目录

[一、 需求概述 1](#_Toc517465018)

[二、 采用的技术 1](#_Toc517465019)

[三、 数据库ER图 2](#_Toc517465020)

[四、 数据库逻辑模型 3](#_Toc517465021)

[五、 物理表 3](#_Toc517465022)

[六、 触发器： 4](#_Toc517465023)

[七、 类图： 5](#_Toc517465024)

[八、 所用到windows控件 10](#_Toc517465025)

[九、 数据库访问： 10](#_Toc517465026)

[十、 图形界面： 17](#_Toc517465027)

[十一、 委托： 21](#_Toc517465028)

[十二、 调用系统函数 21](#_Toc517465029)

[十三、 使用多并发执行： 22](#_Toc517465030)

[十四、 C# 反射机制 25](#_Toc517465031)

[十五、 WPF窗口： 26](#_Toc517465032)

# 需求概述

图片素材管理程序可以方便画师或者从事图片开发以及其他的工作者，通过使用标签分类高效管理图片素材。支持广泛的图片格式，支持截屏，以及在不同的参考图之间切换。

# 采用的技术

Net Framework 4.7.1、WinForm、WPF、CSkin第三方图形库，SQLite3数据库、NuGet还包括：Imazen.WebP、System.Drawing.PSD。

监视Windows消息，重载WndProc方法，用于实现热键响应。

实现了IEnumerator接口，模拟了类似Lua的协程的实现。

大量使用了 new EventHandler(delegate (object sender, EventArgs e)

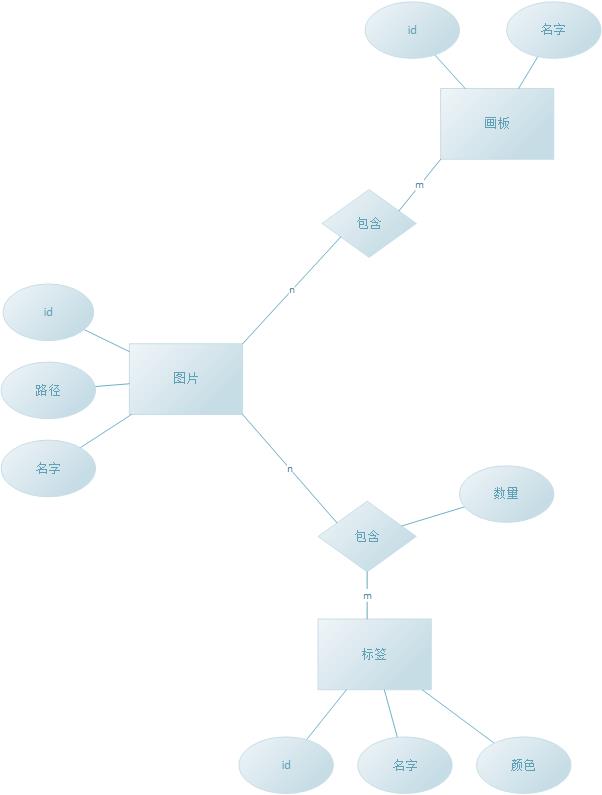
{

//do samethins.

});

进行了匿名委托。

# 数据库ER图



# 数据库逻辑模型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表名 | 列明 | 数据类型 | 说明 | 约束 |
| images | id | INTEGER |  | 主键、自动递增、不为空 |
| title | TEXT | 标题 | 不为空 |
| path | TEXT | 路径 | 不为空 |
| add\_time | TEXT | 添加时间 | 不为空 |
| edit\_time | TEXT | 修改时间 | 不为空 |
| MD5 | TEXT | MD5码 | 不为空、唯一 |
|  |  |  |  |  |
| labels | id | INTEGER |  | 主键、自动递增、不为空 |
| name | TEXT | 标签名 | 唯一、不为空 |
| color | INTEGER | 颜色 | 不为空 |
| num | INTEGER | 数量 | 默认为0 |
|  |  |  |  |  |
| label\_contain | label\_id | INTEGER | 标签外键 | 主键、外键、不为空 |
| image\_id | INTEGER | 图片外键 | 主键、外键、不为空 |

# 物理表

image表

CREATE TABLE "images" (

"id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

"path" TEXT NOT NULL,

"title" TEXT NOT NULL,

"add\_time" TEXT NOT NULL,

"edit\_time" TEXT NOT NULL,

"md5" TEXT NOT NULL,

CONSTRAINT "uq\_images\_md5" UNIQUE ("md5" ASC)

);

label\_contain表

CREATE TABLE "label\_contain" (

"label\_id" INTEGER NOT NULL,

"image\_id" INTEGER NOT NULL,

PRIMARY KEY ("label\_id", "image\_id"),

FOREIGN KEY ("label\_id") REFERENCES "labels" ("id") ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY ("image\_id") REFERENCES "images" ("id") ON DELETE CASCADE ON UPDATE CASCADE

);

labels表

CREATE TABLE "labels" (

"id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

"name" TEXT NOT NULL,

"color" integer NOT NULL,

"num" INTEGER DEFAULT 0,

CONSTRAINT "labels\_name\_u" UNIQUE ("name" ASC)

);

# 触发器：

删除image表时候触发器：

CREATE TRIGGER "tr\_images\_d"

BEFORE DELETE

ON "label\_contain"

FOR EACH ROW

BEGIN

UPDATE labels SET num = num-1

WHERE id = old.label\_id;

END;

插入image表时候的触发器：

CREATE TRIGGER "tr\_images\_i"

BEFORE INSERT

ON "label\_contain"

FOR EACH ROW

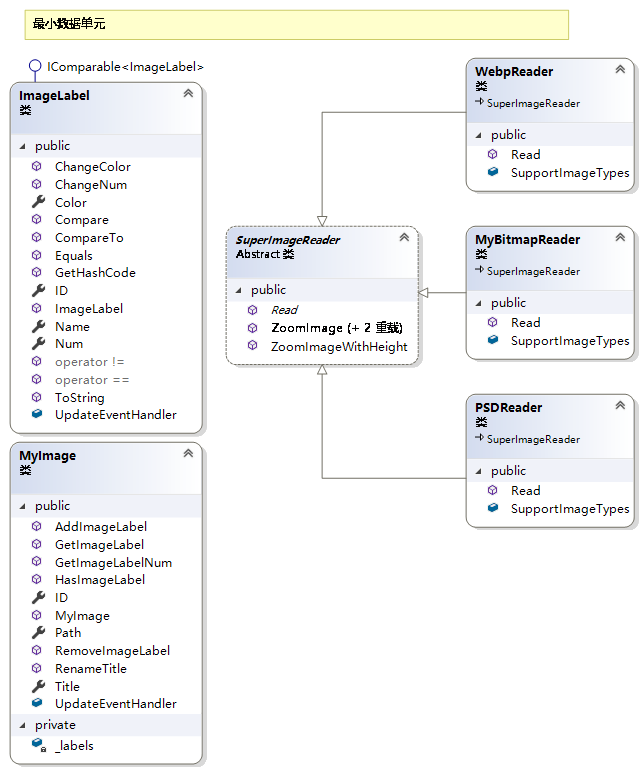
BEGIN

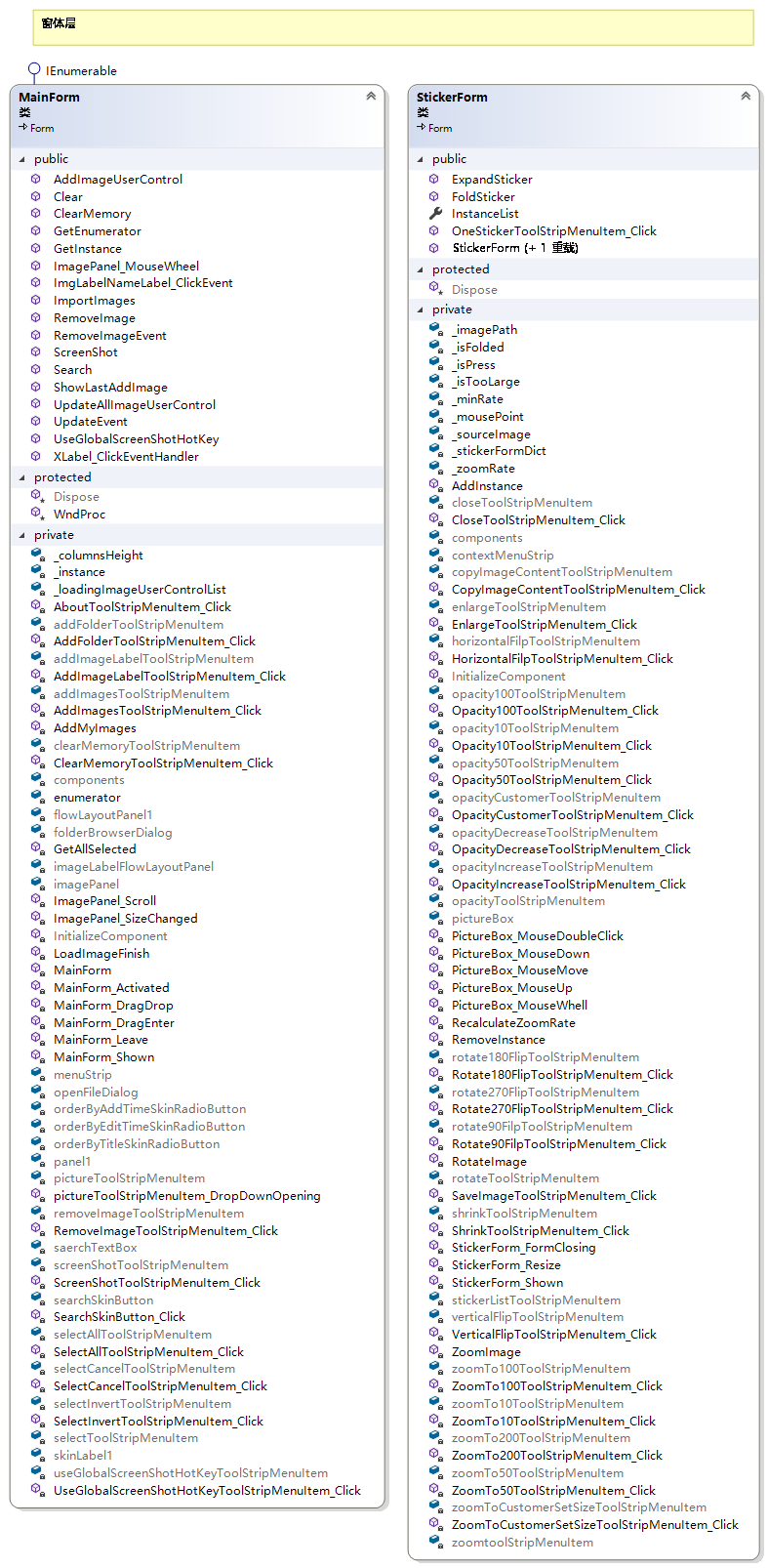
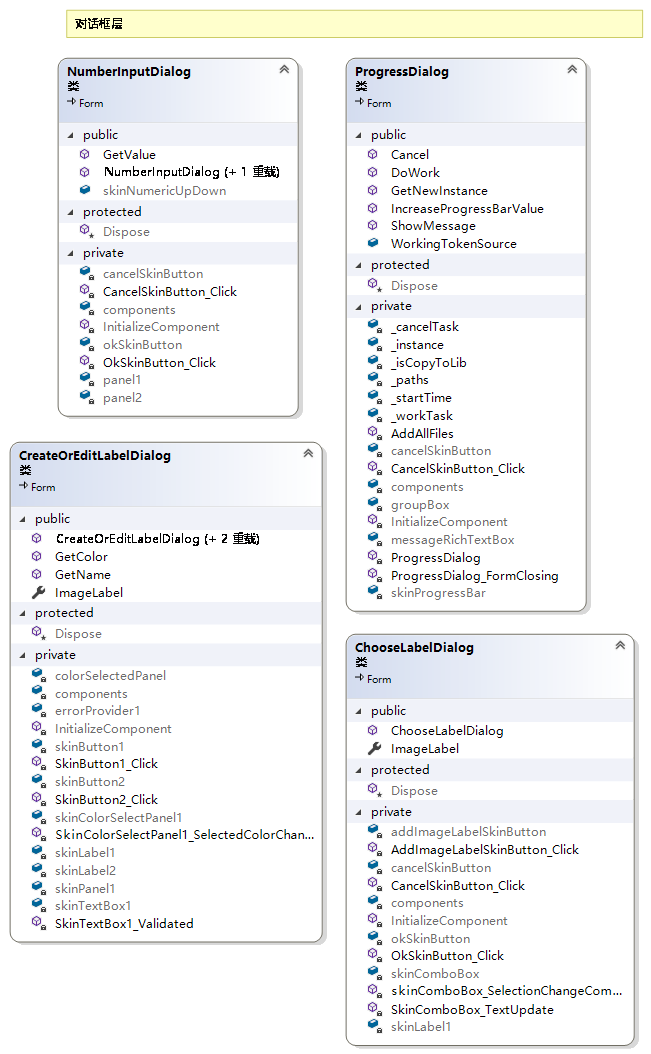
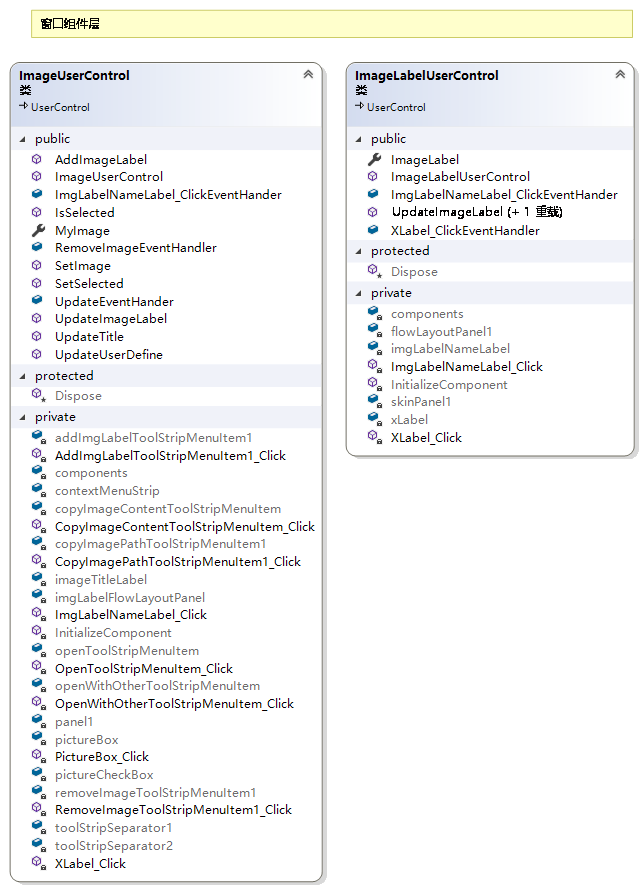
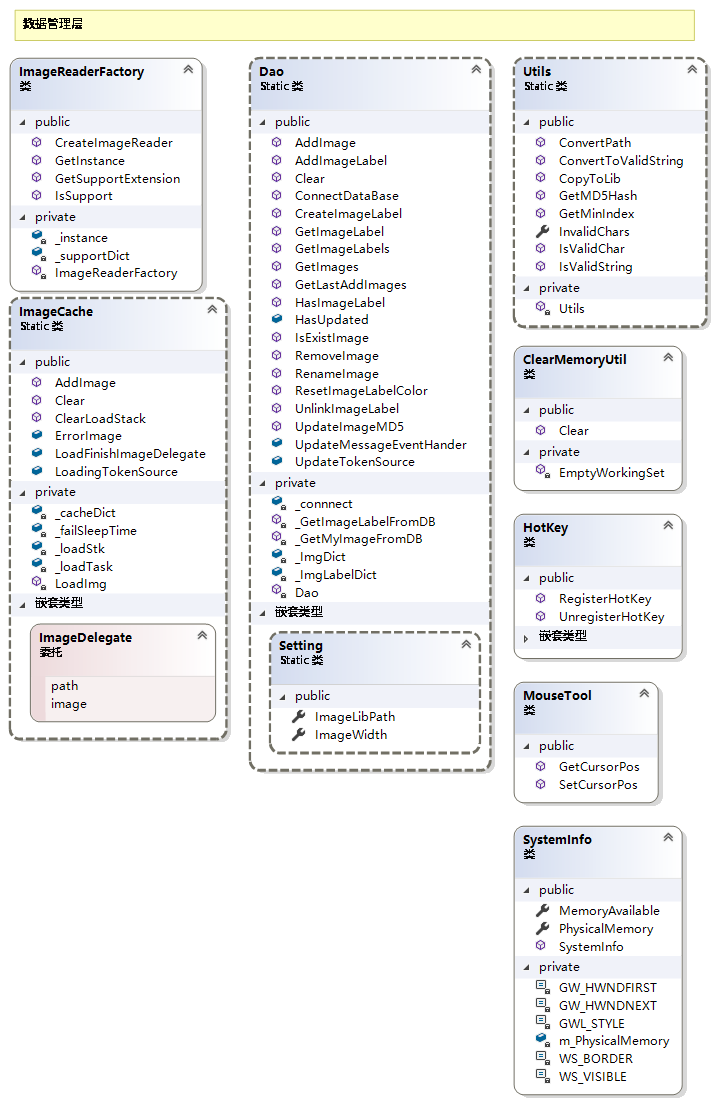
UPDATE labels SET num = num+1

WHERE id = new.label\_id;

END;

# 类图：





# 所用到windows控件

Button、CheckBox、ComboBox、ContextMenuStrip、ErrorProvider、FlowLayoutPanel、FolderBorwserDialog、GroupBox、Label、MenuStrip、OpenFileDialog、Panel、PictureBox、ProgressBar、RadioButton、RichTextBox、TextBox、Timer、ToolStrip、MessageBox

# 数据库访问：

数据库采用SQLite数据库。版本为3.

数据库的连接由Dao 类负责。

数据库连接关键代码：

/// <summary>

/// 连接数据库

/// 启动时会自动连接

/// </summary>

public static void ConnectDataBase()

{

if (!Directory.Exists(Setting.ImageLibPath))

{

Directory.CreateDirectory(Setting.ImageLibPath);

}

var databaseFileName = Setting.ImageLibPath + "\\database.sqlite";

var needCreateDatabase = false;

if (!File.Exists(databaseFileName))

{

SQLiteConnection.CreateFile(databaseFileName);

needCreateDatabase = true;

}

\_connnect = new SQLiteConnection($"Data Source={databaseFileName};Version=3;");

\_connnect.Open();

if (needCreateDatabase)

{

string sql = Resources.CreateDatabaseSql;

using (var command = new SQLiteCommand

{

Connection = \_connnect,

CommandText = Resources.CreateDatabaseSql

})

{

command.ExecuteNonQuery();

}

}

}

数据库查询采用Command + DataReader：

其中几个函数为：

/// <summary>

/// 查找所有图片

/// </summary>

/// <param name="orderby">排序关键列</param>

/// <param name="pageIndex">页号</param>

/// <param name="keywords">关键词</param>

/// <param name="imageLabels">包含的标签</param>

public static MyImage[] GetImages(string orderby = "title", int pageIndex = 0, string keywords = "", ImageLabel[] imageLabels = null)

{

var sqliteParameterList = new LinkedList<SQLiteParameter>();

var pageNum = Settings.Default.PageNumber;

var sql = "";

// 时候限定检索标签

if (imageLabels != null)

{

sql = "SELECT images.id AS images\_id, images.path AS images\_path, images.title AS images\_title FROM images JOIN label\_contain ON images.id = label\_contain.image\_id JOIN labels ON label\_contain.label\_id = labels.id WHERE (labels.name in ";

var imgLabelStringBuilder = new StringBuilder("(");

for (var index = 0; index < imageLabels.Length; index++)// (var imgLabel in imageLabels)

{

var imgLabel = imageLabels[index];

imgLabelStringBuilder.Append($"@imgLabel\_Name\_{index},");

sqliteParameterList.AddLast(new SQLiteParameter($"@imgLabel\_Name\_{index}", imgLabel.Name));

}

sql += imgLabelStringBuilder.ToString(0, imgLabelStringBuilder.Length - 1) + ")) AND ";

}

else

{

sql = "SELECT images.id AS images\_id, images.path AS images\_path, images.title AS images\_title FROM images WHERE ";

}

// 根据关键词检索

keywords = keywords.Trim();

var keywordArray = keywords.Split(new char[] { ' ' });

var keywordStringBuilder = new StringBuilder();

if (keywordArray.Length == 1)

{

keywordStringBuilder.Append($"images.title LIKE '%'||@word||'%' ");

sqliteParameterList.AddLast(new SQLiteParameter("@word", keywordArray[0]));

}

else {

keywordStringBuilder.Append("(");

for (var index = 0; index < keywordArray.Length; index++)//foreach(var word in keywordArray)

{

var word = keywordArray[index];

if (index != keywordArray.Length-1)

{

keywordStringBuilder.Append($"images.title LIKE '%'||@word\_{index}||'%' OR ");

sqliteParameterList.AddLast(new SQLiteParameter($"@word\_{index}", word));

}

else

{

keywordStringBuilder.Append($"images.title LIKE '%'||@word\_{index}||'%' ");

sqliteParameterList.AddLast(new SQLiteParameter($"@word\_{index}", word));

}

}

keywordStringBuilder.Append(")");

}

sql += keywordStringBuilder.ToString();

sql += $" ORDER BY @orderby ";

sqliteParameterList.AddLast(new SQLiteParameter("@orderby", orderby));

sql += $"LIMIT @start, @pageNum";

sqliteParameterList.AddLast(new SQLiteParameter("@start", pageIndex \* pageNum));

sqliteParameterList.AddLast(new SQLiteParameter("@pageNum", pageNum));

//Debug.WriteLine("查询的SQL语句为：" + sql);

//var sql = $"SELECT images.id, images.path, images.title FROM images LIMIT {pageIndex\*pageNum}, {pageNum}";

//var sql= "SELECT images.id, images.path, images.title FROM images JOIN label\_contain ON images.id = label\_contain.image\_id JOIN labels ON label\_contain.label\_id = labels.id WHERE images.title LIKE '%[aa|bb]%' AND labels.name in ('aaa','bbb') LIMIT 0, 10";

var myImageLinkedList = new LinkedList<MyImage>();

using (var command = new SQLiteCommand(sql, \_connnect))

{

var sqliteParameters = new SQLiteParameter[sqliteParameterList.Count];

sqliteParameterList.CopyTo(sqliteParameters, 0);

command.Parameters.AddRange(sqliteParameters);

using (var reader = command.ExecuteReader())

{

var imgIdOrd = reader.GetOrdinal("images\_id");

var imgTitleOrd = reader.GetOrdinal("images\_title");

var imgPathOrd = reader.GetOrdinal("images\_path");

while (reader.Read())

{

var imgId = reader.GetInt64(imgIdOrd);

MyImage img;

if (!\_ImgDict.ContainsKey(imgId))

{

img = \_GetMyImageFromDB(imgId);

}

else

{

img = \_ImgDict[imgId];

}

myImageLinkedList.AddLast(img);

}

}

var imgArray = new MyImage[myImageLinkedList.Count];

myImageLinkedList.CopyTo(imgArray, 0);

return imgArray;

}

}

/// <summary>

/// 从数据库读取MyImage或者根据指定的值来创建MyImage

/// </summary>

/// <param name="id"></param>

/// <param name="title"></param>

/// <param name="path"></param>

/// <returns></returns>

private static MyImage \_GetMyImageFromDB(Int64 id, String title = null, String path = null)

{

var sql = "";

if (title == null || path == null)

{

sql = $"SELECT images.id AS images\_id, images.path AS images\_path, images.title AS images\_title FROM images WHERE images.id = @id ";

using (var command = new SQLiteCommand(sql, \_connnect))

{

command.Parameters.AddWithValue("@id", id);

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

title = reader.GetString(reader.GetOrdinal("images\_title"));

path = reader.GetString(reader.GetOrdinal("images\_path"));

}

else

{

return null;

}

}

}

}

sql = $"SELECT labels.id AS labels\_id, labels.name AS labels\_name, labels.color AS labels\_color, labels.num AS labels\_num FROM labels JOIN label\_contain ON label\_contain.label\_id = labels.id WHERE label\_contain.image\_id = @id";

using (var labelCommand = new SQLiteCommand(sql, \_connnect))

{

labelCommand.Parameters.AddWithValue("@id", id);

using (var labelReader = labelCommand.ExecuteReader())

{

var labelLinkedList = new LinkedList<ImageLabel>();

var labIdOrd = labelReader.GetOrdinal("labels\_id");

var labNameOrd = labelReader.GetOrdinal("labels\_name");

var labColorOrd = labelReader.GetOrdinal("labels\_color");

var labNumOrd = labelReader.GetOrdinal("labels\_num");

while (labelReader.Read())

{

var labName = labelReader.GetString(labNameOrd);

ImageLabel label;

if (!\_ImgLabelDict.ContainsKey(labName))

{

label = \_GetImageLabelFromDB(name: labName);

}

else

{

label = \_ImgLabelDict[labName];

}

labelLinkedList.AddLast(label);

}

var labelArray = new ImageLabel[labelLinkedList.Count];

labelLinkedList.CopyTo(labelArray, 0);

Array.Sort(labelArray);

return \_ImgDict[id] = new MyImage(id, title, path, labelArray);

}

}

}

/// <summary>

/// 根据关键词获取标签

/// </summary>

/// <param name="key">关键词</param>

public static ImageLabel[] GetImageLabels(string key)

{

var sql = $"SELECT labels.id AS labels\_id , labels.name AS labels\_name, labels.color AS labels\_color, labels.num AS labels\_num FROM labels WHERE labels.name LIKE '%'||@key||'%'";

using (var labelCommand = new SQLiteCommand(sql, \_connnect))

{

labelCommand.Parameters.AddWithValue("@key", key);

using (var labelReader = labelCommand.ExecuteReader())

{

var labIdOrd = labelReader.GetOrdinal("labels\_id");

var labNameOrd = labelReader.GetOrdinal("labels\_name");

var labColorOrd = labelReader.GetOrdinal("labels\_color");

var labNumOrd = labelReader.GetOrdinal("labels\_num");

var labelLinkedList = new LinkedList<ImageLabel>();

while (labelReader.Read())

{

var name = labelReader.GetString(labNameOrd);

ImageLabel label;

if (!\_ImgLabelDict.ContainsKey(name))

label = \_GetImageLabelFromDB(labelReader.GetInt64(labIdOrd), labelReader.GetString(labNameOrd), labelReader.GetInt32(labColorOrd), labelReader.GetInt64(labNumOrd));

else

label = \_ImgLabelDict[name];

labelLinkedList.AddLast(label);

}

var labelArray = new ImageLabel[labelLinkedList.Count];

labelLinkedList.CopyTo(labelArray, 0);

Array.Sort(labelArray);

return labelArray;

}

}

}

/// <summary>

/// 从数据库读取标签或者根据所给的数值建立标签

/// </summary>

/// <param name="id"></param>

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

/// <param name="color"></param>

/// <param name="num"></param>

/// <returns>如果标签不存在返回null</returns>

private static ImageLabel \_GetImageLabelFromDB(Int64 id = -1, string name = null, int color = 0, Int64 num = 0)

{

SQLiteParameter sQLiteParameter;

var sql = "";

if (name == null || id == -1)

{

sql = "SELECT labels.id AS labels\_id, labels.name AS labels\_name, labels.color AS labels\_color, labels.num AS labels\_num FROM labels WHERE ";

if (id != -1)

{

sql += $"labels.id = @id ";

sQLiteParameter = new SQLiteParameter("@id", id);

}

else if (name != null)

{

sql += $"labels.name = @name ";

sQLiteParameter = new SQLiteParameter("@name", name);

}

else

return null;

using (var command = new SQLiteCommand(sql, \_connnect))

{

command.Parameters.Add(sQLiteParameter);

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

id = reader.GetInt64(reader.GetOrdinal("labels\_id"));

name = reader.GetString(reader.GetOrdinal("labels\_name"));

color = reader.GetInt32(reader.GetOrdinal("labels\_color"));

num = reader.GetInt64(reader.GetOrdinal("labels\_num"));

}

else

{

return null;

}

}

}

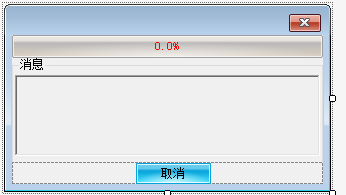
}

return \_ImgLabelDict[name] = new ImageLabel(id, name, Color.FromArgb(color), num);

}

# 图形界面：

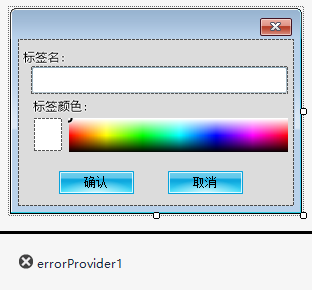
**进度对话框**



**选择标签对话框：**



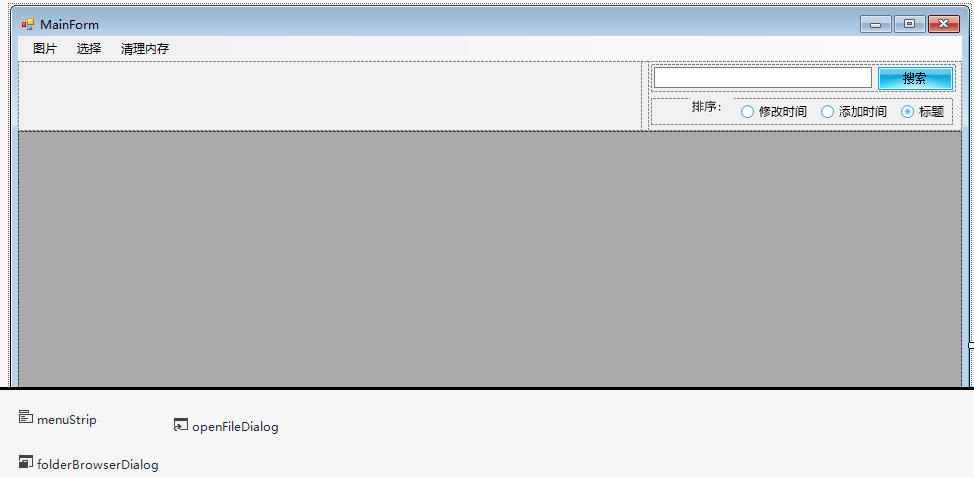
**新建或者编辑标签对话框；**

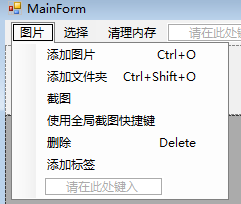


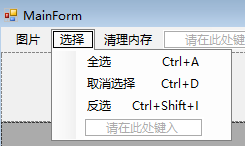
**数值选择对话框：**



**主界面：**



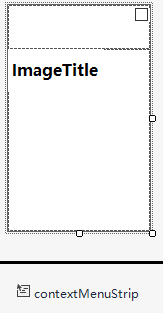




**屏幕贴片窗口：**



**图片组件：**



**图片标签组件：**



# 委托：

部分委托相关的代码：

/// <summary>

/// 图片委托

/// </summary>

/// <param name="path"></param>

/// <param name="image"></param>

public delegate void ImageDelegate(String path, Image image);

/// <summary>

/// 加载完成事件通知

/// </summary>

public static ImageDelegate LoadFinishImageDelegate;

匿名无参数委托：

messageRichTextBox.BeginInvoke((MethodInvoker)delegate

{

messageRichTextBox.AppendText(msg + "\r\n");

messageRichTextBox.ScrollToCaret();

});

EventHander<T>作为委托：

/// <summary>

/// 更新消息事件

/// </summary>

public static EventHandler<String> UpdateMessageEventHander;

imagePanel.MouseWheel += new MouseEventHandler(delegate (object sender, MouseEventArgs e)

{

ImagePanel\_Scroll(sender, null);

});

# 调用系统函数

示例，释放内存部分代码：

[DllImport("psapi.dll")]

static extern int EmptyWorkingSet(IntPtr hwProc);

/// <summary>

/// 释放内存

/// </summary>

public static void Clear()

{

GC.Collect();

GC.WaitForPendingFinalizers();

Process[] processes = Process.GetProcesses();

foreach (Process process in processes)

{

//对于系统进程会拒绝访问，导致出错，此处对异常不进行处理。

try

{

EmptyWorkingSet(process.Handle);

}

catch

{

}

}

}

# 使用多并发执行：

示例，加载图片部分代码：

/// <summary>

/// 加载队列

/// </summary>

private static ConcurrentStack<String> \_loadStk = new ConcurrentStack<string>();

/// <summary>

/// 缓冲字典

/// </summary>

private static ConcurrentDictionary<String, Image> \_cacheDict = new ConcurrentDictionary<string, Image>();

/// <summary>

/// 加载任务

/// </summary>

private static Task \_loadTask;

/// <summary>

/// 取消加载任务标记

/// </summary>

public static CancellationTokenSource LoadingTokenSource = new CancellationTokenSource();

/// <summary>

/// 添加需要加载的图片

/// </summary>

/// <param name="path">位置</param>

public static void AddImage(string path)

{

\_loadStk.Push(Utils.ConvertPath(path));

if (\_loadTask == null || \_loadTask.Status == TaskStatus.Canceled || \_loadTask.Status == TaskStatus.Faulted || \_loadTask.Status == TaskStatus.RanToCompletion)

{

\_loadTask = new Task(LoadImg, LoadingTokenSource.Token, TaskCreationOptions.LongRunning);

\_loadTask.Start();

}

/// <summary>

/// 加载图片

/// </summary>

private static void LoadImg()

{

while (!\_loadStk.IsEmpty)

{

// 取消

if (LoadingTokenSource.Token.IsCancellationRequested)

{

break;

}

string path = null;

while (!\_loadStk.TryPop(out path))

{

Thread.Sleep(\_failSleepTime);

}

if (LoadingTokenSource.Token.IsCancellationRequested)

{

break;

}

Image image;

if (!\_cacheDict.ContainsKey(path))

{

var imageReader = ImageReaderFactory.GetInstance().CreateImageReader(path);

image = imageReader.Read(path);

// 取消

if (LoadingTokenSource.Token.IsCancellationRequested)

{

break;

}

if (image == null)

{

if (ErrorImage.Width != Settings.Default.ImageWidth)

{

ErrorImage = SuperImageReader.ZoomImage((Image)Resources.ImageLoadError.Clone());

}

image = ErrorImage;

}

else

{

image = SuperImageReader.ZoomImage(image);

// 取消

if (LoadingTokenSource.Token.IsCancellationRequested)

{

break;

}

if (Settings.Default.EnableImageCache)

{

while (!\_cacheDict.TryAdd(path, image))

{

Thread.Sleep(\_failSleepTime);

}

}

}

}

else

{

image = \_cacheDict[path];

}

//通知加载完成

LoadFinishImageDelegate(path, image);

}

}

# C# 反射机制

private ImageReaderFactory()

{

var types = Assembly.GetCallingAssembly().GetTypes();

var aType = typeof(SuperImageReader);

//Debug.Log(aType.FullName);

//List<A> alist = new List<A>();

foreach (var type in types)

{

var baseType = type.BaseType; //获取基类

while (baseType != null) //获取所有基类

{

//Debug.Log(baseType.Name);

if (baseType.Name == aType.Name)

{

// 获取子类的支持文件类型

var field = type.GetField("SupportImageTypes");

var suports = field.GetValue(null) as string[];

Type objtype = Type.GetType(type.FullName, true);

foreach (var sp\_ext in suports)

{

\_supportDict[sp\_ext.ToLower()] = objtype;

}

break;

}

else

{

baseType = baseType.BaseType;

}

}

}

}

/// <summary>

/// 创建新的图片读取器

/// </summary>

/// <param name="path">路径</param>

public SuperImageReader CreateImageReader(string path)

{

path = Utils.ConvertPath(path);

var ext = Path.GetExtension(path);

if (\_supportDict.ContainsKey(ext.ToLower()))

{

Type objtype = \_supportDict[ext.ToLower()];

object obj = Activator.CreateInstance(objtype);

var reader = obj as SuperImageReader;

return reader;

}

else

{

return null;

}

}

# WPF窗口：

部分代码：

<Window x:Class="WPFCaptureScreenShot.CaptureWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="CaptureWindow"

AllowsTransparency="True"

WindowStyle="None"

Background="Black"

Opacity="0.1"

WindowState="Maximized"

MouseDown="CaptureWindow\_MouseDown"

MouseMove="CaptureWindow\_MouseMove" KeyDown="Window\_KeyDown" Closing="Window\_Closing">

<Canvas x:Name="CaptureCanvas">

</Canvas>

</Window>

private void CaptureWindow\_MouseMove(object sender, System.Windows.Input.MouseEventArgs e)

{

if (isMouseDown)

{

// 通过一个矩形来表示目前截图区域

System.Windows.Shapes.Rectangle rect = new System.Windows.Shapes.Rectangle();

double dx = e.GetPosition(null).X;

double dy = e.GetPosition(null).Y;

double rectWidth = Math.Abs(dx - x);

double rectHeight = Math.Abs(dy - y);

SolidColorBrush brush = new SolidColorBrush(Colors.White);

rect.Width = rectWidth;

rect.Height = rectHeight;

rect.Fill = brush;

rect.Stroke = brush;

rect.StrokeThickness = 1;

if (dx < x)

{

Canvas.SetLeft(rect, dx);

Canvas.SetTop(rect, dy);

}

else

{

Canvas.SetLeft(rect, x);

Canvas.SetTop(rect, y);

}

CaptureCanvas.Children.Clear();

CaptureCanvas.Children.Add(rect);

if (e.LeftButton == MouseButtonState.Released)

{

CaptureCanvas.Children.Clear();

// 获得当前截图区域

width = Math.Abs(e.GetPosition(null).X - x);

height = Math.Abs(e.GetPosition(null).Y - y);

if (e.GetPosition(null).X == x || e.GetPosition(null).Y == y)

{

DialogResult = false;

}

else if(e.GetPosition(null).X > x)

{

CaptureScreen(x, y, width, height);

}

else

{

CaptureScreen(e.GetPosition(null).X, e.GetPosition(null).Y, width, height);

}

isMouseDown = false;

DialogResult = true;

//x = 0.0;

//y = 0.0;

//this.Close();

}

}

}