# C#编码规范

## **命名规范**

    像C#这样的面向对象语言允许开发人员采用叙述性的方式较灵活地命名类、方法、字段等。对于这些类型的事物，如果可以更清楚地描述事物的含义，那么千万不要害怕使用较长的名称。大家应该使用完整的单词，而不是缩略语。

    .NET有两种主要类型的字母大小写形式：Pascal和camel。对于Pascal字母大小写形式来说，所创建名称的第一个单词的第一个字母是大写的，该名称中的后续单词也是如此，例如ThisIsPascalCase。而对于camel字母大小写形式来说，所创建名称的第一个单词的第一个字母是小写的，而该名称中的后续单词使用大写的字母，例如thisIsCamelCase。

匈牙利表示法(Hungarian notation)是一种在名称内部使用的类型描述方法，可以对所命名的事物的类型提供一个提示。我们在自己的.NET编码规范内相对保守地使用这种方法。在扩展Exception基类的类的命名(例如IllegalArgumentException)以及GUI组件的前缀(例如sumbitBottom)中将发现这种方法的应用。

## **缩进**

    我们可以将Visual Studio 配置为使用制表符或者空格来进行缩进，同时还可设置缩进的字符单位。日常受用的规范使用4个空格的缩进单位。

    代码的行长对于可读性也非常重要。试着将每一行的代码控制在80个字符之内。当每行的字符不超过80个，而又需要换行时，请遵循下述规则：

    ·在操作符前断开

    ·在逗号后断开

·对齐括号

## **声明**

    在声明类字段、实例字段或者局部变量时，每行只声明一个。而当在一起进行几个声明时，需对准字段或者变量的名称。对于局部变量来说，应该在声明变量时就进行初始化，除非您在初始化变量时，还需要执行一些其他的动作，例如计算。

声明应该位于进行声明的类或者方法的顶部，这样就可以使未来对声明的查看变得更容易。该规则的一个例外就是for循环内局部变量的声明和初始化。

## 语句

    C#中有多种类型的语句。每一行代码包含的内容不应该超过一个语句。

    对于if、if-else、和if else-if else 语句来说，总是使用大括号，而对于for、foreach、while、和do-while语句来说，也总是使用大括号（大括号两部分都应该独占一行）。

在return语句中，一般不要使用括号，除非为了使返回值更加明显。

下面是使用switch语句的格式：

switch (expression)

{

    case constant-expression:

    statements;

    break;

    default:

    statements;

    break;

}

而try-catch-finally语句则采用下述格式：

try

{

    statements;

}

catch (exception)

{

    statements;

}

finally

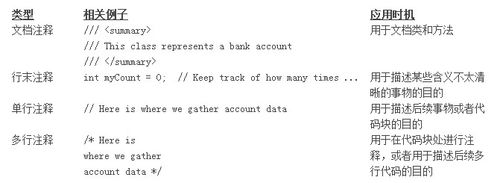
{

    statements;

}

## 注释

C#中可以使用的注释类型也有几种。下表为每一种类型列出了一个例子，说说明了每一种类型的应用时机。



## 空行与空格

    尽管编译器可以忽略空行与空格，但是空行与空格可以将不同逻辑的代码单元分离，从而提高代码的可读性。

#### 在下述几种情况下应该使用一个空行：

   ·方法之间

    ·声明和语句之间

    ·代码的逻辑段之间

    ·单行或者多行注释之前

### 在下述几种情况下应该使用一个空格：

    ·带圆括号的关键字之后

    ·参数列表的逗号之后

    ·数据操作符的前后

    ·在for语句中，用于将语句的3个逻辑段分开

1. 7

解决方案和项目组织

    每个解决方案都应该拥有下述项目：

    ·业务项目（business project）仅存放业务项目。

    ·数据项目（data project）处理数据库和其他用于检索或者更新数据目的的遗留系统的访问。

    ·表示项目（presentiation project）处理使用系统的终端用户所需的所有视图。

    ·测试项目（test project）存放应用程序所有的单元测试。

## 实例代码

privatevoidbutton1\_Click(objectsender, EventArgse)

{

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

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

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

doublesum = 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，并将其转换弧度，放入放入acd中用于计算和检核

for(inti= 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

{

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

doublesumvd= 0;

for(inti = 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(inti= 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

doublex2, y2, x3, y3; //存放已知两个点的x，y坐标

x2 = Convert.ToDouble(dataGridView1.Rows[1].Cells[12].Value);

y2 = Convert.ToDouble(dataGridView1.Rows[1].Cells[13].Value);

x3 = Convert.ToDouble(dataGridView1.Rows[sd.Length -1].Cells[12].Value);

y3 = Convert.ToDouble(dataGridView1.Rows[sd.Length -1].Cells[13].Value);

double[] sl = newdouble[sd.Length -1];//存放观测距离

double[] dx = newdouble[sl.Length];//存放坐标增量

double[] dy = newdouble[sl.Length];

doublesuml = 0, sumdx = 0, sumdy = 0;

for(inti=1;i<sl.Length;i++)

{

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

//将观测距离放到sl数组中

suml += sl[i]; //计算距离总和

dx[i] = sl[i] \* Math.Cos (cr[i]); //利用距离和坐标方位角计算坐标增量

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

sumdx += dx[i]; //计算坐标增量总和

sumdy += dy[i];

}

1. 5