|  |  |  |  |
| --- | --- | --- | --- |
| 文件编号 |  | 作　　者 | Trimble |
| 文档版本 | V1.0 | 最后修改日期 | 2014-07-23 |

Moon.ORM使用及规范

版本：V1.0

编写人员：Trimble

编写日期：2014-07-23

**修订历史记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **撰叙** |
| 2014年07月23日 | V1.0 | 新建 | Trimble |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目 录**

[1. 概述 3](#_Toc393977448)

[1.1. Moon.Orm的追求方向： 3](#_Toc393977449)

[2. Moon.Orm使用说明 3](#_Toc393977450)

[2.1. 使用配置 3](#_Toc393977451)

[2.1.1. Dll引用 3](#_Toc393977452)

[2.1.2. Web.config配置 3](#_Toc393977453)

[2.2. 使用说明 4](#_Toc393977454)

[2.2.1. Entity生成器 4](#_Toc393977455)

[2.2.1.1 表结构 4](#_Toc393977461)

[2.2.1.2 生成器操作 4](#_Toc393977462)

[2.2.1.3 生成代码 6](#_Toc393977463)

[2.2.2. Create 9](#_Toc393977464)

[2.2.3. Retrieve 10](#_Toc393977465)

[4.2.3.1 标准查询 10](#_Toc393977473)

[4.2.3.2 嵌套查询 10](#_Toc393977474)

[4.2.3.3 分组查询 11](#_Toc393977475)

[4.2.3.4 连接查询 12](#_Toc393977476)

[4.2.3.5 Union查询 13](#_Toc393977477)

[2.2.4. Update 13](#_Toc393977478)

[2.2.5. Delete 14](#_Toc393977479)

[2.2.6. Transaction 14](#_Toc393977480)

[2.2.7. Debug 15](#_Toc393977481)

[2.2.8. Cache 15](#_Toc393977482)

[2.2.9. SQL 15](#_Toc393977483)

[2.2.10. Procedure 16](#_Toc393977484)

[3. 性能 17](#_Toc393977485)

[3.1. 压力测试 17](#_Toc393977486)

[3.2. 性能 17](#_Toc393977487)

[4. 不足 18](#_Toc393977488)

[4.1. 针对MS SQL SELECT部分无WITH(NOLOCK) 关键字 18](#_Toc393977489)

[4.2. 参数部分不够严谨 18](#_Toc393977490)

[4.3. 代码注释较少 18](#_Toc393977491)

[4.4. 帮助文档不够全面 18](#_Toc393977492)

# 概述

Moon ORM 基于.NET框架下的ORM 以项目实战而生，无Linq，贴近SQL。

## Moon.Orm的追求方向：

* + 1. 高性能；
    2. 易用性（配置简单、代码生成器辅助）；
    3. 多数据库多数据源支持；
    4. .NET 2.0 原生支持；
    5. 开源

# Moon.Orm使用说明

## 使用配置

## Dll引用

在项目中引用Moon.Orm.dll

## Web.config配置

<connectionStrings>

<add name="DefaultConnection"

connectionString="Server=(local);Database=DataBase\_Products;uid=sa;Password=sa;"

providerName="Moon.Orm,Moon.Orm.SqlServer" />

</connectionStrings>

**说明：**

1. DefaultConnection：Moon.Orm 默认使用的数据库

例：

using (var db = Db.CreateDefaultDb()){

}

多数据库并存（以Oracle为例）：

<add name=" OracleConnection"

connectionString="链接字符串"

providerName="Moon.Orm,Moon.Orm.Oracle" />

using (var db = Db.CreateDbByConfigName("OracleConnection")){

}

connectionString：链接字符串。就是ADO.NET的链接字符串

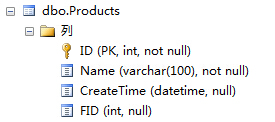
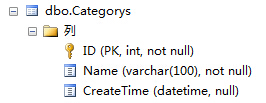
1. providerName：使用的数据库（源）驱动：驱动的dll名称 + 类全名

## 使用说明

## Entity生成器



### 表结构



### 生成器操作



说明：

项目名称：可选、可新建新的名称

数据库类型：MS SQL、MySQL、SQLlIE或其它

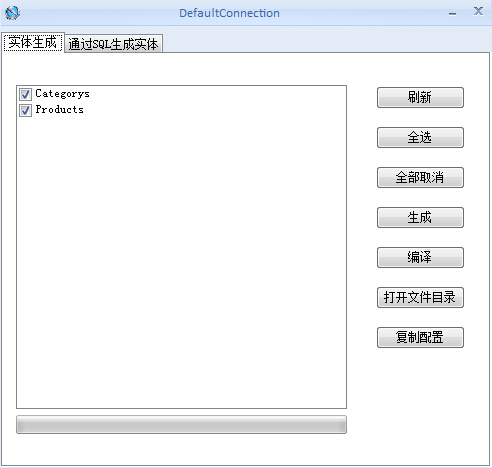
链接字符串：数据库链接字符串

生成路径：生成出的cs文件保存路径

文件选项：可放在一个cs文件 或 每个实体单独一个cs文件

测试链接：顾名思义

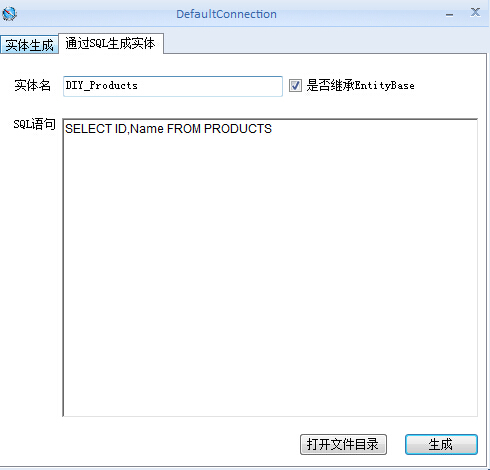
登录：顾名思义



说明：

生成：选择你要生成的表对象后会生成对应的cs文件

编译 ：可把生成的cs文件编译成dll文件



说明：

“通过SQL生成实体”标签：可自定义需要生成的文件

### 生成代码

#### Categorys.cs文件:

[Table("[Categorys]", DbType.SqlServer)]

public class CategorysSet : MQLBase

{

public static new MQLBase Select(params FieldBase[] fields)

{

return MQLBase.Select(DbType.SqlServer,"[Categorys]",fields);

}

public static new MQLBase SelectAll()

{

return MQLBase.SelectAll(DbType.SqlServer,"[Categorys]");

}

public static readonly FieldBase ID = new FieldBase(DbType.SqlServer, "[Categorys]", FieldType.OnlyPrimaryKey, "[ID]");

public static readonly FieldBase Name = new FieldBase(DbType.SqlServer, "[Categorys]", FieldType.Common, "[Name]");

public static readonly FieldBase CreateTime = new FieldBase(DbType.SqlServer, "[Categorys]", FieldType.Common, "[CreateTime]");

}

[Table("[Categorys]", DbType.SqlServer)]

[TablesPrimaryKey(PrimaryKeyType.AutoIncrease, typeof(Int32), "ID")]

public class Categorys : EntityBase

{

/// <summary>

/// 主键ID

/// </summary>

public Int32 ID

{

get { return GetPropertyValue<Int32>("ID"); }

set { SetPropertyValue("ID", value); }

}

/// <summary>

/// 品名

/// </summary>

public String Name

{

get { return GetPropertyValue<String>("Name"); }

set { SetPropertyValue("Name", value); }

}

/// <summary>

/// 创建时间

/// </summary>

public DateTime? CreateTime

{

get { return GetPropertyValue<DateTime?>("CreateTime"); }

set { SetPropertyValue("CreateTime", value); }

}

}

#### Products.cs文件:

[Table("[Products]", DbType.SqlServer)]

public class ProductsSet : MQLBase

{

public static new MQLBase Select(params FieldBase[] fields)

{

return MQLBase.Select(DbType.SqlServer,"[Products]",fields);

}

public static new MQLBase SelectAll()

{

return MQLBase.SelectAll(DbType.SqlServer,"[Products]");

}

public static readonly FieldBase ID = new FieldBase(DbType.SqlServer, "[Products]", FieldType.OnlyPrimaryKey, "[ID]");

public static readonly FieldBase Name = new FieldBase(DbType.SqlServer, "[Products]", FieldType.Common, "[Name]");

public static readonly FieldBase CreateTime = new FieldBase(DbType.SqlServer, "[Products]", FieldType.Common, "[CreateTime]");

public static readonly FieldBase FID = new FieldBase(DbType.SqlServer, "[Products]", FieldType.Common, "[FID]");

}

[Table("[Products]", DbType.SqlServer)]

[TablesPrimaryKey(PrimaryKeyType.AutoIncrease, typeof(Int32), "ID")]

public class Products : EntityBase

{

/// <summary>

/// 主键ID

/// </summary>

public Int32 ID

{

get { return GetPropertyValue<Int32>("ID"); }

set { SetPropertyValue("ID", value); }

}

/// <summary>

/// 名字

/// </summary>

public String Name

{

get { return GetPropertyValue<String>("Name"); }

set { SetPropertyValue("Name", value); }

}

/// <summary>

/// 创建时间

/// </summary>

public DateTime? CreateTime

{

get { return GetPropertyValue<DateTime?>("CreateTime"); }

set { SetPropertyValue("CreateTime", value); }

}

/// <summary>

/// 外键ID（指向商品品类表主键ID）

/// </summary>

public Int32? FID

{

get { return GetPropertyValue<Int32?>("FID"); }

set { SetPropertyValue("FID", value); }

}

}

#### DIY\_Products.cs文件:

public class DIY\_Products : EntityBase

{

public Int32 ID

{

get { return GetPropertyValue<Int32>("ID"); }

set { SetPropertyValue("ID", value); }

}

public String Name

{

get { return GetPropertyValue<String>("Name"); }

set { SetPropertyValue("Name", value); }

}

}

## Create

public object Create(Products obj)

{

using (var db = Db.CreateDefaultDb())

{

return db.Add(obj); ;

}

}

说明：返回类型是表的主键ID，若表的类型是int 自增字段，就返回此字段值，若是GUID类型为字符类型

## Retrieve



### 标准查询

public Products GetModel(int ID)

{

using (var db = Db.CreateDefaultDb())

{

var mql = ProductsSet.SelectAll()

.Where(ProductsSet.ID.Equal(ID));

return db.GetEntity<Products>(mql);

}

}

说明：mql 是具体的查询对象，在此函数中包含 根据主键ID查询表中的所有字段，还有针对指定列的查询 Select(ProductsSet.Name, ProductsSet.ID)

public List<Products> GetList()

{

using (var db = Db.CreateDefaultDb())

{

var mql = ProductsSet.SelectAll()

.Where(ProductsSet.ID.SmallerThan(1000));

return db.GetEntities<Products>(mql);

}

}

说明：db.GetEntites泛型复数返回List对象

### 嵌套查询

/// <summary>

/// 嵌套查询

/// </summary>

/// <returns></returns>

public List<Products> NestedQueries()

{

using (var db = Db.CreateDefaultDb())

{

var mql = ProductsSet.SelectAll().Where(

ProductsSet.FID.In(

CategorysSet.Select(CategorysSet.ID).Where(

CategorysSet.Name.Equal("数码").Or(CategorysSet.Name.Equal("电子设备"))

)

)

).Top(20);

return db.GetEntities<Products>(mql);

}

}

说明：关键字Equal 等价于 SQL “=” 号；Top=TOP； IN=IN

SQL:

SELECT TOP 20 \*

FROM dbo.Products

WHERE FID IN ( SELECT ID

FROM dbo.Categorys

WHERE ( Name = '数码' OR Name = '电子设备')

);

### 分组查询

/// <summary>

/// 分组查询

/// </summary>

/// <returns></returns>

public List<Products> GroupBy()

{

using (var db = Db.CreateDefaultDb())

{

var mql = ProductsSet.Select(ProductsSet.ID.Sum().AS("IDSUM"), ProductsSet.Name).Where(

ProductsSet.FID.BiggerThanOrEqual(10)

).GroupBy(ProductsSet.Name).Having(

ProductsSet.ID.Sum().BiggerThan(300)

);

return db.GetEntities<Products>(mql);

}

}

说明：关键字BiggerThanOrEqual（大于等于）；BiggerThan（大于）

SQL:

SELECT SUM(ID) AS IDSUM ,Name

FROM dbo.Products

WHERE FID >= 10

GROUP BY Name

HAVING SUM(ID) > 300

### 连接查询

/// <summary>

/// 链接查询

/// </summary>

/// <returns></returns>

public List<Products> InnerJoin()

{

using (var db = Db.CreateDefaultDb())

{

var mql = ProductsSet.Select(ProductsSet.ID, ProductsSet.Name).

InnerJoin(CategorysSet.Select(CategorysSet.Name)).

ON(ProductsSet.FID.Equal(CategorysSet.ID)).

Where(ProductsSet.Name.Contains("笔记本"));

return db.GetEntities<Products>(mql);

}

}

说明：关键字Contains (Like)

SQL:

SELECT dbo.Products.ID ,

dbo.Products.Name ,

dbo.Categorys.Name

FROM dbo.Products

INNER JOIN dbo.Categorys

ON dbo.Products.FID = dbo.Categorys.ID

WHERE ( dbo.Products.Name LIKE '%笔记本%' );

### Union查询

/// <summary>

/// Union查询

/// </summary>

/// <returns></returns>

public List<Products> Union()

{

using (var db = Db.CreateDefaultDb())

{

var mql = ProductsSet.SelectAll().UnionAll(CategorysSet.Select(CategorysSet.ID, CategorysSet.Name, CategorysSet.CreateTime, CategorysSet.ID.AS("FID")));

return db.GetEntities<Products>(mql);

}

}

说明：关键字UnionAll (Union ALL)

SQL:

SELECT \* FROM dbo.Products

UNION ALL

SELECT \*,ID AS FID FROM dbo.Categorys

## Update

public bool Update(Products obj)

{

using (var db = Db.CreateDefaultDb())

{

Products c = new Products();

c.Name = obj.Name;

c.FID = obj.FID;

c.CreateTime = obj.CreateTime;

c.WhereExpression = ProductsSet.ID.Equal(obj.ID);

return (db.Update(c)>0);

}

}

说明：c.WhereExpression 条件表达式

## Delete

public bool Delete(int ID)

{

using (var db = Db.CreateDefaultDb())

{

var result = db.Remove(ProductsSet.ID.Equal(ID));

return (result > 0);

}

}

说明：ProductsSet.ID.Equal(ID) 条件表达式

## Transaction

public int Create(List<Products> list)

{

using (var db = Db.CreateDefaultDb())

{

int result = 0;

int id = 0;

db.TransactionEnabled = true;

foreach (var obj in list)

{

var o = db.Add(obj);

if (int.TryParse(o.ToString(), out id)) result++;

}

return result;

}

}

说明：db.TransactionEnabled = true;开启事务即可

## Debug

public List<Products> GetList()

{

using (var db = Db.CreateDefaultDb())

{

db.DebugEnabled = true;

var mql = ProductsSet.SelectAll().Where(ProductsSet.ID.SmallerThan(1000));

string sql = mql.ToDebugSQL();

return db.GetEntities<Products>(mql);

}

}

说明：db.DebugEnabled = true;开启调试

string sql = mql.ToDebugSQL();可以获取具体的sql

## Cache

db.StartCache(int cacheTime);

说明：针对查询操作，查询后立即缓存到当前内存中

## SQL

public List<Products> GetList(int pageIndex, int pageSize, out int recordCount)

{

recordCount = 0;

using (var db = Db.CreateDefaultDb())

{

db.DebugEnabled = true;

string sql = @"SELECT Name ,

CreateTime ,

FID

FROM ( SELECT ROW\_NUMBER() OVER ( ORDER BY ID DESC ) AS ROWNUMBER ,

Name ,

CreateTime ,

FID

FROM dbo.Products WITH ( NOLOCK )

WHERE ( [FID] < @ )

) AS Temp

WHERE ROWNUMBER BETWEEN @ AND @;";

var list = db.ExecuteSqlToOwnList<Products>(sql, 10, ((pageIndex - 1) \* pageSize + 1),(pageIndex \* pageSize));

sql = @"SELECT COUNT(ID) FROM dbo.Products WITH ( NOLOCK );";

object obj = db.ExecuteSqlToScalar(sql);

recordCount = (int)obj;

return list;

}

说明：db.ExecuteSqlToOwnList() 是直接执行SQL;

有两个参数，sql，params :sql中的参数以单个@符号作为占位符，params要保证顺序

## Procedure

public int Update(Products obj)

{

using (var db = Db.CreateDefaultDb())

{

string procName = "proc\_products\_create";

System.Data.Common.DbParameter[] parameter = {};

parameter[0].DbType = System.Data.DbType.Int32;

parameter[0].Direction = System.Data.ParameterDirection.Input;

parameter[0].ParameterName="@ID";

parameter[0].Value = 1;

return db.ExecuteProWithNonQuery(procName, parameter);

}

}

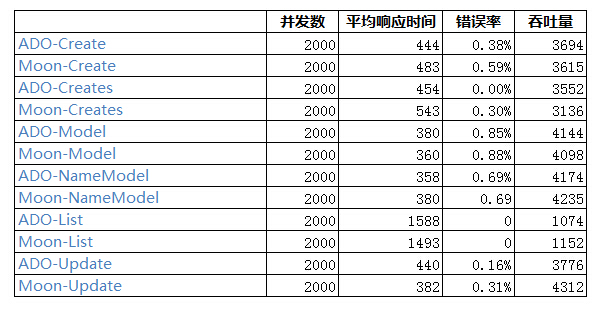
说明：db. ExecuteProWithNonQuery(string procName, params DbParameter[] parameters) 是直接执行SQL;

procName:存储过程名称

parameters：参数集合

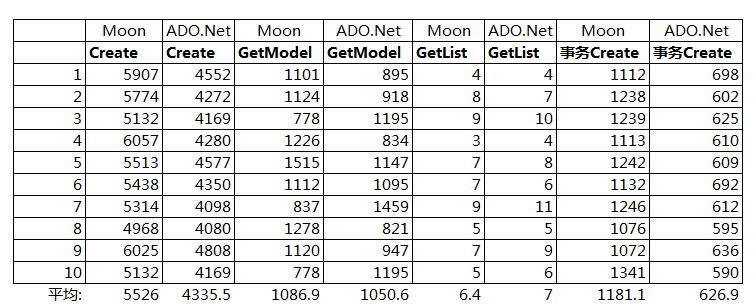
# 性能

## 压力测试



说明：2000并发用户

## 性能



说明：同时请求10000条数据