using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Xml;

using System.Runtime.Serialization;

namespace Huiting.Common

{

public interface IAddRange

{

void AddRange(ArrayList collection);

}

public interface IReSetValue

{

void ReSetValue(ArrayList collection);

}

/// <summary>

/// 迭代器

/// </summary>

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

[Serializable]

public class BEnumerator<T> : IEnumerator<T>

{

int curIndex;

T curObj;

protected IList<T> lstObj;

public BEnumerator(IList<T> LstObj)

{

lstObj = LstObj;

curIndex = -1;

if (lstObj == null)

return;

curObj = default(T);

}

public T Current

{

get { return curObj; }

}

public void Dispose()

{

}

object IEnumerator.Current

{

get { return Current; }

}

public bool MoveNext()

{

if (++curIndex >= lstObj.Count)

return false;

curObj = lstObj[curIndex];

return true;

}

public void Reset()

{

curIndex = -1;

}

}

[Serializable]

public class BasicQueue<T> :ADisposableExtend, IAddRange, ICloneable, IReSetValue, IEnumerable<T>, IList where T : class

{

protected List<T> lstData;

public T this[int index]

{

get

{

if (index < 0||index>=lstData.Count)

return null;

return lstData[index];

}

}

public BasicQueue()

{

lstData = new List<T>();

}

IEnumerator<T> IEnumerable<T>.GetEnumerator()

{

return new BEnumerator<T>(lstData);

}

public IEnumerator GetEnumerator()

{

return new BEnumerator<T>(lstData);

}

public void AddRange(T[] collection)

{

foreach (T t in collection)

{

if (t == null)

continue;

this.Add(t);

}

}

public void AddRange(ArrayList collection)

{

foreach (object item in collection)

{

T t = item as T;

if (t == null)

continue;

this.Add(t);

}

}

public void ReSetValue(ArrayList collection)

{

if (lstData != null)

lstData.Clear();

foreach (object item in collection)

{

T t = item as T;

if (t == null)

continue;

this.Add(t);

}

}

public void Add(T item)

{

if (lstData == null)

lstData = new List<T>();

//IParent<T> pt = item as IParent<T>;

//if (pt != null)

// pt.Parent = this;

lstData.Add(item);

}

public bool Remove(T item)

{

if (lstData.Contains(item))

{

lstData.Remove(item);

return true;

}

return false;

}

public bool RemoveAt(int index)

{

if (index <= lstData.Count - 1)

{

lstData.RemoveAt(index);

return true;

}

return false;

}

public int IndexOf(T t)

{

for (int i = 0; i < lstData.Count; i++)

{

T item = lstData[i];

if (item.Equals(t))

return i;

}

return -1;

}

public void Insert(int index, T item)

{

if (index <= lstData.Count)

lstData.Insert(index, item);

}

#region IList

int IList.Add(object value)

{

T t = value as T;

if (t != null)

{

this.Add(t);

return lstData.Count - 1;

}

return -1;

}

bool IList.Contains(object value)

{

T t = value as T;

if (t != null)

{

if (lstData.Contains(t))

return true;

}

return false;

}

int IList.IndexOf(object value)

{

T t = value as T;

if (t != null)

{

return this.IndexOf(t);

}

return -1;

}

void IList.Insert(int index, object value)

{

T t = value as T;

if (t != null)

{

if (index <= lstData.Count - 1)

lstData.Insert(index, t);

}

}

void IList.Remove(object value)

{

T t = value as T;

if (t != null)

{

if (lstData.Contains(t))

lstData.Remove(t);

}

}

public bool IsReadOnly

{

get { return false; }

}

public bool IsFixedSize

{

get { return false; }

}

void IList.RemoveAt(int index)

{

if (index <= lstData.Count - 1)

lstData.RemoveAt(index);

}

public void CopyTo(Array array, int index)

{

List<T> lstT = new List<T>();

lstT.ToArray();

foreach (object item in lstData)

{

T t = item as T;

if (t != null)

lstT.Add(t);

}

T[] tAry = new T[lstData.Count];

lstT.CopyTo(tAry, index);

array = tAry;

}

public T[] ToArray()

{

return lstData.ToArray();

}

public bool IsSynchronized

{

get { return false; }

}

public object SyncRoot

{

get { return null; }

}

object IList.this[int index]

{

get

{

return this[index];

}

set

{

}

}

public void Clear()

{

if (lstData == null)

return;

lstData.Clear();

}

public int Count

{

get { return lstData.Count; }

}

#endregion

public override string ToString()

{

string strResult = "";

foreach (T item in lstData)

{

if (item == null)

continue;

if (!string.IsNullOrEmpty(strResult))

strResult += ",";

strResult += item.ToString();

}

return strResult;

}

public object Clone()

{

return PublicMethods.DeepClone(this);

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Huiting.Common

{

public class DataGridViewBehaviors

{

/// <summary>

/// 格式化复制

/// </summary>

/// <param name="DataGridView\_cur">目标DataGridView对象</param>

public static void CopyFromDataGridView(DataGridView DataGridView\_cur)

{

DataGridViewSelectedCellCollection selCells = DataGridView\_cur.SelectedCells;

if (selCells.Count <= 0)

return;

int BeginRowIndex = selCells[0].RowIndex;

int BeginColumnIndex = selCells[0].ColumnIndex;

int EndRowIndex = selCells[selCells.Count - 1].RowIndex;

int EndColumnIndex = selCells[selCells.Count - 1].ColumnIndex;

foreach (DataGridViewCell curCell in selCells)

{

if (curCell.RowIndex < BeginRowIndex)

{

BeginRowIndex = curCell.RowIndex;

}

if (curCell.RowIndex > EndRowIndex)

{

EndRowIndex = curCell.RowIndex;

}

if (curCell.ColumnIndex < BeginColumnIndex)

{

BeginColumnIndex = curCell.ColumnIndex;

}

if (curCell.ColumnIndex > EndColumnIndex)

{

EndColumnIndex = curCell.ColumnIndex;

}

}

string RealStr = "";

string TempRowsStr = "";

string Value = "";

for (int i = BeginRowIndex; i <= EndRowIndex; i++)

{

if (!string.IsNullOrEmpty(RealStr))

{

RealStr += "\r\n";

}

TempRowsStr = "";

for (int a = BeginColumnIndex; a <= EndColumnIndex; a++)

{

DataRowView curRow = (DataRowView)DataGridView\_cur.Rows[i].DataBoundItem;

if (curRow == null)

{

continue;

}

if (curRow.Row.RowState == DataRowState.Detached)

{

continue;

}

if (!string.IsNullOrEmpty(TempRowsStr))

{

TempRowsStr += "\t";

}

Value = DataGridView\_cur.Rows[i].Cells[a].Value.ToString();

if (!string.IsNullOrEmpty(Value))

{

TempRowsStr += Value;

}

}

RealStr += TempRowsStr;

}

Clipboard.SetText(RealStr);

}

/// <summary>

/// 格式化粘贴

/// </summary>

/// <param name="sender">目标DataGridView对象</param>

/// <param name="e">如果走出目前行总数，是否自动追加行</param>

public static void PasterToDataGridView(DataGridView DataGridView\_cur, bool EnAutoAddRow)

{

string PasterStr = Clipboard.GetText();

DataTable dt\_CurDataSouce = null;

switch (DataGridView\_cur.DataSource.GetType().Name)

{

case "DataTable":

dt\_CurDataSouce = (DataTable)DataGridView\_cur.DataSource;

break;

case "BindingSource":

dt\_CurDataSouce = (DataTable)((BindingSource)DataGridView\_cur.DataSource).DataSource;

break;

default:

throw new Exception("未处理的数据源类型");

}

if (string.IsNullOrEmpty(PasterStr))

{

return;

}

if (DataGridView\_cur.SelectedCells.Count == 0)

{

PublicMethods.WarnMessageBox(DataGridView\_cur.FindForm(), "请选择要粘贴的单元格！");

//MyMethod.ShowMessage(DataGridView\_cur.FindForm(), "请选择要粘贴的单元格！");

return;

}

//int BeginRowIndex = DataGridView\_cur.CurrentCell.RowIndex;

//int BeginColumnIndex = DataGridView\_cur.CurrentCell.ColumnIndex;

int BeginRowIndex = DataGridView\_cur.SelectedCells[0].RowIndex;

int BeginColumnIndex = DataGridView\_cur.SelectedCells[0].ColumnIndex;

string[] PasterRows = PasterStr.Split('\r');

int RowsAddCount = -1;

int ColumnsAddCount = -1;

string ErrMsg = "";

DataGridView\_cur.CurrentCell = null;

foreach (string curRowStr in PasterRows)

{

if (curRowStr == "\n")

{

continue;

}

RowsAddCount++;

DataRow curRow = null;

if (((BeginRowIndex + RowsAddCount) >= DataGridView\_cur.Rows.Count) || ((DataRowView)DataGridView\_cur.Rows[BeginRowIndex + RowsAddCount].DataBoundItem == null))

{

if (!EnAutoAddRow)

{

break;

}

curRow = dt\_CurDataSouce.NewRow();

dt\_CurDataSouce.Rows.Add(curRow);

}

else

{

curRow = ((DataRowView)DataGridView\_cur.Rows[BeginRowIndex + RowsAddCount].DataBoundItem).Row;

}

string[] RowsList = curRowStr.Split('\t');

ColumnsAddCount = -1;

string RealValue = "";

string strCurColumnName = "";

foreach (string curCellValue in RowsList)

{

ColumnsAddCount++;

if ((BeginColumnIndex + ColumnsAddCount) >= DataGridView\_cur.ColumnCount)

{

break;

}

RealValue = curCellValue.Replace("\n", "");

try

{

strCurColumnName = DataGridView\_cur.Columns[BeginColumnIndex + ColumnsAddCount].DataPropertyName;

if (string.IsNullOrEmpty(RealValue))

{

curRow[strCurColumnName] = DBNull.Value;

}

else

{

curRow[strCurColumnName] = RealValue;

}

}

catch (System.Exception ex)

{

if (!string.IsNullOrEmpty(ErrMsg))

{

ErrMsg += "\n";

}

ErrMsg += "粘贴第" + (RowsAddCount + 1).ToString() + "行数据的[" + DataGridView\_cur.Columns[BeginColumnIndex + ColumnsAddCount].HeaderText + "]列时出错，错误：" + ex.Message;

}

curRow.EndEdit();

}

}

try

{

DataGridView\_cur.CurrentCell = DataGridView\_cur.Rows[BeginRowIndex].Cells[BeginColumnIndex];

}

catch

{

}

if (!string.IsNullOrEmpty(ErrMsg))

{

MessageBox.Show(ErrMsg);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.ComponentModel;

namespace Huiting.Common

{

public class EnumAttrDict<T, K>

where T : struct

where K : Attribute

{

private static EnumAttrDict<T, K> instance = new EnumAttrDict<T, K>();

public static EnumAttrDict<T, K> Instance

{

get { return EnumAttrDict<T, K>.instance; }

}

private Dictionary<T, K> dictStyle = new Dictionary<T, K>();

public Dictionary<T, K> Dictionary

{

get { return dictStyle; }

}

protected EnumAttrDict()

{

System.Reflection.FieldInfo[] fieldAry = typeof(T).GetFields();

foreach (System.Reflection.FieldInfo item in fieldAry)

{

object[] objAry = item.GetCustomAttributes(typeof(K), false);

if (objAry.Length > 0)

{

K da = objAry[0] as K;

dictStyle.Add((T)Enum.Parse(typeof(T), item.Name), da);

}

}

}

public K GetAttribute(T t)

{

if (dictStyle.ContainsKey(t))

return dictStyle[t];

else

return default(K);

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Text;

namespace Huiting.Common

{

[Description("图像处理器")]

[Serializable]

public class ImageOperator

{

private static ImageOperator instance = new ImageOperator();

public static ImageOperator Instance

{

get { return ImageOperator.instance; }

}

private ImageOperator()

{

}

public Bitmap ZoomBitmap(Bitmap originalImage, Size fixedSize, ImageChangedMode changedMode, bool disposeOriginalImage)

{

return ZoomBitmap(originalImage, fixedSize.Width, fixedSize.Height, changedMode, disposeOriginalImage);

}

/// <summary>

/// 获取缩放后图片

/// </summary>

/// <param name="originalImage">原始图片</param>

/// <param name="fixedWidth">固定宽度</param>

/// <param name="fixedHeight">固定高度</param>

/// <param name="changedMode">改变模式</param>

/// <param name="disposeOriginalImage">释放原始照片</param>

/// <returns></returns>

public Bitmap ZoomBitmap(Bitmap originalImage, int fixedWidth, int fixedHeight, ImageChangedMode changedMode, bool disposeOriginalImage)

{

int towidth = fixedWidth;

int toheight = fixedHeight;

int x = 0;

int y = 0;

int ow = originalImage.Width;

int oh = originalImage.Height;

switch (changedMode)

{

case ImageChangedMode.FixedWH://指定高宽缩放（可能变形）

break;

case ImageChangedMode.ZoomHByW://指定宽，高按比例

toheight = originalImage.Height \* fixedWidth / originalImage.Width;

break;

case ImageChangedMode.ZoomWByH://指定高，宽按比例

towidth = originalImage.Width \* fixedHeight / originalImage.Height;

break;

case ImageChangedMode.ZoomBySacle://根据源/目标 的宽高比例，哪个指标大，固定哪个指标，缩放另一个指标

double scaleW = originalImage.Width / fixedWidth;

double scaleH = originalImage.Height / fixedHeight;

if (scaleW > scaleH)

toheight = originalImage.Height \* fixedWidth / originalImage.Width;

else

towidth = originalImage.Width \* fixedHeight / originalImage.Height;

break;

case ImageChangedMode.CutByFixedWH://指定高宽裁减（不变形）

default:

if ((double)originalImage.Width / (double)originalImage.Height > (double)towidth / (double)toheight)

{

oh = originalImage.Height;

ow = originalImage.Height \* towidth / toheight;

y = 0;

x = (originalImage.Width - ow) / 2;

}

else

{

ow = originalImage.Width;

oh = originalImage.Width \* fixedHeight / towidth;

x = 0;

y = (originalImage.Height - oh) / 2;

}

break;

}

//新建一个bmp图片

System.Drawing.Bitmap bitmap = new System.Drawing.Bitmap(towidth, toheight);

//新建一个画板

Graphics g = System.Drawing.Graphics.FromImage(bitmap);

try

{

//设置高质量插值法

g.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.High;

//设置高质量,低速度呈现平滑程度

g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;

//清空画布并以透明背景色填充

g.Clear(Color.Transparent);

//在指定位置并且按指定大小绘制原图片的指定部分

g.DrawImage(originalImage, new Rectangle(0, 0, towidth, toheight),

new Rectangle(x, y, ow, oh),

GraphicsUnit.Pixel);

}

catch (System.Exception e)

{

throw e;

}

finally

{

if (disposeOriginalImage)

originalImage.Dispose();

g.Dispose();

}

return bitmap;

}

}

[Serializable]

public enum ImageChangedMode

{

[Description("固定宽高")]

FixedWH,

[Description("固定宽,同比缩放高")]

ZoomHByW,

[Description("固定高,同比缩放宽")]

ZoomWByH,

[Description("根据源/目标 的宽高比例，哪个指标大，固定哪个指标，缩放另一个指标")]

ZoomBySacle,

[Description("剪切")]

CutByFixedWH

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Huiting.Common

{

public static class ObjectConvert

{

public static int ToInt(this Color color)

{

return color.ToArgb();

}

public static Color ToColor(this int argb)

{

return Color.FromArgb(argb);

}

public static Color ToColor(this string argb)

{

return Color.FromArgb(int.Parse(argb));

}

public static string ToString(this Color color)

{

return color.ToArgb().ToString();

}

public static string ConvertToString(this Padding padding)

{

return padding.Left.ToString() + "," +

padding.Top.ToString() + "," +

padding.Right.ToString() + "," +

padding.Bottom.ToString();

}

public static Padding ToPadding(this string paddingStr)

{

Padding padding = new Padding();

string[] strAry = paddingStr.Split(new char[] { ',' });

if (strAry.Length != 4)

return padding;

padding = new Padding(int.Parse(strAry[0]), int.Parse(strAry[1]), int.Parse(strAry[2]), int.Parse(strAry[3]));

return padding;

}

public static Size ToMaxSize(this SizeF sizef)

{

int Width = (int)(Math.Ceiling(sizef.Width));

int Height = (int)(Math.Ceiling(sizef.Height));

return new Size(Width, Height);

}

public static Size ToMinSize(this SizeF sizef)

{

int Width = (int)(Math.Floor(sizef.Width));

int Height = (int)(Math.Floor(sizef.Height));

return new Size(Width, Height);

}

public static double ToDoubleWithDefaultZero(this Object obj)

{

return obj.ToDouble(0.0);

}

public static double ToDouble(this Object obj, double DefaultValue)

{

double db = DefaultValue;

if (obj == null || obj == System.DBNull.Value)

return db;

bool bol = double.TryParse(obj.ToString(), out db);

return db;

}

public static double ToDouble(this Object obj)

{

double db = 0;

if (obj == null || obj == System.DBNull.Value)

return db;

bool bol = double.TryParse(obj.ToString(), out db);

return db;

}

public static double? ToDoubleOrNull(this Object obj)

{

double? result = null;

if (obj == null || obj == System.DBNull.Value)

return result;

double curDB;

if (double.TryParse(obj.ToString(), out curDB) == false)

return result;

result = curDB;

return result;

}

public static double? ToDoubleOrNull(this Object obj, int decimalPlace)

{

double? result = default(double?);

if (obj == null || obj == System.DBNull.Value)

return result;

result = obj.ToDouble(decimalPlace);

return result;

}

public static double ToDouble(this Object obj, int decimalPlace)

{

double dblValue = obj.ToDouble();

return PublicMethods.Round(dblValue, decimalPlace);

}

public static decimal ToDecimal(this Object obj)

{

decimal db = 0;

if (obj == null)

return db;

bool bol = decimal.TryParse(obj.ToString(), out db);

return db;

}

// 获取日期格式

public static string GetDecimalFormat(int decimalPlace)

{

string format = "0";

for (int i = 0; i < decimalPlace; i++)

{

if (i == 0)

format += ".";

format += "#";

}

return format;

}

public static float ToFloat(this Object obj)

{

float db = float.NaN;

if (obj == null || obj == System.DBNull.Value)

return db;

bool bol = float.TryParse(obj.ToString(), out db);

if (bol)

return db;

else

return float.NaN;

}

////根据出现的可能性排序

//static string[] DateStringAry = {

// "yyyyMM",

// "yyyy/MM/dd","yyyy/MM/dd",

// "yyyy/MM",

// "yyyy/MM/dd hh:mm:ss",

// "yyyy/MM/dd h:mm:ss",

// "yyyy/MM/d h:mm:ss",

// "yyyy/MM/dd ddd hh:mm:ss",

// "yyyy/MM/dd/ddd hh:mm:ss",

// "yyyy/MM/dd ddd",

// "yyyy/MM/dd/ ddd",

// "yyyy",

// "yyyy-MM",

// "yyyy/M", "yyyy-M", "yyyyM",

// "MM/yyyy","MM-yyyy","MMyyyy",

// "M/yyyy", "M-yyyy", "Myyyy",

// "yyyy-MM-dd", "yyyyMMdd",

// "yyyy/M/d", "yyyy-M-d", "yyyyMd",

// "MM/dd/yyyy", "MM-dd-yyyy", "MMddyyyy",

// "M/d/yyyy", "M-d-yyyy", "Mdyyyy",

// "dd/MM/yyyy","dd-MM-yyyy","ddMMyyyy",

// "d/M/yyyy","d-M-yyyy","dMyyyy",

// };

////日期形式

//static string[] DateStringAry = {

// "yy",//年年

// "yyyy", //年年年年

// "yyMM", //年年月月

// "MMyy", //月月年年

// "yy-M", //年年-0月

// "M-yy", //0月-年年

// "yyyyM",

// "Myyyy",

// "yy-MM",

// "MM-yy",

// "yyyyMM",

// "MMyyyy",

// "yyMMdd",

// "MMddyy",

// "ddMMyy",

// "yyyy-M",

// "M-yyyy",

// "yyyy-MM",

// "MM-yyyy",

// "yy-MM-dd",

// "MM-dd-yy",

// "dd-MM-yy",

// "yyyyMMdd",

// "ddMMyyyy",

// "yyyy-M-d",

// "M-d-yyyy",

// "d-M-yyyy",

// "yyyy-MM-dd",

// "MM-dd-yyyy",

// "dd-MM-yyyy",

// "yyyy-M-d hh:mm",

// "M-d-yyyy hh:mm",

// "d-M-yyyy hh:mm",

// "yyyy-MM-dd hh:mm",

// "MM-dd-yyyy hh:mm",

// "dd-MM-yyyy hh:mm",

// "yyyy-MM-dd hh:mm:ss",

// "MM-dd-yyyy hh:mm:ss",

// "dd-MM-yyyy hh:mm:ss",

// "yyyyM",

// "yyyy-M",

// "Myyyy",

// "M-yyyy"

// };

//日期形式

static string[] DateStringAry = {

//年月

"yy-M", //年年-0月

"M-yy", //0月-年年

"yy-MM",

"MM-yy",

"yyyyMM",

"MMyyyy",

"yyyy-M",

"M-yyyy",

"yyyy-MM",

"MM-yyyy",

//年月日

"yyyyMMdd",

"MMddyyyy",

"ddMMyyyy",

"yyMMdd",

"MMddyy",

"ddMMyy",

"yyyy-M-d",

"M-d-yyyy",

"d-M-yyyy",

"yyyy-MM-dd",

"MM-dd-yyyy",

"dd-MM-yyyy",

//年月日时分

"yyyy-M-d h:m",

"M-d-yyyy h:m",

"d-M-yyyy h:m",

"yyyy-M-d hh:mm",

"M-d-yyyy hh:mm",

"d-M-yyyy hh:mm",

"yyyy-MM-dd hh:mm",

"MM-dd-yyyy hh:mm",

"dd-MM-yyyy hh:mm",

//年月日时分秒

"yyyy-M-d h:m:s",

"M-d-yyyy h:m:s",

"d-M-yyyy h:m:s",

"yyyy-M-d hh:mm:ss",

"M-d-yyyy hh:mm:ss",

"d-M-yyyy hh:mm:ss",

"yyyy-MM-dd hh:mm:ss",

"MM-dd-yyyy hh:mm:ss",

"dd-MM-yyyy hh:mm:ss",

};

public static DateTime ToDateTime(this object dateTime)

{

DateTime rightNY = DateTime.Now;

if (dateTime == null)

return rightNY;

//先用常归转换一次

string strDateTime = dateTime.ToString().Trim();

#region 处理字符串有包含非能决定日期的信息，比如周几或星期几等信息

//若关键字符串中包含“周”或"星期"，先替换成关键字周

strDateTime = strDateTime.Replace("星期", "周");

//若关键字符串中包含“周”，先判断前后字符

int index = strDateTime.IndexOf("周");

if (index >= 0)

{

string strBefore = "", strAfter = "";

if (index > 0)

strBefore = strDateTime.Substring(0, index - 1).Trim();

if (index + 2 < strDateTime.Length)

strAfter = strDateTime.Substring(index + 3, strDateTime.Length - index - 3).Trim();

strDateTime = strBefore + ' ' + strAfter;

}

#endregion

if (DateTime.TryParse(strDateTime, out rightNY))

return rightNY;

//将分隔符批量替换成‘-’形式

strDateTime = strDateTime.Replace('/', '-');

//bool bol = DateTime.TryParseExact(strDateTime, "yyyy-M-d h:mm:ss", null, DateTimeStyles.None, out rightNY);

if (DateTime.TryParseExact(strDateTime, DateStringAry, null, DateTimeStyles.None, out rightNY))

return rightNY;

return rightNY;

}

public static bool TryDateTime(this object dateTime)

{

DateTime rightNY;

return dateTime.TryDateTime(out rightNY);

}

public static bool TryDateTime(this object dateTime, out DateTime rightNY)

{

rightNY = DateTime.Now;

if (dateTime == null)

return false;

string strDateTime = dateTime.ToString().Trim();

strDateTime = strDateTime.Replace('/', '-');

if (DateTime.TryParse(strDateTime, out rightNY) == false)

{

if (DateTime.TryParseExact(strDateTime, DateStringAry, null, DateTimeStyles.None, out rightNY) == false)

return false;

}

return true;

}

public static int ToIntWithDefaultValue(this Object obj, int defaultValue)

{

if (obj == null)

return defaultValue;

int m\_int = defaultValue;

if (obj == null || obj == System.DBNull.Value)

return m\_int;

double dbl;

if (double.TryParse(obj.ToString(), out dbl))

return (int)dbl;

bool bol = Int32.TryParse(obj.ToString(), out m\_int);

if (bol)

return m\_int;

return defaultValue;

}

public static int ToInt(this Object obj)

{

return obj.ToIntWithDefaultValue(0);

}

public static int? ToIntOrNull(this Object obj)

{

if (obj == null || obj == null || obj == System.DBNull.Value)

return null;

return obj.ToInt();

}

public static bool ToBoolWithZeroIsFalse(this Object obj)

{

if (obj != null)

{

string strObj = obj.ToString();

if (strObj == "0" || string.IsNullOrEmpty(strObj))

return false;

}

return ToBool(obj);

}

public static bool ToBool(this Object obj)

{

if (obj == null)

return false;

string strObj = obj.ToString();

bool result;

//如果不能转换成bool值，但有值，则返回真

if (bool.TryParse(strObj, out result) == false)

return true;

return result;

}

public static T[] Insert<T>(this T[] ary, int index, T obj)

{

List<T> lstT = new List<T>(ary);

lstT.Insert(index, obj);

return lstT.ToArray();

}

public static void SetColumnNameToLower(this DataTable dt)

{

foreach (DataColumn dc in dt.Columns)

dc.ColumnName = dc.ColumnName.ToLower();

}

public static int IndexOf<T, D>(this Dictionary<T, D> dict, T keyValue)

{

int index = -1;

foreach (KeyValuePair<T, D> item in dict)

{

index++;

object objTmp = item.Key;

object objTmp2 = keyValue;

if (objTmp.ToString() == objTmp2.ToString())

return index;

}

return index;

}

/// <summary>

/// 给表添加拼音列

/// </summary>

/// <param name="dt"></param>

/// <param name="pyColumn"></param>

/// <param name="sourceColumn"></param>

public static void AddPYColumn(this DataTable dt, string pyColumn, string sourceColumn)

{

if (dt.Columns.Contains(pyColumn)==false)

dt.Columns.Add(pyColumn);

if (dt.Columns.Contains(sourceColumn) == false) return;

foreach (DataRow dr in dt.Rows)

dr[pyColumn] = ChineseToPinYin.ToPinYin(dr[sourceColumn].ToString(), null);

}

/// <summary>

/// 首字母小写写

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static string FirstCharToLower(this string input)

{

if (String.IsNullOrEmpty(input))

return input;

string str = input.First().ToString().ToLower() + input.Substring(1);

return str;

}

/// <summary>

/// 首字母大写

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static string FirstCharToUpper(this string input)

{

if (String.IsNullOrEmpty(input))

return input;

string str = input.First().ToString().ToUpper() + input.Substring(1);

return str;

}

}

}

using System;

using System.ComponentModel;

using System.Drawing;

using System.Globalization;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Windows.Forms;

using System.Xml;

using System.IO;

namespace Huiting.Common

{

public class ObjectSerializer

{

private static ObjectSerializer instance = new ObjectSerializer();

public static ObjectSerializer Instance

{

get { return ObjectSerializer.instance; }

}

private ObjectSerializer()

{

}

public bool WriteObjectToXmlFile<A>(object obj, string xmlFile) where A : Attribute

{

if (File.Exists(xmlFile))

File.Delete(xmlFile);

XmlDocument xmlDoc = new XmlDocument();

XmlDeclaration xmlDecl = xmlDoc.CreateXmlDeclaration("1.0", "gb2312", null);

xmlDoc.AppendChild(xmlDecl);

Type type = obj.GetType();

XmlElement xeRoot = xmlDoc.CreateElement(type.Name);

xmlDoc.AppendChild(xeRoot);

Serialize<A>(xeRoot, obj);

xmlDoc.Save(xmlFile);

return true;

}

public bool ReadObjectFromXmlFile<A>(object obj, string xmlFile) where A : Attribute

{

if (!File.Exists(xmlFile))

return false;

XmlDocument xmlDoc = new XmlDocument();

xmlDoc.Load(xmlFile);

if (xmlDoc == null || xmlDoc.ChildNodes.Count <= 0)

return false;

Type type = obj.GetType();

XmlNode xnRoot = xmlDoc.SelectSingleNode(type.Name);

DecSerialize<SerializableAttribute>(xnRoot, obj);

return true;

}

public bool Serialize<A>(XmlNode xnParent, object obj) where A : Attribute

{

Type typeClass = obj.GetType();

PropertyInfo[] piAryClass = typeClass.GetProperties(BindingFlags.Public | BindingFlags.Instance);

if (piAryClass.Length <= 0)

return false;

try

{

foreach (PropertyInfo item in piAryClass)

{

A attribute = PublicMethods.GetAttribute<A>(item);

if (attribute != null)

continue;

if (item.CanWrite == false)

continue;

string pName = item.Name;

object value = item.GetValue(obj, null);

XmlElement xeChild = xnParent.OwnerDocument.CreateElement(item.Name);

object objChildValue = item.GetValue(obj, null);

if (objChildValue == null)

continue;

TypeConverter converter = TypeDescriptor.GetConverter(item.PropertyType);

//如果能通过string与之相互转化，则转化

if (GetConversionSupported(converter, typeof(string)))

{

if (objChildValue != null)

{

xeChild.InnerText = objChildValue.ToString();

xnParent.AppendChild(xeChild);

}

}

else if (item.PropertyType.IsClass)

{

if (Serialize<A>(xeChild, objChildValue))

xnParent.AppendChild(xeChild);

}

else

{

}

}

}

catch (Exception ex)

{

return false;

}

return true;

}

public bool DecSerialize<A>(XmlNode xmlNode, object obj)

{

PropertyInfo[] piAry = obj.GetType().GetProperties(BindingFlags.Instance | BindingFlags.Public);

foreach (PropertyInfo item in piAry)

{

try

{

XmlNode xnChild = xmlNode.SelectSingleNode(item.Name);

if (xnChild == null)

continue;

object objChildValue = null;

TypeConverter converter = TypeDescriptor.GetConverter(item.PropertyType);

if (GetConversionSupported(converter, typeof(string)))

{

objChildValue = converter.ConvertFrom(null, CultureInfo.InvariantCulture, xnChild.InnerText);

item.SetValue(obj, objChildValue, null);

}

else if (item.PropertyType.IsClass)

{

objChildValue = Activator.CreateInstance(item.PropertyType);

DecSerialize<SerializableAttribute>(xnChild, objChildValue);

item.SetValue(obj, objChildValue, null);

}

else

{

}

}

catch (Exception ex)

{

PublicMethods.WarnMessageBox(ex.Message);

}

}

return true;

}

//转换支持

private bool GetConversionSupported(TypeConverter converter, Type conversionType)

{

return (converter.CanConvertFrom(conversionType) && converter.CanConvertTo(conversionType));

}

}

}