Language Integrated Query

学习指南

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

序

“在 Linq To Sql 正式推出之前，很多人只是把 sql 语句形成一个 string，然后，通过 ADO.NET 传给 SQL Server，返回结果集。这里的缺陷就是，假如你 sql 语句写的有问题，只有到运行时才知道， 而 Linq To Sql 语句是在编译期间就做检查”，那么 Linq 是个什么东西呢？它能翻译 SQL 语句？带着疑问…

LINQ：Language Integrated Query 语言集成查询，其本质是对 ADO.NET 结果集通过反射连同泛型特性转换成对象集，实现 OR 模型的转换（类似 JAVA 中 Hibernate 框架，.NET 中的 NHibernate），它完全是基于.NET 2.0 的框架。

学习完 LINQ 完后，你会发现它的优点与缺点。它的优点在于封装了 SQL 语句，只对对象进行操作（添加、删除、修改和查询），代码量大大减少，让我们重点关心业务逻辑，而非代码上，把我们引入到面向对象编程方式上来；缺点在于追求效益的同时牺牲了性能，比起 ADO.NET 性能稍差（ADO.NET 更有优势，不需要进行 OR 转换），另外对一些复杂的 SQL 语句也不好操作（这里 LINQ 支持 SQL 语句），虽然它都支持联合、分组、排序、连接查询等，因此要有选择性的用。

我们学习 LINQ 框架并非全部硬搬，而是有目的的选择，其实学习任何新的东西都要有的放矢，知道它们的得与失，找出平衡点更好的实践在项目中来。

下面的学习文章来自 CNBLOG 网，经整理方便学习研究。

——Joney Liu ^^

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（一）：预备知识

什么是 Linq to sql

Linq to sql（或者叫 DLINQ）是 LINQ（.NET 语言集成查询）的一部分，全称基于关系数据的 .NET 语言集成查询，用于以对象形式管理关系数据，并提供了丰富的查询功能，它和 Linq to xml、Linq to objects、Linq to dataset、Linq to entities 等组成了强大的 LINQ。

要学好 LINQ 查询语法，就不得不先理解 C# 3.0 的一些新特性，下面一一简单介绍。

隐含类型局部变量

```
var age = 26;

var username = "zhuye";

var userlist = new [] {"a","b","c"};

foreach(var user in userlist)

    Console.WriteLine(user);
```

纯粹给懒人用的 var 关键字，告诉编译器（对于 CLR 来说，它是不会知道你是否使用了 var，苦力是编译器出的），你自己推断它的类型吧，我不管了。但是既然让编译器推断类型就必须声明的时候赋值，而且不能是 null 值。注意，这只能用于局部变量，用于字段是不可以的。

匿名类型

```
var data = new {username = "zhuye",age = 26};

    Console.WriteLine("username:{0} age:{1}", data.username, data.age);
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

匿名类型允许开发人员定义行内类型，无须显式定义类型。常和 var 配合使用，var 用于声明匿名类型。定义一个临时的匿名类型在 LINQ 查询句法中非常常见，我们可以很方便的实现对象的转换和投影。

扩展方法

```
public static class helper
{
    public static string MD5Hash(this string s)
    {
        return System.Web.Security.FormsAuthentication.HashPasswordForStoringInConfigFile(s,"MD5");
    }

    public static bool In(this object o, IEnumerable b)
    {
        foreach(object obj in b)
        {
            if(obj==o)
            return true;
        }
```

```
            return false;
        }
    }

// 调用扩展方法
Console.WriteLine("123456".MD5Hash());
Console.WriteLine("1".In(new[]{"1","2","3"}));
```

很多时候我们需要对 CLR 类型进行一些操作，苦于无法扩展 CLR 类型的方法，只能创建一些 helper 方法，或者生成子类。扩展方法使得这些需求得意实现，同时也是实现 LINQ 的基础。定义扩展方法需要注意，只能在静态类中定义并且是静态方法，如果扩展方法名和原有方法名发生冲突，那么扩展方法将失效。

自动属性

```
public class Person
{
    public string username { get; protected set; }
    public int age { get; set; }

    public Person()
    {
        this.username = "zhuye";
```

```
        }
    }

Person p = new Person();
//p.username = "aa";
    Console.WriteLine(p.username);
```

意义不是很大，纯粹解决机械劳动。编译器自动为你生成 get、set 操作以及字段，并且你不能使用字段也不能自定义 get、set 操作，不过你可以分别定义 get 和 set 的访问级别。

对象初始化器

```
    public class Person
    {
        public string username { get; set; }
        public int age { get; set; }

        public override string    ToString()
        {
        return string.Format("username:{0} age:{1}", this.username, this.age);
        }
    }
```

```
Person p = new Person() {username = "zhuye", age=26};
Console.WriteLine(p.ToString());
```

编译器会自动为你做 setter 操作，使得原本几行的属性赋值操作能在一行中完成。这里需要注意：

允许只给一部分属性赋值，包括 internal 访问级别

可以结合构造函数一起使用，并且构造函数初始化先于对象初始化器执行

集合初始化器

```
public class Person
{
    public string username { get; set; }
    public int age { get; set; }

    public override string    ToString()
    {
    return string.Format("username:{0} age:{1}", this.username, this.age);
    }
}
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
var persons = new List<Person> {
      new Person {username = "a", age=1},
      new Person {username = "b", age=2}};
foreach(var p in persons)
     Console.WriteLine(p.ToString());
```

编译器会自动为你做集合插入操作。如果你为 Hashtable 初始化的话就相当于使用了两个对象初始化器。

Lambda 表达式

```
var list = new [] { "aa", "bb", "ac" };
var result = Array.FindAll(list, s => (s.IndexOf("a") > -1));
foreach (var v in result)
     Console.WriteLine(v);
```

其实和 2.0 中的匿名方法差不多，都是用于产生内联方法，只不过 Lambda 表达式的语法更为简洁。语法如下：

(参数列表) => 表达式或者语句块

其中：

参数个数：可以有多个参数，一个参数，或者无参数。

表达式或者语句块：这部分就是我们平常写函数的实现部分（函数体）。

前面的示例分别是 1 个参数的例子，下面结合扩展方法来一个复杂的例子：

```
        public delegate int mydg(int a, int b);


        public static class LambdaTest
        {


            public static int oper(this int a, int b, mydg dg)
            {
                return dg(a, b);
            }
        }


Console.WriteLine(1.oper(2, (a, b) => a + b));
Console.WriteLine(2.oper(1, (a, b) => a - b));
```

查询句法

```
var persons = new List<Person> {
    new Person {username = "a", age=19},
    new Person {username = "b", age=20},
    new Person {username = "a", age=21},
    };
```

```
var selectperson = from p in persons where p.age >= 20 select p.username.ToUpper();

foreach(var p in selectperson)

        Console.WriteLine(p);
```

查询句法是使用标准的 LINQ 查询运算符来表达查询时一个方便的声明式简化写法。该句法能在代码里表达查询时增进可读性和简洁性，读起来容易，也容易让人写对。Visual Studio 对查询句法提供了完整的智能感应和编译时检查支持。编译器在底层把查询句法的表达式翻译成明确的方法调用代码，代码通过新的扩展方法和 Lambda 表达式语言特性来实现。上面的查询句法等价于下面的代码：

```
var selectperson = persons.Where(p=>p.age>=20).Select(p=>p.username.ToUpper());
```

LINQ 查询句法可以实现 90％以上 T-SQL 的功能（由于 T-SQL 是基于二维表的，所以 LINQ 的查询语法会比 T-SQL 更简单和灵活），但是由于智能感应的原因，select 不能放在一开始就输入。

今天就说到这里，再见！

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（二）：DataContext 与实体

DataContext

DataContext 类型（数据上下文）是 System.Data.Linq 命名空间下的重要类型，用于把查询句法翻译成 SQL 语句，以及把数据从数据库返回给调用方和把实体的修改写入数据库。

DataContext 提供了以下一些使用的功能：

以日志形式记录 DataContext 生成的 SQL

执行 SQL（包括查询和更新语句）

创建和删除数据库

DataContext 是实体和数据库之间的桥梁，那么首先我们需要定义映射到数据表的实体。

定义实体类

```
using System.Data.Linq.Mapping;



[Table(Name = "Customers")]

public class Customer

{

    [Column(IsPrimaryKey = true)]

    public string CustomerID {get; set;}
```

```
    [Column(Name = "ContactName")]
    public string Name { get; set; }


    [Column]
    public string City {get; set;}
}
```

以 Northwind 数据库为例，上述 Customers 类被映射成一个表，对应数据库中的 Customers 表。然后在类型中定义了三个属性，对应表中的三个字段。其中，CustomerID 字段是主键，如果没有指定 Column 特性的 Name 属性，那么系统会把属性名作为数据表的字段名，也就是说实体类的属性名就需要和数据表中的字段名一致。

现在，创建一个 ASP.NET 页面，然后在页面上加入一个 GridView 控件，使用下面的代码进行绑定数据：

```
using System.Data.Linq;


DataContext ctx = new
DataContext("server=xxx;database=Northwind;uid=xxx;pwd=xxx");
Table<Customer> Customers = ctx.GetTable<Customer>();
GridView1.DataSource = from c in Customers where c.CustomerID.StartsWith("A") select
new {顾客 ID=c.CustomerID，顾客名=c.Name，城市=c.City};
GridView1.DataBind();
```

使用 DataContext 类型把实体类和数据库中的数据进行关联。你可以直接在 DataContext 的构造方法中定义连接字符串，也可以使用 IDbConnection：

```
using System.Data.SqlClient;



IDbConnection conn = new
SqlConnection("server=xxx;database=Northwind;uid=xxx;pwd=xxx");

DataContext ctx = new DataContext(conn);
```

之后，通过 GetTable 获取表示底层数据表的 Table 类型，显然，数据库中的 Customers 表的实体是 Customer 类型。随后的查询句法，即使你不懂 SQL 应该也能看明白。从 Customers 表中找出 CustomerID 以“A”开头的记录，并把 CustomersID、Name 以及 City 封装成新的匿名类型进行返回。

结果如下图：

顾客ID	顾客名	城市
ALFKI	Maria Anders	Berlin
ANATR	Ana Trujillo	México D.F.
ANTON	Antonio Moreno	México D.F.
AROUT	Thomas Hardy	London

强类型 DataContext

```
public partial class NorthwindDataContext : DataContext

{

    public Table<Customer> Customers;

    public NorthwindDataContext(IDbConnection connection) : base(connection) { }

    public NorthwindDataContext(string connection) : base(connection) { }

}
```

强类型数据上下文使代码更简洁：

```
NorthwindDataContext ctx = new
NorthwindDataContext("server=xxx;database=Northwind;uid=xxx;pwd=xxx");

GridView1.DataSource = from c in ctx.Customers where c.CustomerID.StartsWith("A")
select new { 顾客 ID = c.CustomerID, 顾客名 = c.Name, 城市 = c.City };

GridView1.DataBind();
```

DataContext 其实封装了很多实用的功能，下面一一介绍。

日志功能

```
using System.IO;



NorthwindDataContext ctx = new
NorthwindDataContext("server=xxx;database=Northwind;uid=xxx;pwd=xxx");

StreamWriter sw = new StreamWriter(Server.MapPath("log.txt"), true); // Append

ctx.Log = sw;

GridView1.DataSource = from c in ctx.Customers where c.CustomerID.StartsWith("A")
select new { 顾客 ID = c.CustomerID, 顾客名 = c.Name, 城市 = c.City };

GridView1.DataBind();

sw.Close();
```

运行程序后在网站所在目录生成了 log.txt，每次查询都会把诸如下面的日志追加到文本文件中：

```
SELECT [t0].[CustomerID], [t0].[ContactName], [t0].[City]

FROM [Customers] AS [t0]
```

```
WHERE [t0].[CustomerID] LIKE @p0

-- @p0: Input String (Size = 2; Prec = 0; Scale = 0) [A%]

-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.20706.1
```

应该说这样的日志对于调试程序是非常有帮助的。

探究查询

```
using System.Data.Common;

using System.Collections.Generic;



NorthwindDataContext ctx = new
NorthwindDataContext("server=xxx;database=Northwind;uid=xxx;pwd=xxx");

var select = from c in ctx.Customers where c.CustomerID.StartsWith("A") select new { 顾
客 ID = c.CustomerID, 顾客名 = c.Name, 城市 = c.City };

DbCommand cmd = ctx.GetCommand(select);

Response.Write(cmd.CommandText + "<br/>");

foreach (DbParameter parm in cmd.Parameters)

    Response.Write(string.Format("参数名:{0},参数值:{1}<br/>", parm.ParameterName,
parm.Value));

Customer customer = ctx.Customers.First();

customer.Name = "zhuye";

IList<object> queryText = ctx.GetChangeSet().ModifiedEntities;

Response.Write(((Customer)queryText[0]).Name);
```

在这里，我们通过 DataContext 的 GetCommand 方法获取了查询对应的 DbCommand，并且输出了 CommandText 和所有的 DbParameter。之后，我们又通过 GetChangeSet 方法获取了修改后的实体，并输出了修改内容。

执行查询

```
NorthwindDataContext ctx = new
NorthwindDataContext("server=xxx;database=Northwind;uid=xxx;pwd=xxx");

string newcity = "Shanghai";

ctx.ExecuteCommand("update Customers set City={0} where CustomerID like 'A%'",
newcity);

IEnumerable<Customer> customers = ctx.ExecuteQuery<Customer>("select * from
Customers where CustomerID like 'A%'");

GridView1.DataSource = customers;

GridView1.DataBind();
```

前一篇文章已经说了，虽然 Linq to sql 能实现 90％以上的 TSQL 功能。但是不可否认，对于复杂的查询，使用 TSQL 能获得更好的效率。因此，DataContext 类型也提供了执行 SQL 语句的能力。代码的执行结果如下图：

CustomerID	Name	City
ALFKI	Maria Anders	Shanghai
ANATR	Ana Trujillo	Shanghai
ANTON	Antonio Moreno	Shanghai
AROUT	Thomas Hardy	Shanghai

创建数据库

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
testContext ctx = new testContext("server=xxx;database=testdb;uid=xxx;pwd=xxx");

ctx.CreateDatabase();



[Table(Name = "test")]

public class test

{

    [Column(IsPrimaryKey = true, IsDbGenerated = true)]

    public int ID { get; set; }



    [Column(DbType="varchar(20)")]

    public string Name { get; set; }

}



public partial class testContext : DataContext

{

    public Table<test> test;

    public testContext(string connection) : base(connection) { }

}
```

这段代码在数据库中创建了名为 testdb 的数据库，等同于下面的脚本：

```
CREATE TABLE [dbo].[test](
```

```
    [ID] [int] IDENTITY(1,1) NOT NULL,

    [Name] [varchar](20) COLLATE Chinese_PRC_CI_AS NULL,

 CONSTRAINT [PK_test] PRIMARY KEY CLUSTERED

(

    [ID] ASC

)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]
```

同时，DataContext 还提供了 DeleteDatabase()方法，在这里就不列举了。

使用 DbDataReader 数据源

```
using System.Data.SqlClient;



var conn = new SqlConnection("server=xxx;database=Northwind;uid=xxx;pwd=xxx");

var ctx = new DataContext(conn);

var cmd = new SqlCommand("select * from customers where CustomerID like 'A%'",

conn);

conn.Open();

var reader = cmd.ExecuteReader();

GridView1.DataSource = ctx.Translate<Customer>(reader);

GridView1.DataBind();
```

```
conn.Close();
```

你同样可以选择使用 DataReader 获取数据，增加了灵活性的同时也增加了性能。

看到这里，你可能会觉得手工定义和数据库中表对应的实体类很麻烦，不用担心，VS2008 提供了自动生成实体类以及关系的工具，工具的使用将在以后讲解。今天就讲到这里，和 DataContext 相关的事务、加载选项、并发选项以及关系实体等高级内容也将在以后讲解。

CustomerID	Name	City
ALFKI	Maria Anders	Shanghai
ANATR	Ana Trujillo	Shanghai
ANTON	Antonio Moreno	Shanghai
AROUT	Thomas Hardy	Shanghai

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（三）：增删改

示例数据库

字段名	字段类型	允许空	字段说明
ID	uniqueidentifier		表主键字段
UserName	varchar(50)		留言用户名
PostTime	datetime		留言时间
Message	varchar(400)	√	留言内容
IsReplied	bit		留言是否回复
Reply	varchar(400)	√	留言管理员回复

在数据库中创建一个名为 GuestBook 的数据库，在里面创建一个 tbGuestBook 的表，结构如上表。

生成实体类

右键点击网站项目，选择添加新项，然后选择“Linq to sql Classes”，命名为 GuestBook。然后打开 App_Code 里面的 GuestBook.dbml。设计视图上的文字提示你可以从服务器资源管理器或者攻击箱拖动项到设计界面上来创建实体类。

那么，我们就在服务器资源管理器中创建一个指向 GuestBook 数据库的数据连接，然后把 tbGuestBook 表拖动到 GuestBook.dbml 的设计视图上，按 CTRL+S 保存。打开 GuestBook.designer.cs 可以发现系统自动创建了 GuestBook 数据库中 tbGuestBook 表的映射，如下图：

```
[Table(Name="dbo.tbGuestBook")]
public partial class tbGuestBook : INotifyPropertyChanging, INotifyPropertyChanged
{

    private static PropertyChangingEventArgs emptyChangingEventArgs = new PropertyChangingEventArgs

    private System.Guid _ID;

    private string _UserName;

    private System.DateTime _PostTime;

    private string _Message;

    private bool _IsReplied;

    private string _Reply;

    Extensibility Method Definitions

    public tbGuestBook()
    {
        OnCreated();
    }

    [Column(Storage="_ID", DbType="UniqueIdentifier NOT NULL", IsPrimaryKey=true)]
    public System.Guid ID...

    [Column(Storage="_UserName", DbType="VarChar(50) NOT NULL", CanBeNull=false)]
    public string UserName...

    [Column(Storage="_PostTime", DbType="DateTime NOT NULL")]
    public System.DateTime PostTime...
```

简易留言簿

现在，我们就可以使用 Linq to sql 完成简易留言簿了。实现以下功能：

发表留言（增）

查看留言（查）

管理员回复留言（改）

管理员删除留言（删除）

首先，创建一个 Default.aspx，在页面上加入一些控件：

```
<div>
        姓名
        <asp:TextBox ID="tb_UserName" runat="server"></asp:TextBox><br />
        <br />
        留言
        <asp:TextBox ID="tb_Message" runat="server" Height="100px"
TextMode="MultiLine" Width="300px"></asp:TextBox><br />
        <br />
        <asp:Button ID="btn_SendMessage" runat="server" Text="发表留言"
OnClick="btn_SendMessage_Click" /><br />
        <br />
        <asp:Repeater ID="rpt_Message" runat="server">
        <ItemTemplate>
        <table width="600px" style="border:solid 1px #666666; font-size:10pt;
background-color:#f0f0f0">
            <tr>
            <td align="left" width="400px">
            <%# Eval("Message")%>
            </td>
            <td align="right" width="200px">
            <%# Eval("PostTime")%> - <%# Eval("UserName")%>
```

```
                </td>
                </tr>
                <tr>
                <td colspan="2" align="right">
                <hr width="300px" />
                管理员回复：<%# Eval("IsReplied").ToString() == "False" ? "暂无" :
Eval("Reply")%>
                </td>
                </tr>
            </table>
            <br/>
            </ItemTemplate>
            </asp:Repeater>
        </div>
```

你可能很难想象，使用 Linq to sql 进行数据访问会是这么简单，后台代码：

```
public partial class _Default : System.Web.UI.Page
{
    GuestBookDataContext ctx = new
GuestBookDataContext("server=xxx;database=GuestBook;uid=xxx;pwd=xxx");

    protected void Page_Load(object sender, EventArgs e)
```

```
{
    if (!IsPostBack)
    {
        SetBind();
    }
}
protected void btn_SendMessage_Click(object sender, EventArgs e)
{
    tbGuestBook gb = new tbGuestBook();
    gb.ID = Guid.NewGuid();
    gb.UserName = tb_UserName.Text;
    gb.Message = tb_Message.Text;
    gb.IsReplied = false;
    gb.PostTime = DateTime.Now;
    ctx.tbGuestBooks.Add(gb);
    ctx.SubmitChanges();
    SetBind();
}
private void SetBind()
{
```

```
            rpt_Message.DataSource = from gb in ctx.tbGuestBooks orderby gb.PostTime
descending select gb;

            rpt_Message.DataBind();

        }

}
```

前面创建 Linq to sql Classes 的时候我们输入名字 GuestBook，系统就为我们自动创建了 GuestBookDataContext（你也可以在 GuestBook.Designer.cs 中找到类定义）。在绑定的时候我们使用查询句法查询留言表中所有留言，按照发表时间倒序（天哪？这是数据访问吗？好像仅仅定义了一句 SQL 啊）。在发表留言按钮中，我们为一个 tbGuestBook 赋值，然后把它加入留言表，再提交更改，就这样完成了记录的插入。

运行效果如下图：

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

然后，再创建一个 Admin.aspx，前台代码如下：

```
<div>

        <asp:Repeater ID="rpt_Message" runat="server"
OnItemCommand="rpt_Message_ItemCommand">

        <ItemTemplate>

        <table width="600px" style="border:solid 1px #666666; font-size:10pt;
background-color:#f0f0f0">

            <tr>

            <td align="left" width="400px">
```

```
                <%# Eval("Message")%>
                </td>
                <td align="right" width="200px">
                <%# Eval("PostTime")%> - <%# Eval("UserName")%>
                </td>
                </tr>
                <tr>
                <td colspan="2" align="right">
                <hr width="300px" />
                <asp:Button ID="btn_DeleteMessage" runat="server" Text="删除留言"
CommandName="DeleteMessage" CommandArgument='<%# Eval("ID")%>'/>
                管理员回复：<asp:TextBox runat="server" ID="tb_Reply"
TextMode="MultiLine" Width="300px" Text='<%# Eval("Reply")%>'/>
                <asp:Button ID="btn_SendReply" runat="server" Text="发表回复"
CommandName="SendReply" CommandArgument='<%# Eval("ID")%>'/>
                </td>
                </tr>
            </table>
            <br/>
            </ItemTemplate>
            </asp:Repeater>
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
    </div>
```

后台代码：

```
public partial class Admin : System.Web.UI.Page
{
    GuestBookDataContext ctx = new
GuestBookDataContext("server=xxx;database=GuestBook;uid=xxx;pwd=xxx");

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            SetBind();
        }
    }

    private void SetBind()
    {
        rpt_Message.DataSource = from gb in ctx.tbGuestBooks orderby gb.PostTime
descending select gb;
        rpt_Message.DataBind();
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
    }

    protected void rpt_Message_ItemCommand(object source,
RepeaterCommandEventArgs e)

    {

        if (e.CommandName == "DeleteMessage")

        {

            StreamWriter sw = new StreamWriter(Server.MapPath("log.txt"), true);

            ctx.Log = sw;

            tbGuestBook gb = ctx.tbGuestBooks.Single(b => b.ID == new
Guid(e.CommandArgument.ToString()));

            ctx.tbGuestBooks.Remove(gb);

            ctx.SubmitChanges();

            SetBind();

            sw.Close();

        }

        if (e.CommandName == "SendReply")

        {

            StreamWriter sw = new StreamWriter(Server.MapPath("log.txt"), true);

            ctx.Log = sw;

            tbGuestBook gb = ctx.tbGuestBooks.Single(b => b.ID == new
Guid(e.CommandArgument.ToString()));
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
                gb.Reply = ((TextBox)e.Item.FindControl("tb_Reply")).Text;

                gb.IsReplied = true;

                ctx.SubmitChanges();

                SetBind();

                sw.Close();

            }

        }

}
```

运行效果如下图：

在这里，我们通过 Single 方法获取一条记录，也就是一个 tbGuestBook 实例，更新了一些属性后保存也就完成了改这个操作。删除操作更简单，只需要从表中移除对象。你是不是觉得好像不是在操作数据库，像在操作内存中的对象。

由于写了日志，看看改和删操作会是怎么样的 SQL？

```
UPDATE [dbo].[tbGuestBook]

SET [IsReplied] = @p4, [Reply] = @p5

WHERE ([ID] = @p0) AND ([UserName] = @p1) AND ([PostTime] = @p2) AND
([Message] = @p3) AND (NOT ([IsReplied] = 1)) AND ([Reply] IS NULL)

-- @p0: Input Guid (Size = 0; Prec = 0; Scale = 0)
[00000000-0000-0000-0000-000000000000]

-- @p1: Input String (Size = 4; Prec = 0; Scale = 0) [ghgh]

-- @p2: Input DateTime (Size = 0; Prec = 0; Scale = 0) [2007-8-16 10:20:09]

-- @p3: Input String (Size = 3; Prec = 0; Scale = 0) [ghj]

-- @p4: Input Boolean (Size = 0; Prec = 0; Scale = 0) [True]

-- @p5: Input String (Size = 3; Prec = 0; Scale = 0) [qqq]

-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.20706.1



DELETE FROM [dbo].[tbGuestBook] WHERE ([ID] = @p0) AND ([UserName] = @p1)
AND ([PostTime] = @p2) AND ([Message] = @p3) AND (NOT ([IsReplied] = 1)) AND
([Reply] = @p4)

-- @p0: Input Guid (Size = 0; Prec = 0; Scale = 0)
[158ec941-13ff-4093-bd8b-9fceae152171]
```

```
-- @p1: Input String (Size = 2; Prec = 0; Scale = 0) [44]

-- @p2: Input DateTime (Size = 0; Prec = 0; Scale = 0) [2007-8-16 9:56:19]

-- @p3: Input String (Size = 2; Prec = 0; Scale = 0) [44]

-- @p4: Input String (Size = 3; Prec = 0; Scale = 0) [222]

-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.20706.1
```

今天就讲到这里，下次将系统介绍查询句法。

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（四）：查询句法

select

描述：查询顾客的公司名、地址信息

查询句法：

```
var 构建匿名类型 1 = from c in ctx.Customers
                     select new
                     {
                         公司名 = c.CompanyName,
                         地址 = c.Address
                     };
```

对应 SQL：

```
SELECT [t0].[CompanyName], [t0].[Address]
FROM [dbo].[Customers] AS [t0]
```

描述：查询职员的姓名和雇用年份

查询句法：

```
var 构建匿名类型 2 = from emp in ctx.Employees
                     select new
                     {
                         姓名 = emp.LastName + emp.FirstName,
                         雇用年 = emp.HireDate.Value.Year
                     };
```

对应 SQL：

```
SELECT [t0].[LastName] + [t0].[FirstName] AS [value], DATEPART(Year, [t0].[HireDate])
AS [value2]
```

```
FROM [dbo].[Employees] AS [t0]
```

描述：查询顾客的 ID 以及联系信息（职位和联系人）

查询句法：

```
var 构建匿名类型 3 = from c in ctx.Customers
                     select new
                     {
                         ID = c.CustomerID,
                         联系信息 = new
                         {
                             职位 = c.ContactTitle,
                             联系人 = c.ContactName
                         }
                     };
```

对应 SQL：

```
SELECT [t0].[CustomerID], [t0].[ContactTitle], [t0].[ContactName]
FROM [dbo].[Customers] AS [t0]
```

描述：查询订单号和订单是否超重的信息

查询句法：

```
var select 带条件 = from o in ctx.Orders
                    select new
                    {
                        订单号 = o.OrderID,
                        是否超重 = o.Freight > 100 ? "是" : "否"
                    };
```

对应 SQL：

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
SELECT [t0].[OrderID],
    (CASE
        WHEN [t0].[Freight] > @p0 THEN @p1
        ELSE @p2
     END) AS [value]
FROM [dbo].[Orders] AS [t0]
-- @p0: Input Currency (Size = 0; Prec = 19; Scale = 4) [100]
-- @p1: Input String (Size = 1; Prec = 0; Scale = 0) [是]
-- @p2: Input String (Size = 1; Prec = 0; Scale = 0) [否]
```

where

描述：查询顾客的国家、城市和订单数信息，要求国家是法国并且订单数大于 5

查询句法：

```
var 多条件 = from c in ctx.Customers
                where c.Country == "France" && c.Orders.Count > 5
                select new
                {
                    国家 = c.Country,
                    城市 = c.City,
                    订单数 = c.Orders.Count
                };
```

对应 SQL：

```
SELECT [t0].[Country], [t0].[City], (
    SELECT COUNT(*)
    FROM [dbo].[Orders] AS [t2]
    WHERE [t2].[CustomerID] = [t0].[CustomerID]
    ) AS [value]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
FROM [dbo].[Customers] AS [t0]
WHERE ([t0].[Country] = @p0) AND (((
    SELECT COUNT(*)
    FROM [dbo].[Orders] AS [t1]
    WHERE [t1].[CustomerID] = [t0].[CustomerID]
    )) > @p1)
-- @p0: Input String (Size = 6; Prec = 0; Scale = 0) [France]
-- @p1: Input Int32 (Size = 0; Prec = 0; Scale = 0) [5]
```

orderby

描述：查询所有没有下属雇员的雇用年和名，按照雇用年倒序，按照名正序

查询句法：

```
  var 排序 = from emp in ctx.Employees
                    where emp.Employees.Count == 0
                    orderby emp.HireDate.Value.Year descending, emp.FirstName
ascending
                    select new
                    {
                        雇用年 = emp.HireDate.Value.Year,
                        名 = emp.FirstName
                    };
```

对应 SQL：

```
SELECT DATEPART(Year, [t0].[HireDate]) AS [value], [t0].[FirstName]
FROM [dbo].[Employees] AS [t0]
WHERE ((
    SELECT COUNT(*)
```

```
    FROM [dbo].[Employees] AS [t1]
    WHERE [t1].[ReportsTo] = [t0].[EmployeeID]
    )) = @p0
ORDER BY DATEPART(Year, [t0].[HireDate]) DESC, [t0].[FirstName]
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [0]
```

分页

描述：按照每页 10 条记录，查询第二页的顾客

查询句法：

```
var 分页 = (from c in ctx.Customers select c).Skip(10).Take(10);
```

对应 SQL：

```
SELECT TOP 10 [t1].[CustomerID], [t1].[CompanyName], [t1].[ContactName],
[t1].[ContactTitle], [t1].[Address], [t1].[City], [t1].[Region], [t1].[PostalCode], [t1].[Country],
[t1].[Phone], [t1].[Fax]
FROM (
    SELECT ROW_NUMBER() OVER (ORDER BY [t0].[CustomerID],
[t0].[CompanyName], [t0].[ContactName], [t0].[ContactTitle], [t0].[Address], [t0].[City],
[t0].[Region], [t0].[PostalCode], [t0].[Country], [t0].[Phone], [t0].[Fax]) AS
[ROW_NUMBER], [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax]
    FROM [dbo].[Customers] AS [t0]
    ) AS [t1]
WHERE [t1].[ROW_NUMBER] > @p0
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [10]
```

分组

描述：根据顾客的国家分组，查询顾客数大于 5 的国家名和顾客数

查询句法：

```
var 一般分组 = from c in ctx.Customers
               group c by c.Country into g
               where g.Count() > 5
               orderby g.Count() descending
               select new
               {
                   国家 = g.Key,
                   顾客数 = g.Count()
               };
```

对应 SQL：

```
SELECT [t1].[Country], [t1].[value3] AS [顾客数]
FROM (
    SELECT COUNT(*) AS [value], COUNT(*) AS [value2], COUNT(*) AS [value3],
[t0].[Country]
    FROM [dbo].[Customers] AS [t0]
    GROUP BY [t0].[Country]
    ) AS [t1]
WHERE [t1].[value] > @p0
ORDER BY [t1].[value2] DESC
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [5]
```

描述：根据国家和城市分组，查询顾客覆盖的国家和城市

查询句法：

```
var 匿名类型分组 = from c in ctx.Customers
```

```
                    group c by new { c.City, c.Country } into g
                    orderby g.Key.Country, g.Key.City
                    select new
                    {
                        国家 = g.Key.Country,
                        城市 = g.Key.City
                    };
```

对应 SQL：

```
SELECT [t1].[Country], [t1].[City]
FROM (
    SELECT [t0].[City], [t0].[Country]
    FROM [dbo].[Customers] AS [t0]
    GROUP BY [t0].[City], [t0].[Country]
    ) AS [t1]
ORDER BY [t1].[Country], [t1].[City]
```

描述：按照是否超重条件分组，分别查询订单数量

查询句法：

```
var 按照条件分组 = from o in ctx.Orders
                    group o by new { 条件 = o.Freight > 100 } into g
                    select new
                    {
                        数量 = g.Count(),
                        是否超重 = g.Key.条件 ? "是" : "否"
                    };
```

对应 SQL：

```
SELECT
```

```
    (CASE
        WHEN [t2].[value2] = 1 THEN @p1
        ELSE @p2
     END) AS [value], [t2].[value] AS [数量]
FROM (
    SELECT COUNT(*) AS [value], [t1].[value] AS [value2]
    FROM (
        SELECT
            (CASE
                WHEN [t0].[Freight] > @p0 THEN 1
                WHEN NOT ([t0].[Freight] > @p0) THEN 0
                ELSE NULL
             END) AS [value]
        FROM [dbo].[Orders] AS [t0]
        ) AS [t1]
    GROUP BY [t1].[value]
    ) AS [t2]
-- @p0: Input Currency (Size = 0; Prec = 19; Scale = 4) [100]
-- @p1: Input String (Size = 1; Prec = 0; Scale = 0) [是]
-- @p2: Input String (Size = 1; Prec = 0; Scale = 0) [否]
```

distinct

描述：查询顾客覆盖的国家

查询句法：

```
var 过滤相同项 = (from c in ctx.Customers orderby c.Country select c.Country).Distinct();
```

对应 SQL：

```
SELECT DISTINCT [t0].[Country]
```

```
FROM [dbo].[Customers] AS [t0]
```

union

描述：查询城市是 A 打头和城市包含 A 的顾客并按照顾客名字排序

查询句法：

```
var 连接并且过滤相同项 = (from c in ctx.Customers where c.City.Contains("A") select
c).Union
                (from c in ctx.Customers where c.ContactName.StartsWith("A") select
c).OrderBy(c => c.ContactName);
```

对应 SQL：

```
SELECT [t3].[CustomerID], [t3].[CompanyName], [t3].[ContactName], [t3].[ContactTitle],
[t3].[Address], [t3].[City], [t3].[Region], [t3].[PostalCode], [t3].[Country], [t3].[Phone],
[t3].[Fax]
FROM (
    SELECT [t2].[CustomerID], [t2].[CompanyName], [t2].[ContactName],
[t2].[ContactTitle], [t2].[Address], [t2].[City], [t2].[Region], [t2].[PostalCode], [t2].[Country],
[t2].[Phone], [t2].[Fax]
    FROM (
        SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax]
        FROM [dbo].[Customers] AS [t0]
        WHERE [t0].[City] LIKE @p0
        UNION
        SELECT [t1].[CustomerID], [t1].[CompanyName], [t1].[ContactName],
[t1].[ContactTitle], [t1].[Address], [t1].[City], [t1].[Region], [t1].[PostalCode], [t1].[Country],
```

```
[t1].[Phone], [t1].[Fax]
        FROM [dbo].[Customers] AS [t1]
        WHERE [t1].[ContactName] LIKE @p1
        ) AS [t2]
    ) AS [t3]
ORDER BY [t3].[ContactName]
-- @p0: Input String (Size = 3; Prec = 0; Scale = 0) [%A%]
-- @p1: Input String (Size = 2; Prec = 0; Scale = 0) [A%]
```

concat

描述：查询城市是 A 打头和城市包含 A 的顾客并按照顾客名字排序，相同的顾客信息不会过滤

查询句法：

```
var 连接并且不过滤相同项 = (from c in ctx.Customers where c.City.Contains("A") select c).Concat
            (from c in ctx.Customers where c.ContactName.StartsWith("A") select c).OrderBy(c => c.ContactName);
```

对应 SQL：

```
SELECT [t3].[CustomerID], [t3].[CompanyName], [t3].[ContactName], [t3].[ContactTitle], [t3].[Address], [t3].[City], [t3].[Region], [t3].[PostalCode], [t3].[Country], [t3].[Phone], [t3].[Fax]
FROM (
    SELECT [t2].[CustomerID], [t2].[CompanyName], [t2].[ContactName], [t2].[ContactTitle], [t2].[Address], [t2].[City], [t2].[Region], [t2].[PostalCode], [t2].[Country], [t2].[Phone], [t2].[Fax]
    FROM (
```

```
            SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax]
            FROM [dbo].[Customers] AS [t0]
            WHERE [t0].[City] LIKE @p0
            UNION ALL
            SELECT [t1].[CustomerID], [t1].[CompanyName], [t1].[ContactName],
[t1].[ContactTitle], [t1].[Address], [t1].[City], [t1].[Region], [t1].[PostalCode], [t1].[Country],
[t1].[Phone], [t1].[Fax]
            FROM [dbo].[Customers] AS [t1]
            WHERE [t1].[ContactName] LIKE @p1
            ) AS [t2]
      ) AS [t3]
ORDER BY [t3].[ContactName]
-- @p0: Input String (Size = 3; Prec = 0; Scale = 0) [%A%]
-- @p1: Input String (Size = 2; Prec = 0; Scale = 0) [A%]
```

取相交项

描述：查询城市是 A 打头的顾客和城市包含 A 的顾客的交集，并按照顾客名字排序

查询句法：

```
var 取相交项 = (from c in ctx.Customers where c.City.Contains("A") select c).Intersect
                (from c in ctx.Customers where c.ContactName.StartsWith("A") select
c).OrderBy(c => c.ContactName);
```

对应 SQL：

```
SELECT [t1].[CustomerID], [t1].[CompanyName], [t1].[ContactName], [t1].[ContactTitle],
[t1].[Address], [t1].[City], [t1].[Region], [t1].[PostalCode], [t1].[Country], [t1].[Phone],
[t1].[Fax]
```

```
FROM (
    SELECT DISTINCT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax]
    FROM [dbo].[Customers] AS [t0]
    ) AS [t1]
WHERE (EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [dbo].[Customers] AS [t2]
    WHERE ([t1].[CustomerID] = [t2].[CustomerID]) AND ([t2].[ContactName] LIKE @p0)
    )) AND ([t1].[City] LIKE @p1)
ORDER BY [t1].[ContactName]
-- @p0: Input String (Size = 2; Prec = 0; Scale = 0) [A%]
-- @p1: Input String (Size = 3; Prec = 0; Scale = 0) [%A%]
```

排除相交项

描述：查询城市包含 A 的顾客并从中删除城市以 A 开头的顾客，并按照顾客名字排序

查询句法：

```
var 排除相交项 = (from c in ctx.Customers where c.City.Contains("A") select c).Except
                (from c in ctx.Customers where c.ContactName.StartsWith("A") select
c).OrderBy(c => c.ContactName);
```

对应 SQL：

```
SELECT [t1].[CustomerID], [t1].[CompanyName], [t1].[ContactName], [t1].[ContactTitle],
[t1].[Address], [t1].[City], [t1].[Region], [t1].[PostalCode], [t1].[Country], [t1].[Phone],
[t1].[Fax]
FROM (
```

```
    SELECT DISTINCT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax]
    FROM [dbo].[Customers] AS [t0]
    ) AS [t1]
WHERE (NOT (EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [dbo].[Customers] AS [t2]
    WHERE ([t1].[CustomerID] = [t2].[CustomerID]) AND ([t2].[ContactName] LIKE @p0)
    ))) AND ([t1].[City] LIKE @p1)
ORDER BY [t1].[ContactName]
-- @p0: Input String (Size = 2; Prec = 0; Scale = 0) [A%]
-- @p1: Input String (Size = 3; Prec = 0; Scale = 0) [%A%]
```

子查询

描述：查询订单数超过 5 的顾客信息

查询句法：

```
var 子查询 = from c in ctx.Customers
                    where
                        (from o in ctx.Orders group o by o.CustomerID into o where
o.Count() > 5 select o.Key).Contains(c.CustomerID)
                    select c;
```

对应 SQL：

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName], [t0].[ContactTitle],
[t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country], [t0].[Phone],
[t0].[Fax]
```

```
FROM [dbo].[Customers] AS [t0]
WHERE EXISTS(
    SELECT NULL AS [EMPTY]
    FROM (
        SELECT COUNT(*) AS [value], [t1].[CustomerID]
        FROM [dbo].[Orders] AS [t1]
        GROUP BY [t1].[CustomerID]
        ) AS [t2]
    WHERE ([t2].[CustomerID] = [t0].[CustomerID]) AND ([t2].[value] > @p0)
    )
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [5]
```

in 操作

描述：查询指定城市中的客户

查询句法：

```
        var in 操作  = from c in ctx.Customers
                       where new string[] { "Brandenburg", "Cowes",
"Stavern" }.Contains(c.City)
                       select c;
```

对应 SQL：

```
        SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax]
FROM [dbo].[Customers] AS [t0]
WHERE [t0].[City] IN (@p0, @p1, @p2)
-- @p0: Input String (Size = 11; Prec = 0; Scale = 0) [Brandenburg]
-- @p1: Input String (Size = 5; Prec = 0; Scale = 0) [Cowes]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
-- @p2: Input String (Size = 7; Prec = 0; Scale = 0) [Stavern]
```

join

描述：内连接，没有分类的产品查询不到

查询句法：

```
var innerjoin = from p in ctx.Products
                join c in ctx.Categories
                on p.CategoryID equals c.CategoryID
                select p.ProductName;
```

对应 SQL：

```
SELECT COUNT(*) AS [value]
FROM [dbo].[Products] AS [t0]
INNER JOIN [dbo].[Categories] AS [t1] ON [t0].[CategoryID] = ([t1].[CategoryID])
```

描述：外连接，没有分类的产品也能查询到

查询句法：

```
var leftjoin = from p in ctx.Products
               join c in ctx.Categories
               on p.CategoryID equals c.CategoryID
               into pro
               from x in pro.DefaultIfEmpty()
               select p.ProductName;
```

对应 SQL：

```
SELECT COUNT(*) AS [value]
FROM [dbo].[Products] AS [t0]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
LEFT OUTER JOIN [dbo].[Categories] AS [t1] ON [t0].[CategoryID] = ([t1].[CategoryID])
```

你可能会很奇怪，原先很复杂的 SQL 使用查询句法会很简单（比如按照条件分组）。但是原先觉得很好理解的 SQL 使用查询句法会觉得很复杂（比如连接查询）。其实，我们还可以通过其它方式进行连接操作，在以后说 DataLoadOptions 类型的时候会再说。虽然 Linq to sql 已经非常智能了，但是对于非常复杂的查询还是建议通过存储过程实现，下次讲解如何调用存储过程。

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

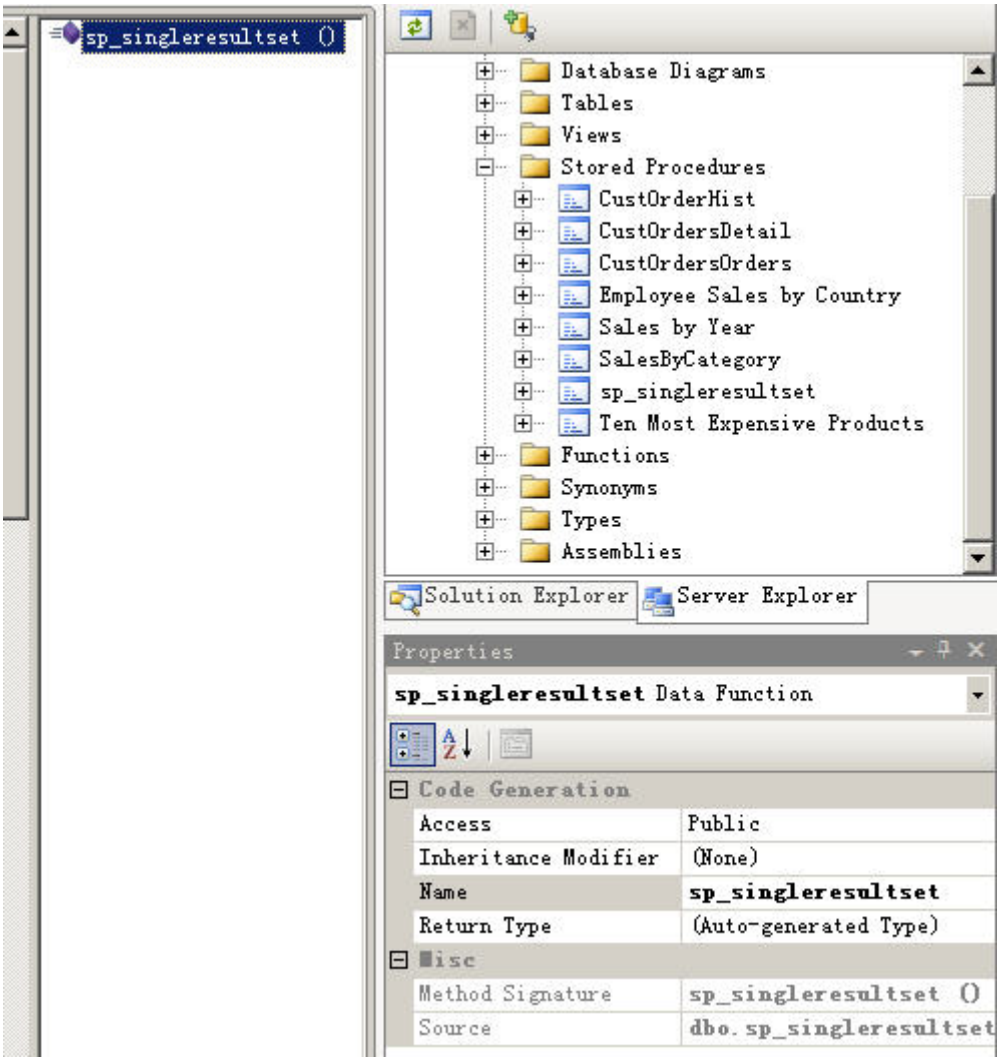
一步一步学 Linq to sql（五）：存储过程

普通存储过程

首先在查询分析器运行下面的代码来创建一个存储过程：

```
create proc sp_singleresultset

as

set nocount on

select * from customers
```

然后打开 IDE 的服务器资源管理器，之前我们从表中拖动表到 dbml 设计视图，这次我们从存储过程中找到刚才创建的存储过程，然后拖动到设计视图。在方法面板中可以看到已经创建了一个 sp_singleresultset 的方法，如下图：



然后打开 Northwind.designer.cs，可以找到下面的代码：

```
[Function(Name="dbo.sp_singleresultset")]

public ISingleResult<sp_singleresultsetResult> sp_singleresultset()

{

    IExecuteResult result = this.ExecuteMethodCall(this,
((MethodInfo)(MethodInfo.GetCurrentMethod())));

    return ((ISingleResult<sp_singleresultsetResult>)(result.ReturnValue));
```

```
}
```

我们可以发现，IDE 为这个存储过程单独生成了返回结果集的实体定义，你可能会觉得很奇怪，IDE 怎么知道这个存储过程将会返回哪些数据那？其实，在把存储过程拖拽入 dbml 设计视图的时候，IDE 就执行了类似下面的命令：

```
SET FMTONLY ON;

exec Northwind.dbo.sp_singleresultset

SET FMTONLY OFF;
```

这样就可以直接获取存储过程返回的元数据而无须执行存储过程。

其实我们存储过程返回的就是顾客表的数据，如果你觉得为存储过程单独设置结果集实体有些浪费的话可以在存储过程的属性窗口中调整返回类型从“自动生成的类型”到 Customer，不过以后你只能通过删除方法面板中的存储过程，然后重新添加来还原到“自动生成的类型”。下面，我们可以写如下的 Linq to object 代码进行查询：

```
var 单结果集存储过程 =
    from c in ctx.sp_singleresultset()
    where c.CustomerID.StartsWith("A")
    select c;
```

在这里确实是 Linq to object 的，因为查询句法不会被整句翻译成 SQL，而是从存储过程的返回对象中再去对对象进行查询。SQL 代码如下：

```
EXEC @RETURN_VALUE = [dbo].[sp_singleresultset]

-- @RETURN_VALUE: Output Int32 (Size = 0; Prec = 0; Scale = 0) [
```

带参数的存储过程

创建如下存储过程：

```
create proc [dbo].[sp_withparameter]

@customerid nchar(5),

@rowcount int output

as

set nocount on

set @rowcount = (select count(*) from customers where customerid = @customerid)
```

使用同样的方法生成存储过程方法，然后使用下面的代码进行测试：

```
int? rowcount = -1;

ctx.sp_withparameter("", ref rowcount);

Response.Write(rowcount);

ctx.sp_withparameter("ALFKI", ref rowcount);

Response.Write(rowcount);
```

结果输出了“01”。说明 ID 为“”的顾客数为 0，而 ID 为“ALFKI”的顾客数为 1。存储过程的输出参数被封装成了 ref 参数，对于 C#语法来说非常合情合理。SQL 代码如下：

```
EXEC @RETURN_VALUE = [dbo].[sp_withparameter] @customerid = @p0, @rowcount = @p1 OUTPUT

-- @p0: Input StringFixedLength (Size = 5; Prec = 0; Scale = 0) []

-- @p1: InputOutput Int32 (Size = 0; Prec = 0; Scale = 0) [-1]

-- @RETURN_VALUE: Output Int32 (Size = 0; Prec = 0; Scale = 0) []
```

带返回值的存储过程

再来创建第三个存储过程：

```
create proc [dbo].[sp_withreturnvalue]

@customerid nchar(5)

as

set nocount on

if exists (select 1 from customers where customerid = @customerid)

return 101

else

return 100
```

生成方法后，可以通过下面的代码进行测试：

```
        Response.Write(ctx.sp_withreturnvalue(""));

        Response.Write(ctx.sp_withreturnvalue("ALFKI"));
```

运行后程序输出“100101”

多结果集的存储过程

再来创建一个多结果集的存储过程：

```
create proc [dbo].[sp_multiresultset]

as

set nocount on

select * from customers
```

```
select * from employees
```

找到生成的存储过程方法：

```
[Function(Name="dbo.sp_multiresultset")]
    public ISingleResult<sp_multiresultsetResult> sp_multiresultset()
    {
        IExecuteResult result = this.ExecuteMethodCall(this,
((MethodInfo)(MethodInfo.GetCurrentMethod())));
        return ((ISingleResult<sp_multiresultsetResult>)(result.ReturnValue));
    }
```

由于现在的 VS2008 会把多结果集存储过程识别为单结果集存储过程（只认识第一个结果集），我们只能对存储过程方法多小动手术，修改为：

```
    [Function(Name="dbo.sp_multiresultset")]
    [ResultType(typeof(Customer))]
    [ResultType(typeof(Employee))]
    public IMultipleResults sp_multiresultset()
    {
        IExecuteResult result = this.ExecuteMethodCall(this,
((MethodInfo)(MethodInfo.GetCurrentMethod())));
        return (IMultipleResults)(result.ReturnValue);
    }
```

然后使用下面的代码测试：

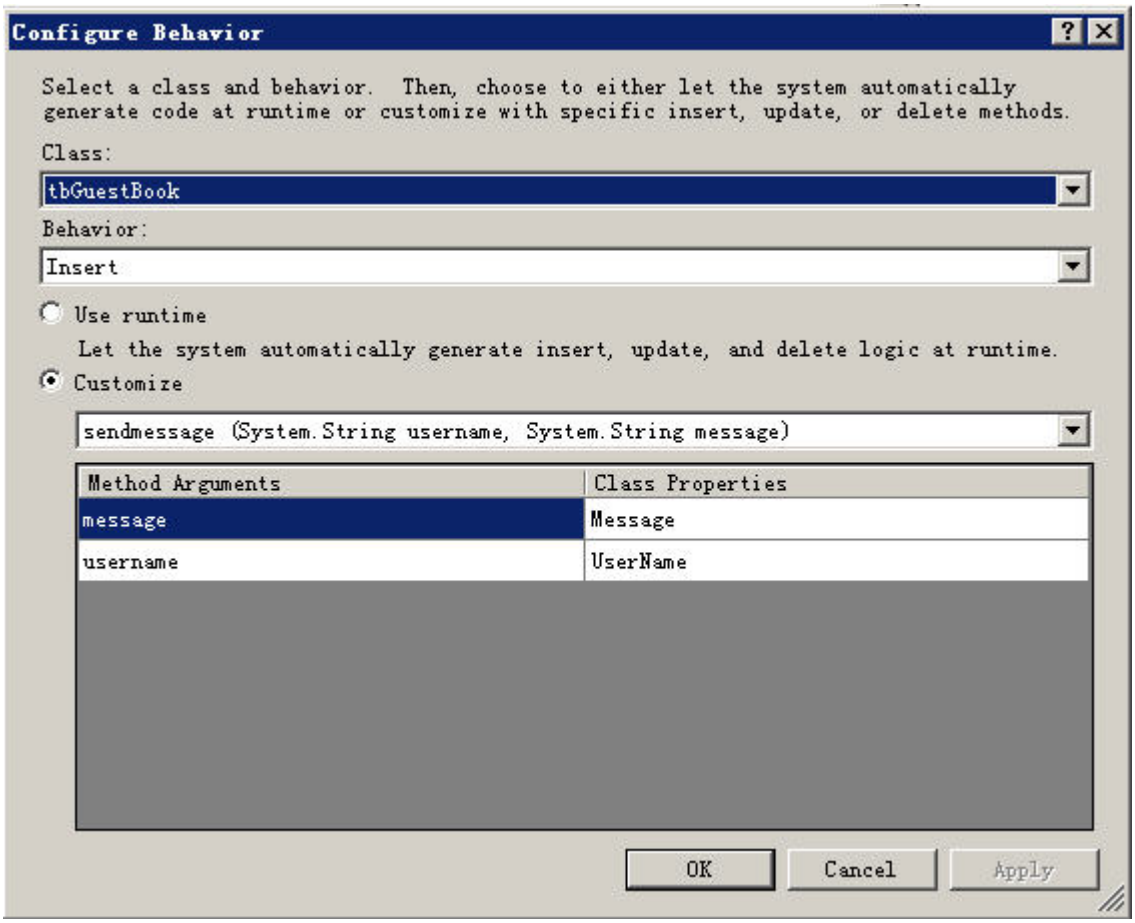
```
            var 多结果集存储过程 = ctx.sp_multiresultset();

            var Customers = 多结果集存储过程.GetResult<Customer>();

            var Employees = 多结果集存储过程.GetResult<Employee>();

            GridView1.DataSource = from emp in Employees where
emp.FirstName.Contains("A") select emp;

            GridView1.DataBind();

            GridView2.DataSource = from c in Customers where
c.CustomerID.StartsWith("A") select c;

            GridView2.DataBind()
```

使用存储过程新增数据

存储过程除了可以直接调用之外，还可以用于实体的增删改操作。还记得在《一步一步学 Linq to sql（三）：增删改》中创建的留言簿程序吗？下面我们就来改造这个程序，使用存储过程而不是系统生成的 SQL 实现实体增删改。首先，我们创建下面的存储过程

```
create proc sendmessage

@username varchar(50),

@message varchar(500)

as

insert into tbguestbook

(id,username,posttime,[message],isreplied,reply)

values

(newid(),@username,getdate(),@message,0,'')
```

然后，打开留言簿 dbml，把存储过程从服务器资源管理器拖拽到设计视图上。右键点击 tbGuestBook 实体类，选择配置行为。如下图，为插入操作选择刚才创建的存储过程方法，并进行参数匹配：



由于我们的存储过程只接受 2 个参数，相应修改以下创建留言的按钮处理事件：

```
protected void btn_SendMessage_Click(object sender, EventArgs e)

{

    tbGuestBook gb = new tbGuestBook();

    gb.UserName = tb_UserName.Text;

    gb.Message = tb_Message.Text;
```

```
            ctx.tbGuestBooks.Add(gb);

            ctx.SubmitChanges();

            SetBind();

        }
```

运行程序后可以发现，在提交修改的时候调用了下面的 SQL：

```
EXEC @RETURN_VALUE = [dbo].[sendmessage] @username = @p0, @message =
@p1

-- @p0: Input AnsiString (Size = 5; Prec = 0; Scale = 0) [zhuye]

-- @p1: Input AnsiString (Size = 11; Prec = 0; Scale = 0) [new message]

-- @RETURN_VALUE: Output Int32 (Size = 0; Prec = 0; Scale = 0) []
```

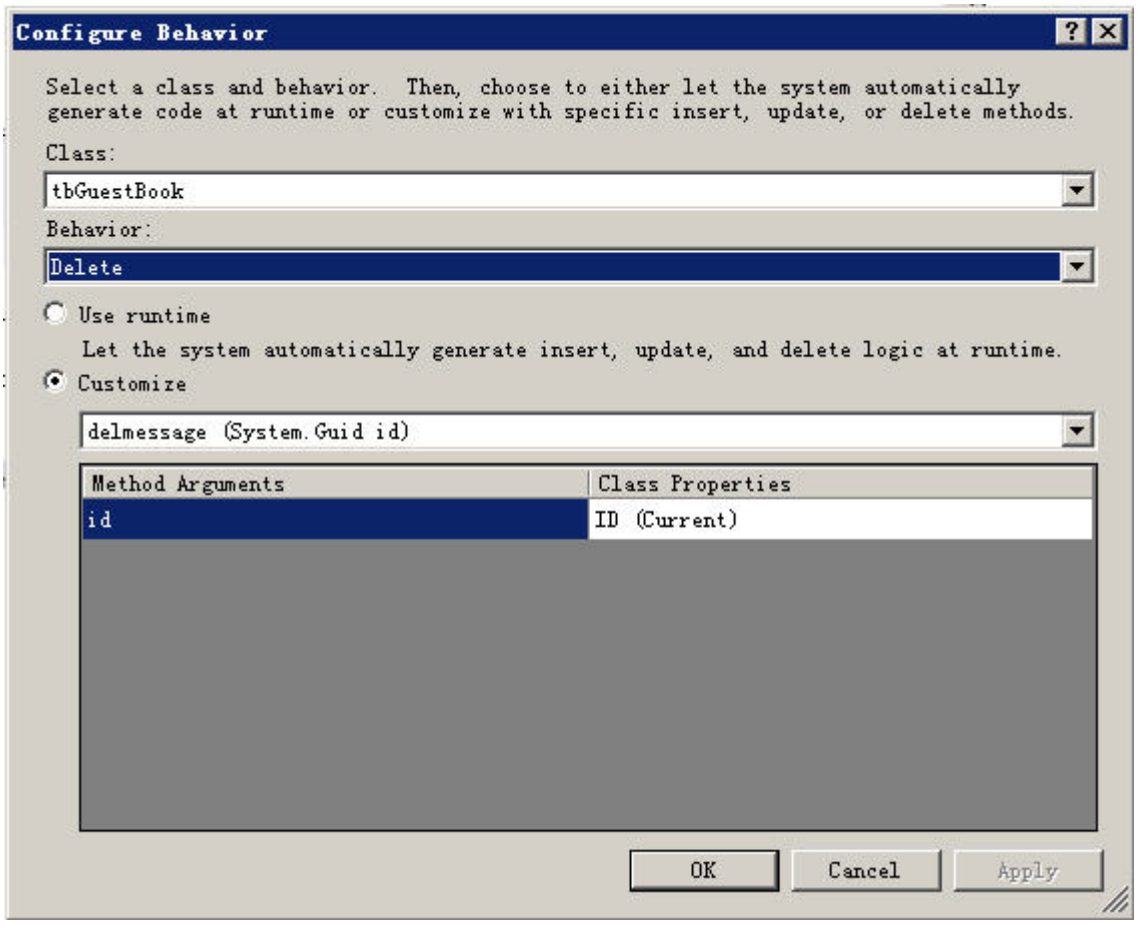
使用存储过程删除数据

创建如下存储过程：

```
create proc delmessage

@id uniqueidentifier

as

delete tbguestbook where id=@id
```

按照前面的步骤生成存储过程方法，并为删除操作执行这个存储过程方法。在选择参数的时候我们可以看到，ID 分当前值和原始值，我们选择当前值即可，如下图：

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816



无须改动任何逻辑代码，进行删除留言操作后可以跟踪到下面的 SQL：

```
EXEC @RETURN_VALUE = [dbo].[delmessage] @id = @p0

-- @p0: Input Guid (Size = 0; Prec = 0; Scale = 0)

[9e3c5ee3-2575-458e-899d-4b0bf73e0849]

-- @RETURN_VALUE: Output Int32 (Size = 0; Prec = 0; Scale = 0) []
```

使用存储过程更改数据

创建如下存储过程：

```
create proc replymessage
```

```
@id uniqueidentifier,

@reply varchar(500)

as

update tbguestbook set reply=@reply,isreplied=1 where id=@id
```

由于更新的时候并不会更新主键，所以我们可以为两个参数都指定当前值。回复留言后可以跟踪到下面的 SQL：

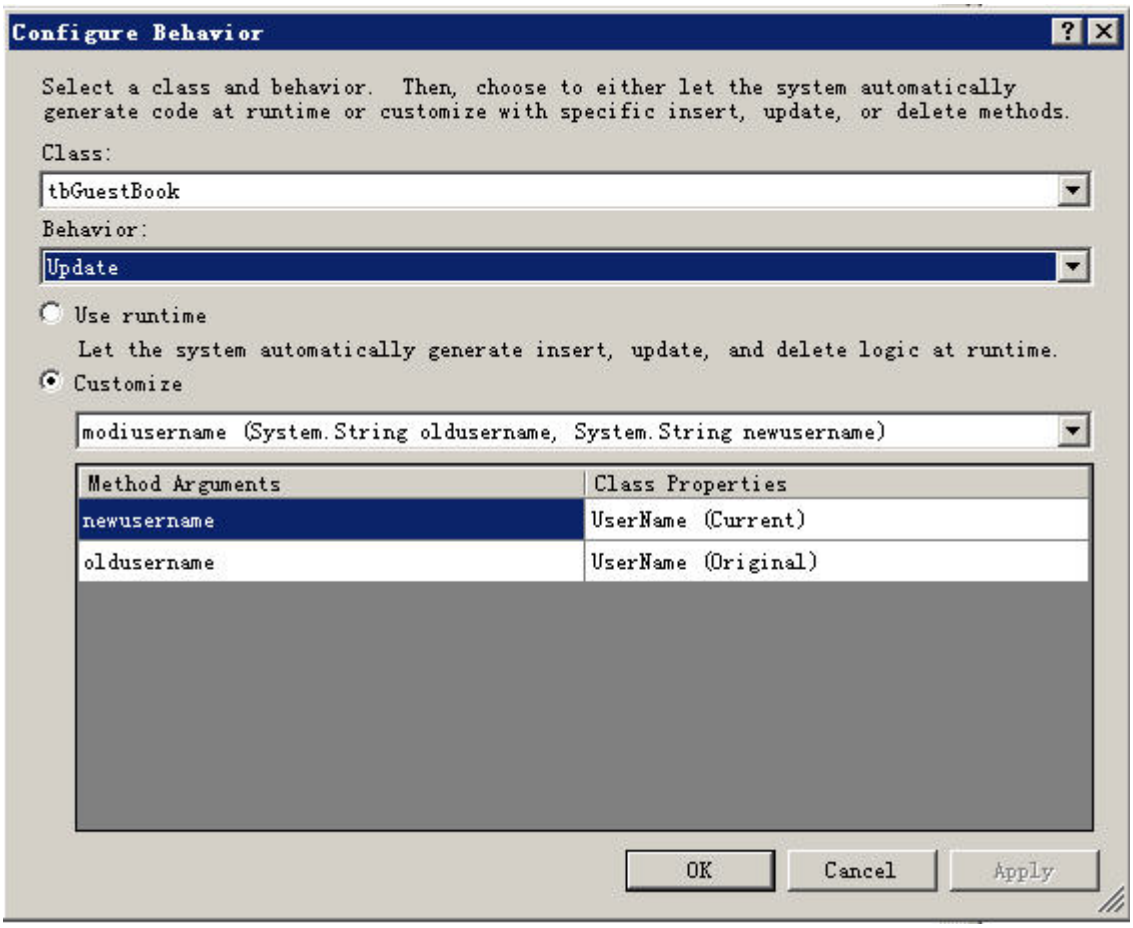
```
EXEC @RETURN_VALUE = [dbo].[replymessage] @id = @p0, @reply = @p1

-- @p0: Input Guid (Size = 0; Prec = 0; Scale = 0)

[67a69d0f-a88b-4b22-8939-fed021eb1cb5]

-- @p1: Input AnsiString (Size = 6; Prec = 0; Scale = 0) [464456]

-- @RETURN_VALUE: Output Int32 (Size = 0; Prec = 0; Scale = 0) []
```

假设有这样一种应用，我们需要修改留言簿中不合法的用户名：

```
create proc modiusername

@oldusername varchar(50),

@newusername varchar(50)

as

update tbguestbook set username=@newusername where username = @oldusername
```

有个网友起名叫“admin”，我们要把所有这个名字修改为“notadmin”。那么，可以如下图设置 update 操作：

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816



然后运行下面的测试代码：

```
var messages = from gb in ctx.tbGuestBooks
                    select gb;
foreach (var gb in messages)
{
    if(gb.UserName == "admin")
        gb.UserName = "notadmin";
}
```

运行程序后能跟踪到下面的 SQL：

```
SELECT [t0].[ID], [t0].[UserName], [t0].[PostTime], [t0].[Message], [t0].[IsReplied],
[t0].[Reply]

FROM [dbo].[tbGuestBook] AS [t0]



EXEC @RETURN_VALUE = [dbo].[modiusername] @oldusername = @p0,
@newusername = @p1

-- @p0: Input AnsiString (Size = 5; Prec = 0; Scale = 0) [admin]

-- @p1: Input AnsiString (Size = 8; Prec = 0; Scale = 0) [notadmin]

-- @RETURN_VALUE: Output Int32 (Size = 0; Prec = 0; Scale = 0) []
```

到这里，你应该能明白当前值和原始值的含义了吧。

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（六）：探究特性

延迟执行

```
IQueryable query = from c in ctx.Customers select c;
```

这样的查询句法不会导致语句立即执行，它仅仅是一个描述，对应一个 SQL。仅仅在需要使用的时候才会执行语句，比如：

```
IQueryable query = from c in ctx.Customers select c;

foreach (Customer c in query)

    Response.Write(c.CustomerID);
```

如果你执行两次 foreach 操作，将会捕获到两次 SQL 语句的执行：

```
IQueryable query = from c in ctx.Customers select c;

foreach (Customer c in query)

    Response.Write(c.CustomerID);

foreach (Customer c in query)

    Response.Write(c.ContactName);
```

对应 SQL：

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName], [t0].[ContactTitle],
[t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country], [t0].[Phone],
[t0].[Fax]

FROM [dbo].[Customers] AS [t0]
```

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName], [t0].[ContactTitle],
[t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country], [t0].[Phone],
[t0].[Fax]

FROM [dbo].[Customers] AS [t0]
```

对于这样的需求，建议你先使用 ToList()等方法把查询结果先进行保存，然后再对集合进行查询：

```
        IEnumerable<Customer> customers = (from c in ctx.Customers select
c).ToList();

        foreach (Customer c in customers)

            Response.Write(c.CustomerID);

        foreach (Customer c in customers)

            Response.Write(c.ContactName);
```

延迟执行的优点在于我们可以像拼接 SQL 那样拼接查询句法，然后再执行：

```
        var query = from c in ctx.Customers select c;

        var newquery = (from c in query select c).OrderBy(c => c.CustomerID);
```

DataLoadOptions

```
       var products = from p in ctx.Products select p;

        foreach (var p in products)

        {

            if (p.UnitPrice > 10)

                ShowDetail(p.Order_Details);
```

```
        }


    private void ShowDetail(EntitySet<Order_Detail> orderdetails)

    {}
```

由于 ShowDetail 方法并没有使用到订单详细信息，所以这个操作只会执行下面的 SQL：

```
SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID],
[t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder],
[t0].[ReorderLevel], [t0].[Discontinued]

FROM [dbo].[Products] AS [t0]
```

现在修改一下 ShowDetail 方法：

```
    private void ShowDetail(EntitySet<Order_Detail> orderdetails)

    {

        foreach (var o in orderdetails)

        {

            Response.Write(o.Quantity + "<br>");

        }

    }
```

你会发现 Linq to sql 对每个价格大于 10 的产品都根据产品号进行了一次查询：

```
SELECT [t0].[OrderID], [t0].[ProductID], [t0].[UnitPrice], [t0].[Quantity], [t0].[Discount]

FROM [dbo].[Order Details] AS [t0]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
WHERE [t0].[ProductID] = @p0

-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [1]
```

这样的语句查询了 N 次。这样的查询不是很合理，我们可以通过设置 DataContext 的 DataLoadOption，来指示 DataContext 再加载产品信息的同时把对应的产品订单信息一起加载：

```
        DataLoadOptions options = new DataLoadOptions();

        options.LoadWith<Product>(p => p.Order_Details);

        ctx.LoadOptions = options;

        var products = from p in ctx.Products select p;

      。。。。。。。。。
```

再执行先前的查询会发现 Linq to sql 进行了左连接：

```
SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID],
[t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder],
[t0].[ReorderLevel], [t0].[Discontinued], [t1].[OrderID], [t1].[ProductID] AS [ProductID2],
[t1].[UnitPrice] AS [UnitPrice2], [t1].[Quantity], [t1].[Discount], (
    SELECT COUNT(*)
    FROM [dbo].[Order Details] AS [t2]
    WHERE [t2].[ProductID] = [t0].[ProductID]
    ) AS [count]
FROM [dbo].[Products] AS [t0]
LEFT OUTER JOIN [dbo].[Order Details] AS [t1] ON [t1].[ProductID] = [t0].[ProductID]
```

```
ORDER BY [t0].[ProductID], [t1].[OrderID]
```

那么，我们怎么限制订单详细表的加载条件那？

```
DataLoadOptions options = new DataLoadOptions();

options.LoadWith<Product>(p => p.Order_Details);

options.AssociateWith<Product>(p => p.Order_Details.Where(od => od.Quantity > 80));

ctx.LoadOptions = options;

var products = from p in ctx.Products select p;
```

这样，就只会有数量大于 80 的订单详细信息会和产品一起加载：

```
SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID],
[t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder],
[t0].[ReorderLevel], [t0].[Discontinued], [t1].[OrderID], [t1].[ProductID] AS [ProductID2],
[t1].[UnitPrice] AS [UnitPrice2], [t1].[Quantity], [t1].[Discount], (
    SELECT COUNT(*)
    FROM [dbo].[Order Details] AS [t2]
    WHERE ([t2].[Quantity] > @p0) AND ([t2].[ProductID] = [t0].[ProductID])
    ) AS [count]
FROM [dbo].[Products] AS [t0]
LEFT OUTER JOIN [dbo].[Order Details] AS [t1] ON ([t1].[Quantity] > @p0) AND
([t1].[ProductID] = [t0].[ProductID])
ORDER BY [t0].[ProductID], [t1].[OrderID]
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [80]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

DataLoadOptions 限制

Linq to sql 对 DataLoadOptions 的使用是有限制的，它只支持 1 个 1 对多的关系。一个顾客可能有多个订单，一个订单可能有多个详细订单：

```
DataLoadOptions options = new DataLoadOptions();
options.LoadWith<Customer>(c => c.Orders);
options.LoadWith<Order>(o => o.Order_Details);
ctx.LoadOptions = options;
IEnumerable<Customer> customers = ctx.Customers.ToList<Customer>();
```

这样的语句执行后会导致下面的 SQL 执行 N 次（参数不同）：

```
SELECT [t0].[OrderID], [t0].[CustomerID], [t0].[EmployeeID], [t0].[OrderDate],
[t0].[RequiredDate], [t0].[ShippedDate], [t0].[ShipVia], [t0].[Freight], [t0].[ShipName],
[t0].[ShipAddress], [t0].[ShipCity], [t0].[ShipRegion], [t0].[ShipPostalCode],
[t0].[ShipCountry], [t1].[OrderID] AS [OrderID2], [t1].[ProductID], [t1].[UnitPrice],
[t1].[Quantity], [t1].[Discount], (
    SELECT COUNT(*)
    FROM [dbo].[Order Details] AS [t2]
    WHERE [t2].[OrderID] = [t0].[OrderID]
    ) AS [count]
FROM [dbo].[Orders] AS [t0]
LEFT OUTER JOIN [dbo].[Order Details] AS [t1] ON [t1].[OrderID] = [t0].[OrderID]
WHERE [t0].[CustomerID] = @x1
```

```
ORDER BY [t0].[OrderID], [t1].[ProductID]

-- @x1: Input StringFixedLength (Size = 5; Prec = 0; Scale = 0) [ALFKI]
```

而对于多对 1 的关系，Linq to sql 对于 DataLoadOptions 没有限制：

```
DataLoadOptions options = new DataLoadOptions();

options.LoadWith<Product>(c => c.Category);

options.LoadWith<Product>(c => c.Order_Details);

options.LoadWith<Order_Detail>(o => o.Order);

ctx.LoadOptions = options;

IEnumerable<Product> products = ctx.Products.ToList<Product>();
```

由于多个产品对应 1 个分类，多个详细订单对应 1 个订单，只有产品和详细订单才是多对 1 的关系，所以也只会有 1 次 SQL（不过这样的操作还是少执行为妙，消耗太大了）：

```
SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID],
[t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock], [t0].[UnitsOnOrder],
[t0].[ReorderLevel], [t0].[Discontinued], [t3].[OrderID], [t3].[ProductID] AS [ProductID2],
[t3].[UnitPrice] AS [UnitPrice2], [t3].[Quantity], [t3].[Discount], [t4].[OrderID] AS [OrderID2],
[t4].[CustomerID], [t4].[EmployeeID], [t4].[OrderDate], [t4].[RequiredDate],
[t4].[ShippedDate], [t4].[ShipVia], [t4].[Freight], [t4].[ShipName], [t4].[ShipAddress],
[t4].[ShipCity], [t4].[ShipRegion], [t4].[ShipPostalCode], [t4].[ShipCountry], (
    SELECT COUNT(*)
    FROM [dbo].[Order Details] AS [t5]
    INNER JOIN [dbo].[Orders] AS [t6] ON [t6].[OrderID] = [t5].[OrderID]
    WHERE [t5].[ProductID] = [t0].[ProductID]
```

```
    ) AS [count], [t2].[test], [t2].[CategoryID] AS [CategoryID2], [t2].[CategoryName],
[t2].[Description], [t2].[Picture]

FROM [dbo].[Products] AS [t0]

LEFT OUTER JOIN (

    SELECT 1 AS [test], [t1].[CategoryID], [t1].[CategoryName], [t1].[Description],
[t1].[Picture]

    FROM [dbo].[Categories] AS [t1]

    ) AS [t2] ON [t2].[CategoryID] = [t0].[CategoryID]

LEFT OUTER JOIN ([dbo].[Order Details] AS [t3]

    INNER JOIN [dbo].[Orders] AS [t4] ON [t4].[OrderID] = [t3].[OrderID]) ON
[t3].[ProductID] = [t0].[ProductID]

ORDER BY [t0].[ProductID], [t2].[CategoryID], [t3].[OrderID]
```

主键缓存

Linq to sql 对查询过的对象进行缓存，之后的如果只根据主键查询一条记录的话会直接从缓存中读取。比如下面的代码：

```
        Customer c1 = ctx.Customers.Single(customer => customer.CustomerID ==
"ANATR");

        c1.ContactName = "zhuye";

        Customer c2 = ctx.Customers.Single(customer => customer.CustomerID ==
"ANATR");

        Response.Write(c2.ContactName);
```

执行后只会产生一条 SQL：

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName], [t0].[ContactTitle],
[t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country], [t0].[Phone],
[t0].[Fax]

FROM [dbo].[Customers] AS [t0]

WHERE [t0].[CustomerID] = @p0

-- @p0: Input String (Size = 5; Prec = 0; Scale = 0) [ANATR]
```

由于没有提交修改，所以数据库中的记录还是没有更新。由于这个特性，我们在使用存储过程作为实体更新方法的时候就要当心了，存储过程书写错误，即使你提交了修改也很可能导致缓存中的数据和数据库中的数据不一致，引起不必要的麻烦。

DataContext 隔离

有的时候我们会把对象从外部传入 DataContext，要求它更新，由于不同的 DataContext 是相对独立的。由于新的 DataContext 中还没有获取实体，我们只能通过附加方式更新数据。

首先把 Customer 表的主键字段加上 IsVersion 标识：

```
    [Column(Storage="_CustomerID", DbType="NChar(5) NOT NULL",
CanBeNull=false, IsPrimaryKey=true, IsVersion = true)]
```

运行下面的测试代码：

```
        Customer c = new Customer { CustomerID = "ALFKI", ContactName = "zhuye",
CompanyName = "1111" };

        ctx.Customers.Attach(c, true);

        ctx.SubmitChanges();
```

会捕捉到下面的 SQL 语句：

```
UPDATE [dbo].[Customers]

SET [CompanyName] = @p2, [ContactName] = @p3, [ContactTitle] = @p4, [Address] =
@p5, [City] = @p6, [Region] = @p7, [PostalCode] = @p8, [Country] = @p9, [Phone] =
@p10, [Fax] = @p11

WHERE ([CustomerID] = @p0) AND ([CustomerID] = @p1)

-- @p0: Input StringFixedLength (Size = 5; Prec = 0; Scale = 0) [ALFKI]

-- @p1: Input String (Size = 5; Prec = 0; Scale = 0) [ALFKI]

-- @p2: Input String (Size = 4; Prec = 0; Scale = 0) [1111]

-- @p3: Input String (Size = 5; Prec = 0; Scale = 0) [zhuye]

-- @p4: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p5: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p6: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p7: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p8: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p9: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p10: Input String (Size = 0; Prec = 0; Scale = 0) []

-- @p11: Input String (Size = 0; Prec = 0; Scale = 0) []
```

今天就到这里，下次讲并发与事务问题。

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（七）：并发与事务

检测并发

首先使用下面的 SQL 语句查询数据库的产品表：

```
select * from products where categoryid=1
```

查询结果如下图：

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	1	Chai	1	1	10 boxes x 20 bags	24.00	38	100	10	0
2	2	Chang	1	1	24 - 12 oz bottles	24.00	38	100	25	0
3	24	Guaraná Fantástica	10	1	12 - 355 ml cans	24.00	38	0	0	1
4	34	Sasquatch Ale	16	1	24 - 12 oz bottles	24.00	38	0	15	0
5	35	Steeleye Stout	16	1	24 - 12 oz bottles	24.00	38	0	15	0
6	38	C?te de Blaye	18	1	12 - 75 cl bottles	24.00	38	0	15	0
7	39	Chartreuse verte	18	1	750 cc per bottle	24.00	38	0	5	0
8	43	Ipoh Coffee	20	1	16 - 500 g tins	24.00	38	10	25	0
9	67	Laughing Lumberjack Lager	16	1	24 - 12 oz bottles	24.00	38	0	10	0
10	70	Outback Lager	7	1	24 - 355 ml bottles	24.00	38	10	30	0
11	75	Rh?nbr?u Klosterbier	12	1	24 - 0.5 l bottles	24.00	38	0	25	0
12	76	Lakkalik??ri	23	1	500 ml	24.00	38	0	20	0

为了看起来清晰，我已经事先把所有分类为 1 产品的价格和库存修改为相同值了。然后执行下面的程序：

```
var query = from p in ctx.Products where p.CategoryID == 1 select p;

foreach (var p in query)

    p.UnitsInStock = Convert.ToInt16(p.UnitsInStock - 1);

ctx.SubmitChanges(); //  在这里设断点
```

我们使用调试方式启动，由于设置了断点，程序并没有进行更新操作。此时，我们在数据库中运行下面的语句：

```
update products

set unitsinstock = unitsinstock -2, unitprice= unitprice + 1

where categoryid = 1
```

然后在继续程序，会得到修改并发（乐观并发冲突）的异常，提示要修改的行不存在或者已经被改动。当客户端提交的修改对象自读取之后已经在数据库中发生改动，就产生了修改并发。解决并发的包括两步，一是查明哪些对象发生并发，二是解决并发。如果你仅仅是希望更新时不考虑并发的话可以关闭相关列的更新验证，这样在这些列上发生并发就不会出现异常：

```
[Column(Storage="_UnitsInStock", DbType="SmallInt", UpdateCheck =
UpdateCheck.Never)]

[Column(Storage="_UnitPrice", DbType="Money", UpdateCheck = UpdateCheck.Never)]
```

为这两列标注不需要进行更新检测。假设现在产品价格和库存分别是 27 和 32。那么，我们启动程序（设置端点），然后运行 UPDATE 语句，把价格+1，库存-2，然后价格和库存分别为 28 和 30 了，继续程序可以发现价格和库存分别是 28 和 31。价格+1 是之前更新的功劳，库存最终是-1 是我们程序之后更新的功劳。当在同一个字段上（库存）发生并发冲突的时候，默认是最后的那次更新获胜。

解决并发

如果你希望自己处理并发的话可以把前面对列的定义修改先改回来，看下面的例子：

```
        var query = from p in ctx.Products where p.CategoryID == 1 select p;

        foreach (var p in query)

            p.UnitsInStock = Convert.ToInt16(p.UnitsInStock - 1);

        try
```

```
        {
            ctx.SubmitChanges(ConflictMode.ContinueOnConflict);
        }
        catch (ChangeConflictException)
        {
            foreach (ObjectChangeConflict cc in ctx.ChangeConflicts)
            {
                Product p = (Product)cc.Object;
                Response.Write(p.ProductID + "<br/>");
                cc.Resolve(RefreshMode.OverwriteCurrentValues); // 放弃当前更新，所有更新以原先更新为准
            }
        }
        ctx.SubmitChanges();
```

首先可以看到，我们使用 try{}catch{}来捕捉并发冲突的异常。在 SubmitChanges 的时候，我们选择了 ConflictMode.ContinueOnConflict 选项。也就是说遇到并发了还是继续。在 catch{}中，我们从 ChangeConflicts 中获取了并发的对象，然后经过类型转化后输出了产品 ID，然后选择的解决方案是 RefreshMode.OverwriteCurrentValues。也就是说，放弃当前的更新，所有更新以原先更新为准。

我们来测试一下，假设现在产品价格和库存分别是 27 和 32。那么，我们启动程序（在 ctx.SubmitChanges(ConflictMode.ContinueOnConflict)这里设置端点），然后运行 UPDATE 语句，把价格+1，库存-2，然后价格和库存分别为 28 和 30 了，继续程序可以发现价格和库存分别是 28 和 30。之前 SQL 语句库存-2 生效了，而我们程序的更新（库存-1）被放弃

了。在页面上也显示了所有分类为 1 的产品 ID（因为我们之前的 SQL 语句是对所有分类为 1 的产品都进行修改的）。

然后，我们来修改一下解决并发的方式：

```
cc.Resolve(RefreshMode.KeepCurrentValues); // 放弃原先更新，所有更新以当前更新为准
```

来测试一下，假设现在产品价格和库存分别是 27 和 32。那么，我们启动程序（在 ctx.SubmitChanges(ConflictMode.ContinueOnConflict)这里设置端点），然后运行 UPDATE 语句，把价格+1，库存-2，然后价格和库存分别为 28 和 30 了，继续程序可以发现价格和库存分别是 27 和 31。产品价格没有变化，库存-1 了，都是我们程序的功劳，SQL 语句的更新被放弃了。

然后，我们再来修改一下解决并发的方式：

```
cc.Resolve(RefreshMode.KeepChanges); // 原先更新有效，冲突字段以当前更新为准
```

来测试一下，假设现在产品价格和库存分别是 27 和 32。那么，我们启动程序（在 ctx.SubmitChanges(ConflictMode.ContinueOnConflict)这里设置端点），然后运行 UPDATE 语句，把价格+1，库存-2，然后价格和库存分别为 28 和 30 了，继续程序可以发现价格和库存分别是 28 和 31。这就是默认方式，在保持原先更新的基础上，对于发生冲突的字段以最后更新为准。

我们甚至还可以针对不同的字段进行不同的处理策略：

```
foreach (ObjectChangeConflict cc in ctx.ChangeConflicts)
{
    Product p = (Product)cc.Object;
    foreach (MemberChangeConflict mc in cc.MemberConflicts)
    {
```

```
            string currVal = mc.CurrentValue.ToString();

            string origVal = mc.OriginalValue.ToString();

            string databaseVal = mc.DatabaseValue.ToString();

            MemberInfo mi = mc.Member;

            string memberName = mi.Name;

            Response.Write(p.ProductID + " " + mi.Name + " " + currVal + " " + origVal +" "+
databaseVal + "<br/>");

            if (memberName == "UnitsInStock")

                mc.Resolve(RefreshMode.KeepCurrentValues); // 放弃原先更新，所有更新
以当前更新为准

            else if (memberName == "UnitPrice")

                mc.Resolve(RefreshMode.OverwriteCurrentValues); // 放弃当前更新，所有
更新以原先更新为准

            else

                mc.Resolve(RefreshMode.KeepChanges); // 原先更新有效，冲突字段以当
前更新为准



        }
}
```

比如上述代码就对库存字段作放弃原先更新处理，对价格字段作放弃当前更新处理。我们来测试一下，假设现在产品价格和库存分别是 27 和 32。那么，我们启动程序（在 ctx.SubmitChanges(ConflictMode.ContinueOnConflict)这里设置端点），然后运行 UPDATE

语句，把价格+1，库存-2，然后价格和库存分别为 28 和 30 了，继续程序可以发现价格和库存分别为 28 和 31 了。说明对价格的处理确实保留了原先的更新，对库存的处理保留了当前的更新。页面上显示的结果如下图：

最后，我们把提交语句修改为：

```
ctx.SubmitChanges(ConflictMode.FailOnFirstConflict);
```

表示第一次发生冲突的时候就不再继续了，然后并且去除最后的 ctx.SubmitChanges(); 语句。来测试一下，在执行了 SQL 后再继续程序可以发现界面上只输出了数字 1，说明在第一条记录失败后，后续的并发冲突就不再处理了。

事务处理

Linq to sql 在提交更新的时候默认会创建事务，一部分修改发生错误的话其它修改也不会生效：

```
        ctx.Customers.Add(new Customer { CustomerID = "abcdf", CompanyName =
"zhuye" });

        ctx.Customers.Add(new Customer { CustomerID = "abcde", CompanyName =
"zhuye" });

        ctx.SubmitChanges();
```

假设数据库中已经存在顾客 ID 为“abcde”的记录，那么第二次插入操作失败将会导致第一次的插入操作失效。执行程序后会得到一个异常，查询数据库发现“abcdf”这个顾客也没有插入到数据库中。

如果每次更新后直接提交修改，那么我们可以使用下面的方式做事务：

```
        if (ctx.Connection != null) ctx.Connection.Open();

        DbTransaction tran = ctx.Connection.BeginTransaction();
```

```
            ctx.Transaction = tran;
            try
            {
                CreateCustomer(new Customer { CustomerID = "abcdf", CompanyName =
"zhuye" });
                CreateCustomer(new Customer { CustomerID = "abcde", CompanyName =
"zhuye" });
                tran.Commit();
            }
            catch
            {
                tran.Rollback();
            }

    private void CreateCustomer(Customer c)
    {
        ctx.Customers.Add(c);
        ctx.SubmitChanges();
    }
```

运行程序后发现增加顾客 abcdf 的操作并没有成功。或者，我们还可以通过 TransactionScope 实现事务：

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
            using (TransactionScope scope = new TransactionScope())
            {
                CreateCustomer(new Customer { CustomerID = "abcdf", CompanyName =
"zhuye" });
                CreateCustomer(new Customer { CustomerID = "abcde", CompanyName =
"zhuye" });
                scope.Complete();
            }
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（八）：继承与关系

论坛表结构

为了演示继承与关系，我们创建一个论坛数据库，在数据库中创建三个表：

1、 论坛版块分类表 dbo.Categories：

字段名	字段类型	可空	备注
CategoryID	int	not null	identity/主键
CategoryName	varchar(50)	not null	

2、 论坛版块表 dbo.Boards：

字段名	字段类型	可空	备注
BoardID	int	not null	identity/主键
BoardName	varchar(50)	not null	
BoardCategory	int	not null	对应论坛版块分类表的 CategoryID

3、 论坛主题表 dbo.Topics：

字段名	字段类型	可空	备注
TopicID	int	not null	identity/主键
TopicTitle	varchar(50)	not null	
TopicContent	varchar(max)	not null	
ParentTopic	int	null	如果帖子是主题贴这

			个字段为 null，否则就是所属主题 id
TopicType	tinyint	not null	0 – 主题贴 1 – 回复帖

实体继承的定义

Linq to sql 支持实体的单表继承，也就是基类和派生类都存储在一个表中。对于论坛来说，帖子有两种，一种是主题贴，一种是回复帖。那么，我们就先定义帖子基类：

```
[Table(Name = "Topics")]
public class Topic
{
    [Column(Name = "TopicID", DbType = "int identity", IsPrimaryKey = true,
IsDbGenerated = true, CanBeNull = false)]
    public int TopicID { get; set; }

    [Column(Name = "TopicType", DbType = "tinyint", CanBeNull = false)]
    public int TopicType { get; set; }

    [Column(Name = "TopicTitle", DbType = "varchar(50)", CanBeNull = false)]
    public string TopicTitle { get; set; }
```

```
    [Column(Name = "TopicContent", DbType = "varchar(max)", CanBeNull = false)]

    public string TopicContent { get; set; }

}
```

这些实体的定义大家应该很熟悉了。下面，我们再来定义两个实体继承帖子基类，分别是主题贴和回复贴：

```
public class NewTopic : Topic

{

    public NewTopic()

    {

        base.TopicType = 0;

    }

}


public class Reply : Topic

{

    public Reply()

    {

        base.TopicType = 1;

    }
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
    [Column(Name = "ParentTopic", DbType = "int", CanBeNull = false)]

    public int ParentTopic { get; set; }

}
```

对于主题贴，在数据库中的 TopicType 就保存为 0，而对于回复贴就保存为 1。回复贴还有一个相关字段就是回复所属主题贴的 TopicID。那么，我们怎么告知 Linq to sql 在 TopicType 为 0 的时候识别为 NewTopic，而 1 则识别为 Reply 那？只需稍微修改一下前面的 Topic 实体定义：

```
[Table(Name = "Topics")]

[InheritanceMapping(Code = 0, Type = typeof(NewTopic), IsDefault = true)]

[InheritanceMapping(Code = 1, Type = typeof(Reply))]

public class Topic

{

    [Column(Name = "TopicID", DbType = "int identity", IsPrimaryKey = true,
IsDbGenerated = true, CanBeNull = false)]

    public int TopicID { get; set; }



    [Column(Name = "TopicType", DbType = "tinyint", CanBeNull = false, IsDiscriminator
= true)]

    public int TopicType { get; set; }



    [Column(Name = "TopicTitle", DbType = "varchar(50)", CanBeNull = false)]
```

```
        public string TopicTitle { get; set; }



        [Column(Name = "TopicContent", DbType = "varchar(max)", CanBeNull = false)]

        public string TopicContent { get; set; }

}
```

为类加了 InheritanceMapping 特性定义，0 的时候类型就是 NewTopic，1 的时候就是 Reply。并且为 TopicType 字段上的特性中加了 IsDiscriminator = true，告知 Linq to sql 这个字段就是用于分类的字段。

实体继承的使用

定义好继承的实体之后，我们就可以使用了。先是自定义一个 DataContext 吧：

```
public partial class BBSContext : DataContext

{

        public Table<BoardCategory> BoardCategories;

        public Table<Board> Boards;

        public Table<Topic> Topics;

        public BBSContext(string connection) : base(connection) { }

}
```

然后，我们来测试一下 Linq to sql 是否能根据 TopicType 识别派生类：

```
            BBSContext ctx = new
BBSContext("server=xxx;database=BBS;uid=xxx;pwd=xxx");

            var query = from t in ctx.Topics select t;
```

```
        foreach (Topic topic in query)
        {
            if (topic is NewTopic)
            {
                NewTopic newtopic = topic as NewTopic;
                Response.Write("标题：" + newtopic.TopicTitle + " 类型：" +
newtopic.TopicType + "<br/>");
            }
            else if (topic is Reply)
            {
                Reply reply = topic as Reply;
                Response.Write("标题：" + reply.TopicTitle + " 类型：" + reply.TopicType
+ " 隶属主题：" + reply.ParentTopic + "<br/>");
            }
        }
```

然后我们往 Topics 表中加一些数据，如下图：

	TopicID	TopicTitle	TopicContent	ParentTopic	TopicType
	4	这个是主题	这个是主题	NULL	0
	6	这个是回复	这个是回复	4	1
	7	还是新主题	还是新主题	NULL	0
▶	8	还是新回复	还是新回复	4	1

启动程序得到如下测试结果：

```
标题：这个是主题 类型：0
标题：这个是回复 类型：1 隶属主题：4
标题：还是新主题 类型：0
标题：还是新回复 类型：1 隶属主题：4
```

当然，你也可以在查询句法中直接查询派生实体：

newtopic 和 replies 是两个 GridView 控件，执行效果如下图：

TopicID	TopicType	TopicTitle	TopicContent
4	0	这个是主题	这个是主题
7	0	还是新主题	还是新主题

ParentTopic	TopicID	TopicType	TopicTitle	TopicContent
4	6	1	这个是回复	这个是回复
4	8	1	还是新回复	还是新回复

再来看看如何进行增删操作：

```
NewTopic nt = new NewTopic() { TopicTitle = "还是新主题", TopicContent = "还是新主题" };

Reply rpl = new Reply() { TopicTitle = "还是新回复", TopicContent = "还是新回复", ParentTopic = 4 };

ctx.Topics.Add(nt);

ctx.Topics.Add(rpl);

ctx.SubmitChanges();

rpl = ctx.Topics.OfType<Reply>().Single(reply => reply.TopicID == 8);
```

```
            ctx.Topics.Remove(rpl);

            ctx.SubmitChanges();
```

实体关系的定义

比如我们的论坛分类表和论坛版块表之间就有关系，这种关系是 1 对多的关系。也就是说一个论坛分类可能有多个论坛版块，这是很常见的。定义实体关系的优势在于，我们无须显式作连接操作就能处理关系表的条件。

首先来看看分类表的定义：

```
[Table(Name = "Categories")]
public class BoardCategory
{
    [Column(Name = "CategoryID", DbType = "int identity", IsPrimaryKey = true,
IsDbGenerated = true, CanBeNull = false)]
    public int CategoryID { get; set; }

    [Column(Name = "CategoryName", DbType = "varchar(50)", CanBeNull = false)]
    public string CategoryName { get; set; }

    private EntitySet<Board> _Boards;

    [Association(OtherKey = "BoardCategory", Storage = "_Boards")]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
    public EntitySet<Board> Boards
    {
        get { return this._Boards; }
        set { this._Boards.Assign(value); }
    }

    public BoardCategory()
    {
        this._Boards = new EntitySet<Board>();
    }
}
```

CategoryID 和 CategoryName 的映射没有什么不同，只是我们还增加了一个 Boards 属性，它返回的是 Board 实体集。通过特性，我们定义了关系外键为 BoardCategory（Board 表的一个字段）。然后来看看 1 对多，多端版块表的实体：

```
[Table(Name = "Boards")]
public class Board
{
    [Column(Name = "BoardID", DbType = "int identity", IsPrimaryKey = true,
IsDbGenerated = true, CanBeNull = false)]
    public int BoardID { get; set; }
```

```
    [Column(Name = "BoardName", DbType = "varchar(50)", CanBeNull = false)]
    public string BoardName { get; set; }

    [Column(Name = "BoardCategory", DbType = "int", CanBeNull = false)]
    public int BoardCategory { get; set; }

    private EntityRef<BoardCategory> _Category;

    [Association(ThisKey = "BoardCategory", Storage = "_Category")]
    public BoardCategory Category
    {
        get { return this._Category.Entity; }
        set
        {
            this._Category.Entity = value;
            value.Boards.Add(this);
        }
    }
}
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

在这里我们需要关联分类，设置了 Category 属性使用 BoardCategory 字段和分类表关联。

实体关系的使用

好了，现在我们就可以在查询句法中直接关联表了（数据库中不一定要设置表的外键关系）：

```
Response.Write("----------查询分类为 1 的版块----------<br/>");
var query1 = from b in ctx.Boards where b.Category.CategoryID == 1 select b;
foreach (Board b in query1)
    Response.Write(b.BoardID + " " + b.BoardName + "<br/>");
Response.Write("----------查询版块大于 2 个的分类----------<br/>");
var query2 = from c in ctx.BoardCategories where c.Boards.Count > 2 select c;
foreach (BoardCategory c in query2)
    Response.Write(c.CategoryID + " " + c.CategoryName + " " +
c.Boards.Count + "<br/>");
```

在数据库中加一些测试数据，如下图：

	CategoryID	CategoryName
1	1	.NET开发
2	2	分布式开发

	BoardID	BoardName	BoardCategory
1	1	C#	1
2	2	ASP.NET	1
3	3	VB.NET	1
4	4	.NET REMOTING	2
5	5	WEB SERVICE	2

运行程序后得到下图的结果：

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
------------查询分类为1的版块-------------
1 C#
2 ASP.NET
3 VB.NET
------------查询版块大于2个的分类-------------
1 .NET开发  3
```

我想定义实体关系的方便我不需要再用语言形容了吧。执行上述的程序会导致下面 SQL 的执行：

```
SELECT [t0].[BoardID], [t0].[BoardName], [t0].[BoardCategory]

FROM [Boards] AS [t0]

INNER JOIN [Categories] AS [t1] ON [t1].[CategoryID] = [t0].[BoardCategory]

WHERE [t1].[CategoryID] = @p0

-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [1]



SELECT [t0].[CategoryID], [t0].[CategoryName]

FROM [Categories] AS [t0]

WHERE ((

    SELECT COUNT(*)

    FROM [Boards] AS [t1]

    WHERE [t1].[BoardCategory] = [t0].[CategoryID]

    )) > @p0

-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [2]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
SELECT [t0].[BoardID], [t0].[BoardName], [t0].[BoardCategory]

FROM [Boards] AS [t0]

WHERE [t0].[BoardCategory] = @p0

-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [1]
```

可以看到，第二个查询并没有做外连接，还记得 DataLoadOptions 吗？我们可以要求 Linq to sql 在读取版块分类信息的时候也把版块信息一起加载：

```
DataLoadOptions options = new DataLoadOptions();

        options.LoadWith<BoardCategory>(c => c.Boards);

        ctx.LoadOptions = options;

        Response.Write("----------查询版块大于 2 个的分类----------<br/>");

        var query2 = from c in ctx.BoardCategories where c.Boards.Count > 2 select c;

        foreach (BoardCategory c in query2)

            Response.Write(c.CategoryID + " " + c.CategoryName + " " +
c.Boards.Count + "<br/>");
```

查询经过改造后会得到下面的 SQL：

```
SELECT [t0].[CategoryID], [t0].[CategoryName], [t1].[BoardID], [t1].[BoardName],
[t1].[BoardCategory], (

    SELECT COUNT(*)

    FROM [Boards] AS [t3]

    WHERE [t3].[BoardCategory] = [t0].[CategoryID]

    ) AS [count]
```

```
FROM [Categories] AS [t0]

LEFT OUTER JOIN [Boards] AS [t1] ON [t1].[BoardCategory] = [t0].[CategoryID]

WHERE ((

    SELECT COUNT(*)

    FROM [Boards] AS [t2]

    WHERE [t2].[BoardCategory] = [t0].[CategoryID]

    )) > @p0

ORDER BY [t0].[CategoryID], [t1].[BoardID]

-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [2]
```

在添加分类的时候，如果这个分类下还有新的版块，那么提交新增分类的时候版块也会新增：

```
BoardCategory dbcat = new BoardCategory() { CategoryName = "Database" };

Board oracle = new Board() { BoardName = "Oracle", Category = dbcat};

ctx.BoardCategories.Add(dbcat);

ctx.SubmitChanges();
```

上述代码导致下面的 SQL 被执行：

```
INSERT INTO [Categories]([CategoryName]) VALUES (@p0)



SELECT [t0].[CategoryID]

FROM [Categories] AS [t0]

WHERE [t0].[CategoryID] = (SCOPE_IDENTITY())
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
-- @p0: Input AnsiString (Size = 8; Prec = 0; Scale = 0) [Database]

INSERT INTO [Boards]([BoardName], [BoardCategory]) VALUES (@p0, @p1)

SELECT [t0].[BoardID]
FROM [Boards] AS [t0]
WHERE [t0].[BoardID] = (SCOPE_IDENTITY())

-- @p0: Input AnsiString (Size = 6; Prec = 0; Scale = 0) [Oracle]
-- @p1: Input Int32 (Size = 0; Prec = 0; Scale = 0) [23]
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（九）：其它补充

外部映射文件

我们可以使用 sqlmetal 命令行工具来生成外部映射文件，使用方法如下：

1、开始菜单 —》 VS2008 —》VS 工具 —》VS2008 命令行提示

2、输入命令：

```
D:\Program Files\Microsoft Visual Studio 9.0\VC>sqlmetal /conn:server=xxx;

database=Northwind;uid=xxx;pwd=xxx /map:c:\northwind.map /code:c:\northwind.cs
```

3、这样，我们就可以在 C 盘下得到一个 xml 映射文件和 C#的实体类代码

4、把.cs 文件添加到项目中来（放到 App_Code 目录），然后使用下面的代码加载映射文件：

```
String path = @"C:\Northwind.map";

XmlMappingSource xms = XmlMappingSource.FromXml(File.ReadAllText(path));

Northwind ctx = new Northwind("server=xxx;database=Northwind;uid=xxx;pwd=xxx",

xms);
```

5、现在就可以照常进行其它工作了。使用 sqlmetal 可以很方便的同步数据库与实体和映射文件。每次修改数据库结构，从 dbml 设计器上删除表、存储过程然后再重新添加也是很麻烦的事情。

处理空值

```
        var count = (from c in ctx.Customers where c.Region == null select c).Count();

         Response.Write(count + "<br/>");
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
var query = from emp in ctx.Employees select emp.ReportsTo;

foreach (Nullable<int> r in query)

{

    Response.Write(r.HasValue ? r.Value.ToString() + "<br/>" : "没有<br/>");

}
```

代码执行后捕获到下面的 SQL 被执行：

```
SELECT COUNT(*) AS [value]

FROM [dbo].[Customers] AS [t0]

WHERE [t0].[Region] IS NULL



SELECT [t0].[ReportsTo]

FROM [dbo].[Employees] AS [t0]
```

已编译查询

对于一些在项目中经常被用到的查询可以封装成已编译查询，这样就能提高执行效率：

```
static class Queries

{

    public static Func<NorthwindDataContext, string, IQueryable<Customer>>

        CustomersByCity = CompiledQuery.Compile((NorthwindDataContext ctx, string

city) => from c in ctx.Customers where c.City == city select c);
```

```
}
```

调用查询方式如下：

```
GridView1.DataSource = Queries.CustomersByCity(ctx, "London");

GridView1.DataBind();
```

获取一些信息

```
var query = from c in ctx.Customers select c;

Response.Write("Provider 类型：" + ctx.Mapping.ProviderType + "<br/>");

Response.Write("数据库：" + ctx.Mapping.DatabaseName + "<br/>");

Response.Write("表：" + ctx.Mapping.GetTable(typeof(Customer)).TableName +
"<br/>");

Response.Write("表达式：" + query.Expression.ToString() + "<br/>");

Response.Write("sql：" + query.Provider.ToString() + "<br/>");
```

上面的代码执行结果如下：

```
Provider 类型：System.Data.Linq.SqlClient.SqlProvider
数据库：Northwind
表：dbo.Customers
表达式：Table(Customer).Select(c => c)
sql：SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region], [t0].[PostalCode], [t0].[Country],
[t0].[Phone], [t0].[Fax] FROM [dbo].[Customers] AS [t0]
```

撤销提交

```
            var customer = ctx.Customers.Single(c => c.CustomerID == "AROUT");

            customer.ContactName = "zhuye";

            customer.Country = "Shanghai";

            Response.Write(string.Format("Name:{0},Country:{1}<br/>",
customer.ContactName, customer.Country));

            customer = ctx.Customers.GetOriginalEntityState(customer);

            Response.Write(string.Format("Name:{0},Country:{1}<br/>",
customer.ContactName, customer.Country));
```

上面的代码执行效果如下：

```
Name:zhuye,Country:Shanghai
Name:Thomas Hardy,Country:UK
```

批量操作

下面的代码会导致提交 N 次 DELETE 操作：

```
            var query = from c in ctx.Customers select c;

            ctx.Customers.RemoveAll(query);

            ctx.SubmitChanges();
```

应该使用 sql 语句进行批操作：

```
            string sql = String.Format("delete from {0}",
ctx.Mapping.GetTable(typeof(Customer)).TableName);

            ctx.ExecuteCommand(sql);
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

对于批量更新操作也是同样道理。

本文将会不断补充其它点滴。最后一篇将会结合分层分布式应用给出一个实际的项目。

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

一步一步学 Linq to sql（十）：分层构架的例子

项目介绍

这节将要把《一步一步学 Linq to sql（三）：增删改》中留言簿的例子修改为使用 WCF 的多层构架。我们将会建立以下项目：

A，网站项目 WebSite：留言簿表现层

B，类库项目 Contract：定义数据访问服务的契约

C，类库项目 Service：定义数据访问服务

D，类库项目 Entity：留言簿实体

E，控制台项目 Host：承载数据访问服务

项目之间的引用如下：

A 引用 B 和 D；

B 引用 D 和 System.ServiceModel 程序集

C 引用 B、D、System.ServiceModel 以及 System.Data.Linq 程序集

D 引用 System.Data.Linq 程序集

E 引用 C 和 System.ServiceModel 程序集

生成映射文件和实体

打开 VS2008 命令行提示，执行以下命令：

```
sqlmetal /conn:server=xxx;database=GuestBook;uid=xxx;pwd=xxx
/map:c:\guestbook.map /code:c:\guestbook.cs /serialization:Unidirectional
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

注意到，这里我们使用了 serialization 开关，告知 sqlmetal 在生成实体的时候自动把它们标记为 WCF 数据对象。生成结束后把 C:\GUESTBOOK.CS 添加到 Entity 项目中。

编写数据访问服务

首先我们可以定义出留言簿数据访问服务的契约（接口），把如下的代码保存为 IDataAccess.cs 放在 Contract 类库项目中：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;


namespace Contract
{
    [ServiceContract]
    public interface IDataAccess
    {
        [OperationContract]
        void SendMessage(TbGuestBook gb);


        [OperationContract]
        List<TbGuestBook> GetData();
```

```
        [OperationContract]
        void DeleteMessage(string ID);

        [OperationContract]
        void SendReply(TbGuestBook gb);
    }
}
```

在这里定义了四个方法：

创建留言

获取所有留言

删除留言

管理员发表回复

然后，我们来实现这个契约，把如下代码保存为 DataAccess.cs 放在 Service 类库项目中：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Contract;
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
using System.Data.Linq.Mapping;

using System.IO;

using System.ServiceModel;



namespace Service

{

    [ServiceBehavior(IncludeExceptionDetailInFaults = true)]

    public class DataAccess : IDataAccess

    {

        GuestBook ctx;



        public DataAccess()

        {

            XmlMappingSource xms =

XmlMappingSource.FromXml(File.ReadAllText("c:\\guestbook.map"));

            ctx = new

GuestBook("server=srv-devdbhost;database=GuestBook;uid=sa;pwd=Abcd1234", xms);

            ctx.Log = Console.Out;

        }



        public void SendMessage(TbGuestBook gb)
```

```
        {
            ctx.TbGuestBook.Add(gb);
            ctx.SubmitChanges();
        }

        public List<TbGuestBook> GetData()
        {
            var query = from gb in ctx.TbGuestBook orderby gb.PostTime descending
select gb;
            return query.ToList();

        }

        public void DeleteMessage(string ID)
        {
            TbGuestBook gb = ctx.TbGuestBook.Single(message => message.ID ==
new Guid(ID));
            ctx.TbGuestBook.Remove(gb);
            ctx.SubmitChanges();
        }
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
        public void SendReply(TbGuestBook gb)
        {
            //ctx.ExecuteCommand("update tbGuestBook set reply={0},isreplied=1 where ID={1}", gb.Reply, gb.ID);
            TbGuestBook record = ctx.TbGuestBook.Single(message => message.ID == gb.ID);
            record.IsReplied = true;
            record.Reply = gb.Reply;
            ctx.SubmitChanges();
        }
    }
}
```

这里需要注意几点：

我们把 DataContext 的操作在控制台输出

在进行发表回复（更新操作）的时候，注释的代码和没有注释的代码虽然都能完成更新操作，但是前者更合理，因为后者会先进行 SELECT 再进行 UPDATE

WCF 服务端与客户端

打开 Host 项目中的 Program.cs，使用下面的代码来实现 WCF 的服务端：

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.ServiceModel;
using Service;
using Contract;


namespace Host
{
    class Program
    {
        static void Main(string[] args)
        {

            Uri uri = new Uri("net.tcp://localhost:8080/DataAccessService");
            using (ServiceHost sh = new ServiceHost(typeof(DataAccess), uri))
            {
                NetTcpBinding ctb = new NetTcpBinding();
                sh.AddServiceEndpoint(typeof(IDataAccess), ctb, string.Empty);
                sh.Opened += delegate { Console.WriteLine("服务已经启动"); };
                sh.Open();
                Console.ReadLine();
```

```
            }

        }

    }

}
```

在 WebSite 项目中的 App_Code 文件夹下创建一个用户调用服务的类，GetService.cs：

```
using System;

using System.Data;

using System.Configuration;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.WebControls.WebParts;

using System.Web.UI.HtmlControls;

using System.Xml.Linq;

using Contract;

using System.ServiceModel.Description;

using System.ServiceModel;
```

```
public class GetService

{

    public static IDataAccess GetDataAccessService()

    {

        ServiceEndpoint sep = new
ServiceEndpoint(ContractDescription.GetContract(typeof(IDataAccess)),

            new NetTcpBinding(),

            new EndpointAddress("net.tcp://localhost:8080/DataAccessService"));



        ChannelFactory<IDataAccess> cf = new ChannelFactory<IDataAccess>(sep);



        return cf.CreateChannel();

    }

}
```

调用服务

最后，就可以调用数据访问服务来进行留言、回复、删除留言等操作了。页面的代码不再贴了，大家可以看第三篇或者下载源代码。我们把 Default.cs 修改成如下：

```
public partial class _Default : System.Web.UI.Page

{

    protected void Page_Load(object sender, EventArgs e)
```

```
{
    if (!IsPostBack)
    {
        SetBind();
    }
}
protected void btn_SendMessage_Click(object sender, EventArgs e)
{
    TbGuestBook gb = new TbGuestBook();
    gb.ID = Guid.NewGuid();
    gb.IsReplied = false;
    gb.PostTime = DateTime.Now;
    gb.UserName = tb_UserName.Text;
    gb.Message = tb_Message.Text;
    GetService.GetDataAccessService().SendMessage(gb);
    SetBind();
}
private void SetBind()
{
    rpt_Message.DataSource = GetService.GetDataAccessService().GetData();
```

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

```
            rpt_Message.DataBind();

        }

}
```

Admin.cs 代码修改成如下：

```
public partial class Admin : System.Web.UI.Page

{

    protected void Page_Load(object sender, EventArgs e)

    {

        if (!IsPostBack)

        {

            SetBind();

        }

    }



    private void SetBind()

    {

        rpt_Message.DataSource = GetService.GetDataAccessService().GetData();

        rpt_Message.DataBind();

    }

    protected void rpt_Message_ItemCommand(object source,
```

```
RepeaterCommandEventArgs e)
    {
        if (e.CommandName == "DeleteMessage")
        {

GetService.GetDataAccessService().DeleteMessage(e.CommandArgument.ToString());
            SetBind();
        }
        if (e.CommandName == "SendReply")
        {
            TbGuestBook gb = new TbGuestBook();
            gb.ID = new Guid(e.CommandArgument.ToString());
            gb.Reply = ((TextBox)e.Item.FindControl("tb_Reply")).Text;
            GetService.GetDataAccessService().SendReply(gb);
            SetBind();
        }
    }
}
```

就这样实现了一个多层构架的留言簿程序。对于 WCF 的一些内容本文不多作解释了。

一步一步学 Linq to sql 到这里就结束了，看到这里应该已经算师父领进门了，后续的提高还要靠大家自己去琢磨。

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816

Beijing ZJS Express Stock Limited Company
Address: The 11th Floor, Zhaowei Building, Jiangtai Road, Chaoyang District of Beijing.
Postcode: 100016 Name: Liu Xiaohui Email: Xiaohui_liu0406@163.com
Tel: 13488810897 Office: 010-84561144-1816