

# 基于.NET的数据访问基类的构建

黄光芳

(湛江师范学院, 广东 湛江 524048)

**摘 要:**为提高数据库应用程序的设计效率,基于VS.NET开发平台,利用ADO.NET的强大数据访问功能,设计了抽象的数据访问基类。主要通过代码的方式阐述了整个基类的创建过程,并重点介绍了创建存储过程参数的操作方法,在调用存储过程中,通过方法重载实现不同的数据库操作。利用该类并结合存储过程的优势构造数据服务层,统一整个系统的数据操作入口,提高开发效率,便于系统的升级、维护和移植。

**关键词:**ADO.NET;数据访问;基类;存储过程

**中图分类号:**TP311.52

**文献标识码:**A

**文章编号:**1673-629X(2008)03-0148-03

## Construction of Data Access Base Class Based on .NET

HUANG Guang-fang

(Zhanjiang Normal University, Zhanjiang 524048, China)

**Abstract:** For enhancing development efficiency of database application, an abstract data access base class was designed with ADO.NET that has strong capacity for data access based on VS.NET development platform. Explain the entire base class construction process by the code mainly, introduce the operating methods that create the stored procedure parameters and implement different database operation through the methods overloading in the course of transferring stored procedure. Utilize this base class and combine advantages of stored procedure to construct data server layer, unify the data operating entry of the whole system, improve the efficiency of development and make the upgrading, maintenance and transplanting of the system convenient.

**Key words:** ADO.NET; data access; base class; stored procedure

### 0 引言

在数据库应用程序中,数据访问界面都要执行相似的数据操作功能,例如查询、删除和修改等,另外,在不同或相同的应用中访问的数据种类不尽相同。所以,为了有效重用代码以提高开发效率,在项目开发中往往需要在数据层设计一个基类,利用此基类的派生类可方便地实现多种功能的数据访问。文中以课程与教学论资源库的开发为例,从数据层开始探讨在.NET开发中数据访问基类的实现过程。

### 1 基类相关技术

#### 1.1 ADO.NET 技术

ADO.NET是.NET中一个核心技术,它提供了非连接的数据操纵功能,是多层数据访问的典型,是.NET Framework中包含的一个功能强大的数据访问

类。ADO.NET库有两个核心的组件(并不是COM意义上的组件):DataSet和.NET数据提供者<sup>[1]</sup>。

DataSet是内存中的高速缓存区,用于高速缓存关系型的数据。为了执行常见的关系型数据任务,例如建立父/子关系,这个类可以包含数据表、数据列和数据行,甚至还可以包含各种表之间的关系。DataSet的另一个主要特点是它对底层的数据源一无所知,而这些数据源可能用于对其进行填充。这是一个分离的用于表示数据集合的独立实体,并且它可通过多层应用程序的不同层由一个组件传递到另一组件,也可作为XML数据流被序列化,因而非常适合于不同类型平台间的数据传输<sup>[2]</sup>。

数据源提供者中常见的接口类有Connection, Command, DataAdapter和DataReader等,每个类中都统一定义了一些方法和属性,以提供对特定类型数据源的托管访问<sup>[3]</sup>。文中以SQL Server数据库为数据源,具体的接口类分别是SqlConnection, SqlCommand, SqlDataAdapter和SqlDataReader等。

#### 1.2 存储过程

存储过程(Stored Procedure)是数据库中的一个重

收稿日期:2007-06-19

基金项目:广东省高校现代教育技术“151工程”项目(GDB045)

作者简介:黄光芳(1982-),男,广东湛江人,助理实验师,研究方向为信息技术的研究及应用。

要对象,是一组为了完成特定功能的 SQL 语句集,经编译后存储在数据库中<sup>[4]</sup>。对于带参数的存储过程,在调用前必须指明该参数的类型,是输入型的还是输出型的或者是输入输出型的还是返回值型的。用户通过相应的存储过程名称以及正确有效的参数便可调用该存储过程了。

使用存储过程访问数据库相对 SQL 语句有很大的优势:第一,大大提高效率。存储过程本身的执行速度非常快,而且,调用存储过程可以大大减少同数据库的交互次数;第二,提高安全性。系统管理员可以限制存储过程执行权限,避免非授权用户对数据的访问,保证数据的安全;第三,有利于 SQL 语句的重用。存储过程在被创建后,可以在程序中被多次调用,而不必重新编写该存储过程的 SQL 语句。

## 2 数据访问基类基本方法的构建

在分布式的应用开发,特别是基于 B/S 的开发结构中,通常采用三层甚至是多层架构。在一些小型的多层架构应用程序中,程序员为了简单方便,往往会把商务逻辑和数据访问放在同一逻辑层中,这在开发的初期虽然简单方便,但随着程序集的增加,重复的代码也越来越多,程序结构性和可维护性也越来越差。所以在资源库开发中,为了更方便处理繁多的数据操作,提高代码的利用率,系统以 SQL Server 为平台数据库建立一个数据层基类。在示例中,基类名为 zykJBaseClass,它是一个抽象类,使用 C# 语言编写,可以作为其他类的基类被继承。图 1 为资源库基类类图。

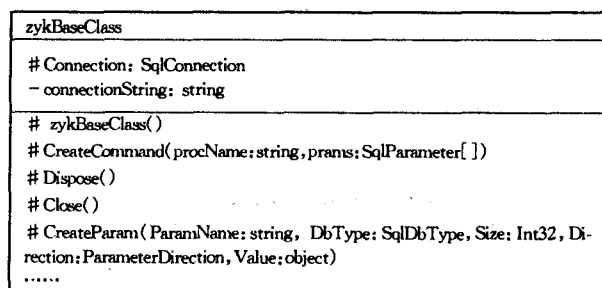


图 1 资源库基类类图

基类的开始首先引入命名空间,声明两个变量,第一个用来储存 SQL 连接,保护类型的,只有它的子类才能访问这一属性。为了进一步保证访问的高效性,在数据访问的基类中,通常采用单体模式,提供全局的唯一连接实例,减少系统开销,此外还应该在 Try/Catch 错误处理结构的内部编写数据库处理代码,从而在出现错误时可以做出响应<sup>[5]</sup>。另一个用来保存数据库连接字符串信息。如下所示:

```
using System;
using System.Data;
```

```
using System.Data.SqlClient;
namespace zyk.config
protected SqlConnection Connection;
private string connectionString;
```

下面是构造函数,通过连接字符串,创建一个 Connection 对象实例,连接信息从 web.config 配置文件读取,方便随时更改数据连接而不必重新构建项目,但是连接并没有打开,以便提高性能。

```
protected zykJBaseClass()
{
    connectionString = ConfigurationSettings.AppSettings
    ["ConnStr"];
    if(Connection == null)
    {
        try{Connection = new SqlConnection(connectionString);}
        catch(Exception ex){ ..... }
    }
}
```

另外基类中 Close 方法用于关闭数据连接,Dispose 方法用于释放资源。下面方法创建执行存储过程操作的 Command 对象并添加 Command 对象执行操作过程中所需要的参数。

```
protected SqlCommand CreateCommand(string procName, SqlParameter[] prams)
{
    SqlCommand cmd = new SqlCommand(procName, Connection);
    cmd.CommandType = CommandType.StoredProcedure;
    //添加存储过程的参数
    if (prams != null)
    {
        foreach (SqlParameter parameter in prams)
        {
            cmd.Parameters.Add(parameter);
        }
    }
    //添加返回参数 ReturnValue
    cmd.Parameters.Add(new SqlParameter(RETURNVALUE,
    SqlDbType.Int, 4, ParameterDirection.ReturnValue, false, 0, 0,
    string.Empty, DataRowVersion.Default, null));
    return cmd; //返回创建的 SqlCommand 对象
}
```

## 3 创建存储过程参数实现不同数据库操作

下面创建一系列的方法实现对表、存储过程、视图、索引和 XML 等数据库对象的操作。因为存储过程是数据库操作中最常用到的数据库对象,也是数据层操作中最为复杂的部分,而且使用存储过程访问数

数据库相对 SQL 语句有很大的优势,所以文中主要是针对存储过程的操作部分展开讨论。

### 3.1 创建存储过程参数

在应用逻辑层调用基类方法操作数据库时,必须先设置好存储过程参数。以下是生成存储过程参数的例子。

```
public SqlParameter CreateParam ( string ParamName, SqlDbType
DbType, Int32 Size, ParameterDirection Direction, object Value)
{
    SqlParameter param;
    if(Size > 0)
    {
        param = new SqlParameter(ParamName, DbType, Size);
    }
    else
    {
        //当参数大小为 0 时,不使用该参数大小值
        param = new SqlParameter(ParamName, DbType);
    }
    .....
    return param;
}
```

在操作存储过程时一个主要环节就是根据存储过程传递不同类型的参数。因此考虑到不同的需要,本系统提供了三种不同的调用方式。例如下面方法创建一个输入类型的参数。

```
public SqlParameter CreateInParam(string ParamName, SqlDbType
DbType, int Size, object Value)
{
    return CreateParam ( ParamName, DbType, Size, ParameterDirec-
tion. Input, Value);
}
```

另外,还可以使用同样的方法,通过更改输入输出类型,创建输出类型的参数和返回类型的参数。

### 3.2 利用重载实现不同的数据库操作

在创建 Command 对象与创建存储过程参数的函数编写完成后,接下来便可以创建不同的函数及存储过程来操作数据库。在执行存储过程中,根据表示层的需要,通过函数重载,可以返回不同类型的数据。下面方法执行一个带有参数集合的存储过程,该函数带有一个输出类型的参数,该参数用于保存从数据库获取的结果。

```
public void RunProc(string procName, SqlParameter[] prams,
out SqlDataReader dataReader)
{
    SqlCommand cmd = CreateCommand ( procName,
prams);
    dataReader = cmd. ExecuteReader( CommandBehav-
```

```
ior. CloseConnection);
}
```

通过重载的方式,还可以创建一系列的数据库操作方法,如 public int RunProc( string procName, SqlParameter[] prams)。它带两个参数:第一个参数表示存储过程名称;第二个参数表示存储过程参数,函数返回受影响的数据行。

### 3.3 利用 SQL 语句直接操作数据库

在课程与教学论资源库开发中,除了在基类中扩充更多的调用储存过程的方法以外,也可以在此类中构造运用 SQL 语句直接操作数据库的方法让其他类继承调用,这种方法因为在数据层没有涉及到参数传递的问题,所以相对存储过程简单很多。下面 GetDataReader 函数就是直接通过 SQL 语句读取数据库内容并返回一个 SqlDataReader 对象。

```
public SqlDataReader GetDataReader(string SqlString)
{
    SqlCommand = new SqlCommand(SqlString, SqlConn);
    SqlConn. Open(); //打开数据库
    SqlDataReader SqlDrow =
    SqlCommand. ExecuteReader
    (System. Data. CommandBehavior. CloseConnection);
    return SqlDrow; //返回 SqlDataReader 对象
}
```

通过为整个数据层构造一个这样的基类,在系统中所有的数据底层操作都将通过这个类进行,便于维护、升级,也可以最大限度的复用,能方便地扩充底层数据操作,而不影响具体数据访问组件操作,从而统一控制了数据访问的入口。

## 4 结束语

文中以 C# 语言、SQL Server 数据库为例,利用 ADO. NET 数据库访问组件,阐述一种基于 ASP. NET 技术的数据库访问基类设计方法,并重点介绍了存储过程的数据库操作。在资源库具体的开发中,还涉及到数据库组件的安全性、异常处理及存储过程权限设置等问题,这里就不再赘述。文中虽然只是针对资源库的基类展开讨论,不过作为基类的一种开发方法,在其他类似的系统开发中都有一定的参考意义。

### 参考文献:

- [1] Ardestani K, Hoffman K, Xie D. 高效掌握 ADO. NET——C# 编程篇[M]. 张哲峰译. 北京:清华大学出版社,2003.
- [2] 章剑林,熊传光. 多层架构下的数据访问基类构造——基于 .net 平台[J]. 安徽理工大学学报:自然科学版,2005,25

(下转第 168 页)

够获得属性名称;

- \* 可以在编译期不知道对象名称的情况下,调用该对象的方法;

- \* 可以操作在编译期不知道大小和类型的数组。

JADE 在执行期用反射 API 来动态加载和执行 Agents, 筛选对象等操作。以上提到的很多带来隐患的机制还可以应用到私有的类、属性、方法。在编译时, 编译程序保证了私有成员的私有特性, 从而, 一个类的私有方法和私有成员变量不能被其他类静态引用。然而, 反射机制使得在运行时可以查询以及访问变量和方法。由于反射是动态的, 因此编译时的检查就不再适用了。一个攻击者利用反射机制只需要一个运行的实例, 他就可以得到正在当前容器活动的一个 Agent 引用, 然后通过 `this.getContainerController()` 读取私有属性 `myImpl`。myImpl 包含产生当前被攻击 Agent 的容器的引用。得到了这个容器的引用, 攻击者就可以进入、操纵该 Agent 的所有数据, 包括私有的。这种攻击方法的应用是典型的 Agent 遭受恶意平台攻击。如何保护 JADE 平台使其不受这种攻击, 或者让攻击损失降到最低, 是值得研究的问题。

### 3.3 解决方案

恶意 Agent 平台可以利用 JADE-S 这一漏洞攻击 Agent, 即 Agent 遭受 Agent 平台攻击, 主要有两种可能性:

- \* 容器和主机是恶意的, 专门捕获迁移过去的 Agent, 导致安全问题(2);

- \* 容器和主机已经被破坏, 成为了一个恶意主机。

解决这个安全问题, 可以从两个角度来进行: 恶意主机的识别和移动 Agent 的保护。

首先从恶意主机的识别的角度出发。前面提到, 每一个容器加入 JADE 平台的时候首先要在主容器注册, 注册的时候, 可以进行一些处理。处理的目的是有两个: 一是识别该容器不是恶意的; 二是确保该容器以后也不会变成恶意的(防止欺骗行为)。首先对该容器进行完整性等安全性检查, 证明合法性后可以给该容器颁布一个证书, 并对这些系统 jar 文件进行签名(防止系统文件被篡改)。每次 Agent 要迁移到某个容器的时候先检查该容器的证书, 确保容器的合法性。

其次从移动 Agent 的保护的角度。关于 Agent 的安全保护问题一直是当前学术界研究的热点问题。这个问题跟传统的安全保护问题不同。传统的保护概念一般指的是保护系统安全, 而 Agent 的保护问题是保护一段能够自由在网内移动的可执行代码集。当前提出的很多方案并不能完全支持许多 Agent 应用所要求的自由迁移和自治能力, 很难广泛应用于实际。这里针对当前的应用, 提出在 Agent 遭受恶意攻击时候能够尽量让损失控制在最小。从上面的论述可以看到, 在 JADE-S 的安全机制下, Agent 携带了私钥, 这样做在带来了效率的同时也带来了安全隐患。解决方法: 一是可以对属性值进行重组<sup>[6]</sup>, 这样做防止 Agent 泄密; 二是 Agent 只携带公钥, 需要私钥的时候根据秘密约定向宿主容器请求私钥。

## 4 结 语

对当前流行的开源移动 Agent 平台 JADE(-S) 的安全性问题进行了较为全面的分析。对它的缺陷提出了一些合理的改进方案。下一步的工作是实现研究方案中的具体细节, 并测试其性能。为进一步研究移动 Agent 的安全和完善 JADE 的 E 安全提供了参考, 有一定的实用价值和前景。

### 参考文献:

- [1] Jansen W, Karygiannis T. Mobile Agent Security[M]. [s. l.]: NIST Special Publication, 2000
- [2] JADE Board. JADE Home Page[EB/OL]. 2007-05-21. <http://jade.cselt.it/>.
- [3] JADE Board. JADE Security Guide: Usage Restricted According to License Agreement[EB/OL]. Copyright (C) 2004 TILAB S.p.A. 2005-02-28. <http://jade.cselt.it/doc>.
- [4] Vila X, Schuster A, Riera A. Security for a Multi-Agent System based on JADE[J]. Computer & Security, 2007, 26: 391-400.
- [5] Campione M, Walrath K. The Java Tutorial, Second Edition: Object Oriented Programming for the Internet[M]. [s. l.]: Addison Wesley Publishing Company, 1998.
- [6] 孟 健, 曹立明, 王小平. Mobile Agent 的两种安全机制[J]. 计算机工程应用, 2003, 29(21): 136-138.

(上接第 150 页)

(1): 18-21.

- [3] 毛 莉, 刘广强. 基于 .NET 的数据访问策略[J]. 微机发展, 2004, 14(10): 52-54.
- [4] 鄢爱兰, 鹿江春. 数据库存储过程应用研究[J]. 南华大学

学报, 2006, 20(2): 100-102.

- [5] MacDonald M. ASP.NET 完全手册[M]. 贾晓军, 于秀山, 吕嘉章, 等译. 北京: 电子工业出版社, 2003.