Linq To Sql Part.2

定义数据模型类

ScottGu

译者：张子阳

jimmyzhang.cnblogs.com

jimmy_dev@163.com

出处：Linq To Sql (Part.2 - Defining Our Model Classes)

术语表

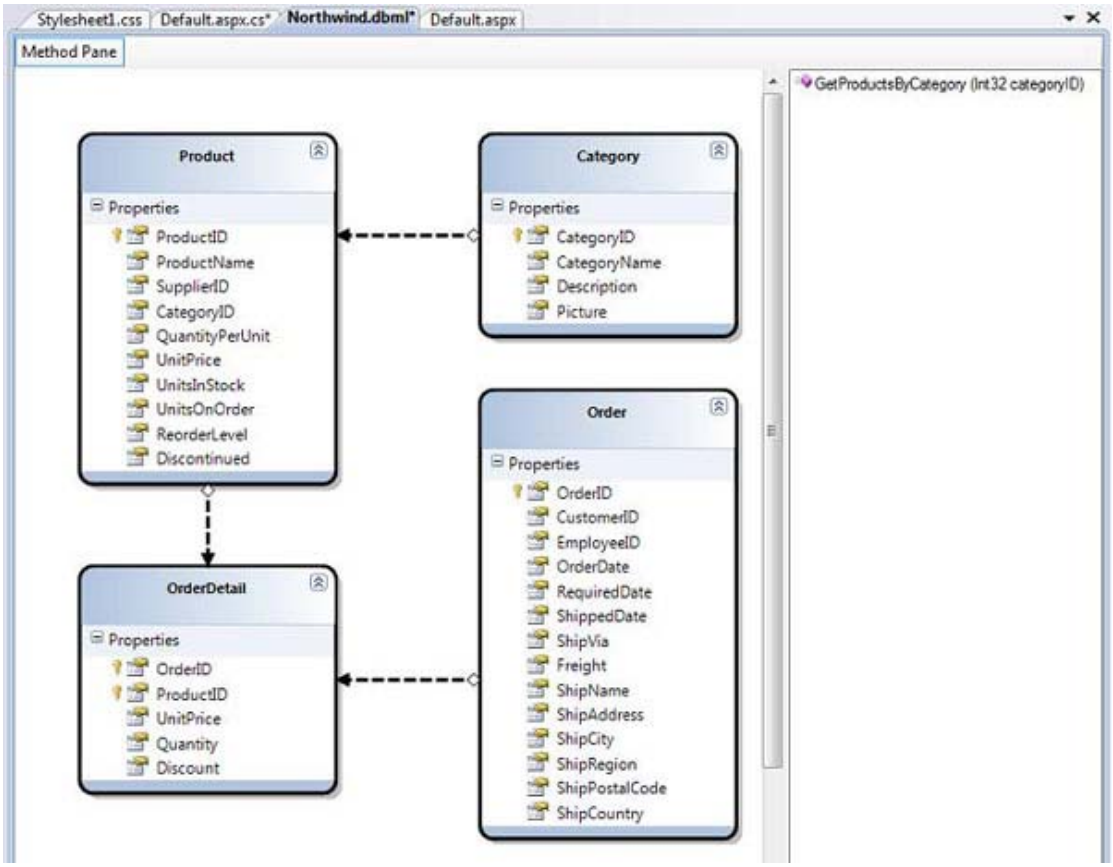
Data Model：数据模型
Designer：设计器
Class Library：类库
Client Project：客户端项目
Conduit：管道
Entity Class：实体类
Instance：实例
Partial Class：部分类
Runtime：运行时
Execution logic：执行逻辑
RelationShip Associations：关系之间的联系
Delay/Lazy：延迟/惰性 加载
Prefectched：预获取

在我博客上的随笔 LINQ TO SQL Part .1中，我讨论了“什么是LINQ TO SQL”这个问题，并提供了一些可以使用它的基本场景。

在我的第一篇随笔中，我提供了一些代码范例，示范了如何使用 LINQ TO SQL 进行一些常见的数据操作任务，其中包括下面几个方面

- 如何对数据库进行查询
- 如何更新数据库中的一行
- 如何在数据库中插入行并给这些行建立关系
- 如何删除数据库中的一行
- 如何调用存储过程
- 如何使用服务器端分页获取数据

我使用一个如下图所示的 LINQ TO SQL 类模型实现了上面的所有操作。



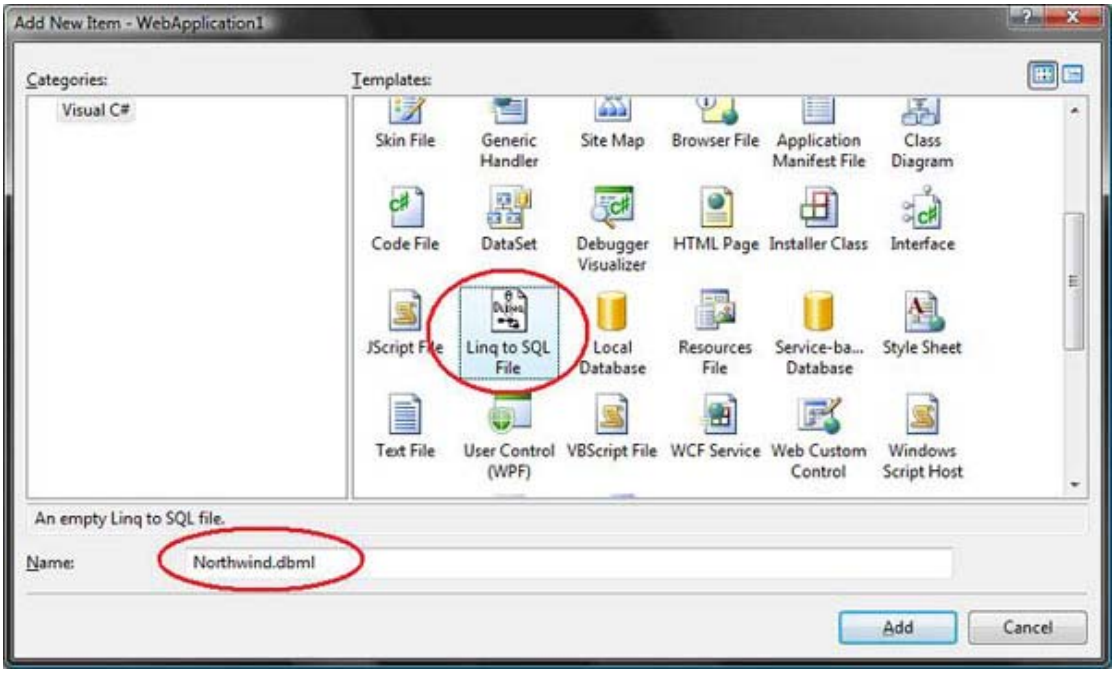
在本文中，也是这一系列的第二篇随笔中，我将更细致地讲解如何创建上图所示的 LINQ TO SQL 数据模型。

LINQ TO SQL，LINQ TO SQL 设计器，以及所有我在这篇随笔中将要讨论的特性都将作为.Net 3.5 的一部分伴随着 Visual Studio “Orcas”版本一起发布。

通过下载 Visual Studio “Orcas” Beta 1 或者 Visual Web Developer Express “Orcas” Beta1，你可以按我下面将要讲解的步骤一步步的学习。它们可以和VS2005 同时安装在同一台机器上。

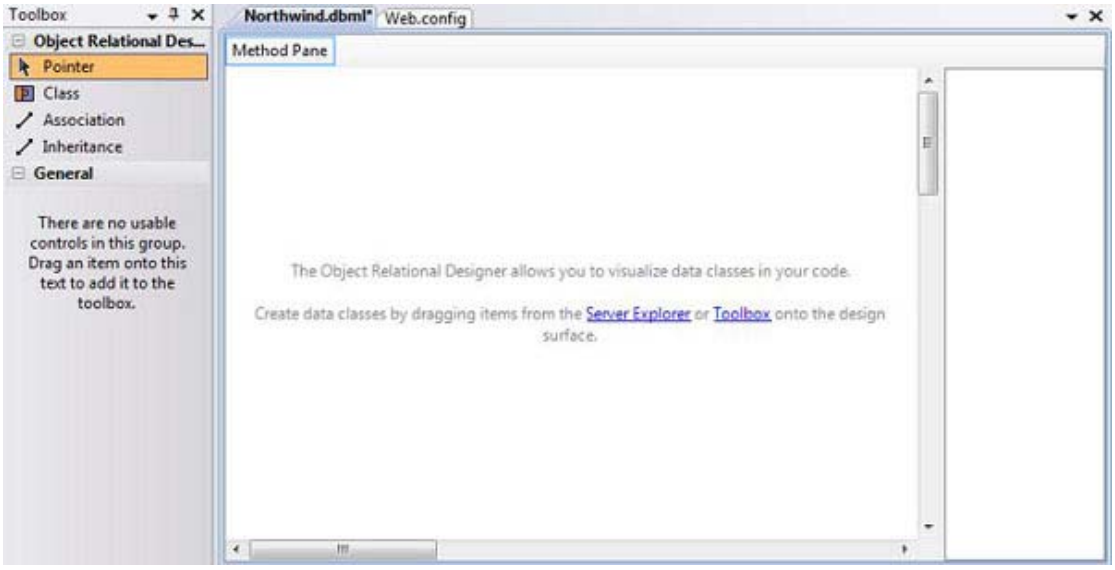
创建一个新的 LINQ TO SQL 数据模型

你可以在 Visual Studio 中点击 “添加新项”选项卡，并且选择其中的“LINQ TO SQL”来添加一个 LINQ TO SQL 数据模型到一个 ASP.NET 类库 或者 Windows 客户端项目中。



选择“LINQ TO SQL”项目将会自动运行 LINQ TO SQL 设计器，并允许你使用它去创建表现关系数据库结构的类。它同样也将创建一个强类型的“DataContext”类，这个类的属性代表着我们为之建立模型的数据库中的每张表和我们所建模型的每个存储过程。如同我在 Part .1 那篇随笔中所描述的那样，DataContext 类是一个主要的管道，我们通过它来对数据库中的实体进行查询以及将对数据的更改存回数据库。

下面是一个空的 LINQ TO SQL ORM (NOTE：对象/关系映射，可以参阅 Part.1)的截图，也是你在创建一个新的 LINQ TO SQL 模型之后将会看到的。



实体类

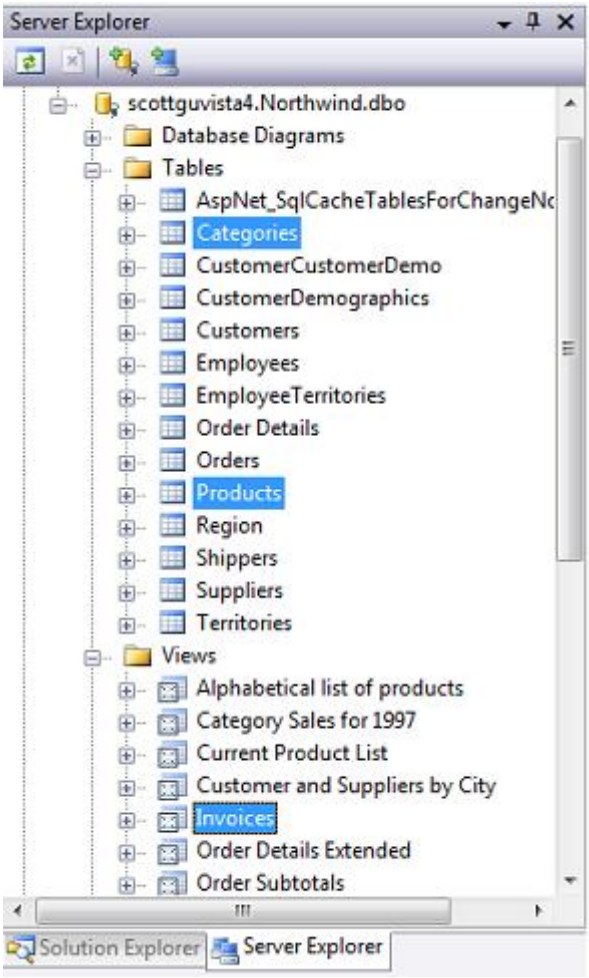
LINQ TO SQL 使你可以创建 由类到数据库，或者由数据库到类的映射。这些类通常都被称

作“实体类”，而他们的实例，被叫做“实体”。实体类映射到数据库中的表。实体类的属性通常映射到表的字段。每个实体类的实例都代表了数据库表中的一行。

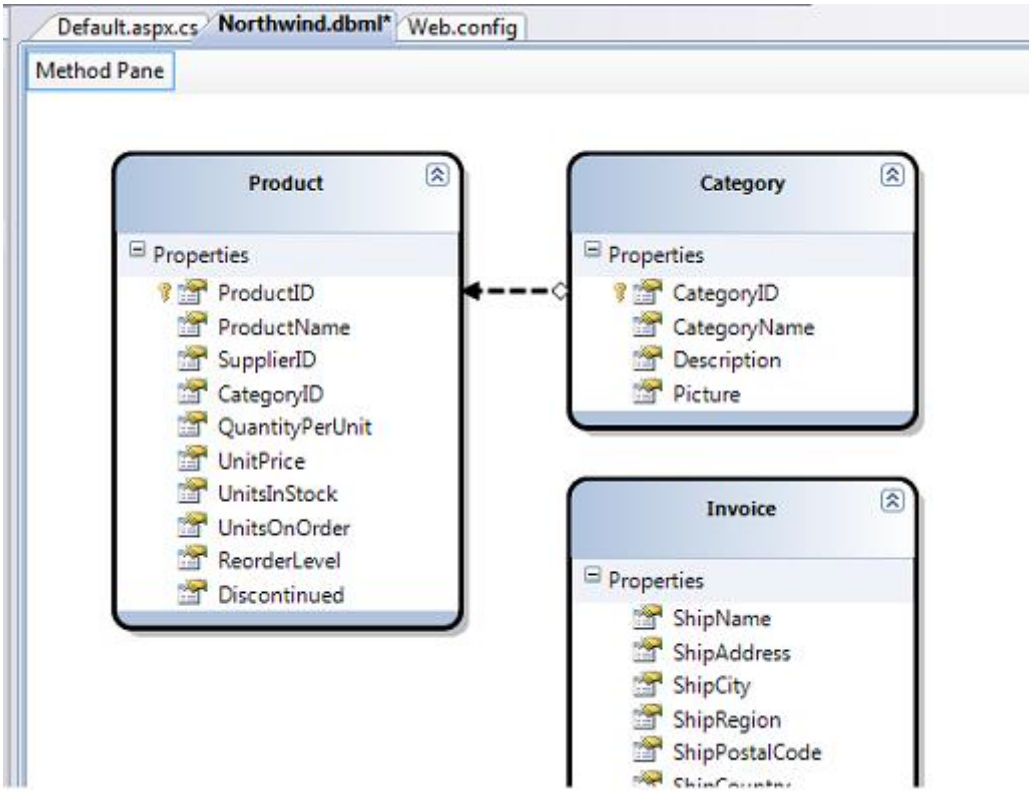
由 LINQ TO SQL 定义的实体类 **不需要** 继承某个特定的基类，这就意味着你可以让它从任何你想要的对象上继承。所有通过使用 LINQ TO SQL 设计器创建的类都被定义为“部分类”，这意味着你可以有选择的深入代码，给它们添加一些额外的属性、方法、事件。

与由 VS2005 中提供的 DataSet/TableAdapter 方式不同，当使用 LINQ TO SQL 设计器时，你在定义你的数据模型和数据访问层时不需要给出 SQL 查询语句。

相反，你只把精力集中在定义你的实体类，它们是如何 映射自数据库/映射到数据库 以及它们之间的关系。LINQ TO QL OR/M 实现将会在你与数据库实体交互或者使用这些实体时体贴地在运行时为你创建合适的 SQL 执行逻辑。你可以使用 LINQ 查询语法明确地指明你想要如何对你的数据模型以一种强类型的方式进行查询。



当你添加了上面所示的来自“Northwind”的两个表(Categories 和 Products)和视图(Invioces)到 LINQ TO SQL 设计器中。根据底层的数据库构架，你将会自动获得下面所示的三个实体类。



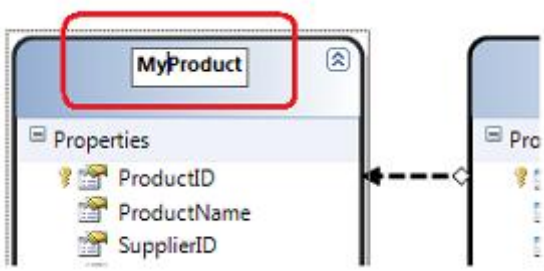
使用上面定义的数据模型类，我现在可以运行所有在 Part 1 中演示的代码范例了(除过存储过程)。我不需要写任何额外的代码或者配置，就可以实现查询、添加、删除、修改以及服务器端分页的操作。

命名和 Pluralization

当你使用 LINQ TO SQL 设计器的时候，应该注意到的一件事就是，当它在根据你的数据库构架为你自动地添加实体类的时候，它也将会自动地为你“Pluralize”不同的表和字段。举个例子，我们上面例子中的“Products”表产生了一个“Product”类，而“Categories”表产生了一个“Category”类。这种类的命名有助于使你的模型与.Net 命名规范相一致，并且我经常会发现使用编辑器做这事是非常方便的(尤其是在为你的模型中添加大量的表的时候)。

如果你不喜欢设计器生成的表或者字段的名字，当然，你可以修改它们并且将它们改成任何你想要的名字。你可以通过编辑设计器中的实体/属性名来修改，也可以通过属性栏。

NOTE：Pluralization，找不到合适的词翻译，根据上下文，可以理解成 “规范命名”。

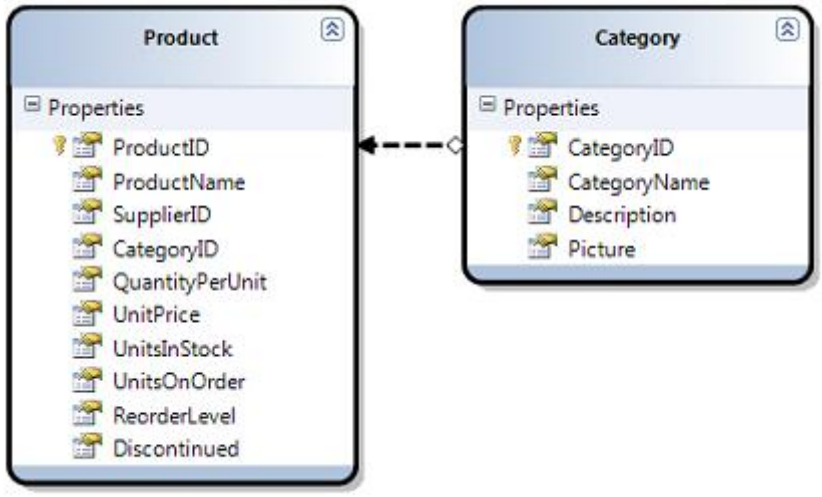


设置 实体/属性/联系 的名字与你数据库构架中的名字不一样的能力，在很多情况下非常有用，特别是：

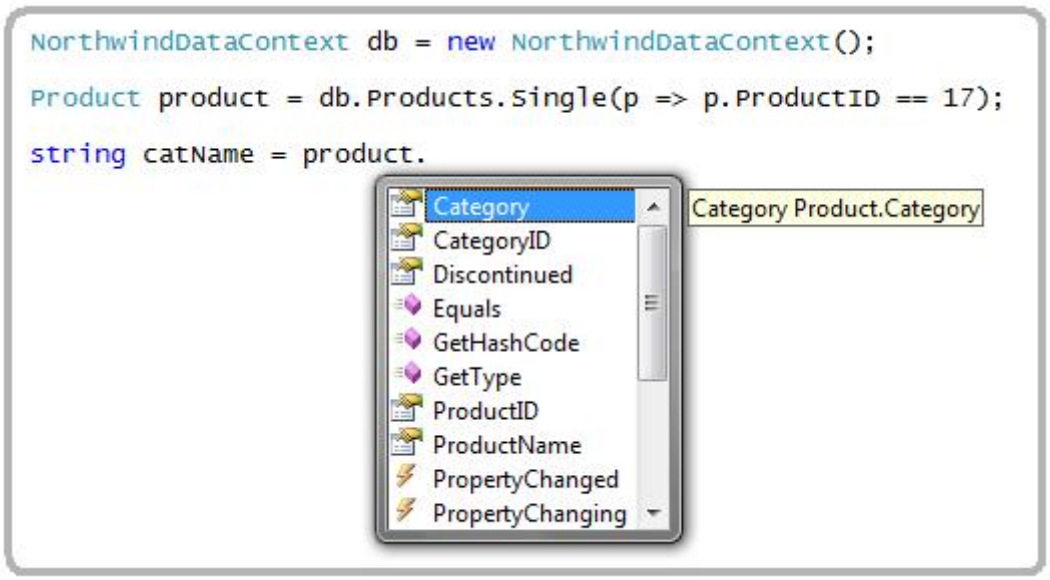
1. 当你的后台数据库 表/字段 的名称改变的时候。因为你的实体模型可以拥有和后台构架不同的名字，你可以决定只更新你的映射规则，但是却不更新你的应用程序或者查询代码，以便使用这个新的 表/字段 名字。
2. 当你有一个数据库构架，其中的对象名字不是非常的“干净”。比如说，使用诸如“au_lname”和“au_fname”这样的名字作为你的实体类的属性名，你可以在你的实体类中将它们更改为“LastName”和“FirstName”，并且根据这个名字进行开发(而不需要更改数据库中的字段名字)。

关系之间的联系

当你从服务器栏拖动一个对象到 LINQ TO SQL 设计器中时，Visual Studio 将会检查对象的主键/外键 关系，并且根据这个关系在它创建的不同实体类之间自动的创建一个默认的“关系之间的联系”。举个例子，当我从 Northwind 中添加 Products 和 Categories 表到我的 LINQ TO SQL 设计器中的时候，你可以看到它们之间的一个一对多的关系建立了(这个是由设计器中的一个箭头示意的)。



上面的联系将会使得 Product 实体类有了一个“Category”属性，通过这个属性，开发者可以访问一个给定 Product 的 Category。它也使得 Category 类有了一个“Products”集合，通过这个集合开发者可以获得一个给定 Category 的所有 Products。



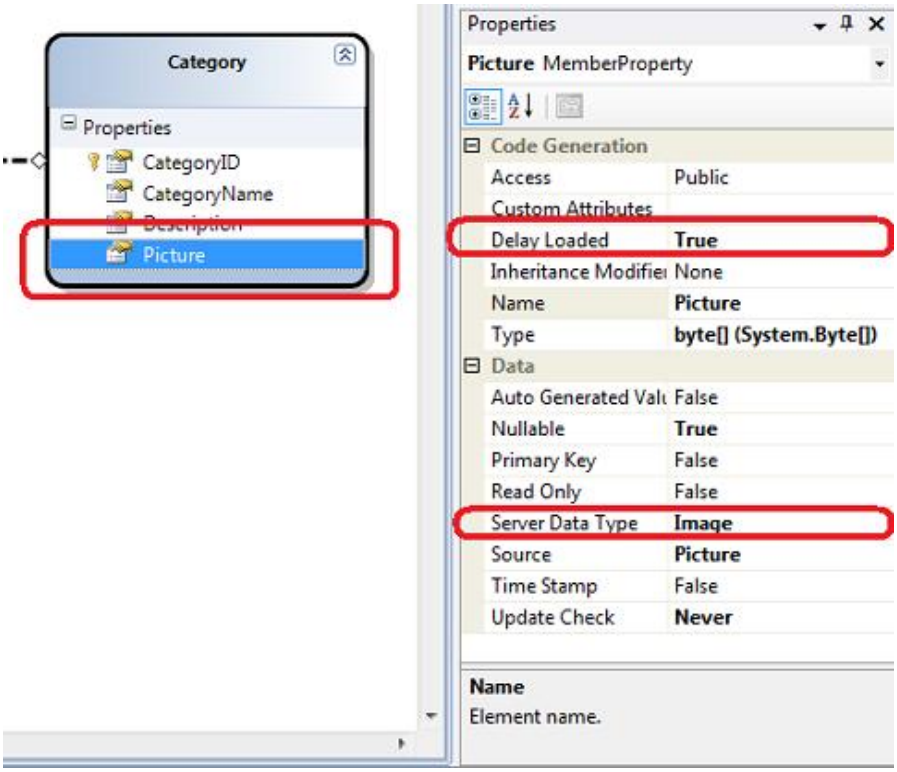
如果你不喜欢设计器建模或者命名关系的这种方式，你可以覆盖它。单击设计其中表示关系的箭头，并且通过属性栏找到它的属性，然后重命名、删除或者修改它。

延迟/惰性 加载

LINQ TO SQL 允许开发者指定实体中的属性在第一次访问的时候应该是 预获取 还是 延迟/惰性加载。你通过在设计器中选择任意实体的属性或者关系来为实体的属性定义默认的预获取/延迟加载 规则，然后将属性栏中的“Delay Loaded”属性设置成 True 或者 false。

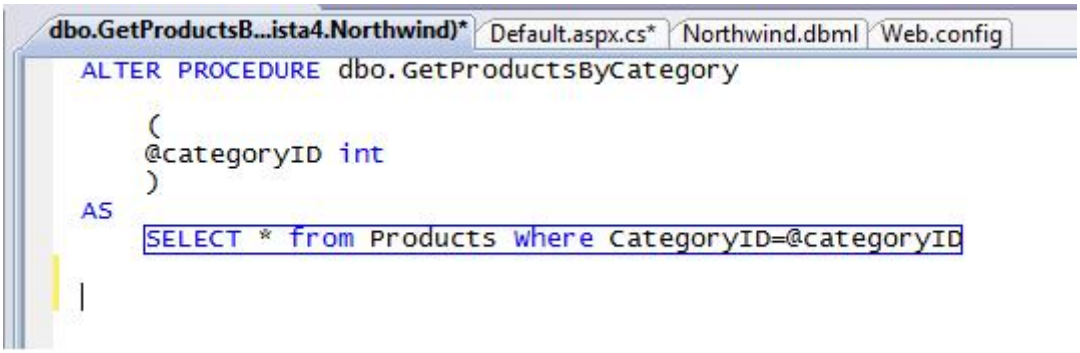
举个例子以说明什么时候我们需要这样做。考虑一下我们刚才建模的“Category”实体类。“Northwind”中有一张“Categories”表，这个表中含有一个“Picture”字段，其中存有(很可能是很大的)表示种类的二进制数据，而当我使用它的时候我只想获取数据库中的二进制数据(而不是进行一个查询将 Category 的名字列出来)。

可以通过在设计器中选择它，然后在属性栏中设置 Picture 的迟加载值来进行配置。

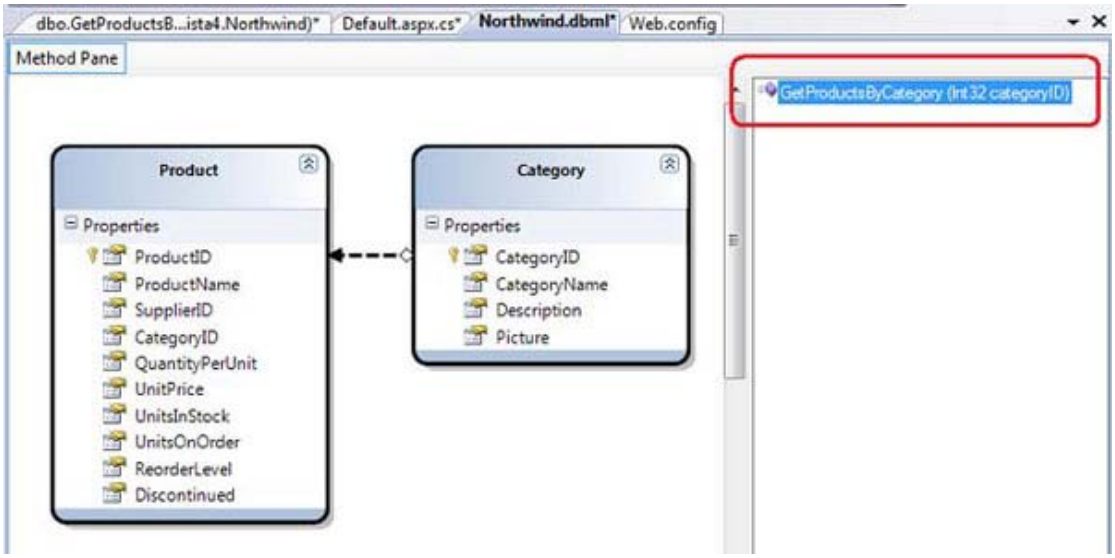


使用存储过程

LINQ TO SQL 允许你有选择的对存储过程进行建模，将它作为 DataContext 类中的方法。举个例子，假设我们定义了一个下面这样简单的存储过程，以根据 CategoryId 获取 Product 的信息：



我可以使用 Visual Studio 中的服务器面板 拖放 存储过程到 LINQ TO SQL 设计器中，以便添加一个强类型的方法，这个方法用于调用这个存储过程。如果我将存储过程拖到设计器的“Product”实体，LINQ TO SQL 设计器将声明这个存储过程返回一个 Ienumerable<Product>：



然后，我可以使用 LINQ 查询语法(这将产生一个实际的 SQL 查询语句)，或者，我可以调用上面添加的存储过程来从数据库中获取 Product。

```
NorthwindDataContext db = new NorthwindDataContext();

// Retrieve products based on an adhoc query

IEnumerable<Product> products = from p in db.Products
                                where p.CategoryID == 1
                                select p;

// Retrieve products instead using a SPROC method

products = db.GetProductsByCategory(1);

// Iterate over results

foreach (Product product in products)
{
    product.
}
```

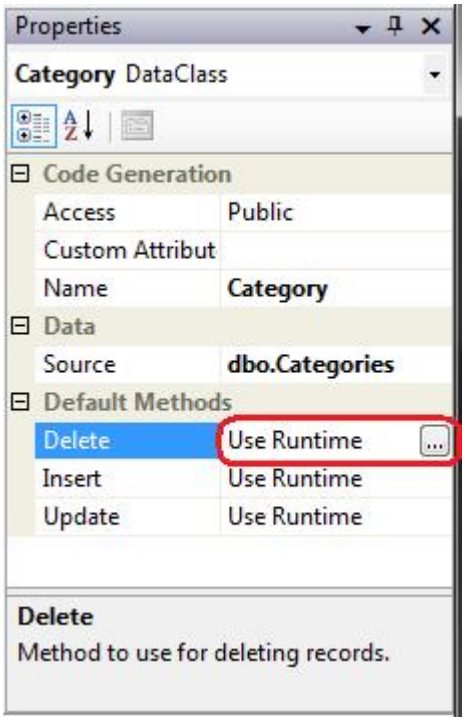
使用存储过程进行 更新/删除/插入 数据

默认情况下，在你插入/更新/删除实体的时候，LINQ TO SQL 将会自动创建合适的 SQL 表达式。举个例子，你写下如下的 LINQ TO SQL 代码以便更新 “Product”实例的一些值。

```
NorthwindDataContext db = new NorthwindDataContext();

Product product = db.Products.Single(p => p.ProductName == "Toy 1");

product.UnitPrice = 99;
product.UnitsInStock = 5;

db.SubmitChanges();
```

默认情况下，当你提交更改时，LINQ TO SQL 将会为你创建合适的“更新”语句(我会在以后的随笔中详细讨论)。

你也可以有选择的定义和使用自定义的 INSERT、UPDATE、DELETE 存储过程。对这些进行设置，点击 LINQ TO SQL 设计器中的实体类，并且在它的属性栏中在 Delete/Update/Insert 属性中点击“…”，然后选择特定的预先写好的存储过程。



改变上面设置做的好处是它纯粹只是在 LINQ TO SQL 的映射层上进行，这就意味着我之前示范的 Update 代码不需要做任何修改就可以继续使用。这也就避免了即使在后来使用自定义的存储过程，开发者也不需要改写代码。

总结

LINQ TO SQL 提供一种良好、清爽的方式对你应用程序中的数据层进行建模。一旦你定义好你的数据模型，你可以简单、高效的通过它实现查询、插入、更新、删除。

通过使用 Visual Studio 和 Visual Web Developer Express 内置的 LINQ TO SQL 设计器，你可以非常快速地创建和管理你的 LINQ TO SQL 数据模型。LINQ TO SQL 设计器同样提供很大的灵活性，允许你自定义默认的行为，并且可以覆盖/扩展这个系统以适应你的特殊需求。

在接下来的随笔中，我将使用我们刚才创建的这个数据模型，深入地探讨查询、插入、更新和删除的特性。在更新、插入和删除的随笔中，我将讨论如何添加自定义的 业务层/数据层 校验到我们设计的实体中，以提供额外的验证逻辑。