**ASP.NET 4.0 与 Entity Framework 4-第四篇-Entity Framework在三层架构中的使用**

LloydSheng 写于 2010年05月8日下午 12:16 | [有3人评论](http://lloydsheng.com/2010/05/aspnet-40-and-entity-framework-4-part-4-3-layered-approach-to-the-entity-framework.html#comments)

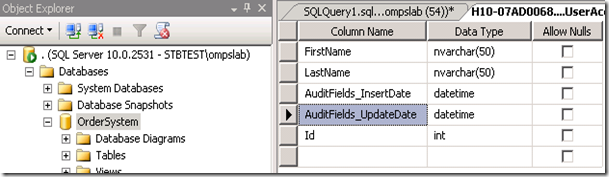
本系列的前3篇文章已经向大家介绍了如何使用Entity Framework4（以下简称EF4）创建数据库并进行增删改查操作，还介绍了存储过程的调用方法。在以前的例子中，我们没对代码进行分层，只是简单的介绍实现各种操作的方式。今天我们就来谈谈在三层架构中使用EF4 ，想必很多人已经对三层有过了解。 当然，三层不一定适合所有应用开发，但分层的思想是值得借鉴的。

本文所有的Web页面都和以前一样，不同的是采用了三层架构。我们对应用进行了逻辑分层，各层分别为：界面层（UI），业务逻辑层（BLL）和数据访问层（DAL）。而在数据交互环境下的实体对象此时将扮演数据访问层（DAL）和业务逻辑层（BLL）数据传输的对象（DTO）。在以前，我们可能需要花时间编写DTO的代码，而现在EF4能帮我们省去这一部分时间了。

好吧，下面跟随我的步骤一起完成这个应用。

**步骤1.创建数据库**

为可简单起见，我们在SQL Server中新建一个OrderSystem数据库，并且创建一个单独的表UserAccounts，该表的字段如下，注意将Id作为主键，并设置为标识列以便自增。

[](http://lloydsheng.com/wp-content/uploads/2010/05/image.png)

仍然采用存储过程操作表的数据，我们创建的存储过程如下：

1.UserAccounts\_SelectAll查询表中所有数据

CREATE PROCEDURE [dbo].[UserAccounts\_SelectAll]

AS

SET NOCOUNT ON

SELECT Id, FirstName, LastName, AuditFields\_InsertDate, AuditFields\_UpdateDate

FROM UserAccounts

RETURN

2.UserAccounts\_SelectById通过Id查询单条表数据

CREATE PROCEDURE [dbo].[UserAccounts\_SelectById]

(

@Id int

)

AS

SET NOCOUNT ON

SELECT Id, FirstName, LastName, AuditFields\_InsertDate, AuditFields\_UpdateDate

FROM UserAccounts

WHERE Id = @Id

RETURN

3.UserAccounts\_Insert插入一条表数据

CREATE PROCEDURE [dbo].[UserAccounts\_Insert]

(

@FirstName nvarchar(50),

@LastName nvarchar(50),

@AuditFields\_InsertDate datetime,

@AuditFields\_UpdateDate datetime

)

AS

INSERT INTO UserAccounts (FirstName, LastName, AuditFields\_InsertDate,

AuditFields\_UpdateDate)

VALUES (@FirstName, @LastName, @AuditFields\_InsertDate,

@AuditFields\_UpdateDate)

SELECT CAST(SCOPE\_IDENTITY() AS INT) AS Id

4.UserAccounts\_Update更新单条表数据

CREATE PROCEDURE [dbo].[UserAccounts\_Update]

(

@Id int,

@FirstName nvarchar(50),

@LastName nvarchar(50),

@AuditFields\_UpdateDate datetime

)

AS

SET NOCOUNT ON

UPDATE UserAccounts

SET FirstName = @FirstName,

LastName = @LastName,

AuditFields\_UpdateDate = @AuditFields\_UpdateDate

WHERE Id = @Id

RETURN

5.UserAccounts\_Delete删除单条表数据

CREATE PROCEDURE [dbo].[UserAccounts\_Delete]

(

@Id int

)

AS

SET NOCOUNT ON

DELETE

FROM UserAccounts

WHERE Id = @Id

RETURN

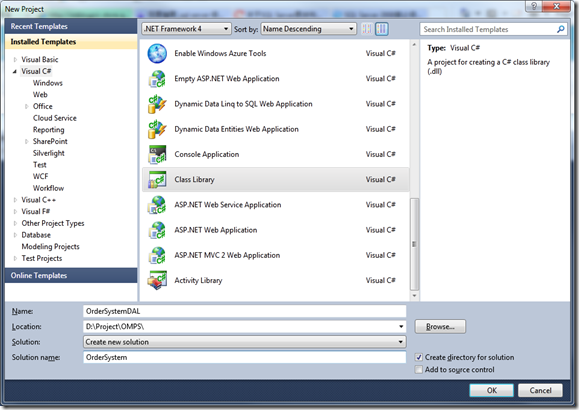
**步骤2.创建数据访问层**

下来在VS 2010中创建一个新的类库项目，它将作为数据访问层。

1.打开VS，选择文件->新项目。

2.选择“C#类库”项目，将名字改为OrederSystemDAL

3.选中“为项目创建目录”，将左边输入框内容改为“OrderSystem”，点“OK”。

[](http://lloydsheng.com/wp-content/uploads/2010/05/image1.png)

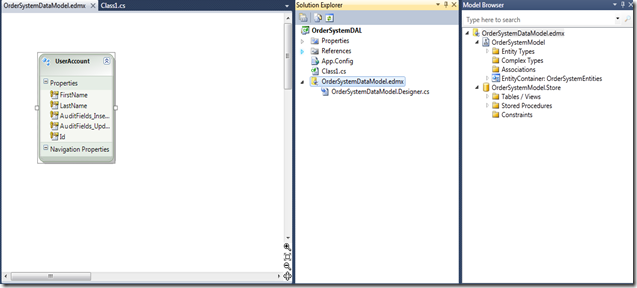
4.添加数据视图模型，OrderSystemDAL项目上右键，选择Add->New Item..

5.从项目模板中选择“ADO.NET Entity Data Mode”，将它名字改为“OrderSystemDataModel.edmx”，点“Add”。

6.将会看到视图数据模型向导，选择“Generate from database”，点“Next”。

7.选择数据库连接，我们创建一个新连接，选择“New connection”，填写好登陆信息并选择OrderSystem数据后，点“OK”，再点“Next”。

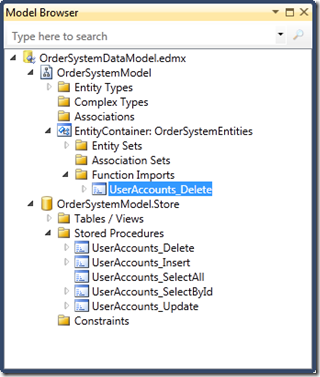
8.选择数据库对象，选择刚创建的UserAccounts表和5个存储过程，点“Finish”。

[](http://lloydsheng.com/wp-content/uploads/2010/05/image2.png)

接下来是创建调用这些存储过程的方法了。

9.展开存储过程文件夹，右击“UserAccounts\_Delete”存储过程，选择“ Add Function Import”。

10.弹出“Function Import”框后，保持默认参数，点击“OK”。

[](http://lloydsheng.com/wp-content/uploads/2010/05/image3.png)

11.UserAccounts\_Insert存储过程上右击，由于它将返回新插入的数据的Id，所以我们要将“Returns a Collection Of”的Scalars参数设置为Int32，然后点“OK”。

12.对其他存储过程重复同样操作，对于SelectAll 和SelectById存储过程，“Returns a Collection Of”中的Entity参数设置为UserAccount，Update存储过程保持默认即可。

实体数据模型就创建好了，下面要在数据访问层中创建一个类来管理这些存储过程的调用。

13.添加一个新的类文件，将其名字改为“UserAccountsData.cs”，将类声明为公共的和静态的。

public static class UserAccountsData

14.给类添加以下方法：

public static int Insert(string firstName, string lastName, DateTime insertDate)

{

using (OrderSystemEntities db = new OrderSystemEntities())

{

return Insert(db, firstName, lastName, insertDate);

}

}

public static int Insert(OrderSystemEntities db, string firstName,

string lastName, DateTime insertDate)

{

return db.UserAccounts\_Insert(firstName, lastName, insertDate, insertDate).ElementAt(0).Value;

}

public static void Update(int id, string firstName, string lastName,

DateTime updateDate)

{

using (OrderSystemEntities db = new OrderSystemEntities())

{

Update(db, id, firstName, lastName, updateDate);

}

}

public static void Update(OrderSystemEntities db, int id, string firstName,

string lastName, DateTime updateDate)

{

db.UserAccounts\_Update(id, firstName, lastName, updateDate);

}

public static void Delete(int id)

{

using (OrderSystemEntities db = new OrderSystemEntities())

{

Delete(db, id);

}

}

public static void Delete(OrderSystemEntities db, int id)

{

db.UserAccounts\_Delete(id);

}

public static UserAccount SelectById(int id)

{

using (OrderSystemEntities db = new OrderSystemEntities())

{

return SelectById(db, id);

}

}

public static UserAccount SelectById(OrderSystemEntities db, int id)

{

return db.UserAccounts\_SelectById(id).ElementAtOrDefault(0);

}

public static List<UserAccount> SelectAll()

{

using (OrderSystemEntities db = new OrderSystemEntities())

{

return SelectAll(db);

}

}

public static List<UserAccount> SelectAll(OrderSystemEntities db)

{

return db.UserAccounts\_SelectAll().ToList();

}

以上就是我们用来操作UserAccounts表数据的类了。

**步骤3.创建业务逻辑层**

业务逻辑层是介于界面层和数据访问层的，它体现了应用的业务逻辑和一些数据验证规则。

1.添加一个新的C#类库项目，将它的名字改为“OrderSystemBLL”

2.给该项目添加DAL项目引用和System.Data.Entity引用。

3.将项目中的“Class1.cs”文件改名为“UserAccountEO”。EO的意思编辑对象（Edit Object），添加EO后缀是为了将其和数据访问层的类区分开。

4.添加命名空间。

using System.Collections;

using OrderSystemDAL;

5.添加属性声明

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime InsertDate { get; set; }

public DateTime UpdateDate { get; set; }

public string FullName

{

get

{

return LastName + ", " + FirstName;

}

}

6.添加下面的代码用来将数据保持到数据库，该方法根据对象Id是否赋值来判断是更新还是插入操作，并且还会进行一系列的验证，如果验证不通过，则不能进行数据的插入或更新，相关的错误还会返回给它的调用者。

public bool Save(ref ArrayList validationErrors)

{

ValidateSave(ref validationErrors);

if (validationErrors.Count == 0)

{

if (Id == 0)

{

Id = UserAccountsData.Insert(FirstName, LastName, DateTime.Now);

}

else

{

UserAccountsData.Update(Id, FirstName, LastName, DateTime.Now);

}

return true;

}

else

{

return false;

}

}

private void ValidateSave(ref ArrayList validationErrors)

{

if (FirstName.Trim() == "")

{

validationErrors.Add("The First Name is required.");

}

if (LastName.Trim() == "")

{

validationErrors.Add("The Last Name is required.");

}

}

7.添加用来删除数据的方法，如果要进行验证可以采取如下的方式

public void Delete(ref ArrayList validationErrors)

{

ValidateDelete(ref validationErrors);

if (validationErrors.Count == 0)

{

UserAccountsData.Delete(Id);

}

}

private void ValidateDelete(ref ArrayList validationErrors)

{

//Check for referential integrity.

}

8.添加用来查询单条数据的方法

public bool Select(int id)

{

UserAccount userAccount = UserAccountsData.SelectById(id);

if (userAccount != null)

{

MapData(userAccount);

return true;

}

else

{

return false;

}

}

internal void MapData(UserAccount userAccount)

{

Id = userAccount.Id;

FirstName = userAccount.FirstName;

LastName = userAccount.LastName;

InsertDate = userAccount.AuditFields\_InsertDate;

UpdateDate = userAccount.AuditFields\_UpdateDate;

}

9.最后是查询表所有数据的方法

public static List<UserAccountEO> SelectAll()

{

List<UserAccountEO> userAccounts

= new List<UserAccountEO>();

List<UserAccountEO> userAccountDTOs = UserAccountsData.SelectAll();

foreach (UserAccount userAccountDTO in userAccountDTOs)

{

UserAccountEO userAccountEO = new UserAccountEO();

userAccountEO.MapData(userAccountDTO);

userAccounts.Add(userAccountEO);

}

return userAccounts;

}

**步骤4.创建Web表单**

下面就是创建数据增删改查的页面了，就是我们所说的界面层

1.新建一个ASP.NET Web项目，命名为“OrderSystemUI”。

2.为该项目添加OrderSystemBLL项目的引用。

3.在web.config中添加connectionStrings配置节，代码如下：

<connectionstrings>

<add name="OrderSystemEntities" connectionstring="你的数据库连接字符串"

providername="System.Data.EntityClient" />

</connectionstrings>

4.添加一个WebForm，命名为“Users.aspx”。

5.转到代码视图，在div之间添加如下html代码

<table>

<tbody>

<tr>

<td colspan="2">

<asp :label id="lblErrorLabel" runat="server" text="Please correct the following issues:" forecolor="Red" visible="false"></asp>

<asp :label id="lblErrorMessages" runat="server" forecolor="Red" visible="false"></asp>

</td>

</tr>

<tr>

<td>Select A User:</td>

<td><asp :dropdownlist id="ddlUsers" runat="server" autopostback="True"></asp></td>

</tr>

<tr>

<td>First Name:</td>

<td><asp :textbox id="txtFirstName" runat="server"></asp></td>

</tr>

<tr>

<td>Last Name:</td>

<td><asp :textbox id="txtLastName" runat="server"></asp></td>

</tr><tr>

<td>Inserted:</td><td><asp :label id="lblInserted" runat="server"></asp></td>

</tr><tr><td>Updated:</td>

<td><asp :label id="lblUpdated" runat="server"></asp></td>

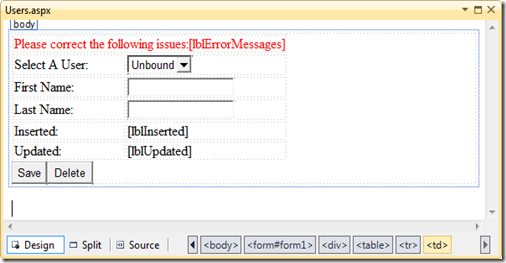
</tr>

</tbody>

</table>

<asp :button id="btnSave" runat="server" text="Save" />

<asp :button id="btnDelete" runat="server" text="Delete" />

[](http://lloydsheng.com/wp-content/uploads/2010/05/image4.png)

**步骤5.将查询的数据加载到Drop Down List**

下面要做的是在页面加载时，将UserAccounts表的数据加载到Drop Down List，同时可以在Drop Down List中选择新增数据。

1.双击页面，创建Page\_Load时间处理方法。

2.添加命名空间using OrderSystemBLL;。

3.Page\_Load处理方法代码如下：

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

LoadUserDropDownList();

}

}

4.添加LoadUserDropDownList方法

private void LoadUserDropDownList()

{

ddlUsers.DataSource = UserAccountEO.SelectAll();

ddlUsers.DataTextField = "FullName";

ddlUsers.DataValueField = "Id";

ddlUsers.DataBind();

ddlUsers.Items.Insert(0, new ListItem("Create New User", ""));

}

**步骤6.插入和更新数据**

设计视图下，双击“Save”按钮，添加点击事件处理代码如下：

protected void btnSave\_Click(object sender, EventArgs e)

{

ArrayList validationErrors = new ArrayList();

UserAccountEO userAccount = new UserAccountEO();

if (ddlUsers.SelectedItem.Value != "")

{

//Updating

userAccount.Id = Convert.ToInt32(ddlUsers.SelectedValue);

}

userAccount.FirstName = txtFirstName.Text;

userAccount.LastName = txtLastName.Text;

userAccount.Save(ref validationErrors);

if (validationErrors.Count > 0)

{

ShowValidationMessages(validationErrors);

}

else

{

//Reload the drop down list

LoadUserDropDownList();

//Select the one the user just saved.

ddlUsers.Items.FindByValue(userAccount.Id.ToString()).Selected = true;

lblErrorLabel.Visible = false;

lblErrorMessages.Visible = false;

}

}

private void ShowValidationMessages(ArrayList validationErrors)

{

if (validationErrors.Count > 0)

{

lblErrorLabel.Visible = true;

lblErrorMessages.Visible = true;

lblErrorMessages.Text = "";

foreach (string message in validationErrors)

{

lblErrorMessages.Text += message + "";

}

}

else

{

lblErrorLabel.Visible = false;

lblErrorMessages.Visible = false;

}

}

这里代码和以前的代码不同之处是，多了一些验证的处理。

**步骤7.查询数据**

设计视图下，双击下拉框，给它添加SelectedIndexChanged事件处理方法，代码如下：

txtFirstName.Text = "";

txtLastName.Text = "";

lblInserted.Text = "";

lblUpdated.Text = "";

if (ddlUsers.SelectedValue != "")

{

int userAccountId = Convert.ToInt32(ddlUsers.SelectedValue);

UserAccountEO userAccount = new UserAccountEO();

if (userAccount.Select(userAccountId))

{

txtFirstName.Text = userAccount.FirstName;

txtLastName.Text = userAccount.LastName;

lblInserted.Text = userAccount.InsertDate.ToString();

lblUpdated.Text = userAccount.UpdateDate.ToString();

}

}

**步骤8.删除数据**

视图设计下，双击“Delete”按钮，给它添加点击事件处理方法，代码如下：

if (ddlUsers.SelectedItem.Value != "")

{

ArrayList validationErrors = new ArrayList();

UserAccountEO userAccount = new UserAccountEO();

userAccount.Id = Convert.ToInt32(ddlUsers.SelectedValue);

userAccount.Delete(ref validationErrors);

if (validationErrors.Count > 0)

{

ShowValidationMessages(validationErrors);

}

else

{

LoadUserDropDownList();

txtFirstName.Text = "";

txtLastName.Text = "";

lblInserted.Text = "";

lblUpdated.Text = "";

lblErrorLabel.Visible = false;

lblErrorMessages.Visible = false;

}

}

**总结**

至此，如何在三层架构中使用EF4就像想你介绍完了，它应该能满足一般应用的要求。在EF4结合VS2010后，我们发现一个最大的好处是DTO对象不再需要我们自己编写了，加上调用存储过程智能感知的支持，确实非常方便。

项目文件：<http://files.cnblogs.com/lloydsheng/OrderSystem4.zip>

独立博客链接：<http://lloydsheng.com/2010/05/aspnet-40-and-entity-framework-4-part-4-3-layered-approach-to-the-entity-framework.html>