编码规范

|  |  |  |  |
| --- | --- | --- | --- |
| *版本* | *日期* | *内容* | *修改人* |
|  |  | *创建C#编码规范* |  |

一**.** 目标

1. 安全

代码完成所需的功能之余,求要产生负作用,即要稳定可靠.

1. 易读

类,实例,成员变量,成员函数的命名一目了然

1. 美观

尽量统一项目组内人员的编程风格,保证一致性,统一性.

二**.**具体注意事项**. (**一**)**命名

1. 命名原则
2. 所有的函数(变量/类/文件名)应该代表其实际的作用.应该使用具有意义的单词或多个词组合,但不要使用人名,项目组名.
3. 所有的函数(变量/类名)一律使用英文.
4. 使用多个单词时不需要使用连线(如下划线) (4)多个词组合较长时,可以使用单词的缩写.

(5)求得使用非常相近的名字类表示几个不同含义的函数(变量/类). (6)命名时请考虑名字的唯一性和含义的准确性.

(7)命名采用三种格式,Pascal(大小写),Camel(大小写混合),首字母全部大写格式如:ID

1. 变量的命名
2. 变量名一般由"类型修饰+代表变量含意的英文单词或单词缩写"等部分组成.

如:strUserName

1. 针对异常捕获过程中的Exception变量命名，在没有冲突的情况下， 统一命名为 e;
2. 函数的命名
3. 使用动宾词组表达函数所做的事.命名采用Pascal形式如：ShowUserInfo(),BoundGridView()等
4. 同名的函数(重载函数)在功能上应该完全相同,在参数上的差别也应一目了然.
5. 不得出现名字非常相近但功能不赋的函数.

如:CreatePage1(),CreatePage2()等.

1. 类命名

(1)名字应该能够标识事物的特性.如Public class Employee() (2)名字尽量不使用缩写,除非它是众所周知的.

(3)名字可以有两个或三个以上单词组成，但通常不应多于三个。(4)在名字中，所有单词第一个字母都大写。缩写都要大写。

(5)不要使用下划线连接符(\_).

如：public classs EmployeeComputeSalary 即采用Pascal大小写形式。

1. 控件命名规则

|  |  |  |
| --- | --- | --- |
| 控件 | 前缀 | 备注 |
| Label | lbl | 例如：lblName |
| TextBox | txt |  |
| Button | btn |  |
| CheckBox | chk |  |
| RadioButton | rdo |  |
| CheckBoxList | cbl |  |
| RadioButtonList | rbt |  |
| ListBox | lst |  |
| DropDownList | ddl |  |
| DataGrid | dg |  |
| DataList | dlt |  |
| Image | img |  |
| Table | tbl |  |
| Panel | pnl |  |
| LinkButton | lbt |  |
| ImageButton | img |  |
| Calender | cld |  |
| RequiredFieldValidator | rfv |  |
| CompareValidator | cvt |  |
| RangeValidator | rvt |  |
| RegularExpressionValidator | rev |  |
| ValidatorSummary | vsm |  |

1.

1. 文件命名
2. 文件命名要具有实际意义。
3. 源文件应尽量使用8.3格式，文件名只能包含字母，数字和下划线，

不得使用其他的字母，超长的文件名应使用缩写方式减少文件名的长度，建议使用如下的缩写的规则（部分情况可以历例外,视具体情况而定）；

\*缩写一般可以去掉无意字母以及不发间字母。

\*单词的首字母一般应该保留。

\*当一个单词必须编写成一个字母时，应该选用最有代表性的字母或首字母。

\*多音节的单词可以去掉后面的音节而只保留有意义的音节。

\*可以使用一些谐音表示一个单词，如2表示to，4表示for等等， (3)当单词数量少，字母少时不需要缩写词。

(4)临时文件必须调用系统函数生成，禁止使用固定文件名，一般情况下，临时文件应在系统临时目录下（通过系统函数取到临时目录名）生成。

1. 参数的命名

一般要采用大小写混合格式。如protect bool login(string userName,string userPass)

1. 接口的命名

以 I 开始，后面加上名词、名词短语、形容词命名。比如：

IDisposable；

1. 常量的命名

所有单词大写，多个单词之间用 "\_" 隔开。public const string PAGE\_TITLE = "Welcome";

1. 命名空间的命名

格式：CompanyName / ProjectName.TechnologyName[.Feature][.Design] a)CompanyName/ProjectName：公司名、项目名称或产品名称； b)TechnologyName：稳定的、公认的技术名称或架构层次名称； c)[.Feature][.Design]：可选的功能与设计；

例如：

//微软公司的Web开发中的UI中的Design namespace Microsoft.Web.UI.Design

**(**二**)**代码格式书写规范

**1**．代码书写格式

1. 所有的缩进TAB键为4个空格，每个单词的首字符大写，其余小写。
2. 在代码中垂直对齐左括号和右括号。例：

if(x==0)

{

Response.Write("用户名必须输入！");

}

不允许以下情况：

if(x==0){

Response.Write("用户名必须输入！");}

或者 if(x==0){Response.Write("用户名必须输入！");}

1. 在大多数运算符之前和之后使用空格，这样做时不会改变代码的的意图却可以使代码容易阅读。

例：int j = i + k;而不应写为：int j=i+k;

1. 缩写SQL语句时，对于关键字使用全部大写，对于数据库元素

（如表，列和视图）使用大小写混合）。

1. 将每个主要的SQL子句放在不同的行上，这样更容易阅读和编辑语句。

**2**注释的写法

1. 在你劳神的地方请加上详细的注释说明，除了最简单的存取成员变量的Set/Get成员函数之外，其余大部分的函数写上注释是良好的习惯，尽量使你的程序让别人很容易看懂。
2. 大多的注释会使很难看，但一些复杂的算法和数据结构和数据结构处还是要加上注释的，这样别人就容易看懂，否则时间长了，你自己都未必卸载明白了。
3. 如果是对某一段程序（算法/结构）的注释，在程序头直接用//再空一格进行说明，一行不要超过80个字符。
4. 为了防止在阅读代码时不得不左右流动源代码编辑器，每行代码或注释在不得超过一显示屏。
5. 使用连续的多个/表示注释行（不要超过80字符）。
6. 文件头部应有注释，简单描述文件的内容
7. 对于程序中的比较关键的算法和函数，必须加注释。

**3**．**cs**文件的书写

1. 各个部分应使用注释行和空行分割，并在必要的地方写上注释。
2. 函数之间用注释各空行分割。中间的内容缩进一个TAB

三**.**其他

1.变量

1. float和bool禁止用"=="判断.bool应该用逻辑运算关系符,而float应该用差值区间来判断"相等";
2. 类型转换一律用显示类型转换(3)类型的长度一律用sizeof()获得;

(4)当声明一个变量时,务必要自已初始化一下变量; 2.函数

1. 功能单一,函数名要名符其实.
2. 要易懂,实现时要根过分追求技巧,优化放到后面去做. (3)长度一般禁止超过200行.
3. 要检查输入值是否合法,实现(成员)函数时务必要求输入参数是在要求范围之内,尤其你定义的(成员)函数给别人调用时,要判断其合法性.
4. 调用函数时要严格按照接口规范调用,调用后要判断执行情况,并做适当的错误处理(尔后会给出错误和异常处理规范)
5. 尽量避免整块复制代码段,如果出现这样的情况要分析原因,如果这段代码完成独立的功能,应考虑使用函数,否则,应考虑使用宏定义,否则因为修改引起的不一致往往是错误的根源.

附**:**

标识符 大小写方式 示例

标识符大小写规则

|  |  |  |
| --- | --- | --- |
| 标识符 | 大小写 | 示例 |
| 类 | Pascal | AppDomain |
| 枚举类型 | Pascal | ErrorLevel |
| 枚举值 | Pascal | FataError |
| 事件 | Pascal | ValueChanged |
| 异常类 | Pascal | WebException |
| 只读的静态字段 | Pascal | ReadValue |
| 接口 | Pascal | IDisposable |
| 方法 | Pascal | ToString |
| 命名空间 | Pascal | System.Drawing |
| 属性 | Pascal | BackColor |
| 参数 | Camel | typeName |

代码

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Data.OleDb;//添加类库

using System.IO;//添加类库 输入输出

using Excel = Microsoft.Office.Interop.Excel;//添加类库

using System.Threading.Tasks;

using System.Windows.Forms;

namespace \_20170668\_刘晨煜\_4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//角度转弧度

public double dmstorad(string s)

{

string[] ss = s.Split(new char[3] { '°', '′', '″' }, StringSplitOptions.RemoveEmptyEntries);

double[] d = new double[ss.Length];

for (int i = 0; i < d.Length; i++)

d[i] = Convert.ToDouble(ss[i]);

double sign = d[0] >= 0.0 ? 1.0 : -1.0;

double rad = 0;

if (d.Length == 1)

rad = Math.Abs(d[0]) \* Math.PI / 180;

else if (d.Length == 2)

rad = (Math.Abs(d[0]) + d[1] / 60) \* Math.PI / 180;

else

rad = (Math.Abs(d[0]) + d[1] / 60 + d[2] / 60 / 60) \* Math.PI / 180;

rad = sign \* rad;

return rad;

}

//弧度转角度

public string radtodms(double rad)

{

double sign = rad >= 0.0 ? 1.0 : -1.0;

rad = Math.Abs(rad) \* 180 / Math.PI;

double[] d = new double[3];

d[0] = (int)rad;

d[1] = (int)((rad - d[0]) \* 60);

d[2] = (rad - d[0] - d[1] / 60) \* 60 \* 60;

d[2] = Math.Round(d[2], 2);

if (d[2] == 60)

{

d[1] += 1;

d[2] -= 60;

if (d[1] == 60)

{

d[0] += 1;

d[1] -= 60;

}

}

d[0] = sign \* d[0];

string s = Convert.ToString(d[0]) + "°" + Convert.ToString(d[1]) + "′" + Convert.ToString(d[2]) + "″";

return s;

}

//坐标方位角推算

public double fangweijiao(double[] sdr, double[] cr)

{

double sum = 0;

for (int i = 1; i < sdr.Length; i++)

{

cr[i] = cr[i - 1] + sdr[i] - Math.PI;

if (cr[i] >= Math.PI \* 2)

cr[i] -= Math.PI \* 2;

else if (cr[i] < 0.0)

cr[i] += Math.PI \* 2;

sum += sdr[i];

}

return sum;

}

private void button1\_Click(object sender, EventArgs e)

{

string[] sd = new string[dataGridView1.RowCount - 5]; //新建一个数组存放观测角度的 原始值

double[] sdr = new double[sd.Length]; //新建一个数组存放观测角度的弧度值

double[] cr = new double[sd.Length]; //新建一个数组存放计算的坐标方位角

double sum = 0;

cr[0] = dmstorad(Convert.ToString(dataGridView1.Rows[0].Cells[4].Value)); //获取第一个坐标方位角，并将其转换成弧度，放入cr[]数组第一个元素中

double acd = dmstorad(Convert.ToString(dataGridView1.Rows[dataGridView1.RowCount - 6].Cells[4].Value)); //获取终边坐标方位角，并将其转换成弧度，放入放入acd中用于计算和检核

for (int i = 1; i < sd.Length; i++) //从第二行开始循环，将观测角度的原始值放入 sd[]数组中,并转换成弧度值存放在sdr数组中

{

sd[i] = Convert.ToString(dataGridView1.Rows[i].Cells[1].Value);

sdr[i] = dmstorad(sd[i]);

}

sum = fangweijiao(sdr, cr); //计算改正前坐标方位角和观测角度总和，分别存储在 cr数组和sum中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[1].Value = radtodms(sum); //将观测角度总和放入表格中

double fd, fdx;

fd = cr[cr.Length - 1] - acd;//计算角度闭合差，单位弧度

fdx = 60 \* Math.Sqrt(sd.Length - 1);//计算角度闭合差限差，单位秒

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[1].Value = Convert.ToString(Math.Round(fd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度闭合差存入表格中

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[1].Value = Convert.ToString(Math.Round(fdx, 2)) + "″";//将角度闭合差限差存入表格中

if (Math.Abs(fd \* 180 / Math.PI \* 3600) > fdx)//检查角度闭合差是否满足要求

MessageBox.Show("角度闭合差超限！");

else

{

double vd = -fd / (sd.Length - 1);//分配角度闭合差（观测左角）

double sumvd = 0;

for (int i = 1; i < sdr.Length; i++)

{

sdr[i] += vd;//计算改正后的观测角度，并存入sdr数组中

sumvd += vd;

dataGridView1.Rows[i].Cells[2].Value = Convert.ToString(Math.Round(vd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数存入表格中

dataGridView1.Rows[i].Cells[3].Value = radtodms(sdr[i]);

}

if (Math.Round(sumvd, 8) != Math.Round(-fd, 8)) //秒保留2位对应弧度是8位

MessageBox.Show("角度改正数分配有误！");

else

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[2].Value = Convert.ToString(Math.Round(sumvd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数总和存入表格中

sum = fangweijiao(sdr, cr);//推算改正后的坐标方位角

if (Math.Round(cr[cr.Length - 1], 8) != Math.Round(acd, 8)) MessageBox.Show("坐标方位角推算有误！");

else

{

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[3].Value = radtodms(sum); //将改正后观测角度总和放入表格中

for (int i = 1; i < cr.Length - 1; i++)//将改正后坐标方位角存入表格

dataGridView1.Rows[i].Cells[4].Value = radtodms(cr[i]);

}

//至此角度调整和计算完毕

}

//坐标增量闭合差计算及其调整

double[] jl =new double[sd.Length-1]; //新建一个数组存放观测的距离

double[] dx = new double[sd.Length-1]; //新建一个数组存放Δx

double[] dy = new double[sd.Length-1]; //新建一个数组存放Δy

double[] ddx = new double[sd.Length-1]; //新建一个数组存放Δx改正值

double[] ddy = new double[sd.Length-1]; //新建一个数组存放Δy改正值

double sumdx = 0;

double sumdy = 0;

double sumjl= 0;

for (int i = 1; i < jl.Length; i++)

{

jl[i] = Convert.ToDouble(dataGridView1.Rows[i].Cells[5].Value);

dx[i] = jl[i] \* Math.Cos(cr[i]);

dy[i] = jl[i] \* Math.Sin(cr[i]);

dataGridView1.Rows[i].Cells[6].Value = Convert.ToString(Math.Round(dx[i], 4));

dataGridView1.Rows[i].Cells[7].Value = Convert.ToString(Math.Round(dy[i], 4));

sumdx += dx[i];

sumdy += dy[i];

sumjl+=jl[i];

}

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[5].Value = Convert.ToString(sumjl);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[6].Value = Convert.ToString(Math.Round(sumdx, 4));

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[7].Value = Convert.ToString(Math.Round(sumdy, 4));

double fx, fy;//坐标增量闭合差

double xa = Convert.ToDouble(dataGridView1.Rows[1].Cells[12].Value);

double xc = Convert.ToDouble(dataGridView1.Rows[dataGridView1.RowCount - 6].Cells[12].Value);

double ya = Convert.ToDouble(dataGridView1.Rows[1].Cells[13].Value);

double yc = Convert.ToDouble(dataGridView1.Rows[dataGridView1.RowCount - 6].Cells[13].Value);

fx = sumdx - (xc - xa);//计算坐标增量闭合差

fy = sumdy - (yc - ya);

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[7].Value = Convert.ToString(Math.Round(fx, 4));

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[7].Value = Convert.ToString(Math.Round(fy, 4));

double fxy, m;//定义道线全长闭合差，导线全长相对闭合差

fxy = Math.Sqrt(fx \* fx + fy \* fy);

m = sumjl / fxy;

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[10].Value = Convert.ToString(Math.Round(fxy, 4));

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[11].Value = Convert.ToString((int)m);

double sumddx = 0;//定义坐标增量改正数的和

double sumddy = 0;

double[] cx = new double[sd.Length - 1]; //定义数组用于存放改正后的坐标增量及总和

double[] cy = new double[sd.Length - 1];

double sumcx = 0;

double sumcy = 0;

double[] x = new double[sd.Length - 1]; //定义数组用于存放x，y坐标

double[] y = new double[sd.Length - 1];

x[1] =xa;

y[1] =ya;

if (m < 2000) //判断导线全长相对闭合差是否超限

MessageBox.Show("导线全长相对闭合差超限！");

else

{

for(int j=1;j<ddx.Length;j++)

{

ddx[j] = -fx \* jl[j] / sumjl; //计算坐标增量改正数

ddy[j] = -fy \* jl[j] / sumjl;

sumddx += ddx[j]; //计算坐标增量改正数总和

sumddy += ddy[j];

dataGridView1.Rows[j].Cells[8].Value = Convert.ToString(Math.Round(ddx[j], 4)); //将坐标增量改正数放入表格

dataGridView1.Rows[j].Cells[9].Value = Convert.ToString(Math.Round(ddy[j], 4));

cx[j] = dx[j] + ddx[j]; //计算改正后坐标增量

cy[j] = dy[j] + ddy[j];

sumcx += cx[j]; //计算改正后坐标增量总和

sumcy += cy[j];

dataGridView1.Rows[j].Cells[10].Value = Convert.ToString(Math.Round(cx[j], 4)); //将改正后坐标增量放入表格

dataGridView1.Rows[j].Cells[11].Value = Convert.ToString(Math.Round(cy[j], 4));

}

if (Math.Round(sumddx, 4) != Math.Round(-fx, 4) || Math.Round(sumddy, 4) != Math.Round(-fy, 4))

MessageBox.Show("坐标增量分配有误！");

if (Math.Round(sumcx, 4) != Math.Round(xc - xa, 4) || Math.Round(sumcy, 4) != Math.Round(yc - ya, 4))//取四位对吗？

MessageBox.Show("改正后坐标增量计算有误！");

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[8].Value = Convert.ToString(Math.Round(sumddx, 4)); //将坐标增量改正数总和放入表格中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[9].Value = Convert.ToString(Math.Round(sumddy, 4));

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[10].Value = Convert.ToString(Math.Round(sumcx, 4)); //将改正后坐标增量总和放入表格中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[11].Value = Convert.ToString(Math.Round(sumcy, 4));

for(int j=1;j<x.Length -1;j++ )

{

x[j + 1] = x[j] + cx[j]; //计算x,y坐标

y[j + 1] = y[j] + cy[j];

dataGridView1.Rows[j+1].Cells[12].Value= Convert.ToString(Math.Round(x[j+1], 3)); //将x,y坐标放入表格

dataGridView1.Rows[j+1].Cells[13].Value= Convert.ToString(Math.Round(y[j+1], 3));

}

if(Math.Round (x[x.Length -1]+cx[cx.Length -1],3)!=Math.Round (xc,3) || Math.Round (y[y.Length -1]+cy[cy.Length -1],3)!=Math .Round ( yc,3))

MessageBox.Show("坐标计算有误！");

}

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void txt文件ToolStripMenuItem\_Click(object sender, EventArgs e)

{

dataGridView1.DataSource = null;

dataGridView1.Rows.Clear();

dataGridView1.Columns.Clear();

OpenFileDialog file = new OpenFileDialog();

file.Filter = "文本文件|\*.txt";

if (file.ShowDialog() == DialogResult.OK)

{

StreamReader sr = new StreamReader(file.FileName, System.Text.Encoding.Default);

textBox1.Text = sr.ReadToEnd();

sr.Close();

}

else

return;

string[] str = textBox1.Text.Split(new string[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);

string[][] k = new string[str.Length][];

for (int i = 0; i < str.Length; i++)

k[i] = str[i].Split(',');

dataGridView1.RowCount = k.Length;

dataGridView1.ColumnCount = k[0].Length;

for (int i = 0; i < k[0].Length; i++)

dataGridView1.Columns[i].HeaderText = k[0][i];

for (int i = 1; i < k.Length; i++)

{

for (int j = 0; j < k[i].Length; j++)

dataGridView1.Rows[i - 1].Cells[j].Value = k[i][j];

}

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void 输出Excel文件ToolStripMenuItem\_Click(object sender, EventArgs e)

{

Excel.Application ex = new Excel.Application(); //声明一个Excel.Application对象 ex

ex.Visible = true; //使ex可见

ex.Application.Workbooks.Add(true); //在ex中增加一个工作簿

for (int i = 0; i < dataGridView1.ColumnCount; i++) //把dataGridView1中的列名存入ex中

{

ex.Cells[1, i + 1] = dataGridView1.Columns[i].HeaderText;

}

for (int i = 0; i < dataGridView1.RowCount; i++) //把dataGridView1中的数据存入ex中

{

for (int j = 0; j < dataGridView1.ColumnCount; j++) ex.Cells[i + 2, j + 1] = dataGridView1.Rows[i].Cells[j].Value;

}

MessageBox.Show("数据输出已完成!");

}

}

}