**C#编程规范**

1.目的

为了保证所每个项目组编写出的程序都符合相同的规范，便于提高代码的可读性和可维护性，便于检查、减少出错概率，有助于项目组成员间交流，保证一致性、统一性而建立的C#程序编码规范。

2.范围

该规范适用于所有基于C#开发的.Net平台项目，为详细设计，代码编写和代码审核提供参考和依据。

3.规范内容

1、代码格式

1.1、要使所有的缩进为一个Tab，即4个空格，使用VS.NET的默认设置。

1.2、在代码中垂直对齐左括号和右括号。

if (x == 0)

{

Response.Write("用户编号必须输入！");

}

不允许以下情况：

if (x == 0) {

Response.Write("用户编号必须输入！")；

}

或者：

if (x == 0){ Response.Write("用户编号必须输入！");}

1.3、为了防止在阅读代码时滚动源代码编辑器，每行代码或注释在1024\*768的分辨率下尽量不超过一显示屏，如超过则应换行，换行后的代码应该缩进一个Tab。

1.4、当一行被分为几行时，将串联运算符放在每一行的末尾而不是开头，清楚地表示没有后面的行是不完整的。

1.5、每一行上放置的语句避免超过一条，避免写 int i = 0;int j = 1;

1.6、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;

1.7、将大的复杂代码节分为较小的、易于理解的模块。

1.8、编写 SQL 语句时，建议（不强求）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。

2、注释规范

注释是为了增加代码的清晰度，保持注释的简洁，不是任何代码都需要注释的，过多的注释反而会影响代码的可读性。

注释不要包括其他的特殊字符，建议先写注释，后写代码，注释和代码一起完成。

如果语句块（比如循环和条件分枝的代码块）代码太长，嵌套太多，则在其结束“｝”要加上注释，标志对应的开始语句。如果分支条件逻辑比较复杂，也要加上注释。

注释规范主要包括：模块（类）注释规范、类属性注释规范、方法注释规范、代码间注释规范。

2.1、模块（类）注释规范

模块开始尽量以以下形式书写模块注释：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2.2、类属性注释规范

在类的属性必须以以下格式编写属性注释：

/// <summary>

///属性说明

/// </summary>

2.3、方法注释规范

在类的方法声明前必须以以下格式编写注释：

/// </summary>

        ///函数的作用

/// <param name="<参数名称>"><参数说明></param>

/// <对方法返回值的说明，该说明必须明确说明返回的值代表什么含义>

        ///<exception>异常类型、异常结果说明（为

#endregion

2.4、代码间注释规范

代码间注释分为单行注释和多行注释。

  单行注释：

//<单行注释>（不提倡使用，尽量都使用/\*\*/）

  多行注释：

/\*多行注释1

 \*多行注释2

 \*多行注释3\*/

代码中遇到语句块（if，case……）或实现一个逻辑功能时，必须添加注释。添加的注释应该能够说明此语句块的作用和实现手段（所用算法等等）。

3、命名规范

3.1、类变量命名:类变量命名一律用下划线开头，多个单词组成，第一个字母小写的，其他单词第一个字母大写。

public class Hello

{

private string \_firstName;

private DateTime \_date;

}

3.2、类命名规范:

3.2.1、名字应该能够标识事物的特性；

3.2.2、首字母大写，缩略语可以全部字母大写；

3.2.3、名字可以有两个或三个单词组成，但通常不应多于三个；

3.2.4、在名字中，每个单词的第一个字母大写。例如：IsSuperUser；包含ID的，ID全部大写，如CustomerID；

3.2.5、使用名词或名词短语命名类；

3.2.6、少用缩写，除非是被广泛使用的；

3.2.7、不要使用下划线字符 (\_)。

3.3、局部变量命名：由小写字母开头，其他单词开头字母大写的各单词组成。

int  localVar;

3.4、函数命名：一律由大写字母开头的各个单词组成。

public void GetValue(int type)

{

//…

}

3.5、枚举类型命名:一律由大写字母”E”开头,第一个字母大写的各个单词组。

public enum Esample

{

EMPTY,

A,

B

}

并确保,枚举类型中有个约定的EMPTY元素。

4、控件命名规则：控件命名=控件缩写前缀+变量具体释义。

开发程序时，往往会有很多人参与，为了使程序的结构与代码风格标准化，以便使每个参与开发的人员尽可能直观地查看和理解其他人编写的代码，需要在编码之前制定一套统一的编码规范。下面介绍了一套C#中常用的程序编码规范供读者参考。

B.1  命 名 规 则

B.1.1  数据类型简写规则

程序中定义常量、变量或方法等内容时，常常需要指定类型。下面介绍一种常见的数据类型简写规则，如表B.1所示。

表B.1  数据类型简写规则

|  |  |  |  |
| --- | --- | --- | --- |
| 数 据 类 型 | 简    写 | 数 据 类 型 | 简    写 |
| 整型 | int | [长整型](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10636914&ss_c=ssc.citiao.link" \t "_blank) | lint |
| 字符串 | str | 单精度浮点型 | flt |
| [布尔型](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=54150574&ss_c=ssc.citiao.link" \t "_blank) | bl | 双精度浮点型 | dbl |
| 短整型 | sint | 字节型 | bt |

B.1.2  变量及对象名称命名规则

根据不同的程序需要，编写代码时都需要定义一定的变量或常量。下面介绍一种常见的变量及常量命名规则，如表B.2所示。

表B.2  变量及常量命名规则

|  |  |  |
| --- | --- | --- |
| 变量及常量级别 | 命 名 规 则 | 举    例 |
| 模块级变量 | M\_+数据类型简写+变量名称 | M\_int\_xx |
| 全局变量 | G\_+数据类型简写+变量名称 | G\_int\_xx |
| 局部变量 | P\_+数据类型简写+变量名称 | P\_dbl\_sl |
| 模块级常量 | Mc\_+数据类型简写+常量名称 | Mc\_str\_xx |
| 全局常量 | Gc\_+数据类型简写+常量名称 | Gc\_str\_xx |
| 过程级常量 | Pc\_+数据类型简写+常量名称 | Pc\_str\_xx |

B.1.3  类命名规则

使用完整的单词，避免使用缩写。

一般使用名词。类通常表示一种对象，而对象一般使用具有名词性质的名称。

不要使用C或下划线作为前缀。尽管在Visual Basic和Visual C++等其他语言中，这是非常普遍的， .NET框架提供的类中没有以C为前缀的类名。

异常类命名通常以Exception结尾，但应尽量避免自定义异常类。

避免将多个类放在一个文件中。

初始化类的实例时，除非十分必要，否则不要赋值为null。

一个类文件里应该有且仅有一个命名空间，避免在一个类文件里包含多个不同的命名空间。

避免在一个类文件里代码超过500行（除去自动生成的代码）。

B.1.4  属性命名规则

属性名称通常都是名词，例如Text、SelectedIndex和Width等。

避免和类型名称重复。如果想要使用一个类型名称作为属性的名称，那么属性必须是该类型的数据。

B.1.5  方法命名规则

方法名通常是动词，因为方法用来实现一个对象的行为功能，例如Read、Write、Start和Stop等。

当重载方法时，参数的顺序和命名要一致。

定义方法名时不要使用缩写。如果认为应用程序中的某些项目应使用缩写，则应将这些情况加上注释，并确保每个人在所有时间内都使用这些缩写。一定不要在某些方法中对某些单词进行缩写，而在别的方法中却不使用缩写。

一个方法的代码长度最好不要超过25行。

避免方法中有超过5个参数的情况，如果超过了，则应使用struct结构来传递多个参数。

避免方法的返回值是错误代码。

B.1.6  接口命名规则

在接口名称前添加前缀I 。

接口名称尽量具有描述性，并使用形容词，因为接口常常用于给一个类添加某种能力，比如[IComparable](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10865979&ss_c=ssc.citiao.link" \t "_blank)和IFormattable等。

避免一个接口中只有一个成员，尽量使每个接口中包含3～5个成员。

接口中的成员不应该超过20个。

避免接口成员中包含事件。

避免使用[抽象方法](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=74295678&ss_c=ssc.citiao.link" \t "_blank)，而使用接口替换。

在类层次中显示接口。

推荐使用显式的接口实现。

B.1.7  控件命名规则

所有的控件名称都应该为自然名称的拼音简写，出现冲突可采用不同的简写规则。另外，在编码过程中涉及不到编码的控件，其名称可以取默认名称。控件命名规则如表B.3所示。

表B.3  控件命名规则

|  |  |
| --- | --- |
| 控    件 | 缩 写 形 式 |
| Form | frm |
| TextBox | txt |
| Button | btn |
| ComboBox | cbox |
| Label | lab |
| DataGridView | dgv |

B.2  代码版式

B.2.1  统一代码缩进

每个项目都应该根据实际情况建立统一的代码缩进标准，比如统一缩进两个字符，这样代码看起来非常美观，而且层次感很清楚，方便代码的阅读。

B.2.2  合理使用空行

在自动生成的命名空间和手动添加的命名空间之间要加空行分隔。

在每个类声明之后、每个方法声明之后都要加空行。

在一个方法体内，逻辑上密切相关的语句之间不加空行，其他地方应该加空行分隔。

B.2.3  代码换行

一行代码只做一件事情，如只定义一个变量，或只编写一条语句。

if、switch、for、foreach、while及do…while等语句独占一行，执行语句不能紧跟其后；而且，不论执行语句有多少都应该加{}。

代码行最大长度应该控制在70～80个字符以内，代码不要过长，否则不便于查看。

长表达式要在低优先级运算符处拆分成新行，运算符放在新行之首（以便突出运算符）。拆分的新行要适当缩进，以便使排版整齐。

B.2.4  空格的使用

关键字后面应该加空格，否则无法解析关键字。

if、for、while等控制语句关键字后面应该加空格，然后再跟左括号"("，以便突出关键字。

方法名后面不要加空格，而应该紧跟"("，以便与关键字进行区分。

"("向后紧跟，")"、","、";"向前紧跟，紧跟处不应该留空格。

赋值运算符、比较运算符、算术运算符、逻辑运算符和位运算符等二元运算符的前后应该加空格。

一元运算符如"!"、"++"、"--"、"&"等的前后不加空格。

类似"[]"、"."、"->"这类符号的前后不加空格。

对于表达式比较长的if语句和for语句，为了看上去更加紧凑，可以适当地去掉一些空格。

B.3  代码注释

B.3.1  注释的目的

文字说明代码的作用（即为什么要编写该代码，而不是如何编写）。

指出该代码的编写思路和逻辑方法。

代码中的重要转折点。

使代码的阅读者不必在他们的头脑中仿真运行代码的执行方法。

B.3.2  代码注释规范

方法体内使用类似"//"的形式进行注释，前面需要使用空格进行对齐时，使用Tab键替代空格键。

对类文件进行属性注释说明。

避免对浅显易懂的语句进行注释说明。

代码应具有解释其自身作用的功能。好的代码可以通过可读的变量和方法名了解该段代码的作用，而不需要过多的注释。

逻辑性较强的地方加入注释，说明这段程序的逻辑是怎样的，以方便其他人理解，并且还可以在一定程度上排除Bug。

使用注释来说明何时可能出错和为什么出错。

在编写代码前进行注释。

纯色字符注释行只用于主要注释。

避免形成注释框。用星号围成的注释框，右边的星号看起来很好，但却给编写或编辑注释的人增加了许多工作。

增强注释的可读性。注释是供人阅读的，而不是让计算机阅读，所以应该使用完整的语句。

对注释进行缩进，使之与后随的语句对齐。注释通常位于它们要说明的代码的前面，为了从视觉上突出注释与代码之间的关系，可将注释缩进，使之与代码处于同一个层次上。

在每个if语句的前面加上注释。

在每个switch语句的前面加上注释。

在每个循环语句的前面加上注释。

代码：

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