段的值的时候，我们就可以使用投影查询了。

投影查询调用以下方法：

FindListByProjection<CType>(NHibernateDynamicQueryGenerator<DomainType> queryBuilder, IProjection projection)

Distinct查询产品所有的category

|  |
| --- |
| public List<CategoriesEntity> GetProductCategoies()  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = GetNewQueryBuilder();  return this.FindListByProjection<CategoriesEntity>(dynamicQueryGenerator, GetDistinctProperty(PROPERTY\_CATEGORYID));  } |

Distinct查询产品所有的productname

|  |
| --- |
| public List<string> GetProductNames()  {  NHibernateDynamicQueryGenerator<ProductsEntity> dynamicQueryGenerator = GetNewQueryBuilder();  return this.FindListByProjection<string>(dynamicQueryGenerator,GetDistinctProperty(PROPERTY\_PRODUCTNAME));  } |

#### HQL查询

##### HQL简介

##### HQL查询示例

##### 带参数的HQL查询

### 记录日志

日志的级别

框架日志支持多种级别的日志。优先级从高到低依次排列如下:

FATAL > ERROR > WARN > INFO > DEBUG

如果一旦开启ERROR级别，那边ERROR级别以及比ERROR级别低的级别的日志都要记录。

实际上，我们可以根据业务场景定记录到那个的级别的日志，在测试环境下面，我们应该开启到最低级别DEBUG的日志，这样方便我们调试，生产环境，我们可以将日志级别定位ERROR级别（调试日志不予以记录）。

此外还有ALL（允许所有的日志请求）和OFF（拒绝所有的日志请求）这两种特殊的级别。

以下五个方法对应5个级别的日志记录。

开启日志的级别可以在配置文件里面定义

数据操作对象内置属性Logger提供日志记录功能。

Logger

|  |
| --- |
| this.Logger.Debug("Messgae");  this.Logger.Error("Messgae");  this.Logger.Fatal("Messgae");  this.Logger.Warn("Messgae");  this.Logger.Info("Messgae"); |

## 业务层类

### 业务层业务包装类简介

### 业务层事务类简介

### 事务管理

### 记录日志

业务层事务对象内置属性Logger提供日志记录功能。

Logger

|  |
| --- |
| this.Logger.Debug("Messgae");  this.Logger.Error("Messgae");  this.Logger.Fatal("Messgae");  this.Logger.Warn("Messgae");  this.Logger.Info("Messgae"); |

## Ado.Net数据操作支持

### 执行SQL语句

### 执行存储过程