# Buffalo项目

*帮助文档编写中...*

*本文介绍了Buffalo的使用方法，及一些常见的问题及处理方法FAQ。FAQ放在文章后面。*

# 简介

## Buffalo项目是什么？

一个C#编写的轻量高效的ORM数据库操作组件。

* 配置方便 *(配置工具完美结合Visual Studio)*
* 操作简单，快速上手 （几步操作即可）
* 支持多数据库 (MySQL,DB2,PostgreSQL,SQLite,Oracle9,MS SQL)

其目的是为了提高工作效率，避免重复地编写似曾相识的代码。屏蔽底层不同数据库的SQL语句及语法差异，一切面向对象。

它支持以下两种最常见也是最流行的使用方式：

1. 使用Buffalo从现有数据库快速地生成相关代码，并完成纯粹的面向对象的数据库操作类；
2. 或者，从现有的已经设计好的类（实体类Class.cs），配置生成数据库库表。

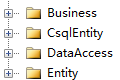
## 这货跟代码生成工具有什么区别？

呃嗯~~~~~~这个~..........这货本身也通过数据库表或实体类这些基础信息，来生成完整的相关的代码，所以也是个代码生成工具。 但是 ，它跟你“印象中”的 SQLHelper 和 DataAccess.dll不一样 ：

* 首先，这货的配置工具完结集成到Visual Studio里成为VS的一部分。从“野工具”回到宇宙中最强大的IDE中。
* 其次，这货完全面向对象，使用者可以根据业务逻辑先完成相关实体类的设计，即可直接配置生成对应的数据库表，省时省力。让你有种“一刀在手，天下我有”的错觉。
* 这货的核心是，通过IL级别的快速反射和映射关系，来生成最终的SQL语句；来给实体对象赋值的。简单，粗暴，有效。

## Buffalo基本结构

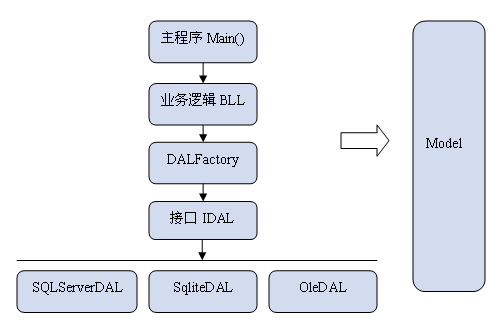
项目分为三层，如下图示：



项目中的Business为业务层，DataAccess为数据层，CsqlEntity为查询辅助类，Entity为实体类，跟表一一对应。

它是如何应对不同数据库？

其数据层使用工厂模式加反射，通过去实现各个数据库的接口，来实现操作不同的数据库。



# 使用

当然了，请先下载Buffalo项目。

使用SVN检出svn://svn.code.sf.net/p/buffalobro/svn/

## （安装）添加工具到Visual Studio

Buffalo的配置工具，是一个VS插件，所以它可以完美地融合到VS里，成为VS的一部分，扩展VS的功能。插件的安装过程非常的简单。

执行目录文件里的安装小工具 \Tools\AddInSetup\AddInSetup.exe，如下图



选择对应的您本机上已经安装的VS版本，点击安装，即可完成安装。

*说明：*

*其实这个“安装插件”的操作，所做的事情是，在 %HomePath%\Documents(xp:Documents and Settings)\Visual Studio {2012}\AddIns 下添加一个名为Buffalo.DBTools.AddIn的文件，该文件描述了插件的信息及存放位置等信息。*

## （卸载）移除插件

如果需要卸载插件，请再次运行 \Tools\AddInSetup\AddInSetup.exe ，卸载即可。

*即删除 %HomePath%\Documents(xp:Documents and Settings)\Visual Studio {2012}\AddIns 下的文件 Buffalo.DBTools.AddIn*

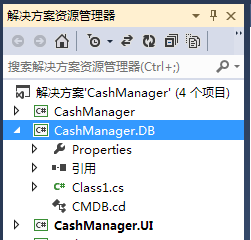
## 快速使用

在Visual Studio 中新建一个类库项目，此类库将作为数据库操作组件（即分层结构里的数据库访问层）。

*下面的例子CashManager项目，演示了从现有数据库，通过配置工具如何创建基于Buffalo框架的数据库操作类*

### 创建一个新类库项目

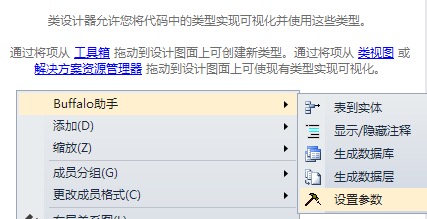
给解决方案创建一个新类库项目 CashManager.DB ，并在 CashManager.DB 下添加新项，一个类图，起名为 CMDB.cd，如下图



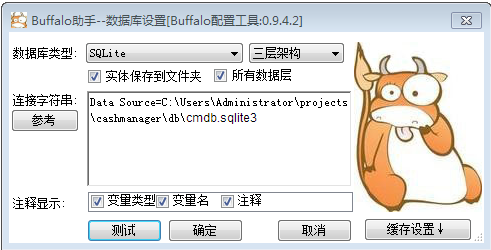
### 设置数据库连接参数。

打开新建的类图 CMDB.cd，在类图的任意一个空白 右击，在右键菜单里选择

Buffalo助手 -> 设置参数，如下图



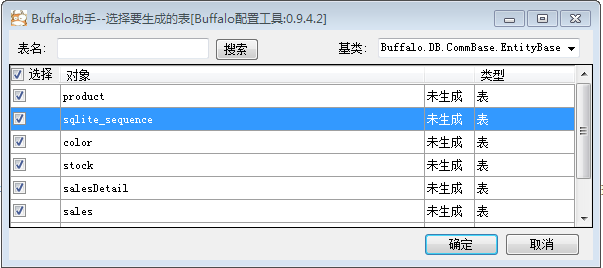
在弹出的对话框里，设置数据库连接信息，如下图示，这里的演示例子里我们使用了 SQLite 数据( 其他数据库类型的操作雷同 )，**设置连接字符串后，点击 测试 连接， 并确保数据库连接成功，能正常访问。**



### 生成数据库表结构的ORM内容

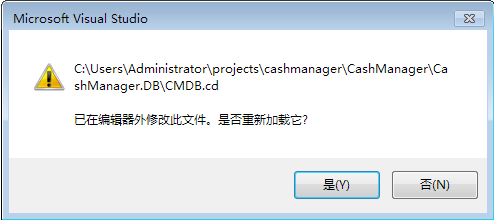
在类图CMDB.cd的任意一个空白 右击，在右键菜单里选择

Buffalo助手 -> 表到实体，如下图

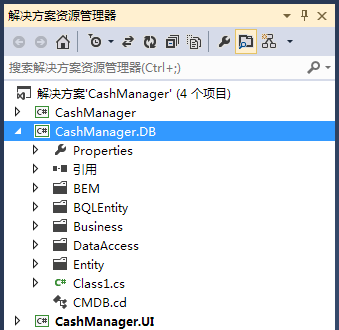


Buffalo助手会把指定的数据库里的库表及视图都读取出来，并显示对应的状态。选择表，并 确定 生成。

此操作结束后，VS需要重新加载类图，如下图示，点击 是 重新加载。

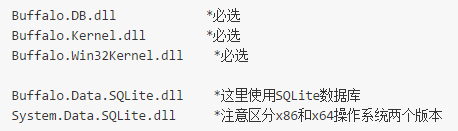


最终结果如下图



### 添加Buffalo DLL引用

在 \Tools\AddInSetup\AddIns 目录下，有.NET各个版本的DLL，选择当前项目对应的.NET版本，将其复制到 CashManager.DB 下，并添加引用。复制的DLL如下：



如果使用MySQL数据库，则下边的两个DLL应该为



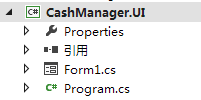
其他类型的数据库依此类推，当然，您也可以复制所有的DLL到项目下，并添加引用。

添加引用后，项目右键->重新生成 CashManager.DB成功。*(只是为了检查有没有错误)*

至此，数据库访问层已经完成。简单否？！接下来，看看怎么使用。

### 操作

在解决方案里新建UI项目，新建Form窗体项目 CashManager.UI *（这里以WINFORM作为例子）*，并 添加引用 Buffalo.DB.dll 和 Buffalo.Data.SQLite.dll



### 初始化数据库

初始化数据库这一步在需要在程序启动前初始化，在项目（Main函数所在，这里是Form窗体项目CashManager.UI）里的 Program.cs，

在Application.Run(new Form1()); 前

添加以下初始化代码，其中的数据库连接字符串，为上面步骤设置的数据库连接字符串。

//Initialize data base infoBuffalo.DB.DataBaseAdapter.DBInfo info = new Buffalo.DB.DataBaseAdapter.DBInfo("CMDB", @"Data Source=C:\Users\Administrator\projects\cashmanager\db\cmdb.sqlite3", "Buffalo.Data.SQLite");info.QueryCache.InitCache(new Buffalo.DB.CacheManager.MemoryAdaper(info), false);info.DataaccessNamespace = new string[] { "DB.DataAccess.Buffalo.Data.SQLite" };Buffalo.DB.DataBaseAdapter.DataAccessLoader.AppendDBInfo(info);CashManager.DB.BQLEntity.CMDB.InitDB();

### 调用

在窗体 Form1.cs 里，添加几个 Button 和一个 DataGridView。

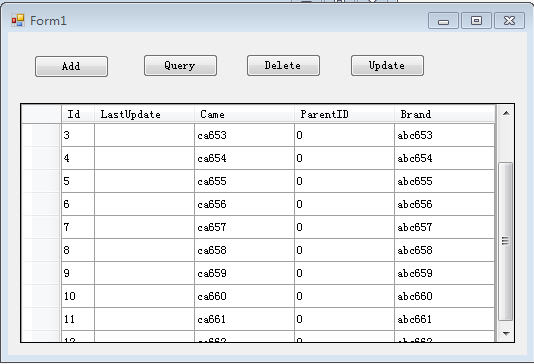
点击Button1 插入记录到库表，Button1 按钮运行以下代码：

private void button1\_Click(object sender, EventArgs e) { int j = DateTime.Now.Millisecond; for (int i = j; i < j + 10; i++) { Categery entity = new Categery(); entity.Brand = "abc" + i; entity.ParentID = 0; entity.Came = "ca" + i; CategeryBusiness bus = new CategeryBusiness(); bus.Insert(entity); } }

Button2 从数据库中查询刚添加的记录，并显示结果。Button2 执行以下代码

private void button4\_Click(object sender, EventArgs e) { ScopeList scope = new ScopeList(); CategeryBusiness bus = new CategeryBusiness(); this.dataGridView1.DataSource = bus.SelectList(scope); }

结果如下图示



（完成）

# 配置连接数据库

Buffalo项目分两个部分，一个是它的配置工具，即VS插件；一个是Buffalo数据库操作组件本身。配置工具用于协助生成代码和配置连接信息。

## 配置工具连接数据库

这里的数据库连接，是指Buffalo的配置工具（VS插件）在需要读取数据库表来生成代码时（或者从已经设计好的类，去数据库生成对应的库表时），需要连接数据库，所以需要给它配置一下。

如何配置，操作同本文节点 [设置数据库连接参数](#_设置数据库连接参数。)

## Buffalo连接数据库

### 数据库连接信息存放位置

数据库连接信息存放形式分两种，一种是把它写在程序的XML配置文件里。另一种是直接写程序的代码里（即在初始化时）。它们的区别是，后一种为了保密数据库连接信息，如果把它写在源代码里，那么编译后，连接信息存在二进制文件里，而不是像XML文件里的明文。这两种形式它们各有优点，使用时各取所需。

### Buffalo内部初始化顺序

在使用Buffalo前，必须对它初始化，初始化时需要数据库的相关信息，比如数据库类型，连接信息等。

在调用静态初始化方法时，Buffalo会查看数据库连接信息是否已经设置，如果已经设置，那么直接初始化。

如果未设置，则默认会去读取程序的配置文件里的appSettings节点，读取节点下的 Buffalo.Config参数值，这个参数值不是直接写有初始化所需要信息，而是保存配置信息的文件路径。Buffalo通过这个路径，去读取配置文件里的初始化信息，再继续初始化。

不管是WebForm还是WinForm，都是在程序启动时，先初始化数据库连接信息即可。只是WebForm和WinForm程序的启动入口代码位置不一样（Program.cs和Global.asax.cs），

所以下面分开说明一下如何配置。

### 需要引用的DLL有哪些

Buffalo.DB.dll (必须)

其他的，看你使用的数据库类型，如里在用SQLite，就添加SQlite.dll,如果使用MySQL,就给添加mydql.dll引用。

最简单的作法是：给项目生成一下，看VS的错误提示，缺少什么，就给它添加对应的dll使用。

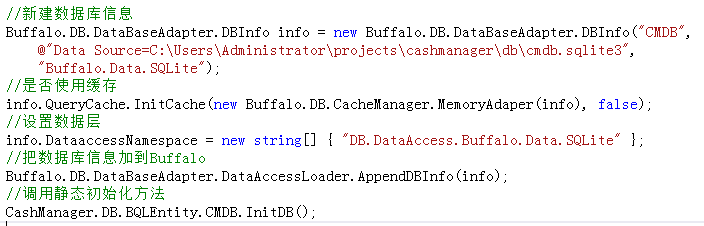
## Win Form调用配置

配置工具生成代码后，会在数据库操作项目下，生成一个对应用的XML配置文件。

比如本文[快速使用章节](#_创建一个新类库项目)举的例子，数据库操作项目是CashManager.DB ，在 CashManager.DB 下用于配置数据库连接信息的类图名为 CMDB.cd，则生成的XML配置文件与它对应 CMDB.xml。它里面保存有数据库类型，及数据库连接信息。

### 连接信息写到源代码里

在 Program.cs 的 main() 函数里，在Application.Run() 前先初始化数据库，添加以下代码：



代码说明：

行1，实例化一个数据库连接信息实体，参数一是数据库名称CMDB（这个名称是在新建类图时写的类图名称CMDB.cd，跟你物理数据库里的名称可以不一样，如果以后需要更改数据库，只需修改连接信息就可以了，只要这里的CMDB不变，你所写的代码都不需要修改），参数二是连接信息字符串（不同的数据库连接信息不一样）。参数三是数据库类型。最简单的方法是从生成的XML文件CMDB.xml里，将对应的内容复制过来，不易写错。

行2，是否使用缓存（看你的项目需要）。Buffalo内部支持全部库表或部分库表缓存，同时也支持外部缓存（连接内存数据库Redis或Memcached)，插件在配置生成时可配置。

行3，指明数据库层使用是的哪个（因Buffalo使用工厂模式来对实现对不同类型的数据库进行操作，如本文前部分所说）

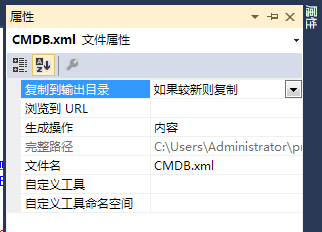
行4，把设置好的数据库连接信息实体，添加到Buffalo里。

行5，调用静态方法初始化。

初始化后，就可以使用了。如果VS错误提示缺少引用，那就添加对应的引用。

### 连接信息存放在XML文件里

1）将Buffalo工具生成的数据库信息XML文件CMDB.xml 复制到你的启动项目下，并设置其属性，复制到输出目录。如下图



2）接着，在程序配置文件 app.config 里的 appSettings节点，添加一个子节点，key=Buffallo.Config , value =CMDB.xml 。说明，key值不可更改，value值为刚复制到项目目录下的配置信息文件CMDB.xml。如下图。（如果这个配置文件CMDB.xml 放在其他地方，那么把它的路径一起写上，这里是在同一目录下，所以只写文件名）



3）在启动项目里的 Program.cs 里的 main 方法， Application.Run() 前先调用静态该当初始化数据库即可。如下

CMDB.InitDB();

Application.Run(Form1());

如有疑问，请看本文的章节 [Buffalo内部初始化顺序](#_Buffalo的初始化顺序)

## Web Form调用配置

### 连接信息写到源代码里

跟Win Form的配置一样，只不过在WEB项目下，代码写在 Global.asax.cs里的 Application\_Start()方法里，初始化代码还是一样的，详情请看本文 Win Form配置章节 [连接信息写到源代码里](#_连接信息写到源代码里)。

### 连接信息存放在XML文件里

跟Win Form的配置一样，只不过在WEB项目下，程序配置文件名为Web.config。详情请看本文的 Win Form 配置章节 [连接信息存放在XML文件里](#_连接信息存放在XML文件里)。

# 使用配置工具（VS插件）

### 从已有库表生成类

请看本文章节 [快速使用](#_快速使用)。

### 从类生成库表

先设计好实体类，从实体类生成对应的数据库表。

操作方法：

同样需要给项目新建一个类图，并把设计好的实体类添加到这个类图里（选中文件直接拖进来）。

1. 先设置数据库连接信息。

在类图的任意一个空白的地方，右键，在弹出的右键菜单中，Buffalo助手 -> 设置参数，设置你的目标数据库，并测试，确保连接成功。

1. 生成数据库表

在类图的任意一个空白的地方，右键，在弹出的右键菜单中，Buffalo助手 -> 生成数据库，在弹出的窗体里，有即将执行的SQL语句，点击执行按钮，生成数据库表，即可。

# 基础查询

### 查询

UsersBusiness bo = new UsersBusiness();

ScopeList lstScope = new ScopeList();’ //查询辅助类

List<Users> lst=bo.SelectList(lstScope); //查询方法

### 条件查询

UsersBusiness bo = new UsersBusiness();

ScopeList lstScope = new ScopeList();

//分页

lstScope.PageContent.PageSize = 10;

lstScope.PageContent.CurrentPage = 1;

//排序

lstScope.OrderBy.Add(ManagementDB.Users.Storage.Id.DESC);

//条件

lstScope.Add(ManagementDB.Users.Id > 1);

lstScope.Add(ManagementDB.Users.Storage.StorageName.Like("aa") | ManagementDB.Users.IsEnable==true);

//查询

List<Users> lst=bo.SelectList(lstScope);

## 插入

### 基本插入

SampleBackBusiness bo = new SampleBackBusiness();

SampleBack sb = CH.Create<SampleBack>();

sb.B\_remark = "aa";

sb.UserId = i % 10;

sb.SampleCode = "SC00" + i;

sb.BackTime = null;

bo. Insert(sb);//插入对象

### 插入并返回自增长

SampleBackBusiness bo = new SampleBackBusiness();

SampleBack sb = CH.Create<SampleBack>();

sb.B\_remark = "aa";

sb.UserId = i % 10;

sb.SampleCode = "SC00" + i;

sb.BackTime = null;

bo. IdentityInsert (sb);//插入对象并把自增长ID存回实体

MessageBox.Show(sb.Id.ToString());

## 更新

### 2.3.1 更新一个对象

UsersBusiness bo = new UsersBusiness();

Users obj = bo.GetEntityById(10);//获取ID为10的实体

if (obj != null)

{

obj.IsEnable = false;

bo.Update(obj);//更新到数据库

}

### 批量更新(慎用)

ArrayList lstId = new ArrayList();//创建一个集合并把in的内容放进集合

lstId.Add(1);

lstId.Add(3);

lstId.Add(5);

lstId.Add(7);

UsersBusiness bo = new UsersBusiness();

ScopeList lstScope = new ScopeList();

lstScope.Add(ManagementDB.Users.Id.In(lstId));//添加in条件

Users objUpdateUser = CH.Create<Users>();

objUpdateUser.IsEnable = false;

bo.Update(objUpdateUser, lstScope);//更新in集合对应的记录

输出SQL: update [users] set [isEnable]=0 where id in(1,3,5,7)

此操作为把ID是1、3、5、7的记录的IsEnable的值改为false，，由于跳过了针对单个对象的业务方法直接操作数据库批量更新记录，所以为比较高危的操作，常用可能导致业务逻辑不连贯

### 强制更新一个BQL值

ValueSetList lstValue = new ValueSetList();

lstValue.Add(MyClass.ManEmployee.Name, MyClass.ManEmployee.Name + MyClass.ManEmployee.IDCard);

lstValue.Add(MyClass.ManEmployee.LastUpdate, BQL.NowDate());

emp.Remark = "23333";

输出：

update [manEmployee] set [remark]=@P0,[name]=([manEmployee].[name]+[manEmployee].[IDCard]),

[lastUpdate]=getdate() where 1=1 and [id]=@P1,@P0='23333' , @P1=2,

## bo.Update(emp, new ScopeList(), lstValue);删除

### 基本删除

UsersBusiness bo = new UsersBusiness();

Users objUser = new Users();

objUser.Id = 25;

bo.Delete(objUser);//删除ID是25的记录

删除ID为25的记录

### 批量删除(慎用)

ArrayList lstId = new ArrayList();//创建一个集合并把in的内容放进集合

lstId.Add(1);

lstId.Add(3);

lstId.Add(5);

lstId.Add(7);

UsersBusiness bo = new UsersBusiness();

ScopeList lstScope = new ScopeList();

lstScope.Add(ManagementDB.Users.Id.In(lstId)); //添加in条件

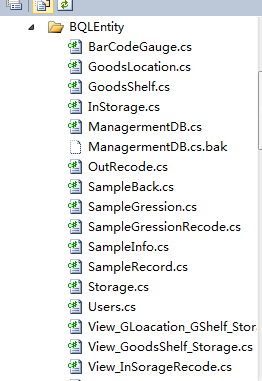
bo.Delete (lstScope);//删除in集合对应的记录

输出SQL: delete from [users] where id in(1,3,5,7)

此操作为把ID是1、3、5、7的记录删除，由于跳过了针对单个对象的业务方法直接操作数据库批量更新记录，所以为比较高危的操作，常用可能导致业务逻辑不连贯甚至删错的危险

## 查询条件类ScopeList介绍

### 条件辅助类 BQLEntity



查询条件辅助类BQLEntity是对应每个实体的属性，用以拼装查询条件、排序条件分组条件等



### 简单条件演示

lstScope.Add(ManagementDB.Users.UserName == "abc");// [users].[userName]='abc'

lstScope.Add(ManagementDB.Users.StorageId > 10);// [users].[storageId]>10

lstScope.Add(ManagementDB.Users.StorageId < 10);// [users].[storageId]<10

lstScope.Add(ManagementDB.Users.UserName.Like("abc"));//[users].[userName] like '%abc%'

lstScope.Add(ManagementDB.Users.UserName.StarWith("abc"));// [users].[userName] like 'abc%'

lstScope.Add(ManagementDB.Users.UserName.EndWith("abc"));// [users].[userName] like '%abc'

DateTime dt1 = new DateTime(2011, 10, 1);

DateTime dt2 = new DateTime(2011, 11, 1);

lstScope.Add(ManagementDB.OutRecode.OutTime.Between(dt1,dt2));

// [outRecode].[outTime] between '2011-10-1' and '2011-11-1'

ArrayList lstId = new ArrayList();

lstId.Add(1);

lstId.Add(3);

lstId.Add(5);

lstId.Add(7);

lstScope.Add(ManagementDB.Users.Id.In(lstId));//添加IN条件

//[outRecode].[id] in (1,3,5,7)

ArrayList lstId = new ArrayList();

lstId.Add(1);

lstId.Add(3);

lstId.Add(5);

lstId.Add(7);

lstScope.Add(ManagementDB.Users.Id.NotIn(lstId));

//[outRecode].[id] not in (1,3,5,7)

### 条件连接

And条件：

lstScope.Add(

ManagementDB.SampleBack.BackTime > DateTime.Parse("2011-10-1")

& ManagementDB.SampleBack.UserId==22);

// [sampleBack].[backTime]>'2011-10-1' and [sampleBack].[userId]=22

Or条件：

lstScope.Add(

ManagementDB.SampleBack.BackTime > DateTime.Parse("2011-10-1")

| ManagementDB.SampleBack.UserId==22);

// [sampleBack].[backTime]>'2011-10-1' or [sampleBack].[userId]=22

### 排序

lstScope.OrderBy.Add(ManagementDB.Users.Storage.Id.DESC);

//order by [users].[storage] desc

lstScope.OrderBy.Add(ManagementDB.Users.Storage.Id.ASC);

//order by [users].[storage]

### 分页

lstScope.PageContent.PageSize = 10; //每页多少条

lstScope.PageContent.CurrentPage = 1; //当前第几条

List<Users> lst=bo.SelectList(lstScope); //查询

dataGridView1.DataSource = lst; //绑定

lab1.Text= "共"+lstScope.PageContent.TotleRecords.ToString()+"条"; //查询后lstScope.PageContent.TotleRecords会自动填入总条数，此为输出总条数

### 业务层使用

3.5.6.1 Exists方法

protected override object Exists(SampleGression entity)

{

ScopeList lstScope = new ScopeList();

if (entity.Id.GetValueOrDefault() > 0)

{

lstScope.Add(ManagementDB.SampleGression.Id != entity.Id.GetValueOrDefault());

}

lstScope.Add(ManagementDB.SampleGression.SampleCode == entity.SampleCode);

bool exists = ExistsRecord(lstScope);

if (exists) //非空则为失败，更新和插入方法都会被中断

{

return "有重复编号";

}

return null;//空则为成功，更新和插入方法都会继续

}

### 业务

3.5.7.1 事务使用



事务使用StartTransation

开启，然后需要用using包着需要事务的动作，在tran之前退出了方法，没执行Commit的话，事务将会自动回滚，

3.5.7.1 事务特性



由于业务需要，有时候单独的带事务业务方法会被嵌在另一个事务里边，此时，里层的事务(UpdateGoodsLocation的事务)为无效事务，回滚和提交的方法都是空方法

### 配置

### 2.5.8.1 不读取配置文件，使用自定义数据库信息

例子：

DBInfo info=new Buffalo.DB.DataBaseAdapter.DBInfo("PermissionDB","server=.;database=School;uid=sa;pwd=111111","Sql2K5");//新建数据库信息

info.QueryCache.InitCache(new MemoryAdaper(info), false);

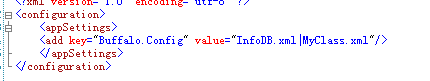
info.DataaccessNamespace = new string[] {

"Buffalo.Permissions.PermissionsInfo.DataAccess.Sql2K5" };//设置数据层

Buffalo.DB.DataBaseAdapter.DataAccessLoader.AppendDBInfo(info);//把数据库信息加到系统

Buffalo.Permissions.PermissionsInfo.BQLEntity.PermissionDB.InitDB();//初始化数据库信息

### 2.5.8.2 一个项目多个数据库



先在配置文件指向多个xml文件



然后再启动时候把程序集加到系统，最后把两个数据库对应的DB类Init一下即可

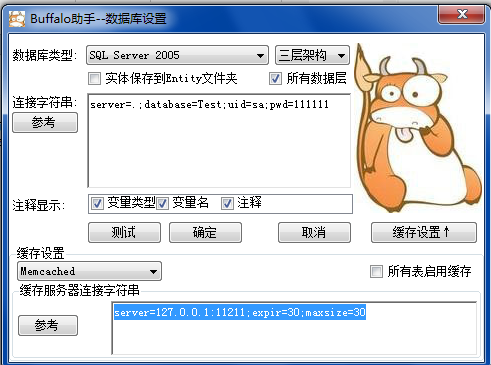
### 使用查询缓存

**2.5.9.1 介绍**

本系统支持内存、memcached、Redis作为查询缓存

**2.5.9.2 配置缓存服务器**

在类图中设置使用何种缓存系统，如果是分布式缓存，需要填入缓存服务器



Server为缓存服务器的IP和端口，expir为缓存数据的保存分钟数，maxsize为连接池最大数量

如果勾选所有表启用缓存后，则所有表都可使用缓存服务器缓存查询，否则只可以让配置为可用缓存的表使用缓存服务器

**2.5.9.3 配置可用缓存的表**



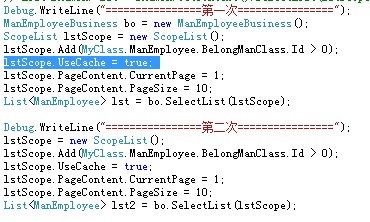
在实体配置窗体中勾选使用缓存则可以让实体对应的表使用缓存

如果表不在类图中则可通过代码实现缓存表的添加：

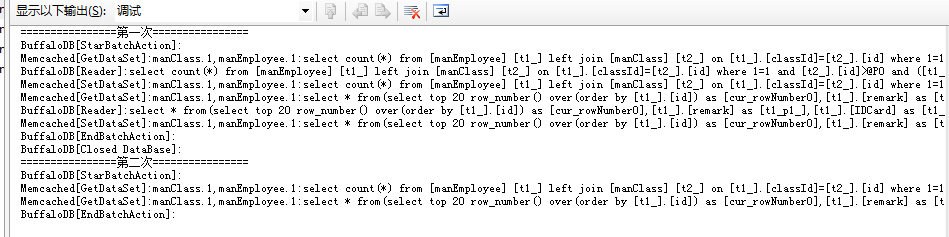
CusManagerDB.GetDBinfo().QueryCache.SetCacheTable("MyTable");

**2.5.9.4 使用缓存**

在查询时候给ScopeList的UseCache设置为true，则可以让查询使用缓存



测试结果显示第二次直接从缓存服务器获取数据：



## 升级数据库

使用DB类可以检查当前类配置跟数据库的差异，并进行增量更新

### 5.6.1 更新特性

可更新：

1. 添加不存在的表
2. 添加不存在的关系约束
3. 添加不存在的字段

无法更新，需要用户生成SQL处理：

1. 存在同名字段，但数据类型不同
2. 存在同名字段但默认值或备注不同

### 5.6.2 利用代码更新到数据库

例如您项目已经有MyDB的库，在进行初始化后使用此代码更新

GameBoxLib.BQLEntity.MyDB.InitDB();//初始化数据库

List<string> lstSql = MyDB.GetDBinfo().CheckDataBase(); //检查数据库并生成需要更新的SQL语句

List<string> resault = DBChecker.ExecuteSQL(MyDB.GetDBinfo().DefaultOperate, lstSql);//批量执行Sql，并返回每条的执行结果