ORM框架使用说明

**(V0.1.0)**

****

2018年5月

**目录**

1. 框架结构简介 3

1.1. 框架的基本约定 4

2. 数据库连接操作 5

2.1. 框架连接Oracle方式 5

2.2. 配置oracle连接 5

3. 使用框架进行数据操作说明 6

3.1. IDataSetAccessor接口使用说明 6

3.1.1. 查询单个实体 6

3.1.2. 查询一组实例 7

3.1.3. 插入一条实例 8

3.1.4. 更新一个实例 8

3.1.5. 删除一条实例 8

3.1.6. 查询实体数量 8

3.1.7. 查询制定表格中单个列的数据 9

3.1.8. 向指定表格的列中插入数据 9

3.1.9. 向指定表格的列中更新数据 9

3.1.10. 在指定表格中删除满足条件的数据 10

3.2. IDataAccessor接口使用说明 10

3.3. 多表联查使用说明 11

# 框架结构简介

ORM框架基本结构如下：



框架的核心是IDataSetAccessor和IDataAccessor，用户通过工厂方法获接口的默认实例并直接对数据进行CURD操作，框架可以自动完成数据库连接打开和关闭操作。二者的主要区别在于

IDataSetAccessor主要使用实体类(PO)信息拼接出SQL语句，在使用的时候只需要提供操作条件和条件对应的参数即可，最大化屏蔽数据库操作细节，适合对于PO对象直接的增删改查操作和对制定表格的增删改查操作。而IDataAccessor接收完整sql语句，并需要提供sql语句中的参数占位符与参数对的集(Dictionary)。该接口使用更加灵活，适合复杂数据库操作场景，用户对于sql操作十分熟悉，可以直接执行用户编写的sql语句。

## 框架的基本约定

框架编写的约定如下：

* 框架只面对oracle数据库设计
* 框架主要面向PO对象的CRUD操作
* 框架不提供多表关联查询，用户需要在DAO层解决关联查询（参考样例）
* 框架不提供事务操作，用户可根据需要，对框架进行扩展
* PO和数据库列名默认映射关系是数据库列名采用PO属性的全小写
* PO和数据库表明默认映射关系是数据库表名采用PO名称

框架提供的数据库类型转换包括：

|  |  |  |
| --- | --- | --- |
| .net类型 | oracle数据类型 | 说明 |
| Short | number |  |
| Int | number |  |
| Long | number |  |
| Float | number |  |
| Double | number |  |
| Char | varchar2 |  |
| String | varchar2 |  |
| bool | int | oracle没有bool类型，框架默认采用0表示false，1表示true |
| Decimal | Decimal | 待测试 |
| DateTime | DateTime | 待测试 |

# 数据库连接操作

## 框架连接Oracle方式

目前了解C#中连接Oracle数据库的方法有3种，分布是微软的System.Data.OracleClient，Oracle的Oracle.DataAccess.Client和Oracle的Oracle.ManagedDataAccess.dll(最优)，本框架采用第三种方式操作oracle数据库。

操作连接方法在OracleConnectionFactory列中，该类提供如下方法：

**public static OracleConnection OpenConn()**

每次操作的时候，都会通过连接工厂获取一个oracle数据库连接。

**public static void CloseConn(OracleConnection conn)**

工厂类提供了关闭连接的方法，框架每次完成操作后，会自动释放连接。

## 配置oracle连接

配置oracle数据库连接在OracleConnectionFactory类中，目前采用硬编码方式，后续可以采用读取配置方式。

连接配置样例如下：

ConnectionString ="DataSource=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=**10.2.32.189**)(PORT=**1521**))(CONNECT\_DATA=(SERVICE\_NAME=**oracle.db.server**)));Persist Security Info=True;User ID=**crscd**;Password=**crscd123@**;"

使用中只需要根据实际情况配置上述字符串中红色标记的部分即可。

# 使用框架进行数据操作说明

## IDataSetAccessor接口使用说明

IDataSetAccessor接口中定义的方法如下：

// 查询对应实体，返回单条记录

T Select<T>(string condition, params object[] paramList) where T : class;

// 查询实体列表

List<T> SelectList<T>(string condition, params object[] paramList) where T : class;

// 插入实体

bool Insert<T>(T t) where T : class;

// 更新实体

bool Update<T>(T t, string condition, params object[] paramList) where T : class;

// 删除实体

bool Del<T>(string condition, params object[] paramList) where T : class;

// 查询实体数量

long GetCount<T>(string condition, params object[] paramList) where T : class;

// 查询单列数据

List<T> GetColumnList<T>(string table, string column, string condition, params object[] paramList);

// 向指定表格插入指定列数据

bool InsertColumnData(string table, Dictionary<string, object> columnDatMap);

// 向指定表格更新指定列的数据

bool UpdateColumnData(string table, Dictionary<string, object> columnDatMap, string condition, params object[] paramList);

// 在指定表格中删除满足条件的数据

bool DelData(string table, string condition, params object[] paramList);

### 查询单个实体

查询单个实体需要指定查询条件和条件中的参数，首先在数据库中建立一个表格person，其中包含三列，id（number），name(varchar2)，note(varchar2)，然后定义一个实体类(PO)Person：

public class Person

{

public int Id { set; get; }

public string Name { set; get; }

public string Note { set; get; }

}

在数据库中存在一个id为1的数据，则从数据库中查询出来person的流程如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

var condition = "id=?";

var person = dataSet.Select<Person>(condition,1);

用户只需要定义查询条件，并在查询的时候按照顺序制定参数值就可以完成查询过程，将符合条件的一条数据查出。

条件的定义约定为:

**[列名(默认为属性的小写)]=？ AND [[列名(默认为属性的小写)]=？]… [约束条件]**

查询语句中的参数采用可变参数方式，当有多个参数的时候，需要按照条件中定义的顺序，将参数写入，参数和？是一一对应的。

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

dataSet.Select<Person>(condition,parm1,parm2,…);

**注意：**如果数据库中不存在符合条件的数据，则会返回null。

当存在多个符合条件的实例时，框架会默认返回从数据库查出的第一行数据。

### 查询一组实例

查询一组实例的操作与查询一个实例的操作类似，流程如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

var condition = "note like '%is%'";

var person = dataSet.Select<Person>(condition);

上述操作会将符合条件的一组信息都查询出来并存放到List中返回。

注意：如果数据库中不存在符合条件的数据，则会返回一个空的List。

### 插入一条实例

插入一条实例的方法如下：

首先定义一个实例对象

var person= new Person{Id=232,Name=”xiaoming”,Note=”test”}

然后调用插入方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

bool flag = dataSet.Insert(person);

返回结果是一个bool类型的结果。

### 更新一个实例

更新实例的方法如下：

首先从数据库查询出一个实例对象

var person= dataSet.Select<Person>(“id=?”,1);

然后对对象进行一定修改：

Person.setNote(“modify note”);

最后调用更新方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

bool flag = dataSet.Update<Person> (person,”id=?”,person.Id);

返回结果是一个bool类型的结果。

### 删除一条实例

删除实例的过程如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

bool flag = dataSet. Del<Person> (“id=?”,1);

返回结果是一个bool类型的结果。

### 查询实体数量

查询符合条件的查询方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

var condition = "note like '%is%'";

var count = dataSet. GetCount<Person>(condition);

### 查询制定表格中单个列的数据

查询单列数据方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

var condition = "id=?";

var noteList = dataSet. GetColumnList<string>(condition,1);

通过该方法可以从数据库中直接查询某个表中的某列数据，返回一个指定类型的列表。

### 向指定表格的列中插入数据

向指定表格插入列的主要用于操作一些不需要构造与数据库表格对应的实体类的场景，例如两个表的关系表等中间表场景。

person\_school\_relation是一张人员与学校对应关系的表格，表中包含两列数据：person\_id和school\_id，向指定表格的指定列中插入数据的方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

Var table = “person\_school\_relation”;

Dictionary <string, object> parms = new Dictionary()；

parms.Add(“person\_id”,1);

parms.Add(“school\_id”,3);

var result = dataSet. InsertColumnData(string table, Dictionary<string, object> columnDatMap);

通过上述操作，可以将id为1的人员与id为3的学校绑定关系存储到关系表中。

### 向指定表格的列中更新数据

向指定表格更新列的主要用于操作一些不需要构造与数据库表格对应的实体类的场景，例如两个表的关系表等中间表场景。

向指定表格的指定列中更新数据的方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

var table = “person\_school\_relation”;

var condition = "person\_id =?";

Dictionary <string, object> parms = new Dictionary()；

parms.Add(“school\_id”,3);

var noteList = dataSet. UpdateColumnData (table, parms, condition,1);

### 在指定表格中删除满足条件的数据

在指定表格删除数据主要用于操作一些不需要构造与数据库表格对应的实体类的场景，例如两个表的关系表等中间表场景。

在指定表格删除满足条件的数据的方法如下：

IDataSetAccessor dataSet = OrmEntryFactory.GetDataSetAccessor();

var table = “person\_school\_relation”;

var condition = " person\_id =?";

var noteList = dataSet. DelData(table, condition, 1);

## IDataAccessor接口使用说明

IDataAccessor接口用于更灵活的数据库操作，支持用户直接编写sql语句并进行执行，接口中定义的方法如下：

// 查询对应实体，返回单条记录

T queryEntity<T>(string sql, Dictionary<string, object> parms) where T : class;

// 查询对应实体，返回多条记录

List<T> queryEntityList<T>(string sql, Dictionary<string, object> parms) where T : class;

// 查询对应的数据，返回单条记录（列名 => 数据）

Dictionary<string, object> queryMap(string sql, Type type, Dictionary<string, object> parms);

// 查询对应的数据，返回多条记录（列名 => 数据）

List<Dictionary<string, object>> queryMapList(string sql, Type type, Dictionary<string, object> parms);

// 查询单列数据，返回一条单列数据（列名 => 数据）

T queryColumn<T>(string sql, Dictionary<string, object> parms);

// 查询单列数据，返回多条单列数据（列名 => 数据）

List<T> queryColumnList<T>(string sql, Dictionary<string, object> parms);

// 查询记录条数，返回总记录数

long queryCount(string sql, Dictionary<string, object> parms);

// 执行更新操作（包括：update、insert、delete），返回所更新的记录数

int update(string sql, Dictionary<string, object> parms);

其中sql语句指的是完整的sql语句，例如：

*Var sql = SELECT id,name,note FROM person WHERE id=:id;*

其中:id是约定的参数占位符，当传入参数的时候，作为key，针对上述语句，构造的参数为：

Dictionary<string, object> parms = new Dictionary<string, object>(){{“:id”,1}};

具备sql语句和对应参数后，可以直接执行查询语句如下：

IDataAccessor dataAccessor = OrmEntryFactory.GetDataAccessor();

var person = dataAccessor. queryEntity<Person>(sql,parms);

## 多表联查使用说明

多表联查的功能主要是指在类的内部定义了其他自定义类型的数据，例如定义一个学校的类型：

public class School{

public int Id { set; get; }

public string Name { set; get; }

public string Address { set; get; }

}

定义一个业务实体类如下:

public class PersonDo：Person

{

public List<School> Schools{set;get;}

}

这样如果要获取上述实体类，需要进行多表查询，才能返回对应的业务实体。

这里以查询和修改为例，进行说明，首先定义服务接口：

public interface IPersionService

{

PersonDo GetPerson(int id);

void SaveSchools(PersonDo person);

}

为了查询出业务实例PersonDo，需要进行组合操作，即先查询出Person，然后利用Person的id查询关联的学校信息。

假设为了避免Person和School的耦合，两个表采用中间表person\_school\_map这张中间表进行关系维护，该表中包含两列，分别为person\_id和school\_id。

首先定义一个通过person\_id在中间表中查询对应的学校id列表：

private List<int> GetSchoolIdsByPersionId(int id)

{

string table = "person\_school\_map";

string column = "school\_id";

string condition = "person\_id=?";

List<int> schoolIdList = dataSet.GetColumnList<int>(table, column, condition, id);

return schoolIdList ?? new List<int>();

}

然后定义通过person\_id获取对应业务实例PersonDo的函数如下：

public PersonDo GetPerson(int id)

{

var condition = "id=?";

Person person = dataSet.Select<Person>(condition, id);

if (person == null)

{

return null;

}

PersonDo p = new PersonDo(person);

p.Schools = GetSchoolsByPersionId(p.Id);

return p;

}

通过上述方法可以查询业务实体PersonDo。

保存修改信息需要考虑封装Schools属性的变化，通过业务代码自动将修改映射到数据库中，函数定义如下：

public void SaveSchools(PersonDo person)

{

List<int> oldSchoolIds = GetSchoolIdsByPersionId(person.Id);

List<int> newSchoolIds = new List<int>();

foreach (var school in person.Schools)

{

newSchoolIds.Add(school.Id);

}

// 添加新关系

foreach (var id in newSchoolIds)

{

if (!oldSchoolIds.Contains(id))

{

AddSchoolIdForPerson(person.Id, id);

}

}

// 删除去除的关系

foreach (var id in oldSchoolIds)

{

if (!newSchoolIds.Contains(id))

{

DelSchoolIdForPerson(person.Id, id);

}

}

}

其中两个辅助函数为：

private void DelSchoolIdForPerson(int pid, int sid)

{

string table = "person\_school\_map";

string condition = "person\_id=? AND school\_id=?";

dataSet.DelData(table, condition, pid, sid);

}

private void AddSchoolIdForPerson(int pid, int sid)

{

string table = "person\_school\_map";

Dictionary<string, object> parms = new Dictionary<string, object> {{"person\_id", pid}, {"school\_id", sid}};

dataSet.InsertColumnData(table, parms);

}