**MT.Frameworks.Data.** **SimpleDataAccessProxy**

**数据库访问框架使用简介**

**一.SimpleDataAccessProxy使用**

该数据库操作代理框架适合于MS-SQL数据库操作，可以使用户更简便操作数据库，减少开发者的对数据库操作那些繁琐的代码片段，有效管理数据库源，支持多个MS-SQL数据库共同为一个项目保存数据。

首先需要导入框架的必須的DLL 文件

1. MT.Framework.Data.dll
2. MT.Framework.Data.SimpleDataAccessLog.dll
3. MT.Framework.Data.SimpleDataAccessProxy.dll
4. NLog.dll

MT.Framework.Data.Dll 是该框架的核心库常用类

|  |  |
| --- | --- |
| **类** | **简介** |
| Procedure | 存储过程 |
| Parametercs | 存储过程参数 |
| SimpleDataTable | 简单数据表 |
| SimpleDataColumn | 简单数据表列 |
| SimpleDataColumnCollection | 简单数据表列集合 |
| SimpleDataRow | 简单数据表行 |
| SimpleDataRowCollection | 简单数据表行集合 |
| DatabaseState | 数据库状态枚举 |
| DatabaseType | 数据库类型枚举 |
| DataType | 数据类型枚举 |
| ParaDirection | 存储过程参数输出、输入方向枚举 |
| TransactionType | 事务类型枚举 |
| SmartDate | 简单的日期数据类型 |
| DataAccessExceptioncs | 数据库操作异常 |

Procedure类

|  |  |
| --- | --- |
| **成员** | **简介** |
| Procedure() | 默认构造函数 |
| Procedure(string text) | 需要存储过程语句构造函数 |
| string Text | 获取或设置存储过程语句 |
| int CommandTimeOut | 获取或设置存储过程语句执行时长 |
| List<Parameter> Parameters | 获取或设置参数对象集合 |
| bool RemoveParameter(string parameterName) | 根据参数名称移除参数对象 |
| object this[string parameterName] | 根据参数名称获取或设置参数值 |

Parametercs类

|  |  |
| --- | --- |
| **成员** | **简介** |
| Parameter() | 默认构造函数 |
| Parameter(string name) | 需要参数名称的构造函数 |
| Parameter(string name, DataType type) | 需要参数名称和数据类型的构造函数 |
| Parameter(string name, DataType type, object value) | 需要参数名称、数据类型和数据值的构造函数 |
| Parameter(string name, ParaDirection dirtion, DataType type) | 需要参数名称、参数方向、参数类型的构造函数 |
| Parameter(string name, ParaDirection dirtion, DataType type, object value) | 需要参数名称、参数方向、参数类型、参数值的构造函数 |
| string Name | 获取或设置参数名称 |
| ParaDirection Direction | 获取或设置参数方向 |
| DataType Type | 获取或设置参数类型 |
| object Value | 获取或设置参数值 |
| bool IsOut() | 检查是否有输出参数 |

SmartDate类

基本和DataTime类相似，不同地方在于它只有日期没有时分秒参数。

SimpleDataTable类

|  |  |
| --- | --- |
| **成员** | **简介** |
| SimpleDataTable() | 默认构造函数 |
| SimpleDataRow CreateRow() | 创建属于改数据表的行对象 |
| SimpleDataRowCollection Rows | 获取或设置行对象集合 |
| SimpleDataColumnCollection Columns | 获取或设置数据表的列对象集合 |

SimpleDataColumn类

|  |  |
| --- | --- |
| **成员** | **简介** |
| SimpleDataColumn() | 默认构造函数 |
| string ColumnName | 获取或设置列名称 |
| Type Type | 获取或设置该列的数据类型 |

DataAccessExceptioncs类

当操作反生异常会通过该异常类抛出告知使用者

MT.Framework.Data.SimpleDataAccessLog.dll 一个操作日志记录通过第三方NLog.dll 框架实现，集合于本框架中

MT.Framework.Data.SimpleDataAccessProxy.DLL 数据库操作库常用类

|  |  |
| --- | --- |
| **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 ExportRunDBLog | 设置是否需要使用日志记录每一个操作 |
| 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 SetDB(string databaseName, string connectionString) | 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 |
| 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** |
| void OpenConnection() | 开启数据库连接 |
| void CloseConnection() | 关闭数据库连接 |
| void BeginTransaction() | 开始执行事务 |
| bool IsInTrancation() | 操作是否在事务中执行 |
| void UseThreadTransaction() | 使用线程级的事务 |
| void UseIndependenceTransaction() | 使用独立事务 |
| void UseTransactionDependDB(IDatabase database) | 使用来自其他数据库的事务 |
| void UseCustomerTransaction(string transactionName) | 使用自定义事务 |
| void Commit() | 提交事务 |
| void Rollback() | 回滚事务 |
| string ConnectionString | 获取连接字符窜 |
| DatabaseType DatabaseType | 获取数据库类型 |
| TransactionManager TransactionManager | 获取当前数据库事务管理器对象 |
| TransactionType TransactionType | 获取事务类型 |
| SimpleDataTable ExcuteSimpleDataTable(Procedure procedure) | 执行数据库读取获取一个资料表数据 |
| SimpleDataTable ExcuteSimpleDataTable(string procedureText, Parameter parameter) | 执行数据库读取获取一个资料表数据 |
| SimpleDataTable ExcuteSimpleDataTable(string procedureText, params Parameter[] parameters) | 执行数据库读取获取一个资料表数据 |
| DataTable ExcuteFullDataTable(Procedure procedure) | 执行数据库读取获取一个资料表数据 |
| DataTable ExcuteFullDataTable(string procedureText, Parameter parameter) | 执行数据库读取获取一个资料表数据 |
| DataTable ExcuteFullDataTable(string procedureText, params Parameter[] parameters) | 执行数据库读取获取一个资料表数据 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(Procedure procedure) | 执行数据库读取获取多个资料表数据 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(string procedureText, Parameter parameter) | 执行数据库读取获取多个资料表数据 |
| IList<SimpleDataTable> ExcuteSimpleDataTables(string procedureText, params Parameter[] parameters) | 执行数据库读取获取多个资料表数据 |
| IList<DataTable> ExcuteFullDataTables(Procedure procedure) | 执行数据库读取获取多个资料表数据 |
| IList<DataTable> ExcuteFullDataTables(string procedureText, Parameter parameter) | 执行数据库读取获取多个资料表数据 |
| IList<DataTable> ExcuteFullDataTables(string procedureText, params Parameter[] parameters) | 执行数据库读取获取多个资料表数据 |
| void ExcuteNotQuery(Procedure procedure) | 执行数据资料操作 |
| void ExcuteNotQuery(string procedureText, Parameter parameter) | 执行数据资料操作 |
| void ExcuteNotQuery(string procedureText, params Parameter[] parameters) | 执行数据资料操作 |
| void ExcuteReader(Procedure procedure, EventHandler<ExcuteReaderEventArgs> excuteReaderEvent) | 进行读取数据资料 |
| void ExcuteReader(string procedureText, Parameter parameter, EventHandler<ExcuteReaderEventArgs> excuteReaderEvent) | 进行读取数据资料 |
| void ExcuteReader(string procedureText, Parameter[] parameters, EventHandler<ExcuteReaderEventArgs> excuteReaderEvent) | 进行读取数据资料 |

**二. SimpleDataAccessProxy.ORM嵌入使用**

加入ORM 反射赋值使SimpleDataAccessProxy 代理操作赋值到资料数据实体更为捷便，但是由于是通过反射模式去实现所以会占用一定资源和时间，优点就是减少获取数据后赋值到实体那一个繁琐步骤。

首先需要添加的DLL文件

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

MT.Framework.ReflectionUtil.dll 是一个反射工具库便于框架里面反射类的操作

MT.Framework.Data.ORM.dll 反射框架的基本库可使日后各种数据库赋值反射扩展

|  |  |
| --- | --- |
| **类** | **简介** |
| MultiTablesMapper | 多表反射机制 |
| ORMException | 反射异常 |

MT.Framework.Data.Attributes.dll 实体类标识库，用于标记实体属性

|  |  |
| --- | --- |
| **成员** | **简介** |
| IDAttribute | 主键标记 |
| FieldAttribute | 属性成员标记 |
| InsertAccessAttribute | 添加资料存储过程语句标记 |
| DeleteAccessAttribute | 撤销资料存储过程语句标记 |
| UpdateAccessAttribute | 更改资料存储过程语句标记 |
| FetchAccessAttribute | 读取资料存储过程语句标记 |

MT.Framework.Data.SimpleDataAccessProxy.ORM.dll SimpleDataAccessProxy的专用反射赋值操作库，实现框架的反射赋值效果。

|  |  |
| --- | --- |
| **类** | **简介** |
| SimpleDataAccessORMUtility | 实体标记读取包 |
| SimpleFetchDataMethod | 资料自动赋值到实体操作包 |

SimpleDataAccessORMUtility类 获取标记内容信息和进行对数据库进行默认的操作

|  |  |
| --- | --- |
| **成员** | **简介** |
| static void Insert(object obj, IDatabase db) | 对新资料数据进行持久化保存 |
| static void Insert(object obj, IDatabase db, EventHandler<ProcedureEventArgs> beforeExcuteEventHandler) | 对新资料数据进行持久化保存 |
| static void Delete(object obj, IDatabase db) | 对资料进行撤销操作 |
| static void Delete(object obj, IDatabase db, EventHandler<ProcedureEventArgs> beforeExcuteEventHandler) | 对资料进行撤销操作 |
| static void Update(object obj, IDatabase db) | 对资料进行更新操作 |
| static void Update(object obj, IDatabase db, EventHandler<ProcedureEventArgs> beforeExcuteEventHandler) | 对资料进行更新操作 |
| 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 |

SimpleFetchDataMethod 类，该类是为了节省SimpleDataAccessORMUtility进行读取资料数据操作时的代码片段。

注：只能用于SimpleDataAccessProxy里面的操作，如果你自己重新实现  
SimpleDataAccessProxy里面机构操作改类未必能实现当前效果。

|  |  |
| --- | --- |
| **成员** | **简介** |
| static IDatabase DB | 设置执行的数据库操作对象 |
| static T Fetch<T>() | 执行读取数据库资料填充对应的实体对象，T为实体类。 |
| static T Fetch<T>(string accessName, params Parameter[] parameters) | 执行读取数据库资料填充对应的实体对象，T为实体类。 |
| static T Fetch<T>(string accessName, List<Parameter> parameters) | 执行读取数据库资料填充对应的实体对象，T为实体类。 |
| static T Fetch<T>(params Parameter[] parameters) | 执行读取数据库资料填充对应的实体对象，T为实体类。 |
| static T Fetch<T>(List<Parameter> parameters) | 执行读取数据库资料填充对应的实体对象，T为实体类。 |
| static void Fetch<T, V>(T tList) | 执行读取数据库资料填充对应的实体对象列表，T为IList<V>,V为实体类。 |
| static void Fetch<T, V>(T tList, string accessName) | 执行读取数据库资料填充对应的实体对象列表，T为IList<V>,V为实体类。 |
| static void Fetch<T, V>(T tList, params Parameter[] parameters) | 执行读取数据库资料填充对应的实体对象列表，T为IList<V>,V为实体类。 |
| static void Fetch<T, V>(T tList, List<Parameter> parameters) | 执行读取数据库资料填充对应的实体对象列表，T为IList<V>,V为实体类。 |
| static void Fetch<T, V>(T tList, string accessName, params Parameter[] parameters) | 执行读取数据库资料填充对应的实体对象列表，T为IList<V>,V为实体类。 |
| static void Fetch<T, V>(T tList, string accessName, List<Parameter> parameters) | 执行读取数据库资料填充对应的实体对象列表，T为IList<V>,V为实体类。 |

**三.使用事例：**

/// <summary>

/// 数据库源配置

/// </summary>

public static class ServerConfig

{

//数据库源Key名称

public const string dataAccessName = "WebDB";

//数据库源连接字符串

private const string connStr = @"Data Source=MICHAEL-PC\SQL2008DEV;Initial Catalog=WebDB;User ID=DEV2008;Password=dev2008";

static ServerConfig()

{

DbManager.SetDB(dataAccessName, connStr);

DbManager.ExportRunDBLog = true;

}

public static IDatabase GetDB()

{

return DbManager.GetDB(dataAccessName);

}

}

[InsertAccess("cObj", "cArticle\_Type")]

[FetchAccess("All", "rArticle\_TypeForAll")]

[FetchAccess("ByParentID", "rArticle\_TypeByParentID")]

[UpdateAccess("uName", "uArticle\_TypeName")]

[UpdateAccess("uEffectiveness", "uArticle\_TypeEffectiveness")]

[Serializable]

public class ArticleTypeEntity

{

#region --- Fields Begin ---

//使用标记可以通过反射方式自动填充资料值

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;

}

}

private bool \_Effectiveness = true;

[Field()]

public bool Effectiveness

{

get

{

return \_Effectiveness;

}

set

{

\_Effectiveness = value;

}

}

#endregion --- Fields End ---

#region ---保存资料 Begin ---

/// <summary>

/// 手动保存资料

/// </summary>

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

public static void SaveArticleTypeA(ArticleTypeEntity articleType)

{

// 手动建立存储过程

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

{

db.BeginTransaction();

db.ExcuteNotQuery(proc);

db.Commit();

}

catch (Exception ex)

{

db.Rollback();

throw ex;

}

}

/// <summary>

/// 通过标记获取存储过程来保存资料

/// </summary>

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

public static void SaveArticleTypeB(ArticleTypeEntity articleType)

{

//根据标记来获取存储过程

Procedure proc = SimpleDataAccessORMUtility.GetInsertProcedure(articleType, "cObj");

//Procedure proc = new Procedure();

//proc.Text = "cArticle\_Type";

proc.Parameters.Clear();

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

{

db.BeginTransaction();

db.ExcuteNotQuery(proc);

db.Commit();

}

catch (Exception ex)

{

db.Rollback();

throw ex;

}

}

#endregion ---保存资料 End ---

#region --- 更¨¹新?资Á¨º料¢? Begin ---

/// <summary>

/// 更新资料

/// </summary>

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

public static void UpdateArticleTypeName(ArticleTypeEntity articleType)

{

//手动方式

Procedure proc = SimpleDataAccessORMUtility.GetFetchAllProcedure(typeof(ArticleTypeEntity), "uName");

//根据标记来获取存储过程

//Procedure proc = SimpleDataAccessORMUtility.GetInsertProcedure(articleType, "uName");

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

{

db.BeginTransaction();

db.ExcuteNotQuery(proc);

db.Commit();

}

catch (Exception ex)

{

db.Rollback();

throw ex;

}

}

/// <summary>

///撤销资料

/// </summary>

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

public static void DeleteArticleType(ArticleTypeEntity 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

{

db.BeginTransaction();

db.ExcuteNotQuery(proc);

db.Commit();

}

catch (Exception ex)

{

db.Rollback();

throw ex;

}

}

#endregion ---更新资料 End ---

#region ---读取资料 Begin ---

/// <summary>

/// 反射方式读取所有资料行

/// </summary>

/// <returns></returns>

public static ArticleTypeEntityList GetArticleTypeEntityListForAllA()

{

ArticleTypeEntityList tagList = new ArticleTypeEntityList();

SimpleFetchDataMethod.DB = ServerConfig.GetDB();

try

{

//通过所有标记来实现获取所有资料行信息并自动填充

SimpleFetchDataMethod.Fetch<ArticleTypeEntityList, ArticleTypeEntity>(tagList, "All");

}

catch

{

return tagList;

}

return tagList;

}

/// <summary>

/// 手动方式读取所有资料行

/// </summary>

/// <returns></returns>

public static ArticleTypeEntityList GetArticleTypeEntityListForAllB()

{

ArticleTypeEntityList tagList = new ArticleTypeEntityList();

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

IDatabase db = ServerConfig.GetDB();

SimpleDataTable sdt = db.ExcuteSimpleDataTable(proc);

foreach (SimpleDataRow dr in sdt.Rows)

{

ArticleTypeEntity tag = new ArticleTypeEntity();

tag.ID = dr.GetInt32("ID").Value;

tag.TypeName = dr.GetString("Name");

tag.ParentID = dr.GetInt32("ParentID").Value;

tag.Seq = dr.GetInt32("Seq");

tag.Effectiveness = dr.GetBoolean("Effectiveness").Value;

tagList.Add(tag);

}

return tagList;

}

#endregion ---读取资料 End ---

}

[Serializable]

public class ArticleTypeEntityList : List<ArticleTypeEntity>

{

}