## 涵越第三方仓储管理系统源代码

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Cache

{

public class Cache : ICache

{

private static readonly Dictionary<object, CacheBody> Dic = null;

private readonly object \_object = new object();

static Cache()

{

if (Dic == null)

{

Dic = new Dictionary<object, CacheBody>();

}

}

/// <summary>

/// 获得缓存容器的长度

/// </summary>

/// <returns></returns>

public int Length()

{

if (Dic != null)

{

return Dic.Count();

}

return 0;

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 过期时间无限延长，不依赖任何文件

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

public int Add(object argKey, object argValue)

{

lock (\_object)

{

if (Dic != null && !Dic.ContainsKey(argKey))

{

CacheBody body = new CacheBody() { Body=argValue,Expiration=DateTime.MaxValue,DependencyFile=string.Empty };

Dic.Add(argKey,body);

return 1;

}

return 0;

}

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

public int Add(object argKey, object argValue, DateTime expiration)

{

lock (\_object)

{

if (Dic != null && !Dic.ContainsKey(argKey))

{

CacheBody body = new CacheBody() { Body = argValue, Expiration = expiration, DependencyFile = string.Empty };

Dic.Add(argKey, body);

return 1;

}

return 0;

}

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

public int Insert(object argKey, object argValue)

{

lock (\_object)

{

if (Dic != null)

{

if (Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

}

else

{

CacheBody body = new CacheBody() { Body = argValue, Expiration = DateTime.MaxValue, DependencyFile = string.Empty };

Dic.Add(argKey, body);

}

return 1;

}

return 0;

}

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

public int Insert(object argKey, object argValue, DateTime expiration)

{

lock (\_object)

{

if (Dic != null)

{

if (Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

}

else

{

CacheBody body = new CacheBody() { Body = argValue, Expiration = expiration, DependencyFile = string.Empty };

Dic.Add(argKey, body);

}

return 1;

}

return 0;

}

}

/// <summary>

/// 移除缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

public int Remove(object argKey)

{

lock (\_object)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

Dic.Remove(argKey);

return 1;

}

return 0;

}

}

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

public int Replace(object argKey, object argValue)

{

lock (\_object)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

return 1;

}

return 0;

}

}

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

public int Replace(object argKey, object argValue, DateTime expiration)

{

lock (\_object)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

Dic[argKey].Expiration = expiration;

return 1;

}

return 0;

}

}

/// <summary>

/// 获得缓存值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

public object Get(object argKey)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

if (DateTime.Now <= Dic[argKey].Expiration)

{

return Dic[argKey].Body;

}

else

{

Remove(argKey);

}

}

return null;

}

/// <summary>

/// 清空所有缓存数据

/// </summary>

public void Clear()

{

lock (\_object)

{

if (Dic != null)

{

Dic.Clear();

}

}

}

/// <summary>

/// 获得所有的key值数组

/// </summary>

/// <returns></returns>

public object[] GetKeys()

{

if (Dic != null)

{

return Dic.Keys.ToArray();

}

return null;

}

/// <summary>

/// 获得所有缓存value值数组

/// </summary>

/// <returns></returns>

public object[] GetValues()

{

if (Dic != null)

{

return Dic.Values.Where(item=>DateTime.Now<=item.Expiration).Select(item => item.Body).ToArray();

}

return null;

}

/// <summary>

/// Key值数组属性

/// </summary>

public object[] Keys

{

get

{

return GetKeys();

}

}

/// <summary>

/// Value值数组属性

/// </summary>

public object[] Values

{

get

{

return GetValues();

}

}

/// <summary>

/// 容器长度属性

/// </summary>

public int Count

{

get

{

return Length();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Cache

{

public class CacheBody

{

public CacheBody()

{

}

/// <summary>

/// 存储数据

/// </summary>

public object Body { get; set; }

/// <summary>

/// 数据有效期

/// </summary>

public DateTime Expiration { get; set; }

/// <summary>

/// 依赖文件

/// </summary>

public string DependencyFile { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Cache

{

public class CacheBody<T>

{

public CacheBody()

{

}

/// <summary>

/// 存储数据

/// </summary>

public T Body { get; set; }

/// <summary>

/// 数据有效期

/// </summary>

public DateTime Expiration { get; set; }

/// <summary>

/// 依赖文件

/// </summary>

public string DependencyFile { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Cache

{

public class Cache<T, V> : ICache<T, V>

{

private static Dictionary<T, CacheBody<V>> Dic = null;

private readonly object \_object = new object();

static Cache()

{

Dic = new Dictionary<T, CacheBody<V>>();

}

/// <summary>

/// 获得缓存容器的长度

/// </summary>

/// <returns></returns>

public int Length()

{

if (Dic != null)

{

return Dic.Count;

}

return 0;

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

public int Add(T argKey, V argValue)

{

lock (\_object)

{

if (Dic != null && !Dic.ContainsKey(argKey))

{

CacheBody<V> body = new CacheBody<V>() { Body=argValue,DependencyFile=string.Empty, Expiration=DateTime.MaxValue };

Dic.Add(argKey, body);

return 1;

}

return 0;

}

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

public int Add(T argKey, V argValue, DateTime expiration)

{

lock (\_object)

{

if (Dic != null && !Dic.ContainsKey(argKey))

{

CacheBody<V> body = new CacheBody<V>() { Body = argValue, DependencyFile = string.Empty, Expiration = expiration };

Dic.Add(argKey, body);

return 1;

}

return 0;

}

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

public int Insert(T argKey, V argValue)

{

lock (\_object)

{

if (Dic != null)

{

if (Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

}

else

{

CacheBody<V> body = new CacheBody<V>() { Body = argValue, DependencyFile = string.Empty, Expiration = DateTime.MaxValue };

Dic.Add(argKey, body);

}

return 1;

}

else

{

return 0;

}

}

}

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

public int Insert(T argKey, V argValue, DateTime expiration)

{

lock (\_object)

{

if (Dic != null)

{

if (Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

}

else

{

CacheBody<V> body = new CacheBody<V>() { Body = argValue, DependencyFile = string.Empty, Expiration = expiration };

Dic.Add(argKey, body);

}

return 1;

}

else

{

return 0;

}

}

}

/// <summary>

/// 移除缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

public int Remove(T argKey)

{

lock (\_object)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

Dic.Remove(argKey);

return 1;

}

return 0;

}

}

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

public int Replace(T argKey, V argValue)

{

lock (\_object)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

return 1;

}

return 0;

}

}

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

public int Replace(T argKey, V argValue, DateTime expiration)

{

lock (\_object)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

Dic[argKey].Body = argValue;

Dic[argKey].Expiration = expiration;

return 1;

}

return 0;

}

}

/// <summary>

/// 获得缓存值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

public V Get(T argKey)

{

if (Dic != null && Dic.ContainsKey(argKey))

{

if (DateTime.Now <= Dic[argKey].Expiration)

{

return Dic[argKey].Body;

}

else

{

Remove(argKey);

}

}

return default(V);

}

/// <summary>

/// 清空所有缓存数据

/// </summary>

public void Clear()

{

if (Dic != null)

{

Dic.Clear();

}

}

/// <summary>

/// 获得所有的key值数组

/// </summary>

/// <returns></returns>

public T[] GetKeys()

{

if (Dic != null)

{

return Dic.Keys.ToArray<T>();

}

return default(T[]);

}

/// <summary>

/// 获得所有缓存value值数组

/// </summary>

/// <returns></returns>

public V[] GetValues()

{

if (Dic != null)

{

return Dic.Values.Where(item => item.Expiration >= DateTime.Now).Select(item => item.Body).ToArray<V>();

}

return default(V[]);

}

/// <summary>

/// Key值数组属性

/// </summary>

public T[] Keys

{

get

{

return GetKeys();

}

}

/// <summary>

/// Value值数组属性

/// </summary>

public V[] Values

{

get

{

return GetValues();

}

}

/// <summary>

/// 容器长度属性

/// </summary>

public int Count

{

get

{

return Length();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Web;

using System.Web.Caching;

namespace AnJie.ERP.Cache

{

public class CacheHelper

{

private static readonly System.Web.Caching.Cache Cache = HttpRuntime.Cache;

/// <summary>

/// 插入缓存，如果存在则替换

/// </summary>

/// <param name="argKey"></param>

/// <param name="argValue"></param>

/// <returns></returns>

public static int Insert(string argKey, object argValue)

{

Cache.Insert(argKey, argValue);

return 1;

}

/// <summary>

/// 插入缓存，如果存在则替换

/// </summary>

/// <param name="argKey"></param>

/// <param name="argValue"></param>

/// <param name="argDependency"></param>

/// <returns></returns>

public static int Insert(string argKey, object argValue, CacheDependency argDependency)

{

Cache.Insert(argKey, argValue, argDependency);

return 1;

}

/// <summary>

/// 插入缓存，如果存在则替换

/// </summary>

/// <param name="argKey"></param>

/// <param name="argValue"></param>

/// <param name="argDependency"></param>

/// <param name="argExpiration"></param>

/// <returns></returns>

public static int Insert(string argKey, object argValue, CacheDependency argDependency, DateTime argExpiration)

{

Cache.Insert(argKey, argValue, argDependency, argExpiration, System.Web.Caching.Cache.NoSlidingExpiration);

return 1;

}

/// <summary>

/// 添加缓存，如果存在则抛出异常

/// </summary>

/// <param name="argKey"></param>

/// <param name="argValue"></param>

/// <returns></returns>

public static int Add(string argKey, object argValue)

{

Cache.Add(argKey, argValue, null, DateTime.MaxValue, System.Web.Caching.Cache.NoSlidingExpiration,

CacheItemPriority.Default, null);

return 1;

}

/// <summary>

/// 添加缓存，如果存在则抛出异常

/// </summary>

/// <param name="argKey"></param>

/// <param name="argValue"></param>

/// <param name="argDependency"></param>

/// <returns></returns>

public static int Add(string argKey, object argValue, CacheDependency argDependency)

{

Cache.Add(argKey, argValue, argDependency, DateTime.MaxValue, System.Web.Caching.Cache.NoSlidingExpiration,

CacheItemPriority.Default, null);

return 1;

}

/// <summary>

/// 添加缓存，如果存在则抛出异常

/// </summary>

/// <param name="argKey"></param>

/// <param name="argValue"></param>

/// <param name="argDependency"></param>

/// <param name="argExpiration"></param>

/// <returns></returns>

public static int Add(string argKey, object argValue, CacheDependency argDependency, DateTime argExpiration)

{

Cache.Add(argKey, argValue, argDependency, argExpiration, System.Web.Caching.Cache.NoSlidingExpiration,

CacheItemPriority.Default, null);

return 0;

}

/// <summary>

/// 返回缓存中的所有数据行

/// </summary>

/// <returns></returns>

public static int Count()

{

return Cache.Count;

}

/// <summary>

/// 根据键值获得缓存值

/// </summary>

/// <param name="argKey"></param>

/// <returns></returns>

public static object Get(string argKey)

{

return Cache[argKey];

}

/// <summary>

/// 根据键值获得特定类型的缓存值

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="argKey"></param>

/// <returns></returns>

public static T Get<T>(string argKey)

{

if (Cache[argKey] != null)

{

return (T) Cache[argKey];

}

return default(T);

}

/// <summary>

/// 移除特定的键值

/// </summary>

/// <param name="argKey"></param>

/// <returns></returns>

public static int Remove(string argKey)

{

Cache.Remove(argKey);

return 1;

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Text;

using System.Web;

using System.Web.Caching;

namespace AnJie.ERP.Cache

{

/// <summary>

/// 服务器缓存帮助类

/// </summary>

public class DataCache

{

/// <summary>

/// 创建缓存项过期

/// </summary>

/// <param name="key">缓存Key</param>

/// <param name="obj">object对象</param>

public static void Insert(string key, object obj)

{

if (obj != null)

{

int expires = int.Parse(ConfigurationManager.AppSettings["TimeCache"]);

Insert(key, obj, expires);

}

}

/// <summary>

/// 创建缓存项过期

/// </summary>

/// <param name="key">缓存Key</param>

/// <param name="obj">object对象</param>

/// <param name="expires">设置时间</param>

public static void Insert(string key, object obj, int expires)

{

if (obj != null)

{

HttpContext.Current.Cache.Insert(key, obj, null, System.Web.Caching.Cache.NoAbsoluteExpiration,

new TimeSpan(0, expires, 0));

}

}

/// <summary>

/// 判断缓存对象是否存在

/// </summary>

/// <param name="strKey">缓存键值名称</param>

/// <returns>是否存在true 、false</returns>

public static bool IsExist(string strKey)

{

return HttpContext.Current.Cache[strKey] != null;

}

/// <summary>

/// 获取缓存对象

/// </summary>

/// <param name="key">缓存Key</param>

/// <returns>object对象</returns>

public static object Get(string key)

{

if (string.IsNullOrEmpty(key))

return null;

if (ConfigurationManager.AppSettings["IsCache"] == "false")

{

return null;

}

return HttpContext.Current.Cache.Get(key);

}

/// <summary>

/// 移除指定数据缓存

/// </summary>

public static void RemoveAllCache(string cacheKey)

{

System.Web.Caching.Cache cache = HttpRuntime.Cache;

cache.Remove(cacheKey);

}

/// <summary>

/// 移除全部缓存

/// </summary>

public static void RemoveAllCache()

{

System.Web.Caching.Cache cache = HttpRuntime.Cache;

IDictionaryEnumerator cacheEnum = cache.GetEnumerator();

while (cacheEnum.MoveNext())

{

cache.Remove(cacheEnum.Key.ToString());

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Cache

{

public interface ICache

{

/// <summary>

/// 获得缓存容器的长度

/// </summary>

/// <returns></returns>

int Length();

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

int Add(object argKey, object argValue);

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

int Add(object argKey, object argValue, DateTime expiration);

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

int Insert(object argKey, object argValue);

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

int Insert(object argKey, object argValue, DateTime expiration);

/// <summary>

/// 移除缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

int Remove(object argKey);

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

int Replace(object argKey, object argValue);

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

int Replace(object argKey, object argValue, DateTime expiration);

/// <summary>

/// 获得缓存值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

object Get(object argKey);

/// <summary>

/// 清空所有缓存数据

/// </summary>

void Clear();

/// <summary>

/// 获得所有的key值数组

/// </summary>

/// <returns></returns>

object[] GetKeys();

/// <summary>

/// 获得所有缓存value值数组

/// </summary>

/// <returns></returns>

object[] GetValues();

/// <summary>

/// Key值数组属性

/// </summary>

object[] Keys { get; }

/// <summary>

/// Value值数组属性

/// </summary>

object[] Values { get; }

/// <summary>

/// 容器长度属性

/// </summary>

int Count { get; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Cache

{

public interface ICache<T, V>

{

/// <summary>

/// 获得缓存容器的长度

/// </summary>

/// <returns></returns>

int Length();

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

int Add(T argKey, V argValue);

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

int Add(T argKey, V argValue, DateTime expiration);

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

int Insert(T argKey, V argValue);

/// <summary>

/// 添加缓存值 当容器中不存在key值时添加

/// 当容器中存在key值修改

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

int Insert(T argKey, V argValue, DateTime expiration);

/// <summary>

/// 移除缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

int Remove(T argKey);

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <returns></returns>

int Replace(T argKey, V argValue);

/// <summary>

/// 替换缓存容器中的值

/// </summary>

/// <param name="argKey">key值</param>

/// <param name="argValue">value值</param>

/// <param name="expiration">过期时间</param>

/// <returns></returns>

int Replace(T argKey, V argValue, DateTime expiration);

/// <summary>

/// 获得缓存值

/// </summary>

/// <param name="argKey">key值</param>

/// <returns></returns>

V Get(T argKey);

/// <summary>

/// 清空所有缓存数据

/// </summary>

void Clear();

/// <summary>

/// 获得所有的key值数组

/// </summary>

/// <returns></returns>

T[] GetKeys();

/// <summary>

/// 获得所有缓存value值数组

/// </summary>

/// <returns></returns>

V[] GetValues();

/// <summary>

/// Key值数组属性

/// </summary>

T[] Keys { get; }

/// <summary>

/// Value值数组属性

/// </summary>

V[] Values { get; }

/// <summary>

/// 容器长度属性

/// </summary>

int Count { get; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Business

{

public enum YesNoStatus

{

/// <summary>

/// 否

/// </summary>

[Description("否")] No = 0,

/// <summary>

/// 是

/// </summary>

[Description("是")] Yes = 1

}

/// <summary>

/// 订单状态

/// </summary>

public enum PrintStatus

{

/// <summary>

/// 待打印

/// </summary>

[Description("待打印")]

WaitPrint = 0,

/// <summary>

/// 打印中

/// </summary>

[Description("打印中")]

Printing = 1,

/// <summary>

/// 已打印

/// </summary>

[Description("已打印")]

Printed = 2,

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data.Common;

using System.Data.SqlClient;

using System.Text;

using System.Linq;

using System.Web;

using AnJie.ERP.DataAccess;

using AnJie.ERP.Utilities;

namespace AnJie.ERP.DataAccess.Common

{

/// <summary>

/// 多条件动态查询属性

/// </summary>

public class Condition

{

/// <summary>

/// 左边括号

/// </summary>

public string LeftBrace { get; set; }

/// <summary>

/// 项目名称（字段名）

/// </summary>

public string ParamName { get; set; }

/// <summary>

/// 比较（操作符）

/// </summary>

public ConditionOperate Operation { get; set; }

/// <summary>

/// 比较值（字段值）

/// </summary>

public object ParamValue { get; set; }

/// <summary>

/// 右边括号

/// </summary>

public string RightBrace { get; set; }

/// <summary>

/// 逻辑符：AND/OR

/// </summary>

public string Logic { get; set; }

///// <summary>

///// 有效：如果等于0 就是无效，不拼接SQL条件

///// </summary>

//public string Enabled { get; set; }

}

// 查询所用到的运算符

public enum ConditionOperate : byte

{

/// <summary>

/// 等于

/// </summary>

Equal,

/// <summary>

/// 不等于

/// </summary>

NotEqual,

/// <summary>

/// 大于

/// </summary>

Greater,

/// <summary>

/// 大于等于

/// </summary>

GreaterThan,

/// <summary>

/// 小于

/// </summary>

Less,

/// <summary>

/// 小于等于

/// </summary>

LessThan,

/// <summary>

/// 为空

/// </summary>

Null,

/// <summary>

/// 不为空

/// </summary>

NotNull,

/// <summary>

/// 包含

/// </summary>

Like,

/// <summary>

/// 不包含

/// </summary>

NotLike,

/// <summary>

/// 左包含

/// </summary>

LeftLike,

/// <summary>

/// 右包含

/// </summary>

RightLike,

/// <summary>

/// 昨天

/// </summary>

Yesterday,

/// <summary>

/// 今天

/// </summary>

Today,

/// <summary>

/// 明天

/// </summary>

Tomorrow,

/// <summary>

/// 上周

/// </summary>

LastWeek,

/// <summary>

/// 本周

/// </summary>

ThisWeek,

/// <summary>

/// 下周

/// </summary>

NextWeek,

/// <summary>

/// 上月

/// </summary>

LastMonth,

/// <summary>

/// 本月

/// </summary>

ThisMonth,

/// <summary>

/// 下月

/// </summary>

NextMonth,

/// <summary>

/// 今天之前（天）

/// </summary>

BeforeDay,

/// <summary>

/// 今天之后（天）

/// </summary>

AfterDay,

}

/// <summary>

/// 拼接查询条件SQL语句

/// </summary>

public class ConditionBuilder

{

/// <summary>

/// 动态查询条件

/// </summary>

/// <param name="conditions">条件参数集合</param>

/// <param name="parameter">要返回Sql参数</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序分类：DESC、ASC</param>

/// <returns></returns>

public static string GetWhereSql(IList conditions, out List<DbParameter> parameter, string orderField = "",

string orderType = "")

{

DateTime startTime;

DateTime endTime;

string ParamKey = DbFactory.CreateDbParmCharacter();

List<DbParameter> ParamList = new List<DbParameter>();

StringBuilder sbWhere = new StringBuilder();

if (conditions.Count > 0)

{

sbWhere.Append(" AND");

}

int indexrow = 0;

foreach (Condition item in conditions)

{

if (item.ParamValue == null)

continue;

string Logic = "";

if (string.IsNullOrEmpty(item.Logic)) Logic = "";

else Logic = item.Logic == "AND" ? "AND" : "OR";

if (conditions.Count - 1 == indexrow)

{

Logic = "";

}

string fieldName = item.ParamName;

string paramName = item.ParamName + indexrow;

int index = (int) item.Operation;

switch (item.Operation)

{

case ConditionOperate.Equal:

sbWhere.Append(" " + item.LeftBrace + fieldName + " = " + ParamKey + paramName + item.RightBrace +

" " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue));

break;

case ConditionOperate.NotEqual:

sbWhere.Append(" " + item.LeftBrace + fieldName + " <> " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue));

break;

case ConditionOperate.Greater:

sbWhere.Append(" " + item.LeftBrace + fieldName + " > " + ParamKey + paramName + item.RightBrace +

" " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue));

break;

case ConditionOperate.GreaterThan:

sbWhere.Append(" " + item.LeftBrace + fieldName + " >= " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue));

break;

case ConditionOperate.Less:

sbWhere.Append(" " + item.LeftBrace + fieldName + " < " + ParamKey + paramName + item.RightBrace +

" " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue));

break;

case ConditionOperate.LessThan:

sbWhere.Append(" " + item.LeftBrace + fieldName + " <= " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue));

break;

case ConditionOperate.Null:

sbWhere.Append(string.Format(" " + item.LeftBrace + "{0} is null ", fieldName) + item.RightBrace +

" " + Logic);

break;

case ConditionOperate.NotNull:

sbWhere.Append(string.Format(" " + item.LeftBrace + "{0} is not null ", fieldName) +

item.RightBrace + " " + Logic);

break;

case ConditionOperate.Like:

sbWhere.Append(" " + item.LeftBrace + fieldName + " like " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, "%" + item.ParamValue + "%"));

break;

case ConditionOperate.NotLike:

sbWhere.Append(" " + item.LeftBrace + fieldName + " not like " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, "%" + item.ParamValue + "%"));

break;

case ConditionOperate.LeftLike:

sbWhere.Append(" " + item.LeftBrace + fieldName + " like " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, "%" + item.ParamValue));

break;

case ConditionOperate.RightLike:

sbWhere.Append(" " + item.LeftBrace + fieldName + " like " + ParamKey + paramName +

item.RightBrace + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + paramName, item.ParamValue + "%"));

break;

case ConditionOperate.Yesterday:

startTime = CommonHelper.GetDateTime(DateTime.Now.AddDays(-1).ToString("yyyy-MM-dd") + " 00:00");

endTime = CommonHelper.GetDateTime(DateTime.Now.AddDays(-1).ToString("yyyy-MM-dd") + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.Today:

startTime = CommonHelper.GetDateTime(DateTimeHelper.GetToday() + " 00:00");

endTime = CommonHelper.GetDateTime(DateTimeHelper.GetToday() + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.Tomorrow:

startTime = CommonHelper.GetDateTime(DateTime.Now.AddDays(1).ToString("yyyy-MM-dd") + " 00:00");

endTime = CommonHelper.GetDateTime(DateTime.Now.AddDays(1).ToString("yyyy-MM-dd") + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.LastWeek:

startTime =

CommonHelper.GetDateTime(

DateTime.Now.AddDays(Convert.ToInt32(1 - Convert.ToInt32(DateTime.Now.DayOfWeek)) - 7)

.ToString("yyyy-MM-dd") + " 00:00");

endTime = CommonHelper.GetDateTime(startTime.AddDays(6).ToString("yyyy-MM-dd") + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.ThisWeek:

startTime =

CommonHelper.GetDateTime(

DateTime.Now.AddDays(1 - Convert.ToInt32(DateTime.Now.DayOfWeek)).ToString("yyyy-MM-dd") +

" 00:00");

endTime = CommonHelper.GetDateTime(startTime.AddDays(6).ToString("yyyy-MM-dd") + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.NextWeek:

startTime =

CommonHelper.GetDateTime(

DateTime.Now.AddDays(Convert.ToInt32(1 - Convert.ToInt32(DateTime.Now.DayOfWeek)) + 7)

.ToString("yyyy-MM-dd") + " 00:00");

endTime = CommonHelper.GetDateTime(startTime.AddDays(6).ToString("yyyy-MM-dd") + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.LastMonth:

startTime =

CommonHelper.GetDateTime(DateTime.Now.AddMonths(-1).ToString("yyyy-MM-01") + " 00:00");

endTime =

CommonHelper.GetDateTime(

DateTime.Parse(DateTime.Now.ToString("yyyy-MM-01")).AddDays(-1).ToString("yyyy-MM-dd") +

" 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.ThisMonth:

startTime = CommonHelper.GetDateTime(DateTime.Now.ToString("yyyy-MM-01") + " 00:00");

endTime =

CommonHelper.GetDateTime(DateTime.Now.AddMonths(1).AddDays(-1).ToString("yyyy-MM-dd") +

" 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.NextMonth:

startTime = CommonHelper.GetDateTime(DateTime.Now.AddMonths(1).ToString("yyyy-MM-01") + " 00:00");

endTime =

CommonHelper.GetDateTime(

DateTime.Parse(DateTime.Now.ToString("yyyy-MM-01"))

.AddMonths(2)

.AddDays(-1)

.ToString("yyyy-MM-dd") + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.BeforeDay:

startTime =

CommonHelper.GetDateTime(

DateTime.Now.AddDays(double.Parse("-" + item.ParamValue.ToString())) + " 00:00");

endTime = CommonHelper.GetDateTime(DateTimeHelper.GetToday() + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

case ConditionOperate.AfterDay:

startTime =

CommonHelper.GetDateTime(DateTime.Now.AddDays(double.Parse(item.ParamValue.ToString())) +

" 00:00");

endTime = CommonHelper.GetDateTime(DateTimeHelper.GetToday() + " 23:59");

sbWhere.Append(

string.Format(

" " + item.LeftBrace + "{0} between " + ParamKey + "start{1} AND " + ParamKey +

"end\_{2}" + item.RightBrace + "", fieldName, paramName, paramName) + " " + Logic);

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("start{0}", paramName),

startTime));

ParamList.Add(DbFactory.CreateDbParameter(ParamKey + string.Format("end\_{0}", paramName),

endTime));

break;

default:

break;

}

indexrow++;

}

if (!string.IsNullOrEmpty(orderField)) //判断是否有排序功能

{

orderType = orderType.ToLower() == "desc" ? "desc" : "asc";

sbWhere.Append(" Order By " + orderField + " " + orderType + "");

}

parameter = ParamList;

return sbWhere.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.DataAccess.Attributes

{

/// <summary>

/// 主键字段

/// </summary>

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Interface)]

public class PrimaryKeyAttribute : Attribute

{

public PrimaryKeyAttribute()

{

}

public PrimaryKeyAttribute(string name)

{

\_name = name;

}

private string \_name;

public virtual string Name

{

get

{

return \_name;

}

set

{

\_name = value;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.DataAccess.Attributes

{

/// <summary>

/// 主键字段

/// </summary>

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Interface)]

public class TableNameAttribute : Attribute

{

public TableNameAttribute()

{

}

public TableNameAttribute(string name)

{

\_name = name;

}

private string \_name;

public virtual string Name

{

get

{

return \_name;

}

set

{

\_name = value;

}

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Linq;

using System.Text;

using AnJie.ERP.DataAccess.DbExpand;

namespace AnJie.ERP.DataAccess

{

/// <summary>

/// 操作数据库基类

/// </summary>

public class Database : IDatabase, IDisposable

{

#region SqlBulkCopy大批量数据插入

/// <summary>

/// 大批量数据插入

/// </summary>

/// <param name="datatable">资料表</param>

/// <returns></returns>

public bool BulkInsert(DataTable datatable)

{

return false;

}

#endregion

#region 构造函数

public static string ConnString { get; set; }

/// <summary>

/// 构造方法

/// </summary>

public Database(string connstring)

{

var dbhelper = new DbHelper(connstring);

}

/// <summary>

/// 数据库连接对象

/// </summary>

private DbConnection DbConnection { get; set; }

/// <summary>

/// 事务对象

/// </summary>

private DbTransaction IsOpenTrans { get; set; }

/// <summary>

/// 是否已在事务之中

/// </summary>

public bool InTransaction { get; set; }

/// <summary>

/// 事务开始

/// </summary>

/// <returns></returns>

public DbTransaction BeginTrans()

{

if (!InTransaction)

{

DbConnection = DbFactory.CreateDbConnection(DbHelper.ConnectionString);

if (DbConnection.State == ConnectionState.Closed)

{

DbConnection.Open();

}

InTransaction = true;

IsOpenTrans = DbConnection.BeginTransaction();

}

return IsOpenTrans;

}

/// <summary>

/// 提交事务

/// </summary>

public void Commit()

{

if (InTransaction)

{

InTransaction = false;

IsOpenTrans.Commit();

Close();

}

}

/// <summary>

/// 回滚事务

/// </summary>

public void Rollback()

{

if (InTransaction)

{

InTransaction = false;

IsOpenTrans.Rollback();

Close();

}

}

/// <summary>

/// 关闭数据库连接

/// </summary>

public void Close()

{

if (DbConnection != null)

{

DbConnection.Close();

DbConnection.Dispose();

}

if (IsOpenTrans != null)

{

IsOpenTrans.Dispose();

}

DbConnection = null;

IsOpenTrans = null;

}

/// <summary>

/// 内存回收

/// </summary>

public void Dispose()

{

if (DbConnection != null)

{

DbConnection.Dispose();

}

if (IsOpenTrans != null)

{

IsOpenTrans.Dispose();

}

}

#endregion

#region 执行SQL语句

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql)

{

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString());

}

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql, DbTransaction isOpenTrans)

{

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString());

}

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters)

{

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameters);

}

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters, DbTransaction isOpenTrans)

{

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameters);

}

#endregion

#region 执行存储过程

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

public int ExecuteByProc(string procName)

{

return DbHelper.ExecuteNonQuery(CommandType.StoredProcedure, procName);

}

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteByProc(string procName, DbTransaction isOpenTrans)

{

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.StoredProcedure, procName);

}

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int ExecuteByProc(string procName, DbParameter[] parameters)

{

return DbHelper.ExecuteNonQuery(CommandType.StoredProcedure, procName, parameters);

}

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteByProc(string procName, DbParameter[] parameters, DbTransaction isOpenTrans)

{

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.StoredProcedure, procName, parameters);

}

#endregion

#region 插入数据

/// <summary>

/// 插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <returns></returns>

public int Insert<T>(T entity)

{

object val = 0;

var strSql = DatabaseCommon.InsertSql(entity);

var parameter = DatabaseCommon.GetParameter(entity);

val = DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

/// <summary>

/// 插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Insert<T>(T entity, DbTransaction isOpenTrans)

{

object val = 0;

var strSql = DatabaseCommon.InsertSql(entity);

var parameter = DatabaseCommon.GetParameter(entity);

val = DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

/// <summary>

/// 批量插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <returns></returns>

public int Insert<T>(List<T> entity)

{

object val = 0;

var isOpenTrans = BeginTrans();

try

{

foreach (var item in entity)

{

Insert(item, isOpenTrans);

}

Commit();

val = 1;

}

catch (Exception ex)

{

Rollback();

Close();

val = -1;

throw;

}

return Convert.ToInt32(val);

}

/// <summary>

/// 批量插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Insert<T>(List<T> entity, DbTransaction isOpenTrans)

{

object val = 0;

try

{

foreach (var item in entity)

{

Insert(item, isOpenTrans);

}

val = 1;

}

catch (Exception)

{

val = -1;

throw;

}

return Convert.ToInt32(val);

}

/// <summary>

/// 插入数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">哈希表键值</param>

/// <returns></returns>

public int Insert(string tableName, Hashtable ht)

{

object val = 0;

var strSql = DatabaseCommon.InsertSql(tableName, ht);

var parameter = DatabaseCommon.GetParameter(ht);

val = DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

/// <summary>

/// 插入数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">哈希表键值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Insert(string tableName, Hashtable ht, DbTransaction isOpenTrans)

{

object val = 0;

var strSql = DatabaseCommon.InsertSql(tableName, ht);

var parameter = DatabaseCommon.GetParameter(ht);

val = DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

#endregion

#region 修改数据

/// <summary>

/// 修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <returns></returns>

public int Update<T>(T entity)

{

object val = 0;

var strSql = DatabaseCommon.UpdateSql(entity);

var parameter = DatabaseCommon.GetParameter(entity);

val = DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update<T>(T entity, DbTransaction isOpenTrans)

{

object val = 0;

var strSql = DatabaseCommon.UpdateSql(entity);

var parameter = DatabaseCommon.GetParameter(entity);

val = DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public int Update<T>(string propertyName, string propertyValue)

{

object val = 0;

var strSql = new StringBuilder();

var sb = new StringBuilder();

sb.Append("Update ");

sb.Append(DatabaseCommon.GetTableName<T>());

sb.Append(" Set ");

sb.Append(propertyName);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

val = DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

return Convert.ToInt32(val);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update<T>(string propertyName, string propertyValue, DbTransaction isOpenTrans)

{

object val = 0;

var strSql = new StringBuilder();

var sb = new StringBuilder();

sb.Append("Update ");

sb.Append(DatabaseCommon.GetTableName<T>());

sb.Append(" Set ");

sb.Append(propertyName);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

val = DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

return Convert.ToInt32(val);

}

/// <summary>

/// 批量修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <returns></returns>

public int Update<T>(List<T> entity)

{

object val = 0;

var isOpenTrans = BeginTrans();

try

{

foreach (var item in entity)

{

Update(item, isOpenTrans);

}

Commit();

val = 1;

}

catch (Exception ex)

{

Rollback();

Close();

val = -1;

throw ex;

}

return Convert.ToInt32(val);

}

/// <summary>

/// 批量修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update<T>(List<T> entity, DbTransaction isOpenTrans)

{

object val = 0;

try

{

foreach (var item in entity)

{

Update(item, isOpenTrans);

}

val = 1;

}

catch (Exception ex)

{

val = -1;

throw ex;

}

return Convert.ToInt32(val);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">哈希表键值</param>

/// <param name="propertyName">主键字段</param>

/// <returns></returns>

public int Update(string tableName, Hashtable ht, string propertyName)

{

object val = 0;

var strSql = DatabaseCommon.UpdateSql(tableName, ht, propertyName);

var parameter = DatabaseCommon.GetParameter(ht);

val = DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">哈希表键值</param>

/// <param name="propertyName">主键字段</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update(string tableName, Hashtable ht, string propertyName, DbTransaction isOpenTrans)

{

object val = 0;

var strSql = DatabaseCommon.UpdateSql(tableName, ht, propertyName);

var parameter = DatabaseCommon.GetParameter(ht);

val = DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter);

return Convert.ToInt32(val);

}

#endregion

#region 删除数据

/// <summary>

/// 删除数据

/// </summary>

/// <param name="entity">实体类</param>

/// <returns></returns>

public int Delete<T>(T entity)

{

var strSql = DatabaseCommon.DeleteSql(entity);

var parameter = DatabaseCommon.GetParameter(entity);

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="entity">实体类</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete<T>(T entity, DbTransaction isOpenTrans)

{

var strSql = DatabaseCommon.DeleteSql(entity);

var parameter = DatabaseCommon.GetParameter(entity);

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <returns></returns>

public int Delete<T>(object propertyValue)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var pkName = DatabaseCommon.GetKeyField<T>().ToString(); //获取主键

var strSql = DatabaseCommon.DeleteSql(tableName, pkName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + pkName, propertyValue));

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete<T>(object propertyValue, DbTransaction isOpenTrans)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var pkName = DatabaseCommon.GetKeyField<T>().ToString(); //获取主键

var strSql = DatabaseCommon.DeleteSql(tableName, pkName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + pkName, propertyValue));

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public int Delete<T>(string propertyName, string propertyValue)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var strSql = DatabaseCommon.DeleteSql(tableName, propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete<T>(string propertyName, string propertyValue, DbTransaction isOpenTrans)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var strSql = DatabaseCommon.DeleteSql(tableName, propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public int Delete(string tableName, string propertyName, string propertyValue)

{

var strSql = DatabaseCommon.DeleteSql(tableName, propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(string tableName, string propertyName, string propertyValue, DbTransaction isOpenTrans)

{

var strSql = DatabaseCommon.DeleteSql(tableName, propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">键值生成SQL条件</param>

/// <returns></returns>

public int Delete(string tableName, Hashtable ht)

{

var strSql = DatabaseCommon.DeleteSql(tableName, ht);

var parameter = DatabaseCommon.GetParameter(ht);

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">键值生成SQL条件</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(string tableName, Hashtable ht, DbTransaction isOpenTrans)

{

var strSql = DatabaseCommon.DeleteSql(tableName, ht);

var parameter = DatabaseCommon.GetParameter(ht);

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter);

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyValue">主键值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

public int Delete<T>(object[] propertyValue)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var pkName = DatabaseCommon.GetKeyField<T>().ToString(); //获取主键

var strSql = new StringBuilder("DELETE FROM " + tableName + " WHERE " + pkName + " IN (");

try

{

IList<DbParameter> parameter = new List<DbParameter>();

var index = 0;

var str = DbHelper.DbParmChar + "ID" + index;

for (var i = 0; i < propertyValue.Length - 1; i++)

{

var obj2 = propertyValue[i];

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str).Append(",");

parameter.Add(DbFactory.CreateDbParameter(str, obj2));

index++;

}

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str);

parameter.Add(DbFactory.CreateDbParameter(str, propertyValue[index]));

strSql.Append(")");

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

;

}

catch (Exception ex)

{

throw ex;

}

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyValue">主键值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete<T>(object[] propertyValue, DbTransaction isOpenTrans)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var pkName = DatabaseCommon.GetKeyField<T>().ToString(); //获取主键

var strSql =

new StringBuilder("DELETE FROM " + tableName + " WHERE " + DbHelper.DbParmChar + pkName + " IN (");

try

{

IList<DbParameter> parameter = new List<DbParameter>();

var index = 0;

var str = DbHelper.DbParmChar + "ID" + index;

for (var i = 0; i < propertyValue.Length - 1; i++)

{

var obj2 = propertyValue[i];

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str).Append(",");

parameter.Add(DbFactory.CreateDbParameter(str, obj2));

index++;

}

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str);

parameter.Add(DbFactory.CreateDbParameter(str, propertyValue[index]));

strSql.Append(")");

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

;

}

catch (Exception ex)

{

throw ex;

}

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

public int Delete<T>(string propertyName, object[] propertyValue)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var pkName = propertyName;

var strSql =

new StringBuilder("DELETE FROM " + tableName + " WHERE " + DbHelper.DbParmChar + pkName + " IN (");

try

{

IList<DbParameter> parameter = new List<DbParameter>();

var index = 0;

var str = DbHelper.DbParmChar + "ID" + index;

for (var i = 0; i < propertyValue.Length - 1; i++)

{

var obj2 = propertyValue[i];

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str).Append(",");

parameter.Add(DbFactory.CreateDbParameter(str, obj2));

index++;

}

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str);

parameter.Add(DbFactory.CreateDbParameter(str, propertyValue[index]));

strSql.Append(")");

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

}

catch (Exception ex)

{

throw ex;

}

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete<T>(string propertyName, object[] propertyValue, DbTransaction isOpenTrans)

{

var tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

var pkName = propertyName;

var strSql =

new StringBuilder("DELETE FROM " + tableName + " WHERE " + DbHelper.DbParmChar + pkName + " IN (");

try

{

IList<DbParameter> parameter = new List<DbParameter>();

var index = 0;

var str = DbHelper.DbParmChar + "ID" + index;

for (var i = 0; i < propertyValue.Length - 1; i++)

{

var obj2 = propertyValue[i];

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str).Append(",");

parameter.Add(DbFactory.CreateDbParameter(str, obj2));

index++;

}

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str);

parameter.Add(DbFactory.CreateDbParameter(str, propertyValue[index]));

strSql.Append(")");

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

;

}

catch (Exception ex)

{

throw ex;

}

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

public int Delete(string tableName, string propertyName, object[] propertyValue)

{

var pkName = propertyName;

var strSql =

new StringBuilder("DELETE FROM " + tableName + " WHERE " + DbHelper.DbParmChar + pkName + " IN (");

try

{

IList<DbParameter> parameter = new List<DbParameter>();

var index = 0;

var str = DbHelper.DbParmChar + "ID" + index;

for (var i = 0; i < propertyValue.Length - 1; i++)

{

var obj2 = propertyValue[i];

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str).Append(",");

parameter.Add(DbFactory.CreateDbParameter(str, obj2));

index++;

}

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str);

parameter.Add(DbFactory.CreateDbParameter(str, propertyValue[index]));

strSql.Append(")");

return DbHelper.ExecuteNonQuery(CommandType.Text, strSql.ToString(), parameter.ToArray());

;

}

catch (Exception ex)

{

throw ex;

}

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(string tableName, string propertyName, object[] propertyValue, DbTransaction isOpenTrans)

{

var pkName = propertyName;

var strSql =

new StringBuilder("DELETE FROM " + tableName + " WHERE " + DbHelper.DbParmChar + pkName + " IN (");

try

{

IList<DbParameter> parameter = new List<DbParameter>();

var index = 0;

var str = DbHelper.DbParmChar + "ID" + index;

for (var i = 0; i < propertyValue.Length - 1; i++)

{

var obj2 = propertyValue[i];

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str).Append(",");

parameter.Add(DbFactory.CreateDbParameter(str, obj2));

index++;

}

str = DbHelper.DbParmChar + "ID" + index;

strSql.Append(str);

parameter.Add(DbFactory.CreateDbParameter(str, propertyValue[index]));

strSql.Append(")");

return DbHelper.ExecuteNonQuery(isOpenTrans, CommandType.Text, strSql.ToString(), parameter.ToArray());

;

}

catch (Exception ex)

{

throw ex;

}

}

#endregion

#region 查询数据列表、返回List

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

public List<T> FindListTop<T>(int top) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public List<T> FindListTop<T>(int top, string propertyName, string propertyValue) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

strSql.Append(" AND " + propertyName + " = " + DbHelper.DbParmChar + propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameter.ToArray());

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public List<T> FindListTop<T>(int top, string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public List<T> FindListTop<T>(int top, string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameters);

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <returns></returns>

public List<T> FindList<T>() where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public List<T> FindList<T>(string propertyName, string propertyValue) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(" AND " + propertyName + " = " + DbHelper.DbParmChar + propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameter.ToArray());

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public List<T> FindList<T>(string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public List<T> FindList<T>(string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameters);

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public List<T> FindListBySql<T>(string strSql)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql);

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public List<T> FindListBySql<T>(string strSql, DbParameter[] parameters)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql, parameters);

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public List<T> FindListPage<T>(string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

return SqlServerHelper.GetPageList<T>(strSql.ToString(), orderField, orderType, pageIndex, pageSize,

ref recordCount);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public List<T> FindListPage<T>(string whereSql, string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

return SqlServerHelper.GetPageList<T>(strSql.ToString(), orderField, orderType, pageIndex, pageSize,

ref recordCount);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public List<T> FindListPage<T>(string whereSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

return SqlServerHelper.GetPageList<T>(strSql.ToString(), parameters, orderField, orderType, pageIndex,

pageSize, ref recordCount);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public List<T> FindListPageBySql<T>(string strSql, string orderField, string orderType, int pageIndex,

int pageSize, ref int recordCount)

{

return SqlServerHelper.GetPageList<T>(strSql, orderField, orderType, pageIndex, pageSize, ref recordCount);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public List<T> FindListPageBySql<T>(string strSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount)

{

return SqlServerHelper.GetPageList<T>(strSql, parameters, orderField, orderType, pageIndex, pageSize,

ref recordCount);

}

#endregion

#region 查询数据列表、返回DataTable

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

public DataTable FindTableTop<T>(int top) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public DataTable FindTableTop<T>(int top, string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTableTop<T>(int top, string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(top);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameters);

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <returns></returns>

public DataTable FindTable<T>() where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public DataTable FindTable<T>(string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTable<T>(string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameters);

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public DataTable FindTableBySql(string strSql)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql);

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTableBySql(string strSql, DbParameter[] parameters)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql, parameters);

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public DataTable FindTablePage<T>(string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

return SqlServerHelper.GetPageTable(strSql.ToString(), orderField, orderType, pageIndex, pageSize,

ref recordCount);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public DataTable FindTablePage<T>(string whereSql, string orderField, string orderType, int pageIndex,

int pageSize, ref int recordCount) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

return SqlServerHelper.GetPageTable(strSql.ToString(), orderField, orderType, pageIndex, pageSize,

ref recordCount);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public DataTable FindTablePage<T>(string whereSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>();

strSql.Append(whereSql);

return SqlServerHelper.GetPageTable(strSql.ToString(), parameters, orderField, orderType, pageIndex,

pageSize, ref recordCount);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public DataTable FindTablePageBySql(string strSql, string orderField, string orderType, int pageIndex,

int pageSize, ref int recordCount)

{

return SqlServerHelper.GetPageTable(strSql, orderField, orderType, pageIndex, pageSize, ref recordCount);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="orderField">排序字段</param>

/// <param name="orderType">排序类型</param>

/// <param name="pageIndex">当前页</param>

/// <param name="pageSize">页大小</param>

/// <param name="recordCount">返回查询条数</param>

/// <returns></returns>

public DataTable FindTablePageBySql(string strSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount)

{

return SqlServerHelper.GetPageTable(strSql, parameters, orderField, orderType, pageIndex, pageSize,

ref recordCount);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

public DataTable FindTableByProc(string procName)

{

var dr = DbHelper.ExecuteReader(CommandType.StoredProcedure, procName);

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTableByProc(string procName, DbParameter[] parameters)

{

var dr = DbHelper.ExecuteReader(CommandType.StoredProcedure, procName, parameters);

return DatabaseReader.ReaderToDataTable(dr);

}

#endregion

#region 查询数据列表、返回DataSet

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public DataSet FindDataSetBySql(string strSql)

{

return DbHelper.GetDataSet(CommandType.Text, strSql);

}

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataSet FindDataSetBySql(string strSql, DbParameter[] parameters)

{

return DbHelper.GetDataSet(CommandType.Text, strSql, parameters);

}

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

public DataSet FindDataSetByProc(string procName)

{

return DbHelper.GetDataSet(CommandType.StoredProcedure, procName);

}

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataSet FindDataSetByProc(string procName, DbParameter[] parameters)

{

return DbHelper.GetDataSet(CommandType.StoredProcedure, procName, parameters);

}

#endregion

#region 查询对象、返回实体

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <returns></returns>

public T FindEntity<T>(object propertyValue) where T : new()

{

var pkName = DatabaseCommon.GetKeyField<T>().ToString(); //获取主键字段

var strSql = DatabaseCommon.SelectSql<T>(1);

strSql.Append(" AND ").Append(pkName).Append("=").Append(DbHelper.DbParmChar + pkName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + pkName, propertyValue));

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameter.ToArray());

return DatabaseReader.ReaderToModel<T>(dr);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public T FindEntity<T>(string propertyName, object propertyValue) where T : new()

{

var pkName = propertyName;

var strSql = DatabaseCommon.SelectSql<T>(1);

strSql.Append(" AND ").Append(pkName).Append("=").Append(DbHelper.DbParmChar + pkName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + pkName, propertyValue));

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameter.ToArray());

return DatabaseReader.ReaderToModel<T>(dr);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public T FindEntityByWhere<T>(string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(1);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToModel<T>(dr);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public T FindEntityByWhere<T>(string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectSql<T>(1);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameters);

return DatabaseReader.ReaderToModel<T>(dr);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public T FindEntityBySql<T>(string strSql)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql);

return DatabaseReader.ReaderToModel<T>(dr);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public T FindEntityBySql<T>(string strSql, DbParameter[] parameters)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql, parameters);

return DatabaseReader.ReaderToModel<T>(dr);

}

#endregion

#region 查询对象、返回哈希表

/// <summary>

/// 查询对象、返回哈希表

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public Hashtable FindHashtable(string tableName, string propertyName, object propertyValue)

{

var strSql = DatabaseCommon.SelectSql(tableName, 1);

strSql.Append(" AND ").Append(propertyName).Append("=").Append(DbHelper.DbParmChar + propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameter.ToArray());

return DatabaseReader.ReaderToHashtable(dr);

}

/// <summary>

/// 查询对象、返回哈希表

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public Hashtable FindHashtable(string tableName, StringBuilder whereSql)

{

var strSql = DatabaseCommon.SelectSql(tableName, 1);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString());

return DatabaseReader.ReaderToHashtable(dr);

}

/// <summary>

/// 查询对象、返回哈希表

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public Hashtable FindHashtable(string tableName, StringBuilder whereSql, DbParameter[] parameters)

{

var strSql = DatabaseCommon.SelectSql(tableName, 1);

strSql.Append(whereSql);

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), parameters);

return DatabaseReader.ReaderToHashtable(dr);

}

/// <summary>

/// 查询对象、返回哈希表

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public Hashtable FindHashtableBySql(string strSql)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql);

return DatabaseReader.ReaderToHashtable(dr);

}

/// <summary>

/// 查询对象、返回哈希表

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public Hashtable FindHashtableBySql(string strSql, DbParameter[] parameters)

{

var dr = DbHelper.ExecuteReader(CommandType.Text, strSql, parameters);

return DatabaseReader.ReaderToHashtable(dr);

}

#endregion

#region 查询数据、返回条数

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <returns></returns>

public int FindCount<T>() where T : new()

{

var strSql = DatabaseCommon.SelectCountSql<T>();

return Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString()));

}

/// <summary>

/// 查询数据、返回条数

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// </summary>

/// <returns></returns>

public int FindCount<T>(string propertyName, string propertyValue) where T : new()

{

var strSql = DatabaseCommon.SelectCountSql<T>();

strSql.Append(" AND " + propertyName + " = " + DbHelper.DbParmChar + propertyName);

IList<DbParameter> parameter = new List<DbParameter>();

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + propertyName, propertyValue));

return Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString(), parameter.ToArray()));

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public int FindCount<T>(string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectCountSql<T>();

strSql.Append(whereSql);

return Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString()));

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int FindCount<T>(string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectCountSql<T>();

strSql.Append(whereSql);

return Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString(), parameters));

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public int FindCountBySql(string strSql)

{

return Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, strSql));

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int FindCountBySql(string strSql, DbParameter[] parameters)

{

return Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, strSql, parameters));

}

#endregion

#region 查询数据、返回最大数

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <returns></returns>

public object FindMax<T>(string propertyName) where T : new()

{

var strSql = DatabaseCommon.SelectMaxSql<T>(propertyName);

return DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString());

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public object FindMax<T>(string propertyName, string whereSql) where T : new()

{

var strSql = DatabaseCommon.SelectMaxSql<T>(propertyName);

strSql.Append(whereSql);

return DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString());

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public object FindMax<T>(string propertyName, string whereSql, DbParameter[] parameters) where T : new()

{

var strSql = DatabaseCommon.SelectMaxSql<T>(propertyName);

strSql.Append(whereSql);

return DbHelper.ExecuteScalar(CommandType.Text, strSql.ToString(), parameters);

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public object FindMaxBySql(string strSql)

{

return DbHelper.ExecuteScalar(CommandType.Text, strSql);

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public object FindMaxBySql(string strSql, DbParameter[] parameters)

{

return DbHelper.ExecuteScalar(CommandType.Text, strSql, parameters);

}

#endregion

}

}

using AnJie.ERP.DataAccess.Attributes;

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.Common;

using System.Linq;

using System.Reflection;

using System.Text;

namespace AnJie.ERP.DataAccess

{

public class DatabaseCommon

{

#region 对象参数转换DbParameter

/// <summary>

/// 对象参数转换DbParameter

/// </summary>

/// <returns></returns>

public static DbParameter[] GetParameter<T>(T entity)

{

IList<DbParameter> parameter = new List<DbParameter>();

DbType dbtype = new DbType();

Type type = entity.GetType();

PropertyInfo[] props = type.GetProperties();

foreach (PropertyInfo pi in props)

{

if (pi.GetValue(entity, null) != null)

{

switch (pi.PropertyType.ToString())

{

case "System.Nullable`1[System.Int32]":

case "System.Int32":

dbtype = DbType.Int32;

break;

case "System.Nullable`1[System.Decimal]":

case "System.Decimal":

dbtype = DbType.Decimal;

break;

case "System.Nullable`1[System.DateTime]":

case "System.DateTime":

dbtype = DbType.DateTime;

break;

case "System.Boolean":

dbtype = DbType.Boolean;

break;

default:

dbtype = DbType.String;

break;

}

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + pi.Name, pi.GetValue(entity, null),

dbtype));

}

}

return parameter.ToArray();

}

/// <summary>

/// 对象参数转换DbParameter

/// </summary>

/// <returns></returns>

public static DbParameter[] GetParameter(Hashtable ht)

{

IList<DbParameter> parameter = new List<DbParameter>();

DbType dbtype = new DbType();

foreach (string key in ht.Keys)

{

if (ht[key] is DateTime)

dbtype = DbType.DateTime;

else

dbtype = DbType.String;

parameter.Add(DbFactory.CreateDbParameter(DbHelper.DbParmChar + key, ht[key], dbtype));

}

return parameter.ToArray();

}

#endregion

#region 获取实体类自定义信息

/// <summary>

/// 获取实体类主键字段

/// </summary>

/// <returns></returns>

public static object GetKeyField<T>()

{

Type objTye = typeof(T);

string keyField = "";

var name = objTye.Name;

foreach (Attribute attr in objTye.GetCustomAttributes(true))

{

var attrKeyField = attr as PrimaryKeyAttribute;

if (attrKeyField != null)

{

keyField = attrKeyField.Name;

}

}

return keyField;

}

/// <summary>

/// 获取实体类对应表名

/// </summary>

/// <returns></returns>

public static object GetTableName<T>()

{

Type objTye = typeof(T);

string tableName = objTye.Name;

foreach (Attribute attr in objTye.GetCustomAttributes(true))

{

var keyField = attr as TableNameAttribute;

if (keyField != null)

{

tableName = keyField.Name;

}

}

return tableName;

}

/// <summary>

/// 获取实体类主键字段

/// </summary>

/// <returns></returns>

public static string GetKeyField(string className)

{

Assembly asmb =

Assembly.LoadFrom(System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "bin",

"AnJie.ERP.Entity.dll"));

Type objTye = asmb.GetType(className);

string keyField = "";

foreach (Attribute attr in objTye.GetCustomAttributes(true))

{

var attrKeyField = attr as PrimaryKeyAttribute;

if (attrKeyField != null)

keyField = attrKeyField.Name;

}

return keyField;

}

/// <summary>

/// 获取实体类主键字段

/// </summary>

/// <param name="entity">实体类</param>

/// <returns></returns>

public static object GetKeyFieldValue<T>(T entity)

{

Type objTye = typeof(T);

string keyField = "";

var name = objTye.Name;

foreach (Attribute attr in objTye.GetCustomAttributes(true))

{

var attrKeyField = attr as PrimaryKeyAttribute;

if (attrKeyField != null)

{

keyField = attrKeyField.Name;

}

}

PropertyInfo property = objTye.GetProperty(keyField);

return property.GetValue(entity, null).ToString();

}

/// <summary>

/// 获取实体类 字段中文名称

/// </summary>

/// <param name="pi">字段属性信息</param>

/// <returns></returns>

public static string GetFieldText(PropertyInfo pi)

{

string txt = "";

var descAttrs = pi.GetCustomAttributes(typeof(DisplayNameAttribute), true);

if (descAttrs.Any())

{

var descAttr = descAttrs[0] as DisplayNameAttribute;

if (descAttr != null)

{

txt = descAttr.DisplayName;

}

}

else

{

txt = pi.Name;

}

return txt;

}

/// <summary>

/// 获取实体类中文名称

/// </summary>

/// <returns></returns>

public static string GetClassName<T>()

{

Type objTye = typeof(T);

string entityName = "";

var busingessNames = objTye.GetCustomAttributes(true).OfType<DisplayNameAttribute>();

var descriptionAttributes = busingessNames as DisplayNameAttribute[] ?? busingessNames.ToArray();

if (descriptionAttributes.Any())

{

entityName = descriptionAttributes.ToList()[0].DisplayName;

}

else

{

entityName = objTye.Name;

}

return entityName;

}

#endregion

#region 拼接 Insert SQL语句

/// <summary>

/// 哈希表生成Insert语句

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">Hashtable</param>

/// <returns>int</returns>

public static StringBuilder InsertSql(string tableName, Hashtable ht)

{

StringBuilder sb = new StringBuilder();

sb.Append(" Insert Into ");

sb.Append(tableName);

sb.Append("(");

StringBuilder sp = new StringBuilder();

StringBuilder sbPrame = new StringBuilder();

foreach (string key in ht.Keys)

{

if (ht[key] != null)

{

sbPrame.Append("," + key);

sp.Append("," + DbHelper.DbParmChar + "" + key);

}

}

sb.Append(sbPrame.ToString().Substring(1, sbPrame.ToString().Length - 1) + ") Values (");

sb.Append(sp.ToString().Substring(1, sp.ToString().Length - 1) + ")");

return sb;

}

/// <summary>

/// 泛型方法，反射生成InsertSql语句

/// </summary>

/// <param name="entity">实体类</param>

/// <returns>int</returns>

public static StringBuilder InsertSql<T>(T entity)

{

string tableName = DatabaseCommon.GetTableName<T>().ToString();

Type type = entity.GetType();

StringBuilder sb = new StringBuilder();

sb.Append(" Insert Into ");

sb.Append(tableName);

sb.Append("(");

StringBuilder sp = new StringBuilder();

StringBuilder sbPrame = new StringBuilder();

PropertyInfo[] props = type.GetProperties();

foreach (PropertyInfo prop in props)

{

if (prop.GetValue(entity, null) != null)

{

sbPrame.Append("," + (prop.Name));

sp.Append("," + DbHelper.DbParmChar + "" + (prop.Name));

}

}

sb.Append(sbPrame.ToString().Substring(1, sbPrame.ToString().Length - 1) + ") Values (");

sb.Append(sp.ToString().Substring(1, sp.ToString().Length - 1) + ")");

return sb;

}

#endregion

#region 拼接 Update SQL语句

/// <summary>

/// 哈希表生成UpdateSql语句

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">Hashtable</param>

/// <param name="pkName">主键</param>

/// <returns></returns>

public static StringBuilder UpdateSql(string tableName, Hashtable ht, string pkName)

{

StringBuilder sb = new StringBuilder();

sb.Append(" Update ");

sb.Append(tableName);

sb.Append(" Set ");

bool isFirstValue = true;

foreach (string key in ht.Keys)

{

if (ht[key] != null && pkName != key)

{

if (isFirstValue)

{

isFirstValue = false;

sb.Append(key);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + key);

}

else

{

sb.Append("," + key);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + key);

}

}

}

sb.Append(" Where ").Append(pkName).Append("=").Append(DbHelper.DbParmChar + pkName);

return sb;

}

/// <summary>

/// 泛型方法，反射生成UpdateSql语句

/// </summary>

/// <param name="entity">实体类</param>

/// <param name="pkName">主键</param>

/// <returns>int</returns>

public static StringBuilder UpdateSql<T>(T entity, string pkName)

{

string tableName = DatabaseCommon.GetTableName<T>().ToString();

Type type = entity.GetType();

PropertyInfo[] props = type.GetProperties();

StringBuilder sb = new StringBuilder();

sb.Append(" Update ");

sb.Append(tableName);

sb.Append(" Set ");

bool isFirstValue = true;

foreach (PropertyInfo prop in props)

{

if (prop.GetValue(entity, null) != null && GetKeyField<T>().ToString() != prop.Name)

{

if (isFirstValue)

{

isFirstValue = false;

sb.Append(prop.Name);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + prop.Name);

}

else

{

sb.Append("," + prop.Name);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + prop.Name);

}

}

}

sb.Append(" Where ").Append(pkName).Append("=").Append(DbHelper.DbParmChar + pkName);

return sb;

}

/// <summary>

/// 泛型方法，反射生成UpdateSql语句

/// </summary>

/// <param name="entity">实体类</param>

/// <returns>int</returns>

public static StringBuilder UpdateSql<T>(T entity)

{

string tableName = DatabaseCommon.GetTableName<T>().ToString();

string pkName = GetKeyField<T>().ToString();

Type type = entity.GetType();

PropertyInfo[] props = type.GetProperties();

StringBuilder sb = new StringBuilder();

sb.Append("Update ");

sb.Append(tableName);

sb.Append(" Set ");

bool isFirstValue = true;

foreach (PropertyInfo prop in props)

{

if (prop.GetValue(entity, null) != null && pkName != prop.Name)

{

if (isFirstValue)

{

isFirstValue = false;

sb.Append(prop.Name);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + prop.Name);

}

else

{

sb.Append("," + prop.Name);

sb.Append("=");

sb.Append(DbHelper.DbParmChar + prop.Name);

}

}

}

sb.Append(" Where ").Append(pkName).Append("=").Append(DbHelper.DbParmChar + pkName);

return sb;

}

#endregion

#region 拼接 Delete SQL语句

/// <summary>

/// 拼接删除SQL语句

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="pkName">字段主键</param>

/// <returns></returns>

public static StringBuilder DeleteSql(string tableName, string pkName)

{

return

new StringBuilder("Delete From " + tableName + " Where " + pkName + " = " + DbHelper.DbParmChar + pkName +

"");

}

/// <summary>

/// 拼接删除SQL语句

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">多参数</param>

/// <returns></returns>

public static StringBuilder DeleteSql(string tableName, Hashtable ht)

{

StringBuilder sb = new StringBuilder("Delete From " + tableName + " Where 1=1");

foreach (string key in ht.Keys)

{

sb.Append(" AND " + key + " = " + DbHelper.DbParmChar + "" + key + "");

}

return sb;

}

/// <summary>

/// 拼接删除SQL语句

/// </summary>

/// <param name="entity">实体类</param>

/// <returns></returns>

public static StringBuilder DeleteSql<T>(T entity)

{

string tableName = DatabaseCommon.GetTableName<T>().ToString();

Type type = entity.GetType();

PropertyInfo[] props = type.GetProperties();

StringBuilder sb = new StringBuilder("Delete From " + tableName + " Where 1=1");

foreach (PropertyInfo prop in props)

{

if (prop.GetValue(entity, null) != null)

{

sb.Append(" AND " + prop.Name + " = " + DbHelper.DbParmChar + "" + prop.Name + "");

}

}

return sb;

}

#endregion

#region 拼接 Select SQL语句

/// <summary>

/// 拼接 查询 SQL语句

/// </summary>

/// <returns></returns>

public static StringBuilder SelectSql<T>() where T : new()

{

string tableName = DatabaseCommon.GetTableName<T>().ToString();

PropertyInfo[] props = GetProperties(new T().GetType());

StringBuilder sbColumns = new StringBuilder();

foreach (PropertyInfo prop in props)

{

string propertytype = prop.PropertyType.ToString();

sbColumns.Append(prop.Name + ",");

}

if (sbColumns.Length > 0) sbColumns.Remove(sbColumns.ToString().Length - 1, 1);

string strSql = "SELECT {0} FROM {1} WHERE 1=1 ";

strSql = string.Format(strSql, sbColumns.ToString(), tableName + " ");

return new StringBuilder(strSql);

}

/// <summary>

/// 拼接 查询 SQL语句

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

public static StringBuilder SelectSql<T>(int top) where T : new()

{

string tableName = DatabaseCommon.GetTableName<T>().ToString();

PropertyInfo[] props = GetProperties(new T().GetType());

StringBuilder sbColumns = new StringBuilder();

foreach (PropertyInfo prop in props)

{

sbColumns.Append(prop.Name + ",");

}

if (sbColumns.Length > 0) sbColumns.Remove(sbColumns.ToString().Length - 1, 1);

string strSql = "SELECT top {0} {1} FROM {2} WHERE 1=1 ";

strSql = string.Format(strSql, top, sbColumns.ToString(), tableName + " ");

return new StringBuilder(strSql);

}

/// <summary>

/// 拼接 查询 SQL语句

/// </summary>

/// <param name="tableName">表名</param>

/// <returns></returns>

public static StringBuilder SelectSql(string tableName)

{

StringBuilder strSql = new StringBuilder();

strSql.Append("SELECT \* FROM " + tableName + " WHERE 1=1 ");

return strSql;

}

/// <summary>

/// 拼接 查询 SQL语句

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="top">显示条数</param>

/// <returns></returns>

public static StringBuilder SelectSql(string tableName, int top)

{

StringBuilder strSql = new StringBuilder();

strSql.Append("SELECT top " + top + " \* FROM " + tableName + " WHERE 1=1 ");

return strSql;

}

/// <summary>

/// 拼接 查询条数 SQL语句

/// </summary>

/// <returns></returns>

public static StringBuilder SelectCountSql<T>() where T : new()

{

string tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

return new StringBuilder("SELECT Count(1) FROM " + tableName + " WHERE 1=1 ");

}

/// <summary>

/// 拼接 查询最大数 SQL语句

/// </summary>

/// <param name="propertyName">属性字段</param>

/// <returns></returns>

public static StringBuilder SelectMaxSql<T>(string propertyName) where T : new()

{

string tableName = DatabaseCommon.GetTableName<T>().ToString(); //获取表名

return new StringBuilder("SELECT MAX(" + propertyName + ") FROM " + tableName + " WHERE 1=1 ");

}

#endregion

#region 扩展

/// <summary>

/// 获取访问元素

/// </summary>

/// <param name="type"></param>

/// <returns></returns>

public static PropertyInfo[] GetProperties(Type type)

{

return type.GetProperties(BindingFlags.Public | BindingFlags.Instance);

}

#endregion

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Reflection;

using System.Text;

namespace AnJie.ERP.DataAccess

{

/// <summary>

/// 利用反射实现通用的DataReader转List、DataReader转实体类

/// </summary>

public class DatabaseReader

{

/// <summary>

/// 将IDataReader转换为DataTable

/// </summary>

/// <param name="dr"></param>

/// <returns></returns>

public static DataTable ReaderToDataTable(IDataReader dr)

{

using (dr)

{

DataTable objDataTable = new DataTable("Table");

int intFieldCount = dr.FieldCount;

for (int intCounter = 0; intCounter < intFieldCount; ++intCounter)

{

objDataTable.Columns.Add(dr.GetName(intCounter).ToLower(), dr.GetFieldType(intCounter));

}

objDataTable.BeginLoadData();

object[] objValues = new object[intFieldCount];

while (dr.Read())

{

dr.GetValues(objValues);

objDataTable.LoadDataRow(objValues, true);

}

dr.Close();

objDataTable.EndLoadData();

return objDataTable;

}

}

/// <summary>

/// 将IDataReader转换为DataTable,表头字段大小写不变化

/// </summary>

/// <param name="dr"></param>

/// <returns></returns>

public static DataTable ReaderToDataTableOriginal(IDataReader dr)

{

using (dr)

{

DataTable objDataTable = new DataTable("Table");

int intFieldCount = dr.FieldCount;

for (int intCounter = 0; intCounter < intFieldCount; ++intCounter)

{

objDataTable.Columns.Add(dr.GetName(intCounter), dr.GetFieldType(intCounter));

}

objDataTable.BeginLoadData();

object[] objValues = new object[intFieldCount];

while (dr.Read())

{

dr.GetValues(objValues);

objDataTable.LoadDataRow(objValues, true);

}

dr.Close();

objDataTable.EndLoadData();

return objDataTable;

}

}

/// <summary>

/// 将IDataReader转换为 集合

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="dr"></param>

/// <returns></returns>

public static List<T> ReaderToList<T>(IDataReader dr)

{

using (dr)

{

List<string> field = new List<string>(dr.FieldCount);

for (int i = 0; i < dr.FieldCount; i++)

{

field.Add(dr.GetName(i).ToLower());

}

List<T> list = new List<T>();

while (dr.Read())

{

T model = Activator.CreateInstance<T>();

foreach (PropertyInfo property in model.GetType().GetProperties(BindingFlags.GetProperty | BindingFlags.Public | BindingFlags.Instance))

{

if (field.Contains(property.Name.ToLower()))

{

if (!IsNullOrDBNull(dr[property.Name]))

{

property.SetValue(model, HackType(dr[property.Name], property.PropertyType), null);

}

}

}

list.Add(model);

}

dr.Close();

return list;

}

}

public static List<T> DataTableToList<T>(DataTable dt)

{

using (dt)

{

List<string> field = new List<string>(dt.Columns.Count);

for (int i = 0; i < dt.Columns.Count; i++)

{

field.Add(dt.Columns[i].ColumnName.ToLower());

}

List<T> list = new List<T>();

foreach (DataRow item in dt.Rows)

{

T model = Activator.CreateInstance<T>();

foreach (

PropertyInfo property in

model.GetType()

.GetProperties(BindingFlags.GetProperty | BindingFlags.Public | BindingFlags.Instance))

{

if (field.Contains(property.Name.ToLower()))

{

if (!IsNullOrDBNull(item[property.Name]))

{

property.SetValue(model, HackType(item[property.Name], property.PropertyType), null);

}

}

}

list.Add(model);

}

return list;

}

}

/// <summary>

/// 将IDataReader转换为 实体类

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="dr"></param>

/// <returns></returns>

public static T ReaderToModel<T>(IDataReader dr)

{

using (dr)

{

List<string> field = new List<string>(dr.FieldCount);

for (int i = 0; i < dr.FieldCount; i++)

{

field.Add(dr.GetName(i).ToLower());

}

bool flag = false;

T model = Activator.CreateInstance<T>();

while (dr.Read())

{

foreach (PropertyInfo pi in model.GetType().GetProperties(BindingFlags.GetProperty | BindingFlags.Public | BindingFlags.Instance))

{

if (field.Contains(pi.Name.ToLower()))

{

if (!IsNullOrDBNull(dr[pi.Name]))

{

pi.SetValue(model, HackType(dr[pi.Name], pi.PropertyType), null);

if (!flag)

{

flag = true;

}

}

}

}

}

dr.Close();

return !flag ? default(T) : model;

}

}

/// <summary>

/// 将IDataReader转换为 哈希表

/// </summary>

/// <param name="dr"></param>

/// <returns></returns>

public static Hashtable ReaderToHashtable(IDataReader dr)

{

using (dr)

{

Hashtable ht = new Hashtable();

while (dr.Read())

{

for (int i = 0; i < dr.FieldCount; i++)

{

string strfield = dr.GetName(i).ToLower();

ht[strfield] = dr[strfield];

}

}

dr.Close();

return ht;

}

}

//这个类对可空类型进行判断转换，要不然会报错

public static object HackType(object value, Type conversionType)

{

if (conversionType.IsGenericType && conversionType.GetGenericTypeDefinition() == typeof(Nullable<>))

{

if (value == null)

return null;

System.ComponentModel.NullableConverter nullableConverter =

new System.ComponentModel.NullableConverter(conversionType);

conversionType = nullableConverter.UnderlyingType;

}

return Convert.ChangeType(value, conversionType);

}

/// <summary>

///

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public static bool IsNullOrDBNull(object obj)

{

// return ((obj is DBNull) || string.IsNullOrEmpty(obj.ToString())) ? true : false;

return obj is DBNull ? true : false;

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Linq;

using System.Text;

namespace AnJie.ERP.DataAccess.DbExpand

{

/// <summary>

/// 扩展方法

/// </summary>

public class SqlServerHelper

{

#region 数据分页

/// <summary>

/// 摘要:

/// 数据分页

/// 参数：

/// sql：传入要执行sql语句

/// param：参数化

/// orderField：排序字段

/// orderType：排序类型

/// pageIndex：当前页

/// pageSize：页大小

/// count：返回查询条数

/// </summary>

public static DataTable GetPageTable(string sql, DbParameter[] param, string orderField, string orderType,

int pageIndex, int pageSize, ref int count)

{

StringBuilder strSql = new StringBuilder();

if (pageIndex == 0)

{

pageIndex = 1;

}

int num = (pageIndex - 1)\*pageSize;

int num1 = (pageIndex)\*pageSize;

string orderBy = "";

if (!string.IsNullOrEmpty(orderField))

orderBy = "Order By " + orderField + " " + orderType + "";

else

orderBy = "order by (select 0)";

strSql.Append("Select \* From (Select ROW\_NUMBER() Over (" + orderBy + ")");

strSql.Append(" As rowNum, \* From (" + sql + ") As T ) As N Where rowNum > " + num + " And rowNum <= " +

num1 + "");

count =

Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, "Select Count(1) From (" + sql + ") As t",

param));

IDataReader dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), param);

return DatabaseReader.ReaderToDataTable(dr);

}

/// <summary>

/// 摘要:

/// 数据分页

/// 参数：

/// sql：传入要执行sql语句

/// orderField：排序字段

/// orderType：排序类型

/// pageIndex：当前页

/// pageSize：页大小

/// count：返回查询条数

/// </summary>

public static DataTable GetPageTable(string sql, string orderField, string orderType, int pageIndex,

int pageSize, ref int count)

{

return GetPageTable(sql, null, orderField, orderType, pageIndex, pageSize, ref count);

}

/// <summary>

/// 摘要:

/// 数据分页

/// 参数：

/// sql：传入要执行sql语句

/// param：参数化

/// orderField：排序字段

/// orderType：排序类型

/// pageIndex：当前页

/// pageSize：页大小

/// count：返回查询条数

/// </summary>

public static List<T> GetPageList<T>(string sql, DbParameter[] param, string orderField, string orderType,

int pageIndex, int pageSize, ref int count)

{

StringBuilder strSql = new StringBuilder();

if (pageIndex == 0)

{

pageIndex = 1;

}

int num = (pageIndex - 1)\*pageSize;

int num1 = (pageIndex)\*pageSize;

string orderBy = "";

if (!string.IsNullOrEmpty(orderField))

orderBy = "Order By " + orderField + " " + orderType + "";

else

orderBy = "Order By (select 0)";

strSql.Append("Select \* From (Select ROW\_NUMBER() Over (" + orderBy + ")");

strSql.Append(" As rowNum, \* From (" + sql + ") As T ) As N Where rowNum > " + num + " And rowNum <= " +

num1 + "");

count =

Convert.ToInt32(DbHelper.ExecuteScalar(CommandType.Text, "Select Count(1) From (" + sql + ") As t",

param));

IDataReader dr = DbHelper.ExecuteReader(CommandType.Text, strSql.ToString(), param);

return DatabaseReader.ReaderToList<T>(dr);

}

/// <summary>

/// 摘要:

/// 数据分页

/// 参数：

/// sql：传入要执行sql语句

/// orderField：排序字段

/// orderType：排序类型

/// pageIndex：当前页

/// pageSize：页大小

/// count：返回查询条数

/// </summary>

public static List<T> GetPageList<T>(string sql, string orderField, string orderType, int pageIndex,

int pageSize, ref int count)

{

return GetPageList<T>(sql, null, orderField, orderType, pageIndex, pageSize, ref count);

}

#endregion

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Data.SqlClient;

namespace AnJie.ERP.DataAccess

{

/// <summary>

/// 数据库服务工厂。

/// </summary>

public class DbFactory

{

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来获取命令参数中的参数符号oracle为":",sqlserver为"@"

/// </summary>

/// <returns></returns>

public static string CreateDbParmCharacter()

{

return "@";

}

/// <summary>

/// 根据配置文件中所配置的数据库类型和传入的

/// 数据库链接字符串来创建相应数据库连接对象

/// </summary>

/// <param name="connectionString"></param>

/// <returns></returns>

public static DbConnection CreateDbConnection(string connectionString)

{

DbConnection conn = new SqlConnection(connectionString);

return conn;

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库命令对象

/// </summary>

/// <returns></returns>

public static DbCommand CreateDbCommand()

{

return new SqlCommand();

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库适配器对象

/// </summary>

/// <returns></returns>

public static IDbDataAdapter CreateDataAdapter()

{

return new SqlDataAdapter();

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 和传入的命令对象来创建相应数据库适配器对象

/// </summary>

/// <returns></returns>

public static IDbDataAdapter CreateDataAdapter(DbCommand cmd)

{

return new SqlDataAdapter((SqlCommand) cmd);

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库的参数对象

/// </summary>

/// <returns></returns>

public static DbParameter CreateDbParameter()

{

return new SqlParameter();

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库的参数对象

/// </summary>

/// <returns></returns>

public static DbParameter CreateDbParameter(string paramName, object value)

{

var param = CreateDbParameter();

param.ParameterName = paramName;

param.Value = value;

return param;

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库的参数对象

/// </summary>

/// <returns></returns>

public static DbParameter CreateDbParameter(string paramName, object value, DbType dbType)

{

var param = CreateDbParameter();

param.DbType = dbType;

param.ParameterName = paramName;

param.Value = value;

return param;

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库的参数对象

/// </summary>

/// <returns></returns>

public static DbParameter CreateDbParameter(string paramName, object value, DbType dbType, int size)

{

var param = CreateDbParameter();

param.DbType = dbType;

param.ParameterName = paramName;

param.Value = value;

param.Size = size;

return param;

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库的参数对象

/// </summary>

/// <returns></returns>

public static DbParameter CreateDbParameter(string paramName, object value, int size)

{

var param = CreateDbParameter();

param.ParameterName = paramName;

param.Value = value;

param.Size = size;

return param;

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 来创建相应数据库的参数对象

/// </summary>

/// <returns></returns>

public static DbParameter CreateDbOutParameter(string paramName, int size)

{

DbParameter param = CreateDbParameter();

param.Direction = ParameterDirection.Output;

param.ParameterName = paramName;

param.Size = size;

return param;

}

/// <summary>

/// 根据配置文件中所配置的数据库类型

/// 和传入的参数来创建相应数据库的sql语句对应参数对象

/// </summary>

/// <returns></returns>

public static DbParameter[] CreateDbParameters(int size)

{

int i = 0;

var param = new DbParameter[size];

while (i < size)

{

param[i] = new SqlParameter();

i++;

}

return param;

}

}

}

using System;

using System.Configuration;

using System.Data;

using System.Data.Common;

using AnJie.ERP.Utilities;

namespace AnJie.ERP.DataAccess

{

/// <summary>

/// 数据库操作基类

/// </summary>

public class DbHelper

{

public DbHelper(string connstring)

{

var conStringDesEncrypt = ConfigurationManager.AppSettings["ConStringDESEncrypt"];

ConnectionString = ConfigurationManager.ConnectionStrings[connstring].ConnectionString;

if (conStringDesEncrypt == "true")

{

ConnectionString = DESEncrypt.Decrypt(ConnectionString);

}

DbParmChar = DbFactory.CreateDbParmCharacter();

}

/// <summary>

/// 调试日志

/// </summary>

private static LogHelper Log

{

get { return LogFactory.GetLogger(typeof(DbHelper)); }

}

/// <summary>

/// 连接字符串

/// </summary>

public static string ConnectionString { get; set; }

/// <summary>

/// 数据库命名参数符号

/// </summary>

public static string DbParmChar { get; set; }

/// <summary>

/// 执行 SQL 语句，并返回受影响的行数。

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns></returns>

public static int ExecuteNonQuery(CommandType cmdType, string cmdText, params DbParameter[] parameters)

{

var num = 0;

try

{

var cmd = DbFactory.CreateDbCommand();

using (var conn = DbFactory.CreateDbConnection(ConnectionString))

{

PrepareCommand(cmd, conn, null, cmdType, cmdText, parameters);

num = cmd.ExecuteNonQuery();

cmd.Parameters.Clear();

}

}

catch (Exception ex)

{

num = -1;

Log.Error(ex.Message);

throw;

}

return num;

}

/// <summary>

/// 执行 SQL 语句，并返回受影响的行数。

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <returns></returns>

public static int ExecuteNonQuery(CommandType cmdType, string cmdText)

{

var num = 0;

try

{

var cmd = DbFactory.CreateDbCommand();

using (var conn = DbFactory.CreateDbConnection(ConnectionString))

{

PrepareCommand(cmd, conn, null, cmdType, cmdText, null);

num = cmd.ExecuteNonQuery();

cmd.Parameters.Clear();

}

}

catch (Exception ex)

{

num = -1;

Log.Error(ex.Message);

}

return num;

}

/// <summary>

/// 执行 SQL 语句，并返回受影响的行数。

/// </summary>

/// <param name="connection">数据库连接对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns></returns>

public static int ExecuteNonQuery(DbConnection connection, CommandType cmdType, string cmdText,

params DbParameter[] parameters)

{

var num = 0;

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, connection, null, cmdType, cmdText, parameters);

num = cmd.ExecuteNonQuery();

cmd.Parameters.Clear();

}

catch (Exception ex)

{

num = -1;

Log.Error(ex.Message);

}

return num;

}

/// <summary>

/// 执行 SQL 语句，并返回受影响的行数。

/// </summary>

/// <param name="connection">数据库连接对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <returns></returns>

public static int ExecuteNonQuery(DbConnection connection, CommandType cmdType, string cmdText)

{

var num = 0;

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, connection, null, cmdType, cmdText, null);

num = cmd.ExecuteNonQuery();

cmd.Parameters.Clear();

}

catch (Exception ex)

{

num = -1;

Log.Error(ex.Message);

}

return num;

}

/// <summary>

/// 执行 SQL 语句，并返回受影响的行数。

/// </summary>

/// <param name="isOpenTrans">事务对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns></returns>

public static int ExecuteNonQuery(DbTransaction isOpenTrans, CommandType cmdType, string cmdText,

params DbParameter[] parameters)

{

var num = 0;

try

{

var cmd = DbFactory.CreateDbCommand();

if (isOpenTrans == null || isOpenTrans.Connection == null)

{

using (var conn = DbFactory.CreateDbConnection(ConnectionString))

{

PrepareCommand(cmd, conn, isOpenTrans, cmdType, cmdText, parameters);

num = cmd.ExecuteNonQuery();

}

}

else

{

PrepareCommand(cmd, isOpenTrans.Connection, isOpenTrans, cmdType, cmdText, parameters);

num = cmd.ExecuteNonQuery();

}

cmd.Parameters.Clear();

}

catch (Exception ex)

{

num = -1;

Log.Error(ex.Message);

throw;

}

return num;

}

/// <summary>

/// 执行 SQL 语句，并返回受影响的行数。

/// </summary>

/// <param name="isOpenTrans">事务对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <returns></returns>

public static int ExecuteNonQuery(DbTransaction isOpenTrans, CommandType cmdType, string cmdText)

{

var num = 0;

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, isOpenTrans.Connection, isOpenTrans, cmdType, cmdText, null);

num = cmd.ExecuteNonQuery();

cmd.Parameters.Clear();

}

catch (Exception ex)

{

num = -1;

Log.Error(ex.Message);

throw;

}

return num;

}

/// <summary>

/// 使用提供的参数，执行有结果集返回的数据库操作命令、并返回SqlDataReader对象

/// </summary>

/// <param name="isOpenTrans">事务对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns>返回SqlDataReader对象</returns>

public static IDataReader ExecuteReader(DbTransaction isOpenTrans, CommandType cmdType, string cmdText,

params DbParameter[] parameters)

{

var cmd = DbFactory.CreateDbCommand();

var conn = DbFactory.CreateDbConnection(ConnectionString);

try

{

PrepareCommand(cmd, conn, isOpenTrans, cmdType, cmdText, parameters);

IDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

cmd.Parameters.Clear();

return rdr;

}

catch (Exception ex)

{

conn.Close();

cmd.Dispose();

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 使用提供的参数，执行有结果集返回的数据库操作命令、并返回SqlDataReader对象

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns>返回SqlDataReader对象</returns>

public static IDataReader ExecuteReader(CommandType cmdType, string cmdText, params DbParameter[] parameters)

{

var cmd = DbFactory.CreateDbCommand();

var conn = DbFactory.CreateDbConnection(ConnectionString);

try

{

PrepareCommand(cmd, conn, null, cmdType, cmdText, parameters);

IDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

cmd.Parameters.Clear();

return rdr;

}

catch (Exception ex)

{

conn.Close();

cmd.Dispose();

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 使用提供的参数，执行有结果集返回的数据库操作命令、并返回SqlDataReader对象

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <returns>返回SqlDataReader对象</returns>

public static IDataReader ExecuteReader(CommandType cmdType, string cmdText)

{

var cmd = DbFactory.CreateDbCommand();

var conn = DbFactory.CreateDbConnection(ConnectionString);

try

{

PrepareCommand(cmd, conn, null, cmdType, cmdText, null);

IDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

cmd.Parameters.Clear();

return rdr;

}

catch (Exception ex)

{

conn.Close();

cmd.Dispose();

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 查询数据填充到数据集DataSet中

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">命令文本</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns>数据集DataSet对象</returns>

public static DataSet GetDataSet(CommandType cmdType, string cmdText, params DbParameter[] parameters)

{

var ds = new DataSet();

var cmd = DbFactory.CreateDbCommand();

var conn = DbFactory.CreateDbConnection(ConnectionString);

try

{

PrepareCommand(cmd, conn, null, cmdType, cmdText, parameters);

var sda = DbFactory.CreateDataAdapter(cmd);

sda.Fill(ds);

return ds;

}

catch (Exception ex)

{

conn.Close();

cmd.Dispose();

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 查询数据填充到数据集DataSet中

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">命令文本</param>

/// <returns>数据集DataSet对象</returns>

public static DataSet GetDataSet(CommandType cmdType, string cmdText)

{

var ds = new DataSet();

var cmd = DbFactory.CreateDbCommand();

var conn = DbFactory.CreateDbConnection(ConnectionString);

try

{

PrepareCommand(cmd, conn, null, cmdType, cmdText, null);

var sda = DbFactory.CreateDataAdapter(cmd);

sda.Fill(ds);

return ds;

}

catch (Exception ex)

{

conn.Close();

cmd.Dispose();

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 依靠数据库连接字符串connectionString,

/// 使用所提供参数，执行返回首行首列命令

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns>返回一个对象，使用Convert.To{Type}将该对象转换成想要的数据类型。</returns>

public static object ExecuteScalar(CommandType cmdType, string cmdText, params DbParameter[] parameters)

{

try

{

var cmd = DbFactory.CreateDbCommand();

using (var connection = DbFactory.CreateDbConnection(ConnectionString))

{

PrepareCommand(cmd, connection, null, cmdType, cmdText, parameters);

var val = cmd.ExecuteScalar();

cmd.Parameters.Clear();

return val;

}

}

catch (Exception ex)

{

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 依靠数据库连接字符串connectionString,

/// 使用所提供参数，执行返回首行首列命令

/// </summary>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <returns>返回一个对象，使用Convert.To{Type}将该对象转换成想要的数据类型。</returns>

public static object ExecuteScalar(CommandType cmdType, string cmdText)

{

try

{

var cmd = DbFactory.CreateDbCommand();

using (var connection = DbFactory.CreateDbConnection(ConnectionString))

{

PrepareCommand(cmd, connection, null, cmdType, cmdText, null);

var val = cmd.ExecuteScalar();

cmd.Parameters.Clear();

return val;

}

}

catch (Exception ex)

{

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 依靠数据库连接字符串connectionString,

/// 使用所提供参数，执行返回首行首列命令

/// </summary>

/// <param name="connection">数据库连接对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns>返回一个对象，使用Convert.To{Type}将该对象转换成想要的数据类型。</returns>

public static object ExecuteScalar(DbConnection connection, CommandType cmdType, string cmdText,

params DbParameter[] parameters)

{

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, connection, null, cmdType, cmdText, parameters);

var val = cmd.ExecuteScalar();

cmd.Parameters.Clear();

return val;

}

catch (Exception ex)

{

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 依靠数据库连接字符串connectionString,

/// 使用所提供参数，执行返回首行首列命令

/// </summary>

/// <param name="connection">数据库连接对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <returns>返回一个对象，使用Convert.To{Type}将该对象转换成想要的数据类型。</returns>

public static object ExecuteScalar(DbConnection connection, CommandType cmdType, string cmdText)

{

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, connection, null, cmdType, cmdText, null);

var val = cmd.ExecuteScalar();

cmd.Parameters.Clear();

return val;

}

catch (Exception ex)

{

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 依靠数据库连接字符串connectionString,

/// 使用所提供参数，执行返回首行首列命令

/// </summary>

/// <param name="conn">数据库连接对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="isOpenTrans"></param>

/// <returns>返回一个对象，使用Convert.To{Type}将该对象转换成想要的数据类型。</returns>

public static object ExecuteScalar(DbConnection conn, DbTransaction isOpenTrans, CommandType cmdType,

string cmdText)

{

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, conn, isOpenTrans, cmdType, cmdText, null);

var val = cmd.ExecuteScalar();

cmd.Parameters.Clear();

return val;

}

catch (Exception ex)

{

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 依靠数据库连接字符串connectionString,

/// 使用所提供参数，执行返回首行首列命令

/// </summary>

/// <param name="isOpenTrans">事务</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行</param>

/// <param name="parameters">执行命令所需的sql语句对应参数</param>

/// <returns>返回一个对象，使用Convert.To{Type}将该对象转换成想要的数据类型。</returns>

public static object ExecuteScalar(DbTransaction isOpenTrans, CommandType cmdType, string cmdText,

params DbParameter[] parameters)

{

try

{

var cmd = DbFactory.CreateDbCommand();

PrepareCommand(cmd, isOpenTrans.Connection, isOpenTrans, cmdType, cmdText, parameters);

var val = cmd.ExecuteScalar();

cmd.Parameters.Clear();

return val;

}

catch (Exception ex)

{

Log.Error(ex.Message);

throw;

}

}

/// <summary>

/// 为即将执行准备一个命令

/// </summary>

/// <param name="cmd">SqlCommand对象</param>

/// <param name="conn">SqlConnection对象</param>

/// <param name="isOpenTrans">DbTransaction对象</param>

/// <param name="cmdType">执行命令的类型（存储过程或T-SQL，等等）</param>

/// <param name="cmdText">存储过程名称或者T-SQL命令行, e.g. Select \* from Products</param>

/// <param name="cmdParms">SqlParameters to use in the command</param>

private static void PrepareCommand(DbCommand cmd, DbConnection conn, DbTransaction isOpenTrans,

CommandType cmdType, string cmdText, DbParameter[] cmdParms)

{

if (conn.State != ConnectionState.Open)

conn.Open();

cmd.Connection = conn;

cmd.CommandText = cmdText;

if (isOpenTrans != null)

cmd.Transaction = isOpenTrans;

cmd.CommandType = cmdType;

if (cmdParms != null)

{

cmd.Parameters.AddRange(cmdParms);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.DataAccess.DbProvider

{

/// <summary>

/// 有关数据库操作的定义。

/// </summary>

public enum DbOperation

{

/// <summary>

/// 查询

/// </summary>

Select,

/// <summary>

/// 插入

/// </summary>

Insert,

/// <summary>

/// 更新

/// </summary>

Update,

/// <summary>

/// 删除

/// </summary>

Delete,

/// <summary>

/// 截取

/// </summary>

Truncate

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.DataAccess

{

/// <summary>

/// 提交数据库信息

/// 应用单件模式，保存状态

/// </summary>

public class DbResultMsg

{

/// <summary>

/// 错误信息

/// </summary>

public static string ReturnMsg { get; set; }

/// <summary>

/// 耗时

/// </summary>

public static string TimeConsuming { get; set; }

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Linq;

using System.Text;

namespace AnJie.ERP.DataAccess

{

/// <summary>

/// 操作数据库基类 接口

/// </summary>

public interface IDatabase : IDisposable

{

bool InTransaction { get; set; }

DbTransaction BeginTrans();

void Commit();

void Rollback();

void Close();

bool BulkInsert(DataTable dt);

int ExecuteBySql(StringBuilder strSql);

int ExecuteBySql(StringBuilder strSql, DbTransaction isOpenTrans);

int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters);

int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters, DbTransaction isOpenTrans);

int ExecuteByProc(string procName);

int ExecuteByProc(string procName, DbTransaction isOpenTrans);

int ExecuteByProc(string procName, DbParameter[] parameters);

int ExecuteByProc(string procName, DbParameter[] parameters, DbTransaction isOpenTrans);

int Insert<T>(T entity);

int Insert<T>(T entity, DbTransaction isOpenTrans);

int Insert<T>(List<T> entity);

int Insert<T>(List<T> entity, DbTransaction isOpenTrans);

int Insert(string tableName, Hashtable ht);

int Insert(string tableName, Hashtable ht, DbTransaction isOpenTrans);

int Update<T>(T entity);

int Update<T>(T entity, DbTransaction isOpenTrans);

int Update<T>(string propertyName, string propertyValue);

int Update<T>(string propertyName, string propertyValue, DbTransaction isOpenTrans);

int Update<T>(List<T> entity);

int Update<T>(List<T> entity, DbTransaction isOpenTrans);

int Update(string tableName, Hashtable ht, string propertyName);

int Update(string tableName, Hashtable ht, string propertyName, DbTransaction isOpenTrans);

int Delete<T>(T entity);

int Delete<T>(T entity, DbTransaction isOpenTrans);

int Delete<T>(object propertyValue);

int Delete<T>(object propertyValue, DbTransaction isOpenTrans);

int Delete<T>(string propertyName, string propertyValue);

int Delete<T>(string propertyName, string propertyValue, DbTransaction isOpenTrans);

int Delete(string tableName, string propertyName, string propertyValue);

int Delete(string tableName, string propertyName, string propertyValue, DbTransaction isOpenTrans);

int Delete(string tableName, Hashtable ht);

int Delete(string tableName, Hashtable ht, DbTransaction isOpenTrans);

int Delete<T>(object[] propertyValue);

int Delete<T>(object[] propertyValue, DbTransaction isOpenTrans);

int Delete<T>(string propertyName, object[] propertyValue);

int Delete<T>(string propertyName, object[] propertyValue, DbTransaction isOpenTrans);

int Delete(string tableName, string propertyName, object[] propertyValue);

int Delete(string tableName, string propertyName, object[] propertyValue, DbTransaction isOpenTrans);

List<T> FindListTop<T>(int top) where T : new();

List<T> FindListTop<T>(int top, string propertyName, string propertyValue) where T : new();

List<T> FindListTop<T>(int top, string whereSql) where T : new();

List<T> FindListTop<T>(int top, string whereSql, DbParameter[] parameters) where T : new();

List<T> FindList<T>() where T : new();

List<T> FindList<T>(string propertyName, string propertyValue) where T : new();

List<T> FindList<T>(string whereSql) where T : new();

List<T> FindList<T>(string whereSql, DbParameter[] parameters) where T : new();

List<T> FindListBySql<T>(string strSql);

List<T> FindListBySql<T>(string strSql, DbParameter[] parameters);

List<T> FindListPage<T>(string orderField, string orderType, int pageIndex, int pageSize, ref int recordCount)

where T : new();

List<T> FindListPage<T>(string whereSql, string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount) where T : new();

List<T> FindListPage<T>(string whereSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount) where T : new();

List<T> FindListPageBySql<T>(string strSql, string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount);

List<T> FindListPageBySql<T>(string strSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount);

DataTable FindTableTop<T>(int top) where T : new();

DataTable FindTableTop<T>(int top, string whereSql) where T : new();

DataTable FindTableTop<T>(int top, string whereSql, DbParameter[] parameters) where T : new();

DataTable FindTable<T>() where T : new();

DataTable FindTable<T>(string whereSql) where T : new();

DataTable FindTable<T>(string whereSql, DbParameter[] parameters) where T : new();

DataTable FindTableBySql(string strSql);

DataTable FindTableBySql(string strSql, DbParameter[] parameters);

DataTable FindTablePage<T>(string orderField, string orderType, int pageIndex, int pageSize, ref int recordCount)

where T : new();

DataTable FindTablePage<T>(string whereSql, string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount) where T : new();

DataTable FindTablePage<T>(string whereSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount) where T : new();

DataTable FindTablePageBySql(string strSql, string orderField, string orderType, int pageIndex, int pageSize,

ref int recordCount);

DataTable FindTablePageBySql(string strSql, DbParameter[] parameters, string orderField, string orderType,

int pageIndex, int pageSize, ref int recordCount);

DataTable FindTableByProc(string procName);

DataTable FindTableByProc(string procName, DbParameter[] parameters);

DataSet FindDataSetBySql(string strSql);

DataSet FindDataSetBySql(string strSql, DbParameter[] parameters);

DataSet FindDataSetByProc(string procName);

DataSet FindDataSetByProc(string procName, DbParameter[] parameters);

T FindEntity<T>(object propertyValue) where T : new();

T FindEntity<T>(string propertyName, object propertyValue) where T : new();

T FindEntityByWhere<T>(string whereSql) where T : new();

T FindEntityByWhere<T>(string whereSql, DbParameter[] parameters) where T : new();

T FindEntityBySql<T>(string strSql);

T FindEntityBySql<T>(string strSql, DbParameter[] parameters);

Hashtable FindHashtable(string tableName, string propertyName, object propertyValue);

Hashtable FindHashtable(string tableName, StringBuilder whereSql);

Hashtable FindHashtable(string tableName, StringBuilder whereSql, DbParameter[] parameters);

Hashtable FindHashtableBySql(string strSql);

Hashtable FindHashtableBySql(string strSql, DbParameter[] parameters);

int FindCount<T>() where T : new();

int FindCount<T>(string propertyName, string propertyValue) where T : new();

int FindCount<T>(string whereSql) where T : new();

int FindCount<T>(string whereSql, DbParameter[] parameters) where T : new();

int FindCountBySql(string strSql);

int FindCountBySql(string strSql, DbParameter[] parameters);

object FindMax<T>(string propertyName) where T : new();

object FindMax<T>(string propertyName, string whereSql) where T : new();

object FindMax<T>(string propertyName, string whereSql, DbParameter[] parameters) where T : new();

object FindMaxBySql(string strSql);

object FindMaxBySql(string strSql, DbParameter[] parameters);

}

}

using AnJie.ERP.DataAccess;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Repository

{

/// <summary>

/// 操作数据库工厂

/// </summary>

public class DataFactory

{

private static readonly object Locker = new object();

private static Database \_db;

/// <summary>

/// 获取指定的数据库连接

/// </summary>

/// <param name="connString"></param>

/// <returns></returns>

public static IDatabase Database(string connString)

{

if (\_db == null)

{

lock (Locker)

{

\_db = new Database(connString);

}

}

return \_db;

}

/// <summary>

/// 获取指定的数据库连接

/// </summary>

/// <returns></returns>

public static IDatabase Database()

{

return Database("ERP\_SqlServer");

}

}

}

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Text;

using AnJie.ERP.Utilities;

namespace AnJie.ERP.Repository

{

/// <summary>

/// 定义通用的Repository接口

/// </summary>

/// <typeparam name="T"></typeparam>

public interface IRepository<T> where T : new()

{

#region SqlBulkCopy大批量数据插入

/// <summary>

/// 大批量数据插入

/// </summary>

/// <param name="datatable">资料表</param>

/// <returns></returns>

bool BulkInsert(DataTable datatable);

#endregion

#region 事务

/// <summary>

/// 事务开始

/// </summary>

/// <returns></returns>

DbTransaction BeginTrans();

/// <summary>

/// 提交事务

/// </summary>

void Commit();

/// <summary>

/// 回滚事务

/// </summary>

void Rollback();

/// <summary>

/// 关闭数据库连接

/// </summary>

void Close();

#endregion

#region 执行SQL语句

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

int ExecuteBySql(StringBuilder strSql);

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int ExecuteBySql(StringBuilder strSql, DbTransaction isOpenTrans);

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters);

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters, DbTransaction isOpenTrans);

#endregion

#region 执行存储过程

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

int ExecuteByProc(string procName);

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int ExecuteByProc(string procName, DbTransaction isOpenTrans);

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

int ExecuteByProc(string procName, DbParameter[] parameters);

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int ExecuteByProc(string procName, DbParameter[] parameters, DbTransaction isOpenTrans);

#endregion

#region 插入数据

/// <summary>

/// 插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <returns></returns>

int Insert(T entity);

/// <summary>

/// 插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Insert(T entity, DbTransaction isOpenTrans);

/// <summary>

/// 批量插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <returns></returns>

int Insert(List<T> entity);

/// <summary>

/// 批量插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Insert(List<T> entity, DbTransaction isOpenTrans);

#endregion

#region 修改数据

/// <summary>

/// 修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <returns></returns>

int Update(T entity);

/// <summary>

/// 修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Update(T entity, DbTransaction isOpenTrans);

/// <summary>

/// 修改数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

int Update(string propertyName, string propertyValue);

/// <summary>

/// 修改数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Update(string propertyName, string propertyValue, DbTransaction isOpenTrans);

/// <summary>

/// 批量修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <returns></returns>

int Update(List<T> entity);

/// <summary>

/// 批量修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Update(List<T> entity, DbTransaction isOpenTrans);

#endregion

#region 删除数据

/// <summary>

/// 删除数据

/// </summary>

/// <param name="entity">实体类</param>

/// <returns></returns>

int Delete(T entity);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="entity">实体类</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Delete(T entity, DbTransaction isOpenTrans);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <returns></returns>

int Delete(object propertyValue);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Delete(object propertyValue, DbTransaction isOpenTrans);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

int Delete(string propertyName, string propertyValue);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Delete(string propertyName, string propertyValue, DbTransaction isOpenTrans);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">键值生成SQL条件</param>

/// <returns></returns>

int Delete(string tableName, Hashtable ht);

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">键值生成SQL条件</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Delete(string tableName, Hashtable ht, DbTransaction isOpenTrans);

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyValue">主键值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

int Delete(object[] propertyValue);

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyValue">主键值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Delete(object[] propertyValue, DbTransaction isOpenTrans);

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

int Delete(string propertyName, object[] propertyValue);

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

int Delete(string propertyName, object[] propertyValue, DbTransaction isOpenTrans);

#endregion

#region 查询数据列表、返回List

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

List<T> FindListTop(int top);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

List<T> FindListTop(int top, string propertyName, string propertyValue);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

List<T> FindListTop(int top, string whereSql);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

List<T> FindListTop(int top, string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <returns></returns>

List<T> FindList();

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

List<T> FindList(string propertyName, string propertyValue);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

List<T> FindList(string whereSql);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

List<T> FindList(string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

List<T> FindListBySql(string strSql);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

List<T> FindListBySql(string strSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

List<T> FindListPage(ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

List<T> FindListPage(string whereSql, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

List<T> FindListPage(string whereSql, DbParameter[] parameters, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

List<T> FindListPageBySql(string strSql, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters"></param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

List<T> FindListPageBySql(string strSql, DbParameter[] parameters, ref JqGridParam jqgridparam);

#endregion

#region 查询数据列表、返回DataTable

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

DataTable FindTableTop(int top);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

DataTable FindTableTop(int top, string whereSql);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

DataTable FindTableTop(int top, string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <returns></returns>

DataTable FindTable();

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

DataTable FindTable(string whereSql);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

DataTable FindTable(string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

DataTable FindTableBySql(string strSql);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

DataTable FindTableBySql(string strSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="jqgridparam"></param>

/// <returns></returns>

DataTable FindTablePage(ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

DataTable FindTablePage(string whereSql, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

DataTable FindTablePage(string whereSql, DbParameter[] parameters, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

DataTable FindTablePageBySql(string strSql, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

DataTable FindTablePageBySql(string strSql, DbParameter[] parameters, ref JqGridParam jqgridparam);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

DataTable FindTableByProc(string procName);

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

DataTable FindTableByProc(string procName, DbParameter[] parameters);

#endregion

#region 查询数据列表、返回DataSet

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

DataSet FindDataSetBySql(string strSql);

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

DataSet FindDataSetBySql(string strSql, DbParameter[] parameters);

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

DataSet FindDataSetByProc(string procName);

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

DataSet FindDataSetByProc(string procName, DbParameter[] parameters);

#endregion

#region 查询对象、返回实体

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <returns></returns>

T FindEntity(object propertyValue);

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

T FindEntity(string propertyName, object propertyValue);

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

T FindEntityByWhere(string whereSql);

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

T FindEntityByWhere(string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

T FindEntityBySql(string strSql);

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

T FindEntityBySql(string strSql, DbParameter[] parameters);

#endregion

#region 查询数据、返回条数

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <returns></returns>

int FindCount();

/// <summary>

/// 查询数据、返回条数

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// </summary>

/// <returns></returns>

int FindCount(string propertyName, string propertyValue);

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

int FindCount(string whereSql);

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

int FindCount(string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

int FindCountBySql(string strSql);

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

int FindCountBySql(string strSql, DbParameter[] parameters);

#endregion

#region 查询数据、返回最大数

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <returns></returns>

object FindMax(string propertyName);

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

object FindMax(string propertyName, string whereSql);

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

object FindMax(string propertyName, string whereSql, DbParameter[] parameters);

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

object FindMaxBySql(string strSql);

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

object FindMaxBySql(string strSql, DbParameter[] parameters);

#endregion

}

}

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Text;

using AnJie.ERP.Utilities;

namespace AnJie.ERP.Repository

{

/// <summary>

/// AnJie.ERP.ORM 定义通用的Repository

/// </summary>

/// <typeparam name="T">定义泛型，约束其是一个类</typeparam>

public class Repository<T> : IRepository<T> where T : new()

{

#region SqlBulkCopy大批量数据插入

/// <summary>

/// 大批量数据插入

/// </summary>

/// <param name="datatable">资料表</param>

/// <returns></returns>

public bool BulkInsert(DataTable datatable)

{

return DataFactory.Database().BulkInsert(datatable);

}

#endregion

#region 事务

/// <summary>

/// 事务开始

/// </summary>

/// <returns></returns>

public DbTransaction BeginTrans()

{

return DataFactory.Database().BeginTrans();

}

/// <summary>

/// 提交事务

/// </summary>

public void Commit()

{

DataFactory.Database().Commit();

}

/// <summary>

/// 回滚事务

/// </summary>

public void Rollback()

{

DataFactory.Database().Rollback();

}

/// <summary>

/// 关闭数据库连接

/// </summary>

public void Close()

{

DataFactory.Database().Close();

}

#endregion

#region 执行SQL语句

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql)

{

return DataFactory.Database().ExecuteBySql(strSql);

}

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql, DbTransaction isOpenTrans)

{

return DataFactory.Database().ExecuteBySql(strSql, isOpenTrans);

}

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters)

{

return DataFactory.Database().ExecuteBySql(strSql, parameters);

}

/// <summary>

/// 执行SQL语句

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteBySql(StringBuilder strSql, DbParameter[] parameters, DbTransaction isOpenTrans)

{

return DataFactory.Database().ExecuteBySql(strSql, parameters, isOpenTrans);

}

#endregion

#region 执行存储过程

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

public int ExecuteByProc(string procName)

{

return DataFactory.Database().ExecuteByProc(procName);

}

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteByProc(string procName, DbTransaction isOpenTrans)

{

return DataFactory.Database().ExecuteByProc(procName, isOpenTrans);

}

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int ExecuteByProc(string procName, DbParameter[] parameters)

{

return DataFactory.Database().ExecuteByProc(procName, parameters);

}

/// <summary>

/// 执行存储过程

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int ExecuteByProc(string procName, DbParameter[] parameters, DbTransaction isOpenTrans)

{

return DataFactory.Database().ExecuteByProc(procName, parameters, isOpenTrans);

}

#endregion

#region 插入数据

/// <summary>

/// 插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <returns></returns>

public int Insert(T entity)

{

return DataFactory.Database().Insert(entity);

}

/// <summary>

/// 插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Insert(T entity, DbTransaction isOpenTrans)

{

return DataFactory.Database().Insert(entity, isOpenTrans);

}

/// <summary>

/// 批量插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <returns></returns>

public int Insert(List<T> entity)

{

return DataFactory.Database().Insert(entity);

}

/// <summary>

/// 批量插入数据

/// </summary>

/// <param name="entity">实体类对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Insert(List<T> entity, DbTransaction isOpenTrans)

{

return DataFactory.Database().Insert(entity, isOpenTrans);

}

#endregion

#region 修改数据

/// <summary>

/// 修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <returns></returns>

public int Update(T entity)

{

return DataFactory.Database().Update(entity);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update(T entity, DbTransaction isOpenTrans)

{

return DataFactory.Database().Update(entity, isOpenTrans);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public int Update(string propertyName, string propertyValue)

{

return DataFactory.Database().Update<T>(propertyName, propertyValue);

}

/// <summary>

/// 修改数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update(string propertyName, string propertyValue, DbTransaction isOpenTrans)

{

return DataFactory.Database().Update<T>(propertyName, propertyValue, isOpenTrans);

}

/// <summary>

/// 批量修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <returns></returns>

public int Update(List<T> entity)

{

return DataFactory.Database().Update(entity);

}

/// <summary>

/// 批量修改数据

/// </summary>

/// <param name="entity">实体对象</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Update(List<T> entity, DbTransaction isOpenTrans)

{

return DataFactory.Database().Update(entity, isOpenTrans);

}

#endregion

#region 删除数据

/// <summary>

/// 删除数据

/// </summary>

/// <param name="entity">实体类</param>

/// <returns></returns>

public int Delete(T entity)

{

return DataFactory.Database().Delete(entity);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="entity">实体类</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(T entity, DbTransaction isOpenTrans)

{

return DataFactory.Database().Delete(entity, isOpenTrans);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <returns></returns>

public int Delete(object propertyValue)

{

return DataFactory.Database().Delete<T>(propertyValue);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(object propertyValue, DbTransaction isOpenTrans)

{

return DataFactory.Database().Delete<T>(propertyValue, isOpenTrans);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public int Delete(string propertyName, string propertyValue)

{

return DataFactory.Database().Delete<T>(propertyName, propertyValue);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(string propertyName, string propertyValue, DbTransaction isOpenTrans)

{

return DataFactory.Database().Delete<T>(propertyName, propertyValue, isOpenTrans);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">键值生成SQL条件</param>

/// <returns></returns>

public int Delete(string tableName, Hashtable ht)

{

return DataFactory.Database().Delete(tableName, ht);

}

/// <summary>

/// 删除数据

/// </summary>

/// <param name="tableName">表名</param>

/// <param name="ht">键值生成SQL条件</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(string tableName, Hashtable ht, DbTransaction isOpenTrans)

{

return DataFactory.Database().Delete(tableName, ht, isOpenTrans);

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyValue">主键值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

public int Delete(object[] propertyValue)

{

return DataFactory.Database().Delete<T>(propertyValue);

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyValue">主键值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(object[] propertyValue, DbTransaction isOpenTrans)

{

return DataFactory.Database().Delete<T>(propertyValue, isOpenTrans);

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <returns></returns>

public int Delete(string propertyName, object[] propertyValue)

{

return DataFactory.Database().Delete<T>(propertyName, propertyValue);

}

/// <summary>

/// 批量删除数据

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值：数组1,2,3,4,5,6.....</param>

/// <param name="isOpenTrans">事务对象</param>

/// <returns></returns>

public int Delete(string propertyName, object[] propertyValue, DbTransaction isOpenTrans)

{

return DataFactory.Database().Delete<T>(propertyName, propertyValue, isOpenTrans);

}

#endregion

#region 查询数据列表、返回List

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

public List<T> FindListTop(int top)

{

return DataFactory.Database().FindListTop<T>(top);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public List<T> FindListTop(int top, string propertyName, string propertyValue)

{

return DataFactory.Database().FindListTop<T>(top, propertyName, propertyValue);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public List<T> FindListTop(int top, string whereSql)

{

return DataFactory.Database().FindListTop<T>(top, whereSql);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public List<T> FindListTop(int top, string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindListTop<T>(top, whereSql, parameters);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <returns></returns>

public List<T> FindList()

{

return DataFactory.Database().FindList<T>();

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public List<T> FindList(string propertyName, string propertyValue)

{

return DataFactory.Database().FindList<T>(propertyName, propertyValue);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public List<T> FindList(string whereSql)

{

return DataFactory.Database().FindList<T>(whereSql);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public List<T> FindList(string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindList<T>(whereSql, parameters);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public List<T> FindListBySql(string strSql)

{

return DataFactory.Database().FindListBySql<T>(strSql);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public List<T> FindListBySql(string strSql, DbParameter[] parameters)

{

return DataFactory.Database().FindListBySql<T>(strSql, parameters);

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public List<T> FindListPage(ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var list = DataFactory.Database().FindListPage<T>(orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return list;

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public List<T> FindListPage(string whereSql, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var list = DataFactory.Database()

.FindListPage<T>(whereSql, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return list;

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public List<T> FindListPage(string whereSql, DbParameter[] parameters, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var list = DataFactory.Database()

.FindListPage<T>(whereSql, parameters, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return list;

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public List<T> FindListPageBySql(string strSql, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var list = DataFactory.Database()

.FindListPageBySql<T>(strSql, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return list;

}

/// <summary>

/// 查询数据列表、返回List

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters"></param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public List<T> FindListPageBySql(string strSql, DbParameter[] parameters, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var list = DataFactory.Database()

.FindListPageBySql<T>(strSql, parameters, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return list;

}

#endregion

#region 查询数据列表、返回DataTable

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <returns></returns>

public DataTable FindTableTop(int top)

{

return DataFactory.Database().FindTableTop<T>(top);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public DataTable FindTableTop(int top, string whereSql)

{

return DataFactory.Database().FindTableTop<T>(top, whereSql);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="top">显示条数</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTableTop(int top, string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindTableTop<T>(top, whereSql, parameters);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <returns></returns>

public DataTable FindTable()

{

return DataFactory.Database().FindTable<T>();

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public DataTable FindTable(string whereSql)

{

return DataFactory.Database().FindTable<T>(whereSql);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTable(string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindTable<T>(whereSql, parameters);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public DataTable FindTableBySql(string strSql)

{

return DataFactory.Database().FindTableBySql(strSql);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTableBySql(string strSql, DbParameter[] parameters)

{

return DataFactory.Database().FindTableBySql(strSql, parameters);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="jqgridparam"></param>

/// <returns></returns>

public DataTable FindTablePage(ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var dt = DataFactory.Database().FindTablePage<T>(orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return dt;

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public DataTable FindTablePage(string whereSql, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var dt = DataFactory.Database()

.FindTablePage<T>(whereSql, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return dt;

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public DataTable FindTablePage(string whereSql, DbParameter[] parameters, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var dt = DataFactory.Database()

.FindTablePage<T>(whereSql, parameters, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return dt;

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public DataTable FindTablePageBySql(string strSql, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var dt = DataFactory.Database()

.FindTablePageBySql(strSql, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return dt;

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <param name="jqgridparam">分页参数</param>

/// <returns></returns>

public DataTable FindTablePageBySql(string strSql, DbParameter[] parameters, ref JqGridParam jqgridparam)

{

var orderField = jqgridparam.sidx;

var orderType = jqgridparam.sord;

var pageIndex = jqgridparam.page;

var pageSize = jqgridparam.rows;

var totalRow = jqgridparam.records;

var dt = DataFactory.Database()

.FindTablePageBySql(strSql, parameters, orderField, orderType, pageIndex, pageSize, ref totalRow);

jqgridparam.records = totalRow;

return dt;

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

public DataTable FindTableByProc(string procName)

{

return DataFactory.Database().FindTableByProc(procName);

}

/// <summary>

/// 查询数据列表、返回 DataTable

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataTable FindTableByProc(string procName, DbParameter[] parameters)

{

return DataFactory.Database().FindTableByProc(procName, parameters);

}

#endregion

#region 查询数据列表、返回DataSet

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public DataSet FindDataSetBySql(string strSql)

{

return DataFactory.Database().FindDataSetBySql(strSql);

}

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataSet FindDataSetBySql(string strSql, DbParameter[] parameters)

{

return DataFactory.Database().FindDataSetBySql(strSql, parameters);

}

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="procName">存储过程</param>

/// <returns></returns>

public DataSet FindDataSetByProc(string procName)

{

return DataFactory.Database().FindDataSetByProc(procName);

}

/// <summary>

/// 查询数据列表、返回DataSet

/// </summary>

/// <param name="procName">存储过程</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public DataSet FindDataSetByProc(string procName, DbParameter[] parameters)

{

return DataFactory.Database().FindDataSetByProc(procName, parameters);

}

#endregion

#region 查询对象、返回实体

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="propertyValue">主键值</param>

/// <returns></returns>

public T FindEntity(object propertyValue)

{

return DataFactory.Database().FindEntity<T>(propertyValue);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// <returns></returns>

public T FindEntity(string propertyName, object propertyValue)

{

return DataFactory.Database().FindEntity<T>(propertyName, propertyValue);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public T FindEntityByWhere(string whereSql)

{

return DataFactory.Database().FindEntity<T>(whereSql);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public T FindEntityByWhere(string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindEntity<T>(whereSql, parameters);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public T FindEntityBySql(string strSql)

{

return DataFactory.Database().FindEntityBySql<T>(strSql);

}

/// <summary>

/// 查询对象、返回实体

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public T FindEntityBySql(string strSql, DbParameter[] parameters)

{

return DataFactory.Database().FindEntityBySql<T>(strSql, parameters);

}

#endregion

#region 查询数据、返回条数

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <returns></returns>

public int FindCount()

{

return DataFactory.Database().FindCount<T>();

}

/// <summary>

/// 查询数据、返回条数

/// <param name="propertyName">实体属性名称</param>

/// <param name="propertyValue">字段值</param>

/// </summary>

/// <returns></returns>

public int FindCount(string propertyName, string propertyValue)

{

return DataFactory.Database().FindCount<T>(propertyName, propertyValue);

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public int FindCount(string whereSql)

{

return DataFactory.Database().FindCount<T>(whereSql);

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int FindCount(string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindCount<T>(whereSql, parameters);

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public int FindCountBySql(string strSql)

{

return DataFactory.Database().FindCountBySql(strSql);

}

/// <summary>

/// 查询数据、返回条数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public int FindCountBySql(string strSql, DbParameter[] parameters)

{

return DataFactory.Database().FindCountBySql(strSql, parameters);

}

#endregion

#region 查询数据、返回最大数

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <returns></returns>

public object FindMax(string propertyName)

{

return DataFactory.Database().FindMax<T>(propertyName);

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="whereSql">条件</param>

/// <returns></returns>

public object FindMax(string propertyName, string whereSql)

{

return DataFactory.Database().FindMax<T>(propertyName, whereSql);

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="propertyName">实体属性名称</param>

/// <param name="whereSql">条件</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public object FindMax(string propertyName, string whereSql, DbParameter[] parameters)

{

return DataFactory.Database().FindMax<T>(propertyName, whereSql, parameters);

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <returns></returns>

public object FindMaxBySql(string strSql)

{

return DataFactory.Database().FindMaxBySql(strSql);

}

/// <summary>

/// 查询数据、返回最大数

/// </summary>

/// <param name="strSql">Sql语句</param>

/// <param name="parameters">sql语句对应参数</param>

/// <returns></returns>

public object FindMaxBySql(string strSql, DbParameter[] parameters)

{

return DataFactory.Database().FindMaxBySql(strSql, parameters);

}

#endregion

}

}

using AnJie.ERP.DataAccess;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Repository

{

/// <summary>

/// 通用的Repository工厂

/// </summary>

/// <typeparam name="T"></typeparam>

public class RepositoryFactory<T> where T : new()

{

/// <summary>

/// 定义通用的Repository

/// </summary>

/// <returns></returns>

public IRepository<T> Repository()

{

return new Repository<T>();

}

}

}

using AnJie.ERP.DataAccess;

using AnJie.ERP.Entity;

using AnJie.ERP.Repository;

using System;

using System.Collections.Generic;

using System.Data.Common;

using System.Linq;

using System.Text;

namespace AnJie.ERP.Business

{

/// <summary>

/// 模块按钮

/// </summary>

public class BaseButtonBll : RepositoryFactory<Base\_Button>

{

/// <summary>

/// 获取按钮列表

/// </summary>

/// <param name="moduleId">模块ID</param>

/// <param name="category">分类：1-工具栏，2：右击栏</param>

/// <returns></returns>

public List<Base\_Button> GetList(string moduleId, string category)

{

StringBuilder strSql = new StringBuilder();

strSql.Append("SELECT \* FROM Base\_Button WHERE 1=1");

strSql.Append(" AND ModuleId = @ModuleId ");

strSql.Append(" AND Category = @Category ");

strSql.Append(" ORDER BY SortCode ASC");

var parameter = new List<DbParameter>

{

DbFactory.CreateDbParameter("@ModuleId", moduleId),

DbFactory.CreateDbParameter("@Category", category)

};

return Repository().FindListBySql(strSql.ToString(), parameter.ToArray());

}

}

}

using AnJie.ERP.DataAccess;

using AnJie.ERP.Entity;

using AnJie.ERP.Repository;

using AnJie.ERP.Utilities;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Text;

namespace AnJie.ERP.Business

{

/// <summary>

/// 操作按钮权限表

/// </summary>

public class BaseButtonPermissionBll : RepositoryFactory<Base\_ButtonPermission>

{

/// <summary>

/// 按钮权限列表

/// </summary>

/// <param name="objectId">对象主键</param>

/// <param name="category">对象分类:1-部门2-角色</param>

/// <returns></returns>

public DataTable GetList(string objectId, string category)

{

StringBuilder strSql = new StringBuilder();

List<DbParameter> parameter = new List<DbParameter>();

if (!ManageProvider.Provider.Current().IsSystem)

{

strSql.Append(@"SELECT b.ButtonId ,

b.ModuleId ,

b.Code ,

b.FullName ,

b.Category ,

b.Icon ,

b.SortCode ,

cp.ModuleButtonId AS ObjectId

FROM Base\_Button b INNER JOIN ( SELECT DISTINCT ModuleButtonId FROM Base\_ButtonPermission");

strSql.Append(" WHERE ObjectId IN ('" + ManageProvider.Provider.Current().ObjectId.Replace(",", "','") +

"')) bp ON B.ButtonId = bp.ModuleButtonId");

strSql.Append(" LEFT JOIN ( SELECT DISTINCT ModuleButtonId FROM Base\_ButtonPermission");

strSql.Append(" WHERE ObjectId = @ObjectId ) cp ON cp.ModuleButtonId = b.ButtonId");

}

else

{

strSql.Append(@"SELECT b.ButtonId ,

b.ModuleId ,

b.Code ,

b.FullName ,

b.Category ,

b.Icon ,

b.SortCode ,

bp.ObjectId

FROM Base\_Button b

LEFT JOIN Base\_ButtonPermission bp ON bp.ModuleButtonId = b.ButtonId

AND bp.ObjectId = @ObjectId");

}

strSql.Append(" order by b.SortCode ASC");

parameter.Add(DbFactory.CreateDbParameter("@ObjectId", objectId));

return Repository().FindTableBySql(strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 加载按钮权限

/// </summary>

/// <param name="objectId">对象主键</param>

/// <param name="moduleId">模块主键</param>

/// <returns></returns>

public List<Base\_Button> GetButtonList(string objectId, string moduleId)

{

StringBuilder strSql = new StringBuilder();

List<DbParameter> parameter = new List<DbParameter>();

if (!ManageProvider.Provider.Current().IsSystem)

{

strSql.Append(@"SELECT DISTINCT B.\* FROM Base\_Button B");

strSql.Append(" INNER JOIN Base\_ButtonPermission BP ON B.ButtonId = BP.ModuleButtonId");

strSql.AppendFormat(" WHERE (BP.ObjectId = '{0}'", ManageProvider.Provider.Current().UserId);

strSql.Append(" OR ObjectId IN ('" + ManageProvider.Provider.Current().ObjectId.Replace(",", "','") +

"'))");

}

else

{

strSql.Append(@"SELECT \* FROM Base\_Button B WHERE 1=1 ");

}

strSql.Append(" AND B.ModuleId = @ModuleId");

strSql.Append(" ORDER BY B.SortCode ASC ");

parameter.Add(DbFactory.CreateDbParameter("@ModuleId", moduleId));

return DataFactory.Database().FindListBySql<Base\_Button>(strSql.ToString(), parameter.ToArray());

}

/// <summary>

/// 根据对象Id获取模块按钮权限列表

/// </summary>

/// <param name="objectId">对象ID</param>

/// <returns></returns>

public DataTable GetButtonePermission(string objectId)

{

StringBuilder strSql = new StringBuilder();

strSql.Append(@"SELECT \*

FROM ( SELECT m.ModuleId AS ID ,

m.ParentId ,

m.FullName ,

m.Icon ,

m.SortCode,

m.Category,

'模块' AS Sort

FROM Base\_Module m

LEFT JOIN Base\_ModulePermission mp ON mp.ModuleId = m.ModuleId");

strSql.Append(@" WHERE mp.ObjectId IN ('" + objectId.Replace(",", "','") + "')");

strSql.Append(@" UNION SELECT b.ButtonId AS ID ,

b.ModuleId AS ParentId ,

b.FullName ,

b.Icon ,

b.SortCode,

b.Category,

'按钮' AS Sort

FROM Base\_Button b

LEFT JOIN Base\_ButtonPermission bp ON bp.ModuleButtonId = b.ButtonId");

strSql.Append(" WHERE bp.ObjectId IN ('" + objectId.Replace(",", "','") + "')) A");

strSql.Append(" ORDER BY SortCode ASC ");

return Repository().FindTableBySql(strSql.ToString());

}

}

}

using AnJie.ERP.DataAccess;

using AnJie.ERP.Entity;

using AnJie.ERP.Repository;

using AnJie.ERP.Utilities;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Data.Common;

using System.Text;

namespace AnJie.ERP.Business

{

/// <summary>

/// 角色管理

/// </summary>

public class BaseRolesBll : RepositoryFactory<Base\_Roles>

{

/// <summary>

/// 根据公司id获取角色 列表

/// </summary>

/// <param name="companyId">公司ID</param>

/// <param name="jqgridparam">分页条件</param>

/// <returns></returns>

public DataTable GetPageList(string companyId, ref JqGridParam jqgridparam)

{

StringBuilder strSql = new StringBuilder();

List<DbParameter> parameter = new List<DbParameter>();

strSql.Append(@"SELECT \*

FROM ( SELECT r.RoleId, r.CompanyId, c.FullName AS CompanyName, r.Code,

r.FullName, ISNULL(U.Qty, 0) AS MemberCount, r.Category,

r.Enabled, r.SortCode, r.Remark

FROM Base\_Roles r

LEFT JOIN Base\_Company c ON c.CompanyId = r.CompanyId

LEFT JOIN ( SELECT COUNT(1) AS Qty, ObjectId

FROM Base\_ObjectUserRelation

WHERE Category = '2'

GROUP BY ObjectId

) U ON U.ObjectId = r.RoleId

) T

WHERE 1 = 1 ");

if (!string.IsNullOrEmpty(companyId))

{

strSql.Append(" AND CompanyId = @CompanyId");

parameter.Add(DbFactory.CreateDbParameter("@CompanyId", companyId));

}

if (!ManageProvider.Provider.Current().IsSystem)

{

strSql.Append(" AND ( RoleId IN ( SELECT ResourceId FROM Base\_DataScopePermission WHERE");

strSql.Append(" ObjectId IN ('" + ManageProvider.Provider.Current().ObjectId.Replace(",", "','") + "') ");

strSql.Append(" ) )");

}

return Repository().FindTablePageBySql(strSql.ToString(), parameter.ToArray(), ref jqgridparam);

}

}

}

using AnJie.ERP.Entity;

using AnJie.ERP.Utilities;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.IO;

using System.Linq;

using System.Text;

using System.Xml;

namespace AnJie.ERP.Business

{

/// <summary>

/// 代码生成器

/// </summary>

public class CodeMaticBll

{

private readonly BaseDataBaseBLL \_baseDatabasebll = new BaseDataBaseBLL();

/// <summary>

/// 加载所有数据表

/// </summary>

/// <returns></returns>

public XmlNodeList GetTableName()

{

XmlDocument myXmlDocument = new XmlDocument();

myXmlDocument.Load(DirFileHelper.MapPath("~/CodeMatic/AnJie.ERP.Framework.pdm"));

// 获取表格

string selectPath =

"/Model/\*[local-name()='RootObject' and namespace-uri()='object'][1]/\*[local-name()='Children' and namespace-uri()='collection'][1]/\*[local-name()='Model' and namespace-uri()='object'][1]/\*[local-name()='Tables' and namespace-uri()='collection'][1]";

XmlNodeList myXmlNodeList = myXmlDocument.SelectSingleNode(selectPath).ChildNodes;

return myXmlNodeList;

}

/// <summary>

/// 获取某一个表的所有字段

/// </summary>

/// <param name="tableCode">查询指定表</param>

/// <returns></returns>

public DataTable GetColumns(string tableCode)

{

StringBuilder strSql = new StringBuilder();

if (!string.IsNullOrEmpty(tableCode))

{

DataTable dt = new DataTable();

DataColumn dc1 = new DataColumn("column\_Name", Type.GetType("System.String"));

DataColumn dc2 = new DataColumn("comments", Type.GetType("System.String"));

DataColumn dc3 = new DataColumn("char\_col\_decl\_length", Type.GetType("System.String"));

DataColumn dc4 = new DataColumn("data\_type", Type.GetType("System.String"));

dt.Columns.Add(dc1);

dt.Columns.Add(dc2);

dt.Columns.Add(dc3);

dt.Columns.Add(dc4);

XmlDocument myXmlDocument = new XmlDocument();

myXmlDocument.Load(DirFileHelper.MapPath("~/CodeMatic/AnJie.ERP.Framework.pdm"));

// 获取表格

string selectPath =

"/Model/\*[local-name()='RootObject' and namespace-uri()='object'][1]/\*[local-name()='Children' and namespace-uri()='collection'][1]/\*[local-name()='Model' and namespace-uri()='object'][1]/\*[local-name()='Tables' and namespace-uri()='collection'][1]";

XmlNodeList myXmlNodeList = myXmlDocument.SelectSingleNode(selectPath).ChildNodes;

foreach (XmlNode myXmlNode in myXmlNodeList)

{

if (myXmlNode.ChildNodes[2].InnerText.Equals(tableCode))

{

XmlNodeList myXmlNodeList\_field = myXmlNode.ChildNodes[10].ChildNodes;

foreach (XmlNode myXmlNode\_field in myXmlNodeList\_field)

{

int count = myXmlNode\_field.ChildNodes.Count;

DataRow dr = dt.NewRow();

dr["column\_Name"] = myXmlNode\_field.ChildNodes[2].InnerText;

dr["comments"] = myXmlNode\_field.ChildNodes[1].InnerText;

dr["data\_type"] = myXmlNode\_field.ChildNodes[9].InnerText;

if (count > 9)

{

try

{

dr["char\_col\_decl\_length"] = myXmlNode\_field.ChildNodes[10].InnerText;

}

catch (Exception)

{

}

finally

{

}

}

dt.Rows.Add(dr);

}

break;

}

}

return dt;

}

return null;

}

/// <summary>

/// 获取某一个表的主键字段

/// </summary>

/// <param name="tableCode">查询指定表</param>

/// <returns></returns>

public string GetPrimaryKey(string tableCode)

{

StringBuilder strSql = new StringBuilder();

if (!string.IsNullOrEmpty(tableCode))

{

XmlDocument myXmlDocument = new XmlDocument();

myXmlDocument.Load(DirFileHelper.MapPath("~/CodeMatic/AnJie.ERP.Framework.pdm"));

// 获取表格

string selectPath =

"/Model/\*[local-name()='RootObject' and namespace-uri()='object'][1]/\*[local-name()='Children' and namespace-uri()='collection'][1]/\*[local-name()='Model' and namespace-uri()='object'][1]/\*[local-name()='Tables' and namespace-uri()='collection'][1]";

XmlNodeList myXmlNodeList = myXmlDocument.SelectSingleNode(selectPath).ChildNodes;

foreach (XmlNode myXmlNode in myXmlNodeList)

{

if (myXmlNode.ChildNodes[2].InnerText.Equals(tableCode))

{

XmlNodeList myXmlNodeList\_field = myXmlNode.ChildNodes[11].ChildNodes;

foreach (XmlNode myXmlNode\_field in myXmlNodeList\_field)

{

return myXmlNode\_field.ChildNodes[2].InnerText;

}

}

}

}

return "";

}

#region 代码拼接组件

#region C#实体数据类型

/// <summary>

/// C#实体数据类型

/// </summary>

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

/// <returns></returns>

public string FindModelsType(string name)

{

name = name.ToLower();

if (name == "int" || name == "number" || name == "integer" || name == "smallint")

{

return "int?";

}

else if (name == "tinyint")

{

return "byte?";

}

else if (name == "numeric" || name == "real" || name == "float")

{

return "Single?";

}

else if (name == "float")

{

return "float?";

}

else if (name == "decimal" || name == "number(8,2)")

{

return "decimal?";

}

else if (name == "char" || name == "varchar" || name == "nvarchar2" || name == "text" || name == "nchar" ||

name == "nvarchar" || name == "ntext")

{

return "string";

}

else if (name == "bit")

{

return "bool?";

}

else if (name == "datetime" || name == "date" || name == "smalldatetime")

{

return "DateTime?";

}

else if (name == "money" || name == "smallmoney")

{

return "double?";

}

else

{

return "string";

}

}

#endregion

#region 公共变量

/// <summary>

/// 类名备注

/// </summary>

public string ClassName { get; set; }

/// <summary>

/// 实体类名称

/// </summary>

public string EntityName { get; set; }

/// <summary>

/// 数据类名称

/// </summary>

public string ServiceName { get; set; }

/// <summary>

/// 业务类名称

/// </summary>

public string BusinessName { get; set; }

/// <summary>

/// 表单页名称

/// </summary>

public string PageFormName { get; set; }

/// <summary>

/// 表单页名称

/// </summary>

public string PageFormDetailName { get; set; }

/// <summary>

/// 列表页名称

/// </summary>

public string PageListName { get; set; }

/// <summary>

/// 控制器名称

/// </summary>

public string ControllerName { get; set; }

/// <summary>

/// 业务区域

/// </summary>

public string AreasName { get; set; }

/// <summary>

/// 作者

/// </summary>

public string Author { get; set; }

/// <summary>

/// 项目名称

/// </summary>

public string ProjectName { get; set; }

/// <summary>

/// 创建年份

/// </summary>

public string CreateYear { get; set; }

/// <summary>

/// 公司

/// </summary>

public string Company { get; set; }

/// <summary>

/// 创建日期

/// </summary>

public string CreateDate { get; set; }

#endregion

#region 生成实体类

/// <summary>

/// 生成实体类

/// </summary>

/// <param name="table">表</param>

/// <param name="Key">表主键</param>

/// <param name="Entity\_dt">字段</param>

/// <returns></returns>

public string CodeBuilder\_Entity(string table, string Key, DataTable Entity\_dt)

{

StringBuilder sb\_Entity = new StringBuilder();

//sb\_Entity.Append("//=====================================================================================\r\n");

//sb\_Entity.Append("// All Rights Reserved , Copyright @ " + Company + " " + CreateYear + "\r\n");

//sb\_Entity.Append("// Software Developers @ " + Company + " " + CreateYear + "\r\n");

//sb\_Entity.Append("//=====================================================================================\r\n\r\n");

sb\_Entity.Append("using AnJie.ERP.DataAccess.Attributes;\r\n");

sb\_Entity.Append("using AnJie.ERP.Utilities;\r\n");

sb\_Entity.Append("using System;\r\n");

sb\_Entity.Append("using System.ComponentModel;\r\n");

sb\_Entity.Append("using System.ComponentModel.DataAnnotations;\r\n");

sb\_Entity.Append("using System.Text;\r\n\r\n");

sb\_Entity.Append("namespace " + ProjectName + "\r\n");

sb\_Entity.Append("{\r\n");

sb\_Entity.Append(" /// <summary>\r\n");

sb\_Entity.Append(" /// " + ClassName + "\r\n");

//sb\_Entity.Append(" /// <author>\r\n");

//sb\_Entity.Append(" /// <name>" + Author + "</name>\r\n");

//sb\_Entity.Append(" /// <date>" + CreateDate + "</date>\r\n");

//sb\_Entity.Append(" /// </author>\r\n");

sb\_Entity.Append(" /// </summary>\r\n");

sb\_Entity.Append(" [Description(\"" + ClassName + "\")]\r\n");

sb\_Entity.Append(" [PrimaryKey(\"" + Key + "\")]\r\n");

sb\_Entity.Append(" public class " + EntityName + " : BaseEntity\r\n");

sb\_Entity.Append(" {\r\n");

if (!DataHelper.IsExistRows(Entity\_dt))

{

sb\_Entity.Append(" #region 获取/设置 字段值\r\n");

for (int i = 0; i < Entity\_dt.Rows.Count; i++)

{

string column = Entity\_dt.Rows[i]["column\_name"].ToString();

string datatype = FindModelsType(Entity\_dt.Rows[i]["data\_type"].ToString());

string comments = Entity\_dt.Rows[i]["comments"].ToString();

sb\_Entity.Append(" /// <summary>\r\n");

sb\_Entity.Append(" /// " + comments + "\r\n");

sb\_Entity.Append(" /// </summary>\r\n");

sb\_Entity.Append(" /// <returns></returns>\r\n");

sb\_Entity.Append(" [DisplayName(\"" + comments + "\")]\r\n");

sb\_Entity.Append(" public " + datatype + " " + column + " { get; set; }\r\n\r\n");

}

sb\_Entity.Append(" #endregion\r\n\r\n");

sb\_Entity.Append(" #region 扩展操作\r\n");

sb\_Entity.Append(" /// <summary>\r\n");

sb\_Entity.Append(" /// 新增调用\r\n");

sb\_Entity.Append(" /// </summary>\r\n");

sb\_Entity.Append(" public override void Create()\r\n");

sb\_Entity.Append(" {\r\n");

sb\_Entity.Append(" this." + Key + " = CommonHelper.GetGuid;\r\n");

sb\_Entity.Append(" " + IsCreateDate(Entity\_dt) + "");

sb\_Entity.Append(" " + IsCreateUserId(Entity\_dt) + "");

sb\_Entity.Append(" " + IsCreateUserName(Entity\_dt) + "");

sb\_Entity.Append(" }\r\n");

sb\_Entity.Append(" /// <summary>\r\n");

sb\_Entity.Append(" /// 编辑调用\r\n");

sb\_Entity.Append(" /// </summary>\r\n");

sb\_Entity.Append(" /// <param name=\"KeyValue\"></param>\r\n");

sb\_Entity.Append(" public override void Modify(string KeyValue)\r\n");

sb\_Entity.Append(" {\r\n");

sb\_Entity.Append(" this." + Key + " = KeyValue;\r\n");

sb\_Entity.Append(" " + IsModifyDate(Entity\_dt) + "");

sb\_Entity.Append(" " + IsModifyUserId(Entity\_dt) + "");

sb\_Entity.Append(" " + IsModifyUserName(Entity\_dt) + "");

sb\_Entity.Append(" }\r\n");

sb\_Entity.Append(" #endregion\r\n");

}

sb\_Entity.Append(" }\r\n");

sb\_Entity.Append("}");

WriteCodeBuilder(table + "\\" + EntityName + ".cs", sb\_Entity.ToString());

return sb\_Entity.ToString();

}

public string IsCreateDate(DataTable dt)

{

DataTable newdt = DataHelper.GetNewDataTable(dt, "column\_name = 'CreateDate'");

if (newdt.Rows.Count > 0)

{

return "this.CreateDate = DateTime.Now;\r\n";

}

return "";

}

public string IsCreateUserId(DataTable dt)

{

DataTable newdt = DataHelper.GetNewDataTable(dt, "column\_name = 'CreateUserId'");

if (newdt.Rows.Count > 0)

{

return "this.CreateUserId = ManageProvider.Provider.Current().UserId;\r\n";

}

return "";

}

public string IsCreateUserName(DataTable dt)

{

DataTable newdt = DataHelper.GetNewDataTable(dt, "column\_name = 'CreateUserName'");

if (newdt.Rows.Count > 0)

{

return "this.CreateUserName = ManageProvider.Provider.Current().UserName;\r\n";

}

return "";

}

public string IsModifyDate(DataTable dt)

{

DataTable newdt = DataHelper.GetNewDataTable(dt, "column\_name = 'ModifyDate'");

if (newdt.Rows.Count > 0)

{

return "this.ModifyDate = DateTime.Now;\r\n";

}

return "";

}

public string IsModifyUserId(DataTable dt)

{

DataTable newdt = DataHelper.GetNewDataTable(dt, "column\_name = 'ModifyUserId'");

if (newdt.Rows.Count > 0)

{

return "this.ModifyUserId = ManageProvider.Provider.Current().UserId;\r\n";

}

return "";

}

public string IsModifyUserName(DataTable dt)

{

DataTable newdt = DataHelper.GetNewDataTable(dt, "column\_name = 'ModifyUserName'");

if (newdt.Rows.Count > 0)

{

return "this.ModifyUserName = ManageProvider.Provider.Current().UserName;\r\n";

}

return "";

}

#endregion

#region 业务逻辑类

/// <summary>

/// 生成业务逻辑类

/// </summary>

/// <param name="table">表</param>

/// <returns></returns>

public string GetCodeBuilderBusiness(string table)

{

StringBuilder sb\_Business = new StringBuilder();

//sb\_Business.Append("//=====================================================================================\r\n");

//sb\_Business.Append("// All Rights Reserved , Copyright @ " + Company + " " + CreateYear + "\r\n");

//sb\_Business.Append("// Software Developers @ " + Company + " " + CreateYear + "\r\n");

//sb\_Business.Append("//=====================================================================================\r\n\r\n");

sb\_Business.Append("using AnJie.ERP.Entity;\r\n");

sb\_Business.Append("using AnJie.ERP.Repository;\r\n");

sb\_Business.Append("using AnJie.ERP.Utilities;\r\n");

sb\_Business.Append("using System.Collections;\r\n");

sb\_Business.Append("using System.Collections.Generic;\r\n");

sb\_Business.Append("using System.Text;\r\n\r\n");

sb\_Business.Append("namespace " + ProjectName + "\r\n");

sb\_Business.Append("{\r\n");

sb\_Business.Append(" /// <summary>\r\n");

sb\_Business.Append(" /// " + ClassName + "\r\n");

sb\_Business.Append(" /// <author>\r\n");

sb\_Business.Append(" /// <name>" + Author + "</name>\r\n");

sb\_Business.Append(" /// <date>" + CreateDate + "</date>\r\n");

sb\_Business.Append(" /// </author>\r\n");

sb\_Business.Append(" /// </summary>\r\n");

sb\_Business.Append(" public class " + BusinessName + " : RepositoryFactory<" + EntityName + ">\r\n");

sb\_Business.Append(" {\r\n");

sb\_Business.Append(" }\r\n");

sb\_Business.Append("}");

WriteCodeBuilder(table + "\\" + BusinessName + ".cs", sb\_Business.ToString());

return sb\_Business.ToString();

}

#endregion

#region 页面表单

/// <summary>

/// 页面表单

/// </summary>

/// <param name="table">表名</param>

/// <param name="FromJson">显示表单字段</param>

/// <param name="ColumnCount">显示列数模式</param>

/// <param name="FormCss">表单css样式</param>

/// <returns></returns>

public string GetCodeBuilderFrom(string table, string FromJson, int ColumnCount, string FormCss)

{

StringBuilder sb\_From = new StringBuilder();

sb\_From.Append("@{\r\n");

sb\_From.Append(" ViewBag.Title = \"" + ClassName + "》表单页面\";\r\n");

sb\_From.Append(" Layout = \"~/Views/Shared/\_LayoutForm.cshtml\";\r\n");

sb\_From.Append("}\r\n");

sb\_From.Append("<script type=\"text/javascript\">\r\n");

sb\_From.Append(" var KeyValue = GetQuery('KeyValue');//主键\r\n");

sb\_From.Append(" $(function () {\r\n");

sb\_From.Append(" InitControl();\r\n");

sb\_From.Append(" })\r\n");

sb\_From.Append(" //得到一个对象实体\r\n");

sb\_From.Append(" function InitControl() {\r\n");

sb\_From.Append(" if (!!GetQuery('KeyValue')) {\r\n");

sb\_From.Append(" AjaxJson(\"/" + AreasName + "/" + StringHelper.DelLastLength(ControllerName, 10) +

"/SetForm\", { KeyValue: KeyValue }, function (data) {\r\n");

sb\_From.Append(" SetWebControls(data);\r\n");

sb\_From.Append(" });\r\n");

sb\_From.Append(" }\r\n");

sb\_From.Append(" }\r\n");

sb\_From.Append(" //保存事件\r\n");

sb\_From.Append(" function AcceptClick() {\r\n");

sb\_From.Append(" if (!CheckDataValid('#form1')) {\r\n");

sb\_From.Append(" return false;\r\n");

sb\_From.Append(" }\r\n");

sb\_From.Append(" var postData = GetWebControls(\"#form1\");\r\n");

sb\_From.Append(" AjaxJson(\"/" + AreasName + "/" + StringHelper.DelLastLength(ControllerName, 10) +

"/SubmitForm?KeyValue=\" + KeyValue, postData, function (data) {\r\n");

sb\_From.Append(" tipDialog(data.Message, 3, data.Code);\r\n");

sb\_From.Append(" top.frames[tabiframeId()].windowload();\r\n");

sb\_From.Append(" closeDialog();\r\n");

sb\_From.Append(" });\r\n");

sb\_From.Append(" }\r\n");

sb\_From.Append("</script>\r\n");

sb\_From.Append("<form id=\"form1\" style=\"margin: 1px\">\r\n");

sb\_From.Append(" <div id=\"message\" style=\"display: none\"></div>\r\n");

List<Base\_FormAttribute> ListData = new List<Base\_FormAttribute>();

ListData = (from itementity in FromJson.JonsToList<Base\_FormAttribute>()

where itementity.Enabled == 1

orderby itementity.SortCode ascending

select itementity).ToList<Base\_FormAttribute>();

sb\_From.Append(" " +

Base\_FormAttributeBll.Instance.CreateBuildFormTable(ColumnCount, ListData)

.Replace("Build\_", ""));

sb\_From.Append("</form>\r\n");

WriteCodeBuilder(

table + "\\" + StringHelper.DelLastLength(ControllerName, 10) + "\\" + PageFormName + ".cshtml",

sb\_From.ToString());

return sb\_From.ToString();

}

#endregion

#region 页面明细

/// <summary>

/// 页面明细

/// </summary>

/// <param name="table">表名</param>

/// <param name="fromJson">显示表单字段</param>

/// <param name="columnCount">显示列数模式</param>

/// <param name="formCss">表单css样式</param>

/// <returns></returns>

public string GetCodeBuilderFromDetail(string table, string fromJson, int columnCount, string formCss)

{

StringBuilder sbFrom = new StringBuilder();

sbFrom.Append("@{\r\n");

sbFrom.Append(" ViewBag.Title = \"" + ClassName + "》明细页面\";\r\n");

sbFrom.Append(" Layout = \"~/Views/Shared/\_LayoutForm.cshtml\";\r\n");

sbFrom.Append("}\r\n");

sbFrom.Append("<script type=\"text/javascript\">\r\n");

sbFrom.Append(" $(function () {\r\n");

sbFrom.Append(" InitControl();\r\n");

sbFrom.Append(" })\r\n");

sbFrom.Append(" //得到一个对象实体\r\n");

sbFrom.Append(" function InitControl() {\r\n");

sbFrom.Append(" if (!!GetQuery('KeyValue')) {\r\n");

sbFrom.Append(" AjaxJson(\"/" + AreasName + "/" + StringHelper.DelLastLength(ControllerName, 10) +

"/SetForm\", { KeyValue: GetQuery('KeyValue') }, function (data) {\r\n");

sbFrom.Append(" SetWebControls(data);\r\n");

sbFrom.Append(" });\r\n");

sbFrom.Append(" }\r\n");

sbFrom.Append(" }\r\n");

sbFrom.Append("</script>\r\n");

sbFrom.Append("<form id=\"form1\" style=\"margin: 1px\">\r\n");

var listData = (from itementity in fromJson.JonsToList<Base\_FormAttribute>()

where itementity.Enabled == 1

orderby itementity.SortCode ascending

select itementity).ToList<Base\_FormAttribute>();

sbFrom.Append(" " +

Base\_FormAttributeBll.Instance.CreateBuildFormTable(columnCount, listData)

.Replace("Build\_", ""));

sbFrom.Append("</form>\r\n");

WriteCodeBuilder(

table + "\\" + StringHelper.DelLastLength(ControllerName, 10) + "\\" + PageFormDetailName + ".cshtml",

sbFrom.ToString());

return sbFrom.ToString();

}

#endregion

#region 页面列表

/// <summary>

/// 表头显示/隐藏

/// </summary>

/// <param name="field"></param>

/// <returns></returns>

public string IsShow\_Field(string field)

{

if (field == "1")

{

return ",hidden: true";

}

return "";

}

/// <summary>

/// 页面表格列表

/// </summary>

/// <param name="table">表名</param>

/// <param name="showFieldJson">显示字段</param>

/// <param name="allowOrder">是否排序</param>

/// <param name="orderType">排序类型</param>

/// <param name="orderField">排序字段</param>

/// <param name="allowPageing">是否分页</param>

/// <param name="pageSize">分页大小</param>

/// <param name="pageLayout">页面布局</param>

/// <param name="method">操作</param>

/// <returns></returns>

public string GetCodeBuilderList(string table,

string showFieldJson, string allowOrder, string orderType, string orderField, string allowPageing,

string pageSize, string pageLayout, Hashtable method)

{

string PrimaryKeyColumns = \_baseDatabasebll.GetPrimaryKey(table);

StringBuilder sb\_List = new StringBuilder();

StringBuilder sb\_colModel = new StringBuilder();

List<JqGridColumn> ListData = showFieldJson.JonsToList<JqGridColumn>();

var query = from entity in ListData

orderby entity.SortCode ascending

/\*descending\*/

select entity;

if (ListData != null)

{

foreach (JqGridColumn item in query)

{

if (item.label == null && item.name == null)

{

continue;

}

string propertyName = item.label;

string controlId = item.name;

int width = item.width;

string align = item.align;

string hidden = item.hidden;

string sortable = item.sortable == "1" ? "true" : "false";

string Formatter = item.formatter;

int Enabled = CommonHelper.GetInt(item.Enabled);

if (Enabled == 1)

{

sb\_colModel.Append(" { label: '" + propertyName + "', name: '" + controlId +

"', index: '" + controlId + "', width: " + width + ", align: '" + align +

"',sortable: " + sortable + " " + IsShow\_Field(hidden) + " },\r\n");

}

}

}

sb\_List.Append("@{\r\n");

sb\_List.Append(" ViewBag.Title = \"" + ClassName + "\";\r\n");

sb\_List.Append(" Layout = \"~/Views/Shared/\_LayoutIndex.cshtml\";\r\n");

sb\_List.Append("}\r\n");

sb\_List.Append("<script type=\"text/javascript\">\r\n");

sb\_List.Append(" $(function () {\r\n");

sb\_List.Append(" GetGrid();\r\n");

sb\_List.Append(" })\r\n");

sb\_List.Append(" //加载表格\r\n");

sb\_List.Append(" function GetGrid() {\r\n");

sb\_List.Append(" $(\"#gridTable\").jqGrid({\r\n");

if (allowPageing == "1")

{

sb\_List.Append(" url: \"@Url.Content(\"~/" + AreasName + "/" +

StringHelper.DelLastLength(ControllerName, 10) + "/GridPageJson\")\",\r\n");

}

else

{

sb\_List.Append(" url: \"@Url.Content(\"~/" + AreasName + "/" +

StringHelper.DelLastLength(ControllerName, 10) + "/GridJson\")\",\r\n");

}

sb\_List.Append(" datatype: \"json\",\r\n");

sb\_List.Append(" height: $(window).height() - 149,\r\n");

sb\_List.Append(" autowidth: true,\r\n");

sb\_List.Append(" colModel: [\r\n" + sb\_colModel + " ],\r\n");

if (allowPageing == "1")

{

sb\_List.Append(" pager: \"#gridPager\",\r\n");

}

else

{

sb\_List.Append(" pager: false,\r\n");

}

sb\_List.Append(" sortname: '" + orderField + "',\r\n");

sb\_List.Append(" sortorder: '" + orderType + "',\r\n");

sb\_List.Append(" rownumbers: true,\r\n");

sb\_List.Append(" shrinkToFit: false,\r\n");

sb\_List.Append(" gridview: true,\r\n");

sb\_List.Append(" });\r\n");

sb\_List.Append("}\r\n");

if (method["AllowInsert"].ToString() == "checked")

{

sb\_List.Append(" //新增\r\n");

sb\_List.Append(" function btn\_add() {\r\n");

sb\_List.Append(" var url = \"/" + AreasName + "/" +

StringHelper.DelLastLength(ControllerName, 10) + "/Form\"\r\n");

sb\_List.Append(" openDialog(url, \"Form\", \"新增" + ClassName +

"\", 770, 395, function (iframe) {\r\n");

sb\_List.Append(" top.frames[iframe].AcceptClick()\r\n");

sb\_List.Append(" });\r\n");

sb\_List.Append(" }\r\n");

}

if (method["AllowUpdate"].ToString() == "checked")

{

sb\_List.Append(" //编辑\r\n");

sb\_List.Append(" function btn\_edit() {\r\n");

sb\_List.Append(" var KeyValue = GetJqGridRowValue(\"#gridTable\", \"" + PrimaryKeyColumns +

"\");\r\n");

sb\_List.Append(" if (IsChecked(KeyValue)) {\r\n");

sb\_List.Append(" var url = \"/" + AreasName + "/" +

StringHelper.DelLastLength(ControllerName, 10) + "/Form?KeyValue=\" + KeyValue;\r\n");

sb\_List.Append(" openDialog(url, \"Form\", \"编辑" + ClassName +

"\", 770, 395, function (iframe) {\r\n");

sb\_List.Append(" top.frames[iframe].AcceptClick();\r\n");

sb\_List.Append(" });\r\n");

sb\_List.Append(" }\r\n");

sb\_List.Append(" }\r\n");

}

if (method["AllowDelete"].ToString() == "checked")

{

sb\_List.Append(" //删除\r\n");

sb\_List.Append(" function btn\_delete() {\r\n");

sb\_List.Append(" var KeyValue = GetJqGridRowValue(\"#gridTable\", \"" + PrimaryKeyColumns +

"\");\r\n");

sb\_List.Append(" if (IsSelectData(KeyValue)) {\r\n");

sb\_List.Append(" var delparm = 'KeyValue=' + KeyValue;\r\n");

sb\_List.Append(" delConfig('/" + AreasName + "/" +

StringHelper.DelLastLength(ControllerName, 10) +

"/Delete', delparm, KeyValue.split(\",\").length);\r\n");

sb\_List.Append(" }\r\n");

sb\_List.Append(" }\r\n");

}

if (method["AlloLookup"].ToString() == "checked")

{

sb\_List.Append(" //明细\r\n");

sb\_List.Append(" function btn\_detail() {\r\n");

sb\_List.Append(" var KeyValue = GetJqGridRowValue(\"#gridTable\", \"" + PrimaryKeyColumns +

"\");\r\n");

sb\_List.Append(" if (IsChecked(KeyValue)) {\r\n");

sb\_List.Append(" var url = \"/" + AreasName + "/" +

StringHelper.DelLastLength(ControllerName, 10) + "/Detail?KeyValue=\" + KeyValue;\r\n");

sb\_List.Append(" Dialog(url, \"Detail\", \"" + ClassName +

"明细\", 820, 500, function (iframe) {\r\n");

sb\_List.Append(" top.frames[iframe].AcceptClick();\r\n");

sb\_List.Append(" });\r\n");

sb\_List.Append(" }\r\n");

sb\_List.Append(" }\r\n");

}

sb\_List.Append(" //刷新\r\n");

sb\_List.Append(" function windowload() {\r\n");

sb\_List.Append(" $(\"#gridTable\").trigger(\"reloadGrid\"); //重新载入\r\n");

sb\_List.Append(" \r\n");

sb\_List.Append(" }\r\n");

sb\_List.Append("</script>\r\n");

if (pageLayout == "1") //显示列表

{

#region 显示列表

sb\_List.Append(

"<div class=\"leftline rightline QueryArea\" style=\"margin: 1px; margin-top: 0px; margin-bottom: 0px;\">\r\n");

sb\_List.Append(" <table border=\"0\" class=\"form-find\" style=\"height: 45px;\">\r\n");

sb\_List.Append(" <tr>\r\n");

sb\_List.Append(" <th>关键字：</th>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"keywords\" type=\"text\" class=\"txt\" style=\"width: 200px\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"btnSearch\" type=\"button\" class=\"btnSearch\" value=\"搜 索\" onclick=\"btn\_Search()\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" </tr>\r\n");

sb\_List.Append(" </table>\r\n");

sb\_List.Append("</div>\r\n");

sb\_List.Append("<div class=\"topline rightline\" style=\"margin: 1px; margin-top: -1px;\">\r\n");

sb\_List.Append(" <table id=\"gridTable\"></table>\r\n");

if (allowPageing == "1") //分页

{

sb\_List.Append(" <div id=\"gridPager\"></div>\r\n");

}

sb\_List.Append("</div>\r\n");

#endregion

}

else if (pageLayout == "2") //显示列表+工具栏按钮

{

#region 显示列表+工具栏按钮

sb\_List.Append("<!--工具栏-->\r\n");

sb\_List.Append(

"<div class=\"tools\_bar leftline rightline\" style=\"margin: 1px; margin-bottom: 0px;\">\r\n");

sb\_List.Append(" <div class=\"PartialButton\">\r\n");

sb\_List.Append(" @Html.Partial(\"\_PartialButton\")\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append("</div>\r\n");

sb\_List.Append(

"<div class=\"leftline rightline QueryArea\" style=\"margin: 1px; margin-top: 0px; margin-bottom: 0px;\">\r\n");

sb\_List.Append(" <table border=\"0\" class=\"form-find\" style=\"height: 45px;\">\r\n");

sb\_List.Append(" <tr>\r\n");

sb\_List.Append(" <th>关键字：</th>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"keywords\" type=\"text\" class=\"txt\" style=\"width: 200px\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"btnSearch\" type=\"button\" class=\"btnSearch\" value=\"搜 索\" onclick=\"btn\_Search()\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" </tr>\r\n");

sb\_List.Append(" </table>\r\n");

sb\_List.Append("</div>\r\n");

sb\_List.Append("<div class=\"topline rightline\" style=\"margin: 1px; margin-top: -1px;\">\r\n");

sb\_List.Append(" <table id=\"gridTable\"></table>\r\n");

if (allowPageing == "1") //分页

{

sb\_List.Append(" <div id=\"gridPager\"></div>\r\n");

}

sb\_List.Append("</div>\r\n");

#endregion

}

else if (pageLayout == "3") //左边目录+显示列表

{

#region 左边目录+显示列表

sb\_List.Append("<div id=\"layout\" class=\"layout\">\r\n");

sb\_List.Append(" <!--左边-->\r\n");

sb\_List.Append(" <div class=\"layoutPanel layout-west\">\r\n");

sb\_List.Append(" <div class=\"btnbartitle\">\r\n");

sb\_List.Append(" <div>\r\n");

sb\_List.Append(" 组织结构\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <div class=\"ScrollBar\" id=\"ItemsTree\"></div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <!--中间-->\r\n");

sb\_List.Append(" <div class=\"layoutPanel layout-center\">\r\n");

sb\_List.Append(" <div class=\"btnbartitle\">\r\n");

sb\_List.Append(" <div>\r\n");

sb\_List.Append(" 用户列表 <span id=\"CenterTitle\"></span>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <!--列表-->\r\n");

sb\_List.Append(" <div id=\"grid\_List\">\r\n");

sb\_List.Append(

" <div class=\"bottomline QueryArea\" style=\"margin: 1px; margin-top: 0px; margin-bottom: 0px;\">\r\n");

sb\_List.Append(" <table border=\"0\" class=\"form-find\" style=\"height: 45px;\">\r\n");

sb\_List.Append(" <tr>\r\n");

sb\_List.Append(" <th>关键字：\r\n");

sb\_List.Append(" </th>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"keywords\" type=\"text\" class=\"txt\" style=\"width: 200px\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"btnSearch\" type=\"button\" class=\"btnSearch\" value=\"搜 索\" onclick=\"btn\_Search()\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" </tr>\r\n");

sb\_List.Append(" </table>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <table id=\"gridTable\"></table>\r\n");

if (allowPageing == "1") //分页

{

sb\_List.Append(" <div id=\"gridPager\"></div>\r\n");

}

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append("</div>\r\n");

#endregion

}

else if (pageLayout == "4") //左边目录+显示列表+工具栏按钮

{

#region 左边目录+显示列表+工具栏按钮

sb\_List.Append("<div id=\"layout\" class=\"layout\">\r\n");

sb\_List.Append(" <!--左边-->\r\n");

sb\_List.Append(" <div class=\"layoutPanel layout-west\">\r\n");

sb\_List.Append(" <div class=\"btnbartitle\">\r\n");

sb\_List.Append(" <div>\r\n");

sb\_List.Append(" 组织结构\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <div class=\"ScrollBar\" id=\"ItemsTree\"></div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <!--中间-->\r\n");

sb\_List.Append(" <div class=\"layoutPanel layout-center\">\r\n");

sb\_List.Append(" <div class=\"btnbartitle\">\r\n");

sb\_List.Append(" <div>\r\n");

sb\_List.Append(" 用户列表 <span id=\"CenterTitle\"></span>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <!--工具栏-->\r\n");

sb\_List.Append(" <div class=\"tools\_bar\" style=\"border-top: none; margin-bottom: 0px;\">\r\n");

sb\_List.Append(" <div class=\"PartialButton\">\r\n");

sb\_List.Append(" @Html.Partial(\"\_PartialButton\")\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <!--列表-->\r\n");

sb\_List.Append(" <div id=\"grid\_List\">\r\n");

sb\_List.Append(

" <div class=\"bottomline QueryArea\" style=\"margin: 1px; margin-top: 0px; margin-bottom: 0px;\">\r\n");

sb\_List.Append(" <table border=\"0\" class=\"form-find\" style=\"height: 45px;\">\r\n");

sb\_List.Append(" <tr>\r\n");

sb\_List.Append(" <th>关键字：\r\n");

sb\_List.Append(" </th>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"keywords\" type=\"text\" class=\"txt\" style=\"width: 200px\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" <td>\r\n");

sb\_List.Append(

" <input id=\"btnSearch\" type=\"button\" class=\"btnSearch\" value=\"搜 索\" onclick=\"btn\_Search()\" />\r\n");

sb\_List.Append(" </td>\r\n");

sb\_List.Append(" </tr>\r\n");

sb\_List.Append(" </table>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" <table id=\"gridTable\"></table>\r\n");

if (allowPageing == "1") //分页

{

sb\_List.Append(" <div id=\"gridPager\"></div>\r\n");

}

sb\_List.Append(" </div>\r\n");

sb\_List.Append(" </div>\r\n");

sb\_List.Append("</div>\r\n");

#endregion

}

WriteCodeBuilder(

table + "\\" + StringHelper.DelLastLength(ControllerName, 10) + "\\" + PageListName + ".cshtml",

sb\_List.ToString());

return sb\_List.ToString();

}

#endregion

#region 控制器

/// <summary>

/// 生成控制器

/// </summary>

/// <param name="table">表名</param>

/// <returns></returns>

public string GetCodeBuilderController(string table)

{

StringBuilder sbController = new StringBuilder();

sbController.Append("using AnJie.ERP.Business;\r\n");

sbController.Append("using AnJie.ERP.Entity;\r\n");

sbController.Append("using AnJie.ERP.Utilities;\r\n");

sbController.Append("using System;\r\n");

sbController.Append("using System.Collections;\r\n");

sbController.Append("using System.Collections.Generic;\r\n");

sbController.Append("using System.Data;\r\n");

sbController.Append("using System.Linq;\r\n");

sbController.Append("using System.Web;\r\n");

sbController.Append("using System.Web.Mvc;\r\n\r\n");

sbController.Append("namespace AnJie.ERP.WebApp.Areas." + AreasName + ".Controllers\r\n");

sbController.Append("{\r\n");

sbController.Append(" /// <summary>\r\n");

sbController.Append(" /// " + ClassName + "控制器\r\n");

sbController.Append(" /// </summary>\r\n");

sbController.Append(" public class " + ControllerName + " : PublicController<" + EntityName + ">\r\n");

sbController.Append(" {\r\n");

sbController.Append(" }\r\n");

sbController.Append("}");

WriteCodeBuilder(table + "\\" + ControllerName + ".cs", sbController.ToString());

string strFilePath = "~/CodeMatic/" + table;

string strZipPath = "~/CodeMatic/" + table + ".zip";

GZipHelper.ZipFile(strFilePath, strZipPath);

DirFileHelper.DeleteDirectory("~/CodeMatic/" + table);

return sbController.ToString();

}

#endregion

/// <summary>

/// 将生成代码写入文件

/// </summary>

/// <param name="filePath">路径</param>

/// <param name="code">代码</param>

public void WriteCodeBuilder(string filePath, string code)

{

string filepath = "/CodeMatic/" + filePath;

DirFileHelper.CreateFile(filepath, code);

}

#endregion

}

}