**MT.Frameworks.Data.** **WebDataAccessProxy**

**数据库访问框架使用介绍**

**一. WebDataAccessProxy使用**

该数据库操作代理框架是前者**SimpleDataAccessProxy** 框架的扩展版相比下要轻便一些，不过**SimpleDataAccessProxy**框架比价适合大型的项目使用，该框架就适合一些小型项目使用，主要用在Website应用程序上。操作方法和**SimpleDataAccessProxy**框架基本相似，所以大部分类的解释不在这里重复说了。

特点：该框架可以支持MS-SQL、Access、My-SQL三个类型数据库

首先需要导入框架的必須的DLL 文件

1. MT.Framework.Data.dll 框架核心
2. MySql.Data.dll My-SQL 的支持
3. MT.Framework.Data.WebDataAccessProxy.dll

MT.Framework.Data.WebDataAccessProxy.dll数据库操作常用类

|  |  |
| --- | --- |
| **类** | **简介** |
| DbManager | 用于设置和获取数据库源连接 |
| IDatabase | 对数据库操作的接口 |

DbManager 类 用于管理数据库源的连接

|  |  |
| --- | --- |
| **成员** | **简介** |
| static bool AutoConnection | 设置是否自动连接数据源 |
| static bool AutoTransaction | 设置是否自动使用事务 |
| static bool CheckDBName(string databaseName) | 检查数据库名称《等同于集合类的键》是否存在 |
| static void SetDatabase(string databaseName, string connectionString, DatabaseType databaseType) | 设置数据库源，databaseName 数据库名称《等同于集合类的键》，connectionString 数据库源连接字符窜，databaseType 数据库源类型 MS-SQL、Access、My-SQL |
| static IDatabase GetDB(string databaseName) | 通过数据库名称《等同于集合类的键》获取数据库操作对象 |

IDatabase 接口 用于执行数据库操作

|  |  |
| --- | --- |
| **成员** | **简介** |
| DatabaseType DataBaseType | 获取当前使用的数据库类型 |
| bool AutoConnection | 获取当前是否自动连接数据源 |
| bool AutoTransaction | 获取当前是否自动使用事务 |
| void OpenConnection() | 打开数据源连接 |
| void CloseConnection() | 关闭数据源连接 |
| void ForceCloseConnection() | 强制关闭数据源连接并释放该数据源连接资源 |
| ConnectionState ExistsConnection() | 检查数据源连接状态 |
| void BeginTransaction() | 开启事务 |
| void Commit() | 提交事务 |
| void Rollback() | 回滚事务 |
| DbCommand BuildCommand(Procedure proc) | 创建实行语句对象 |
| DbCommand BuildCommand(string sql) | 创建实行语句对象 |
| void ExcuteNotQuery(string sql) | 执行没有返回资料的操作语句 |
| void ExcuteNotQuery(Procedure procedure) | 执行没有返回资料的操作存储过程 |
| void ExcuteNotQuery(string procedureText, Parameter parameter) | 执行没有返回资料的操作存储过程 |
| void ExcuteNotQuery(string procedureText, params Parameter[] parameters) | 执行没有返回资料的操作存储过程 |
| SimpleDataTable ExcuteSimpleDataTable(string sql) | 执行并返回资料的操作语句 |
| DataTable ExcuteDataTable(string sql) | 执行并返回资料的操作语句 |
| SimpleDataTable ExcuteSimpleDataTable(Procedure procedure) | 执行并返回资料的操作存储过程 |
| SimpleDataTable ExcuteSimpleDataTable(string procedureText, Parameter parameter) | 执行并返回资料的操作存储过程 |
| SimpleDataTable ExcuteSimpleDataTable(string procedureText, params Parameter[] parameters) | 执行并返回资料的操作存储过程 |
| DataTable ExcuteFullDataTable(Procedure procedure) | 执行并返回资料的操作存储过程 |
| DataTable ExcuteFullDataTable(string procedureText, Parameter parameter) | 执行并返回资料的操作存储过程 |
| DataTable ExcuteFullDataTable(string procedureText, params Parameter[] parameters) | 执行并返回资料的操作存储过程 |
| IList<DataTable> ExcuteDataTables(string sql) | 执行并返回多个资料表的操作语句 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(string sql) | 执行并返回多个资料表的操作语句 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(Procedure procedure) | 执行并返回多个资料表的操作存储过程 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(string procedureText, Parameter parameter) | 执行并返回多个资料表的操作存储过程 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(string procedureText, params Parameter[] parameters) | 执行并返回多个资料表的操作存储过程 |
| IList<DataTable> ExcuteFullDataTables(Procedure procedure) | 执行并返回多个资料表的操作存储过程 |
| IList<DataTable> ExcuteFullDataTables(string procedureText, Parameter parameter) | 执行并返回多个资料表的操作存储过程 |
| IList<DataTable> ExcuteFullDataTables(string procedureText, params Parameter[] parameters) | 执行并返回多个资料表的操作存储过程 |

**二.使用事例：**

[Serializable]

public class WebArticleTypeEntity

{

#region --- Fields Begin ---

private int \_ID = 0;

public int ID

{

get

{

return \_ID;

}

set

{

\_ID = value;

}

}

private string \_TypeName = string.Empty;

public string TypeName

{

get

{

return \_TypeName;

}

set

{

\_TypeName = value;

}

}

private int \_ParentID = 0;

public int ParentID

{

get

{

return \_ParentID;

}

set

{

\_ParentID = value;

}

}

private int? \_Seq = 0;

public int? Seq

{

get

{

return \_Seq;

}

set

{

\_Seq = value;

}

}

private bool \_Effectiveness = true;

public bool Effectiveness

{

get

{

return \_Effectiveness;

}

set

{

\_Effectiveness = value;

}

}

#endregion --- Fields End ---

#region --- 执行SQL语句 Begin ---

protected const string INSERT\_ArticleType = "INSERT INTO Article\_Type (Name,ParentID,Seq,Effectiveness) VALUES({0},{1},{2},1)";

protected const string UPDATE\_ArticleType\_Name = "UPDATE Article\_Type SET [Name]={1} WHERE ID={0}";

protected const string UPDATE\_ArticleType\_\_Effectiveness\_ByID = "UPDATE Article\_Type SET Effectiveness={1} WHERE ID={0} OR ParentID={0}";

protected const string SELECT\_All\_ArticleType = "SELECT ID,Name,ParentID,Seq,Effectiveness from Article\_Type WHERE Effectiveness=1";

#endregion --- 执¡ä行DSQL语®?句? End ---

#region --- 保À¡ê存ä?资Á¨º料¢? Begin ---

/// <summary>

/// 添加资料

/// </summary>

/// <param name="articleType"></param>

public static void SaveArticleType(WebArticleTypeEntity articleType)

{

Procedure proc = new Procedure("cArticle\_Type");

proc.Parameters.Add(new Parameter("Name", DataType.NVarChar, articleType.TypeName));

proc.Parameters.Add(new Parameter("ParentID", DataType.Int, articleType.ParentID));

proc.Parameters.Add(new Parameter("Seq", DataType.Int, articleType.Seq));

IDatabase db = ServerConfig.GetDB();

try

{

//使用存储过程

db.ExcuteNotQuery(proc);

//使用SQL语句

//db.ExcuteNotQuery(string.Format(ArticleTypeEntity.INSERT\_ArticleType, articleType.TypeName, articleType.ParentID, articleType.Seq));

}

catch (Exception ex)

{

throw ex;

}

}

#endregion --- 保存资料 End ---

#region --- 更新资料 Begin ---

/// <summary>

/// 更新资料

/// </summary>

/// <param name="articleType"></param>

public static void UpdateArticleTypeName(WebArticleTypeEntity articleType)

{

Procedure proc = new Procedure();

proc.Text = "uArticle\_TypeName";

proc.Parameters.Add(new Parameter("ID", DataType.Int, articleType.ID));

proc.Parameters.Add(new Parameter("Name", DataType.NVarChar, articleType.TypeName));

IDatabase db = ServerConfig.GetDB();

try

{

//使用存储过程

db.ExcuteNotQuery(proc);

//使用SQL语句

//db.ExcuteNotQuery(string.Format(ArticleTypeEntity.UPDATE\_ArticleType\_Name, articleType.ID, articleType.TypeName));

}

catch (Exception ex)

{

throw ex;

}

}

/// <summary>

/// 撤销资料

/// </summary>

/// <param name="articleType"></param>

public static void DeleteArticleType(WebArticleTypeEntity articleType)

{

Procedure proc = new Procedure("uArticle\_TypeEffectiveness");

proc.Parameters.Add(new Parameter("ID", DataType.Int, articleType.ID));

proc.Parameters.Add(new Parameter("Effectiveness", DataType.Bit, 0));

IDatabase db = ServerConfig.GetDB();

try

{

//使用存储过程

db.ExcuteNotQuery(proc);

//使用SQL语句

// db.ExcuteNotQuery(string.Format(ArticleTypeEntity.UPDATE\_ArticleType\_\_Effectiveness\_ByID, articleType.ID, 0));

}

catch (Exception ex)

{

throw ex;

}

}

#endregion --- 更新料 End ---

#region --- 读取资料 Begin ---

/// <summary>

/// 读取资料

/// </summary>

/// <returns></returns>

public static ArticleTypeEntityList GetArticleTypeEntityListForAll()

{

ArticleTypeEntityList tagList = new ArticleTypeEntityList();

Procedure proc = new Procedure("rArticle\_TypeForAll");

IDatabase db = ServerConfig.GetDB();

try

{

//存储过程方式

DataTable tb = db.ExcuteFullDataTable(proc);

//SQL语句方式

//DataTable tb = db.ExcuteDataTable(ArticleTypeEntity.SELECT\_All\_ArticleType);

foreach (DataRow dr in tb.Rows)

{

WebArticleTypeEntity entity = new WebArticleTypeEntity();

entity.ID = Convert.ToInt32(dr["ID"]);

entity.TypeName = dr["Name"].ToString();

entity.ParentID = Convert.ToInt32(dr["ParentID"]);

if (dr["Seq"] == DBNull.Value)

{

entity.Seq = null;

}

else

{

entity.Seq = Convert.ToInt32(dr["Seq"]);

}

entity.Effectiveness = Convert.ToBoolean(dr["ID"]);

}

}

catch (Exception ex)

{

throw ex;

}

return tagList;

}

#endregion --- 读取资料 End ---

}

[Serializable]

public class ArticleTypeEntityList : List<WebArticleTypeEntity>

{

}

/// <summary>

/// 数据库源设置

/// </summary>

public static class ServerConfig

{

//数据源Key名称

public const string dataAccessName = "WebDB";

//数据源连接字符串

private const string connStr = @"Data Source=.;Initial Catalog=DatabaseName;User ID=sa;Password=sa";

static ServerConfig()

{

DbManager.SetDatabase(dataAccessName, connStr, DatabaseType.MS\_Sql);

//自动开启连接

DbManager.AutoConnection = true;

//自动使用事务

DbManager.AutoTransaction = true;

}

public static IDatabase GetDB()

{

return DbManager.GetDatabase(dataAccessName);

}

}

**三.使用反射赋值。**

导入相关DLL库

1. MT.Framework.ReflectionUtil.dll
2. MT.Framework.Data.Attributes.dll
3. MT.Framework.Data.ORM.dll
4. MT.Framework.Data.WebDataAccessProxy.ORM.dll

和之前SimpleDataAccessProxy.ORM 使用相似，不过这里只能赋值，而SimpleDataAccessProxy.ORM 可以做一些添加参数修改数据资料操作。

编写实体标记的时候只需要标记属性就可以了

如下面代码片段：

private int \_ID = 0;

[ID("ID", DataType.Int)] //主键标记

public int ID

{

get

{

return \_ID;

}

set

{

\_ID = value;

}

}

private string \_TypeName = string.Empty;

[Field("Name", DataType.NChar)] //其他成员标记 注：名称和类型必须和数据库表中的字段匹配。

public string TypeName //注意：在数据库表字段中是没有 TypeName 字段的可以通过标记来标明。

{

get

{

return \_TypeName;

}

set

{

\_TypeName = value;

}

}

private int \_ParentID = 0;

[Field("ParentID")] //如果类型相同 可以省略标记类型

public int ParentID

{

get

{

return \_ParentID;

}

set

{

\_ParentID = value;

}

}

private int? \_Seq = 0;

[Field()] //名称和类型都相同可以省略所有参数

public int? Seq

{

get

{

return \_Seq;

}

set

{

\_Seq = value;

}

}

MT.Framework.Data.WebDataAccessProxy.ORM.dll 反射填充实体对象工具库

|  |  |
| --- | --- |
| **类** | **简介** |
| FetchDataMethod | 反射填充实体对象工具 |

FetchDataMethod 反射填充实体对象工具类成员

|  |  |
| --- | --- |
| **成员** | **简介** |
| static void MapData(object obj, SimpleDataRow dr) | 对资料进行读取并赋值到实体类上 |
| static void MapData(object obj, DataRow dr) | 对资料进行读取并赋值到实体类上 |
| static T MapData<T>(SimpleDataRow dr) | 对资料进行读取并赋值到实体类上 |
| static T MapData<T>(DataRow dr) | 对资料进行读取并赋值到实体类上 |
| static void MapData<T, V>(T tList, SimpleDataTable td) | 对资料进行读取并填充到对应的实体对象列表，T为IList<V>,V为实体类。 |
| static void MapData<T, V>(T tlist, DataTable td) | 对资料进行读取并填充到对应的实体对象列表，T为IList<V>,V为实体类。 |