文档06-2 数据访问技术及其封装

.Net中提供了很多的数据访问技术，其中官方的主要用ADO.NET和EntityFramework两个技术。

## ADO.Net

### 基本知识

ADO.NET主要作用是用来实现数据库访问的，通过ADO.NET提供的对象能够实现数据的增改删查，ADO.NET的框架结构如下：

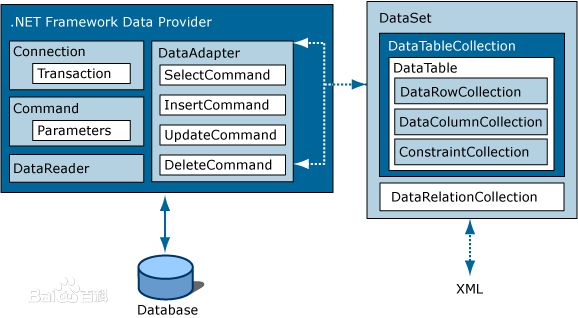


图6-1 ADO.Net框架图

在ADO.Net为了能够实现数据库的操作封装如下经常用到的类（以SqlServer的数据提供程序为例）：

1. SqlConnection

数据库连接对象，通过代码实现数据库连接的打开和关闭。

**主要方法：**

Open 打开连接

Close 关闭连接

BeginTransaction 创建连接对象关联的事务对象

**主要属性：**

ConnectionString 数据库连接串

Status 连接状态

1. SqlCommand

数据库命令执行对象，依靠某个某个数据库连接完成Sql语句或存储过程的执行。

**主要方法：**

ExecuteNonQuery 执行数据修改

ExecuteReader 执行数据查询

ExecuteScalar 执行汇总查询(单个结果)

**主要属性：**

CommandText 命令文本

CommandType 命令类型

Parameters 命令所用参数集合

CommandTimeout命令执行的过期时间

Transaction 事务控制时所用的事务对象

1. SqlParameter

命令参数描述对象，Sql指令中包含几个参数，就需要定义几个SqlParameter对象。

**主要属性：**

ParameterName 参数名

SqlDbType数据值类型

Direction 参数方向

Value 参数值

Size 参数值最大长度

1. SqlTransaction

事务控制对象，让多个SqlCommond命令的操作满足事务要求。

主要方法：

Commit 事务提交

Rollback 事务回滚

1. SqlDataReader

连线数据读取对象，通过SqlCommond对象的ExecuteReader读取查询结果后，通过SqlDataReader进行数据的读取。

主要方法：

Read() 数据指针下移

主要属性

dr[序号]/dr[列名] 获取某个列数据值

1. SqlDataAdapter

数据库数据和内存数据库之间的桥梁，通过该对象能够把数据库数据填充到内存数据库中，并且能够把内存数据库的变更更新会数据库中。

主要属性

SelectCommand 数据填充时所执行的指令

InsertCommand 与查询关联的新增指令

UpdateCommand 与查询关联的更新指令

DeleteCommand 与查询关联的删除指令

主要方法：

Fill 数据填充，Fill(DataTable dt)

Update 数据更新, Update(DataTable dt)

**离线数据保存相关的对象**

1. DataSet

内存数据库的保存对象。

1. DataTable

内存数据表的保存对象，通过SqlDataReader方式读取数据需要连接不关闭，对数据库的压力比较大，通过会把查询的表格数据直接放到DataTable中，方便数据展示与访问。

1. DataRow

内存数据表中一行数据的描述对象。

1. DataColumn

内存数据表中数据列的描述对象。

### 常见代码

1. 离线查询
2. 离线更新
3. 连线查询
4. 连线更新
5. 命令中使用参数
6. 多个命令使用事务

## Dapper

### 概念介绍

ADO.NET数据访问组件，提供了数据访问内容，但是每次使用的是否都需要自己打开连接和关闭连接，编码编写效率不高，Dapper是一个第三方的扩展组件，是对ADO.NET方式的扩展，性能和ADO.NET媲美，在项目中安装Dapper之后，能够提供更加方便的编程方法实现对数据库的相关操作，并且提供了一种更加高效的参数化使用方法。

安装并且在类文件中引入Dapper中，SqlConnection对象中就会多出很多的方法，无需再创建SqlCommond对象就行进行相关语句的执行。

Execute

ExecuteReader

ExecuteScalar

ExecuteScalar<T>

Query<T> ，Query

QueryFirst<T> ，QueryFirst

QueryFirstOrDefault<T> ，QueryFirstOrDefault

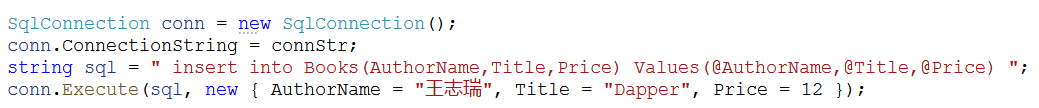
QuerySingle<T> ，QuerySingle

QuerySingleOrDefault<T> ，QuerySingleOrDefault

执行过程中，如果Sql语句中包含参数，可以通过模型类为参数传值，也可以通过匿名类为参数传值，无需创建SqlParameter

### 代码实例

1. 新增的例子



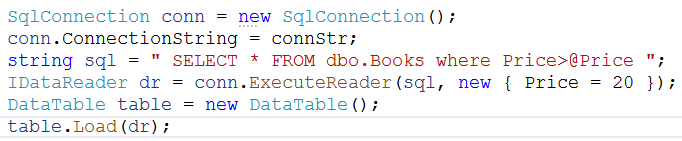
如果数据值保存在C#变量中，则把值换成对应的变量，参数部分格式如下：

New {参数名=C#变量名,参数名=C#变量名}

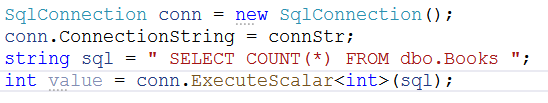
如果C#的变量名和参数名相同，则参数部分可以简写为：

new { C#变量名, C#变量名 }

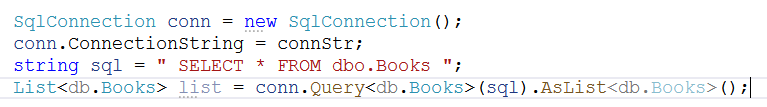
1. 查询的例子



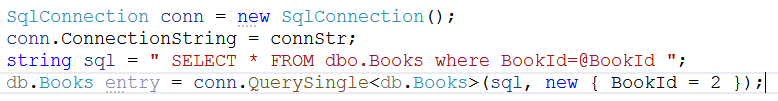
1. 汇总查询



1. 返回对象集合



1. 返回单个对象



## EntityFramework

### 概念介绍

EntityFramework是微软提供的O/R Mapping组件，通过该组件能够实现以面向对象的编程方式实现对数据库的操作，要使用该组件，必须先创建数据库模型。

### EF数据模型的创建方法

要利用EntityFramework创建数据库的实体类，Visual Studio中提供了两种方法，**第一种方式**是采用EDMX文件构建，这是一种图形化的方式，也是Visual Studio内置的一种方式，使用起来特别的方便。创建的基本方法如下：

1）利用【解决方案资源管理器】窗口，打开【添加新项】窗口，在窗口内选择“ADO.NET实体数据模型”，单击“添加”。

2）在【实体模型向导】中选择“从数据库生成”，单击“下一步”，打开【设置数据库连接】窗口。

3）通过此窗口建立到数据库的连接，并将数据库连接信息保存到App.Config文件中，单击“下一步，出现【选择数据口对象和设置】窗口，展开窗口内“表”，“试图”，“存储过程”的下拉列表，选中需要转换为实体层的对象，单击“完成”即可，选中的对象将会被转化为实体类。

在解决方案中展开Model1.edmx将会发现此模型所生成的相关类信息，如图5.2所示。

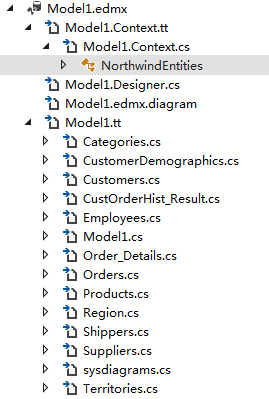


图6.2 EDMX文件结构

在Model1.tt下方包含了所选中的数据库对象，其中每一张表或视图会生成为一个实体类，表的属性会生成为类的属性（字段属性），表之间的关系，会生成为类之间的关系（通过导航属性实现）。存储过程会被生成为一个方法，如果存储过程有返回值，则返回值也会被生成为一个实体类（类名会有\_Result后缀）。

在Model1.Context.cs中有一个NorthwindEntities类，他是一个从DbContext类派生出来的数据库上下文类，要对数据库进行操作，需要首先实例化此类，并对此类中的DbSet类型的属性进行访问，此类内部结构如下5.3所示。

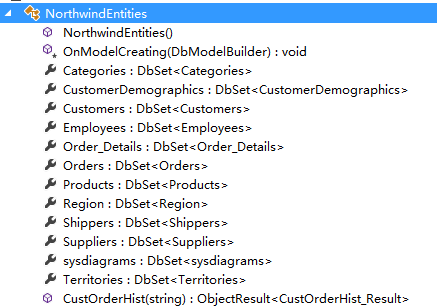


图6.3 上下文类的内部结构

此种方式所生成的文件只有一个，所有的代码都隐含在此文件中，对于追求代码美观的人来说总会感觉有点不适应。

**当数据库发生变更后，需要重新生成EF模型，步骤如下：**

1. 打开模型文件：
2. Ctrl+A 选中
3. Del 把以前的删除
4. 重新添加要生成的表或试图。
5. 保存模型文件（一定要保存）。

**切记不要把实体模型文件删除重新创建文件。**

### LINQ to Entities查询语法

#### 投影查询Select

**用法1：查询所有字段，建议少用，会将有关的导航属性也加载进去，效率差**

using (db.NorthwindEntities dc = new db.NorthwindEntities())

{

var query = from p in dc.Products

select p;

//下边实例只写查询部分的代码

this.dataGridView1.DataSource = query.ToList();

}

等价于：

var query = dc. Products.Select(p=>p);

**用法2：查询单个字段**

var query = from p in dc.Products

select p.ProductName;

//dataGridView显示有问题，可以通过调试查看结果

等价于：

var query = dc.Products.Select(p=>p.ProductName);

**提示：**DatagridView单列模式下显示有问题，会自动去取元素的第一个属性进行显示，可以更换别的输出方式。

**用法3：查询多个字段**

var query = from p in dc.Products

select new { p.ProductName, p.ProductID };

等价于：

var query = dc.Products.Select(p => new { p.ProductName, p.ProductID });

**用法4：字段别名和字段运算**

var query = from p in dc.Employees

select new

{

Name = p.FirstName + " " + p.LastName,

Phone = p.HomePhone

};

等价于：

var query = dc.Employees.Select(p => new{ Name = p.FirstName + " " + p.LastName,

Phone = p.HomePhone });

----------------------------------------------------------------------------------------------------------------------------------

var query =from p in db.Products

select new

{

p.ProductID,

HalfPrice = p.UnitPrice / 2

};

等价于：

var query = dc.Products.Select(p => new { p.ProductID,

HalfPrice = p.UnitPrice / 2 });

----------------------------------------------------------------------------------------------------------------------------------

var query = from c in dc.Customers

select new{

c.CustomerID,

CompanyInfo = new { c.CompanyName, c.City, c.Country },

ContactInfo = new { c.ContactName, c.ContactTitle }

};

等价于：

var query = dc.Customers.Select(c => new{

c.CustomerID,

CompanyInfo = new { c.CompanyName, c.City, c.Country },

ContactInfo = new { c.ContactName, c.ContactTitle }

});

**用法5：获取关联表信息，产品种类的名称**

var query = from p in dc.Products

select new { p.ProductID, p.ProductName, **p.Categories.CategoryName** };

#### 过滤查询Where

**用法1：单条件**

var query = from p in dc.Employees

where p.City = = "London"

select new{

Name = p.FirstName + " " + p.LastName,

Phone = p.HomePhone

};

等价于：

var query = dc.Employees.Where(p => p.City == "London").Select(p => new{

Name = p.FirstName + " " + p.LastName,

Phone = p.HomePhone

});

**用法2：多条件**

var query = from p in dc.Products

where p.UnitsInStock <= p.ReorderLevel && !p.Discontinued

select new { p.ProductID, p.ProductName, p.Categories.CategoryName };

等价于：

var query = dc.Employees.Where(p => p.UnitsInStock <= p.ReorderLevel &&

!p.Discontinued).Select(p =>

new { p.ProductID, p.ProductName, p.Categories.CategoryName });

#### 新增/修改/删除操作

Insert新增操作

using (db.NorthwindEntities dc = new db.NorthwindEntities())

{

//实例化一个实体对象

db.Customers entry = new Customers();

//给实体对象赋值(非空的字段必须赋值)

entry.CustomerID = "BBBBB";

entry.CompanyName = "Test";

//将实体对象插入到Customers集合中

dc.Customers.Add(entry);

//保存更改

dc.SaveChanges();

}

Update更新操作

using (db.NorthwindEntities dc = new db.NorthwindEntities())

{

//利用FirstOrDefault方法获取需要修改的记录

db.Customers entry = dc.Customers.FirstOrDefault (a => a.CustomerID == "BBBBB");

//如果查询有结果，则修改对应字段的值，并保存

if (entry != null)

{

entry.CompanyName = "Update";

entry.ContactName = "123";

dc.SaveChanges();

}

}

删除操作

using (db.NorthwindEntities dc = new db.NorthwindEntities())

{

db.Customers entry = dc.Customers.FirstOrDefault(a => a.CustomerID == "BBBBB");

//如果查询有结果，则从Customers集合中删除，并保存

if (entry != null)

{

dc.Customers.Remove(entry);

dc.SaveChanges();

}

}

#### 查找记录的相关方法

查找记录除了Select方法之外，还有很多其它的方法，可以用来返回唯一的查询结果，这些方法主要有First，FirstOrDefault，Single，SingleOrDefault，Find等。

dc.Customers.First（a=>a.CustomerID == "BBBBB"）根据条件查询，返回查询结果中的第一个元素，如果无查询结果，则抛出异常。

dc.Customers.FirstOrDefault（a=>a.CustomerID == "BBBBB"）根据条件查询，返回查询结果中的第一个元素，如果无查询结果，则返回Null。

dc.Customers.Single（a=>a.CustomerID == "BBBBB"），根据条件查询，如果查询结果非唯一元素，则抛出异常。

dc.Customers.SingleOrDefault（a=>a.CustomerID == "BBBBB"），根据条件查询，如果无查询结果，则返回Null，如果查询结果非唯一元素，则抛出异常。

## 数据访问类的封装

在项目开发过程中，经常要进行数据库访问和修改操作，为了提高代码编写效率，需要对数据库访问的相关方法进行封装，封装过程中需要满足如下几个目标：

1. 方法使用方便，只需要传入必须的参数，就可以完成相关操作。
2. 封装的方法要支持事务，并且要支持**跨方法的事务**，在项目开发过程中，一些复杂的业务处理不可能都放在一个方法内完成，通常会分散在多个方法中，方法之间会相互调用，当方法被UI直接调用的时候，方法内能够满足一个事务要求，当方法被其它方法调用的时候，该方法的代码能够和被调用的方法代码融为一个事务。
3. 在开发过程中，有的时候用Sql方式，有的时候用EF方式，Sql方式适合用的是Dapper方式实现，Linq to Entities方式适合用EF方式，因此封装的包要支持两种方式的混合编写，并且混合编写的时候，也能够支持同一个事务。

### 封装的代码位置

1. 上下文类的扩展代码，对EF上下文对象的扩展，让EF的上下文对象dbEntities中包含连接对象和事务对象。

dbLibrary\lib\db\ruiDbContext.cs

1. 封装的主要文件

dbLibrary\lib\db\efHelper.cs

1. 控制器基类

website\Controller\baseController.cs

业务方法都提供了dc的参数，为了Action中调用业务类是传递参数的方便，创建控制器基类，在基类中内置一个dc对象，所有的控制器都继承该基类，这样保证了控制器中默认就有一个dc对象。

### 封装的实现思路

1、为了能够实现Dapper方式和EF方式共用相同的事务，首要条件就是要共享相同的连接对象和事务对象（封装中采用EF上下文对象实现）。

2、为了能够实现跨方法的事务，因此不同的业务方法之间必须传递相同的连接和事务对象（我们这里只需传递EF上下文对象即可），因此我们编写的业务方法的参数上需要有上下文对象参数，我的代码习惯是把该参数放在方法参数的最后一个，如果方法有默认值参数，则默认值参数放在上下文对象参数后边。

无默认值参数的方法：

getRolePrivResource(string roleCode, string orgCode, **db.dbEntities dc**)

有默认值参数的方法：

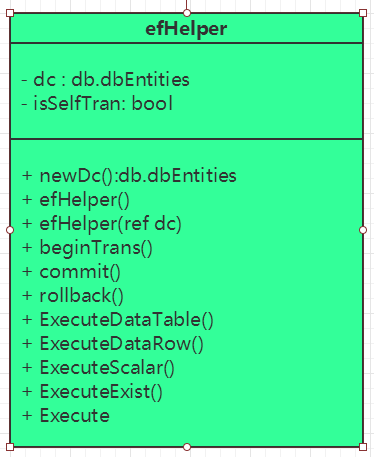
getUserPrivResource(string userCode, string orgCode, db.dbEntities dc, bool hasNoPrivResource = false, bool isUserOnly = false)

如果方法是View中调用的方法，则不需要传递该参数，该方法不会参与业务处理中。

bindDdl(bool has请选择 = false)

3、因此封装的基本思路是，构建一个辅助类，在辅助类内部包含数据库上下文对象，在方法中需要进行数据库操作时，需要创建一个辅助类对象，创建辅助类对象的时候，如果未传入上下文对象，则辅助类内部会自己创建一个上下文对象，如果传入了上下文对象，则辅助类内会共用传入的上下文对象，当调用辅助类内部的相关数据操作方法时，都是借助数据库上下文对象内的连接对象和事务对象实现的操作，因此能够实现跨方法的相关功能。

### 辅助类的代码主要结构



1. dc：数据库上下文对象

辅助类实例内部的数据库上下文对象，该对象可以创建辅助类对象的时候创建（通过无参的构造函数），也可以通过带参的构造函数（efHelper(ref dc)）传入，实现多个辅助类实例对象内所用dc是同一个dc。跨方法事务的时候必须通过参数传递方式让多个方法内的辅助类实例使用相同的dc对象才能实现。

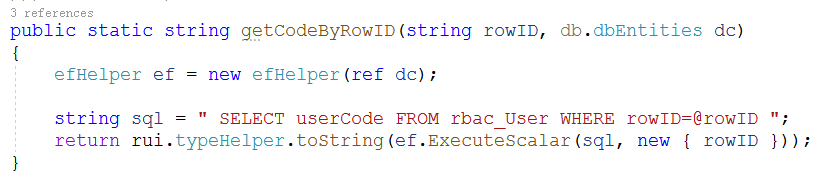
1. isSelfTran：标识辅助类实例内所用事务对象是否辅助类实例自己创建的。
2. newDc()：创建一个新的数据库上下文对象。
3. efHelper()：无参构造函数，创建封装类实例对象时直接创建上下文对象。
4. efHelper(ref dc)：带参数的构造函数，创建辅助类实例对象时传入一个上下文对象，如果传入的是Null对象，则封装类实例对象自己创建上下文对象。
5. beginTrans()：开启事务方法，该方法会根据实例对象内的上下文对象的事务对象是否为Null，来判断是否需要自己创建事务对象，如果不需要则设置isSelfTran为False，如果需要，则通过连接创建事务对象，并设置封装类对象的isSelfTran为True。
6. commit()：事务提交方法，通过辅助类实例对象内部的isSelfTran判断是否实例自己创建的事务，如果是则进行事务的提交，并重置事务对象状态，否则不进行处理。
7. rollback()：事务回滚方法，通过辅助类实例对象内部的isSelfTran判断是否实例自己创建的事务，如果是则进行事务的回滚，并重置事务对象状态，否则不进行处理。
8. ExecuteDataTable：数据查询方法，返回的查询结果是DataTable。
9. ExecuteDataRow：数据查询方法，返回的查询结果是DataRow。
10. ExecuteScalar()：数据查询方法，返回单个查询结果。
11. ExecuteExist()：数据查询方法，判断查询语句是否有结果。
12. Execute()：数据新增、修改，删除执行方法，返回影响的行数。

这些执行方法：第一个参数是Sql语句，第二个参数是Dapper方式的参数。

### 封装包的使用方法

1. 单个方法的普通用法（无事务）

查询数据，并转为字符串返回



1. 单个方法含事务的

数据删除：首先判断是否是否满足删除条件，如果满足则把相关表的数据通过事务一起删除。



1. 跨方法的事务

方法A – 方法B

**情况1：方法A内部启用了事务，方法B未启用事务。**

方法B被非事务方式调用时，方法B代码无事务。

当方法B被事务方法调用时，方法B代码融入上层方法事务。

**情况2：方法A内部启用了事务，方法B内部启用事务。**

方法B被非事务方法调用时，方法B代码内部单独的事务。

方法B被事务方法调用时，方法B代码融入上层方法事务。

### 封装包内的其他常用方法

1. 生成Guid

ef.newGuid()

1. 实体更新-MVC的UI对接用

efHelper.entryUpdate(object entry, db.dbEntities dc, string exceptField = null, string containField = null)

1. 判断是否能够删除

ef.checkCanDelete(string tableName, string fieldName, string fieldValue, string errorMsg, string whereExpr = "")

//检查某表中某个字段是否有等于某个值的，如果有，则显示错误提醒，不允许删除。

ef.checkCanDeleteIn(string tableName, string fieldName, string fieldValue, string errorMsg, string whereExpr = "")

//检查某表中某个字段值中是否包含某个值，如果有，则显示错误提醒，不允许删除。

1. 检查数据是否非空

efHelper.checkNotNull(object value, string fieldName)

1. 检查数据是否等于某值

efHelper.checkEqualValue(object cValue, string value, string fieldName)

1. 检查数据是否不等于某值

efHelper.checkNoEqualValue(object cValue, string value, string fieldName)