基于 ADO. NET 的通用数据访问层设计方法研究①

Research on Common Data Access Layer Design Base on ADO. NET

邱云飞 （辽宁工程技术大学软件学院 辽宁葫芦岛 125100）
邵良杉 （辽宁工程技术大学系统工程研究所 辽宁阜新 123000）

摘要：在. NET 平台上进行应用软件开发时，为了实现与后台具体数据源的相对独立，首先将每个表与一个 DataSet 绑定，其次针对此表的访问需求给出访问接口，接着针对不同的数据源在该接口的基础上实现访问类，最后利用 Factory 模式根据配置文件信息动态实现对不同数据源的访问，从而实现了独立于数据源开发软件的目的。

关键词：ADO. NET Factory 模式 通用数据访问

1 引言

在多层应用程序设计时，通常需要将对数据库的访问集中起来，以保证良好的封装性和可维护性[1]。因此，在利用. NET 作为开发工具时，为了屏蔽多种数据库访问类库的差别为业务逻辑层提供统一的接口，我们有必要对基于 ADO. NET 的数据访问层的设计方法进行深入研究。相关文献资料已经对此问题进行了一定的研究，如孙亚民使用 Factory 设计模式实现了访问类库的动态调用[1]，Silvano Coriani 利用. NET 数据提供程序必须实现的基本接口（IDbConnection、IDbCommand、IDataReader 等）编写透明使用不同数据源的智能应用程序[2]，但这两种设计方法是以损失不同类库操作其对应的 DBMS 的效率为代价的；微软 MSDN 中的 Duwamish 案例对 DataSet 进行子类化扩展作为数据载体，PetShop 案例的数据层则使用了 DataReader[3]，两者比较 Duwamish 整个架构定义非常清晰和严谨，但只实现了 SQL Server 一种数据库的访问，PetShop 程序简洁、轻灵，但与 Duwamish 相比健壮性不足，但两者在性能上与孙亚民、Silvano Coriani 的设计方法相比都具有很大的优势。经过对以上几种数据访问层设计方法的分析比较，本文利用 Factory 设计模式对 Duwamish 案例进行扩展，以使其在保持清晰架构、健壮性和高性能的基础上，实现对多种数据库的访问。

2 相关原理分析

2.1 Factory 设计模式原理

Factory 设计模式要求首先提取不同实现方法的公共部分并定义一个用于创建对象的接口，再在这个接口的基础上实现不同方法类，在运行时让子类决定实例化哪一个类。Factory 设计模式使一个类的实例化延迟到其子类。本文中通用数据访问方法要求对多种数据库进行处理操作，因此，需要首先定义一个操纵数据库的接口，然后，根据数据库的不同，由类工厂决定实例化哪个类。

2.2 ADO. NET 原理分析

ADO. NET 主要包括 Connection，DataAdapter，DataReader，Command 和 DataSet 五个对象，其中 Connection，DataAdapter，DataReader，Command 是与具体数据源相关的，而 DataSet 是与数据源无关的。这就是说我们开发这种柔性的与数据源无关的连接数据源的模块中应该重点关注 Connection，DataAdapter，DataReader，Command 四个数据源相关的对象，对他们进行封装，以统一的界面提供给开发人员，也就达到了我们将连接数据源分离出来的目的了。而前端使用具体语言开发软件时使用的数据只要取自 DataSet 就可以了，Duwamish 案例正是基于这个原理进行的设计。

① 基金项目：教育部博士点基金（20041047006）、国家自然科学基金（70572070）和辽宁工程技术大学校基金（05－122）.

3　通用数据访问层的设计原理分析

我们知道，在使用 ADO. NET 进行应用软件开发时，不管数据存储于何种数据源中，都可将从数据源获得的数据保存在 DataSet 中加以进一步处理，因此可将应用中使用的每一个表与一个 DataSet 绑定，当业务逻辑层对此表中数据有访问需求时将绑定了对应表的 DataSet 的派生对象调入内存供其使用。为了提高数据访问层的可移植性，实现对多种数据源的动态访问，应该采用 Factory 设计模式进行设计，首先提取对每个表的访问需求给出接口，然后针对不同的数据源类型继承该接口给出实现类，在业务逻辑层有访问需求时使用 Factory 工厂根据配置文件给出的信息动态调用实现类对象完成对不同数据源的访问。对于数据源中每个数据表中数据获取的方法如图 1 所示。

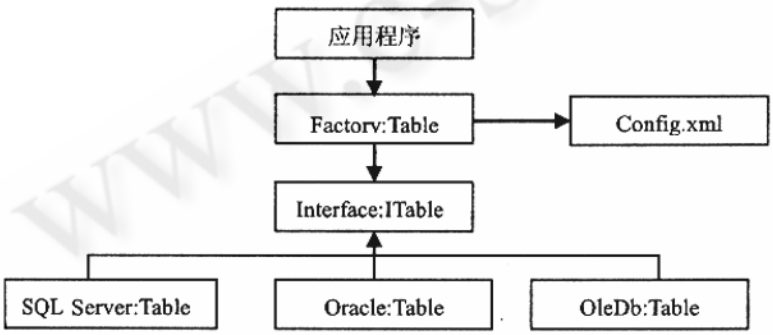

图 1　针对每个表的通用数据访问层类图

4　具体实现

4.1　面向业务逻辑层的通用数据存储结构

首先，我们利用 DataSet 对象基于 XML 技术的原理，在内存中建立每个表的对应存储数据结构类，这个存储数据结构类是派生自 DataSet 对象的，我们的目的是实现数据源中的一个表对应一个 DataSet 中的一个 DataTable。其实现代码如下所示：

```
//定义一个 DataSet 类型的 BookData 对象，用于存储从 Oracle，Sql server 等数据源中提出来的数据，前端应用程序使用的数据是从 BookData 对象中提取的
public class BookData : DataSet
{ //首先定义代替表名和表内字段的常量
  public const String BOOKS_TABLE = "Books";
  public const String PKID_FIELD = "PKId";
  public const String TYPE_ID_FIELD = "TypeId";
  //……
  public BookData()
  { BuildDataTables(); }
  private void BuildDataTables()
  { //本函数中建立一个与数据源中表结构相同的 DataTable 对象
    DataTable table = new DataTable(BOOKS_TABLE);
    DataColumnCollection columns = table.Columns;
    columns.Add(PKID_FIELD, typeof(System.Int32));
    columns.Add(TYPE_ID_FIELD, typeof(System.Int32));
    // ……
    this.Tables.Add(table); }
  }
```

4.2　供业务逻辑层调用的表接口

在对业务逻辑层对表的使用需求进行深入分析的基础上，抽象出供其使用的方法并在接口中定义，其实现代码如下所示：

```
public interface IBooks
  { public BookData GetBooksByCategoryId(int categoryId);//传递查询数据的参数
    //从数据库中提取数据的函数
    private BookData FillBookData(String commandText, String paramName, String paramValue);
    //…… }
```

4.3　继承自接口的连接不同数据源的数据提取类

在供业务逻辑层调用的表接口的基础上定义连接不同数据源的数据提取类，针对每个数据源实现相应的数据提取类，其他数据源类型只要使用相应的 ADO. NET 中的数据提供程序就可以，对于特殊的性能要求也完全可以从本部分给出实现。此处以针对 Oracle 数据库的数据提取类给出实现代码如下：

```
//本部分代码演示提取 Oracle 数据库中数据的方法
```

```
public class OracleBooks:IBooks
  { private OracleDataAdapter dsCommand;
    //本类的构造函数,调用此类时自动连接了 Oracle 数据库
    public OracleBooks(string connectionString)
    { dsCommand = new OracleDataAdapter();
      dsCommand.SelectCommand = new OracleCommand();
      dsCommand.SelectCommand.Connection = new OracleConnection(connectionString); }
    //对接口的实现,传递查询数据的参数
    public BookData GetBooksByCategoryId(int categoryId)
    { return FillBookData("GetBooksByCategoryId", "@CategoryId", categoryId.ToString()); }
    //对接口的实现,按照传递的参数从数据库中将数据提取出来
    public BookData FillBookData(string commandText, string paramName, string paramValue)
    { BookData data = new BookData();
      OracleCommand command = dsCommand.SelectCommand;
      command.CommandText = commandText;
      command.CommandType = CommandType.StoredProcedure; // use stored proc for perf
      OracleParameter param = new OracleParameter(paramName, OracleType.NVarChar, 255);
      param.Value = paramValue;
      command.Parameters.Add(param);
      dsCommand.Fill(data);
      return data; }
  }
```

4.4 在配置文件中定义连接数据库的配置信息(给出配置文件)

为了实现软件部署时对数据库平台的自由选择,我们在配置文件中给出了将要部署的数据库的数据库连接字符串,使用 ConnectionString 键表示。从而使得针对特定数据库的信息不被编译在应用程序中,在部署实施软件时可通过修改配置文件来达到与具体的 DBMS 无关的目的。以下是配置文件的示例:

```
<? xml version="1.0" encoding="utf-8" ? >
<configuration>
  <appSettings>
    <add key="DBType" value="Oracle" />
    <add key="ConnectionString" value="Provider = OraOLEDB.Oracle.1; User ID = system; Data Source = myoracle; Password = system" />
  </appSettings>
</configuration>
```

4.5 利用 factory 模式实现不同数据源的调用

首先,定义代表不同数据源的枚举类型,然后定义以表名为类名的一个提供给前端应用软件使用的将数据源中表数据载入 DataSet 中 DataTable 的初始化数据类,在这个初始化数据类中读取配置文件提供的连接数据源的类型和连接字符串,从而实现了前端应用软件与后台数据源的独立,满足了前端应用软件对通用性的要求。

```
public enum DBProperty
  { OleDB,ODBC,SQLServer,Oracle }
public class Books//前端应用程序调用 Book 表中数据时使用的对象
  { private int queryID;
    public Books(int id)
    { queryID = id; }
    public BookData GetBooks()
    { //从配置文件中读取数据库类型和连接参数
      string connectionString = ConfigurationSettings.AppSettings["ConnectionString"];
      DBProperty dbtype = (DBProperty)Enum.Parse(typeof(DBProperty), ConfigurationSettings.AppSettings["DbType"]);
      //通过 Factory 模式动态调用配置文件中给出的数据源
      IBooks myBooks = DBOperatorFactory.GetDBIBooks(connectionString, dbtype);
      //将数据载入 BookData 对象
      return myBooks.GetBooksByCategoryId(queryID); }
```

(下转第 60 页)

(上接第 56 页)

```
  }
//利用 Factory 模式实现的动态调用各种数据源的类
public class DBOperatorFactory
    { public static IBooks GetDBIBooks( string strCon-
nection, DBProperty p_DBType)
       { IBooks myIBooks = null;
         switch (p_DBType)
         { case DBProperty. Oracle:
            { myIBooks = new OracleBooks( strCon-
nection); break; }
           case DBProperty. SQLServer:
            { myIBooks = new SqlDBBooks( strConnec-
tion); break; }
           //……
           default:
            { //…… }
         }
        return myIBooks;
       }
    }
```

5 小结

本文在对现有的几种. NET 平台上的通用数据访问层设计方法进行认真分析的基础上,给出了一个融合其它几种通用访问数据方法的优点实现方案,并给出了详细的代码实现,对. NET 平台上的多层应用程序设计提供了很好参考。

参考文献

1 孙亚民,使用设计模式构建通用数据库访问类[EB/OL], http://dev. csdn. net/artideY20/20 071. shun, 2003 -03 -20.

2 Silvano Coriani,编写可移植数据访问层[EB/OL], http://www. microsoft. com/china/MSDN/library/data/dataAccess/wriportap2. mspx? pf = true#EOB.

3 漫谈. NetPetShop 和 DuwamishADO. NET 数据库编程[EB/OL], http://www. linuxmine. com/48367. html.