**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。

**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。

**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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=66636822&ss_c=ssc.citiao.link" \t "_blank)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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=142690732&ss_c=ssc.citiao.link" \t "_blank)，每行代码或注释在[1024\*768](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=68525725&ss_c=ssc.citiao.link)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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=142255302&ss_c=ssc.citiao.link" \t "_blank)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。

1.9、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，例如：

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：

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

 \* 文 件 名:

\* 功能描述：

\* [Copyright](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=300434&ss_c=ssc.citiao.link) (c) 2010-2012 \*\*\*\*\*\*\*\*\*公司技术开发部

\* 创 建 人:

\* 日    期:

\* 修 改 人:

\* 修改日期:

\* 修改描述:

\* 版    本:

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

注：这里的修改人，是指对整个类进行修改过，需要在这些添加修改记录，且记录累加。

当模块有重大修改或修改他人代码时，必须添加以下注释：

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

\* 修 改 人:

\* 日    期:

\* 修改描述：<对此修改的描述>

\* 版    本:

\* 修 改 人:

\* 日    期:

\* 修改描述：<对此修改的描述>

\* 版    本:

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

修改时根据需要保留程序员第一次和最近一次修改的代码。具体办法是将原有代码注释。(保留第一次的代码可以反映此部分代码最初要实现的功能及其初始编程思想，保留最新的代码可以反映此代码最近变化要实现的最新功能)

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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=581547&ss_c=ssc.citiao.link" \t "_blank)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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=63796783&ss_c=ssc.citiao.link)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。

3.6、接口命名规范: 和类命名规范相同，唯一区别是接口在名字前加上"I"前缀。

///

///接口样例

///

public interface ISample

{

///

///获取对象

///

object GetValue();

}

3.7、[抽象类](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7766757&ss_c=ssc.citiao.link" \t "_blank)的命名:一律由大写字母”A”开头,第一个字母大写的各单词组成。

///

///抽象类样例

}

3.13、方法（Method）、属性（Atribute）命名规范

3.13.1、方法命名应采用首字母大写的方式；

3.13.2、返回值是Bool值的方法和属性以Is开头；

3.13.3、使用返回值是Bool值的方法（属性）的时候，不要将bool值方法（属性）与if语句放在同一行，如应该这样写：

bool valid = IsValid();

if (valid)

{

}

不应这样写：

if (IsValid())

{

}

3.14、类的实例命名规范：类的实例第一个字母小写，其他单词的第一个字母大写。

FileStream fileStream=new FileStream();

**4、异常处理规范**

异常变量命名：针对异常捕获过程中的[Exception](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7625090&ss_c=ssc.citiao.link" \t "_blank)变量命名，统一命名为ex1，[ex2](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55598959&ss_c=ssc.citiao.link" \t "_blank)……

try

{

// code1

try

{

// code2

}

catch(Exception ex2)

{

//your code

}

}

catch(Exception ex1)

{

//your code

}

如果捕获异常不需要作任何处理，则不需要定义Exception实例。

try

{

//your code

}

catch( Exception )

{

}

不必每个方法都用try-catch。当特定的异常可能发生时才使用。比如，当你写文件时，处理异常FileIOException。

别写太大的 try-catch 模块。如果需要，为每个执行的任务编写单独的 try-catch 模块。 这将帮你找出哪一段代码产生异常，并给用户发出特定的错误消息。

如果应用程序需要，可以编写自己的异常类。自定义异常不应从基类SystemException派生，而要继承于IApplicationException。

**5、控件命名规则**：控件命名=控件缩写前缀+变量具体释义。

|  |  |  |
| --- | --- | --- |
| 控件名称 | 前缀 | 例子 |
| Button | btn | btnSubmit |
| CheckBox | chk | chkReadOnly |
| ComboBox | cbo | cboEnglish |
| Command | cmd | cmdDelete |
| [Connection](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=99672510&ss_c=ssc.citiao.link) | con | conOracle |
| [DataAdapter](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=59252236&ss_c=ssc.citiao.link) | dad | dadDiction |
| [DataColumn](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=69152893&ss_c=ssc.citiao.link) | dcol | dcolDiction |
| DataGrid | dgd | dgdTitles |
| [DataReader](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=9177376&ss_c=ssc.citiao.link) | dr | drDiction |
| [DataRelation](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=71493373&ss_c=ssc.citiao.link) | drel | drelDiction |
| DataRow | drow | drowDiction |
| DataSet | ds | dsDiction |
| [DataTable](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7835293&ss_c=ssc.citiao.link) | dt | dtDiction |
| DataView | dv | dvDiction |
| DateTimePicker | dtp | dtpPublished |
| [DropDownList](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8750484&ss_c=ssc.citiao.link) | drp | drpOffice |
| Form | frm | frmMain |
| ImageList | ils | ilsAllIcons |
| Label | lbl | lblHelpMessage |
| ListBox | lst | lstPolicyCodes |
| ListView | lv | lvHeadings |
| Menu | mnu | mnuFileOpen |
| MonthCalender | mcd | mcdPeriod |
| Parameter | parm | parmIn |
| PictureBox | pic | picVGA |
| [ProgressBar](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=57159109&ss_c=ssc.citiao.link) | prg | prgLoadFile |
| [RadioButton](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55808598&ss_c=ssc.citiao.link) | rad | radGender |
| RichTextBox | rtb | rtbReport |
| StatusBar | stab | stabDateTime |
| TabControl | tab | tabOptions |
| TextBox | txt | txtLastName |
| ToolBar | tlb | tlbActions |
| TreeView | trv | trvOrganization |
| [ScriptManager](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=2362711&ss_c=ssc.citiao.link) | smg | smgContent |
| UpdatePanel | upd | updQuery |
| Panel | pl | plQuery |
| GridView | gv | gvShow |
| UpdateProgress | upp | uppUpdate |
| MultiView | mulv | mulvContent |
| View | vie | vieAdd |
| FormView | fvi | fviAdd |
| HiddenField | hf | hfMian |
| ObjectDataSource | ods | odsData |
| DynamicPopulateExtender | dpe | dpeText |
| FilteredTextBoxExtender | ftbe | ftbeNum |

报表浏览控件 rpt  rptMonth

用户自定义控件 ucl  uclCustom

**6、调用规范**

6.1、类内变量和函数：类内函数和变量的调用一定要在前面加上”this.”；

6.2、类常量和类[静态变量](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=786565&ss_c=ssc.citiao.link" \t "_blank)，以及[静态方法](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10695808&ss_c=ssc.citiao.link)：类内静态元素(元素包括变量和方法)的调用，一定要在前面加上该类的名字；

6.3、系统类的使用：系统类型的使用，.Net中对于基本数值类型,都有相应的引用类型定义，在一般表示的情况下，我们使用小写字母开头的类型，而用到他们方法和数据的时候，则使用引用空间的名称。比如string 有 System.String对应，int 有System.Int32对应。

//注意string和String的使用地方

string strSample = String.Empty;

//注意int和Int32的使用地方

int intSample = 0;

System.Int32.Equals(intSample,3);

6.4、引用类型转换：我们不要使用类型加在前面的转换方式，而使用as，这样当转换不成功时，也不会有异常，但可以根据转换结果进行判断。

object obj;

不要：

ISample sam = (ISample)obj;

而要：

ISample sam = obj as ISample;

if(null == sam)

{ //转换失败

}

**7、DotNet编程提醒**

7.1、循环语句

对一个容器进行循环，如果该容器已经实现了IEnumerable或者IEnumerator接口，那么我们推荐使用foreach…in…语句，例如:ICollection coll中存放了一些ISample对象，那么我们的遍历：

 foreach(ISample samp in coll)

{

//对samp处理

……

}

对于经典的[for循环](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=654572&ss_c=ssc.citiao.link" \t "_blank)：

for(int i=0;i<COLL.COUNT();i++)

{

ISample samp = coll[i] as ISample;

}

我们应该这样写：避免循环的每次都调用coll.Count()方法，而不必要的损失性能。

for(int i=0,size=coll.Count();i< size;++i)

{

ISample samp = coll[i] as ISample;

}

7.2、在需要大量字符串连接的时候,要使用[StringBuilder](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=66238158&ss_c=ssc.citiao.link" \t "_blank)类，而不是使用string+string+string……的方法：

不要：

string strSample = “a”+”b”+”c”+”d”+”e”;

而要：

StringBuilder sb = new StringBuilder();

sb.append(“a”);

sb.append(“b”);

sb.append(“c”);

sb.append(“d”);

sb.append(“e”);

string strSample = sb.ToString();

在需要使用格式化字符串时，使用String.Format而不是使用字符串相加，例如

不要：

string strName = “Shawn”;

string strAge = “23”;

string strSample = “I am ”+strName+”,my age is ”+strAge;

而要：

string strSample =

String.Format(“I am {0},my age is {1}”,strName,strAge);

**8、版本管理**

所有项目的源代码或项目资料，建议采用Microsoft Visual SourceSafe（VSS）或[TortoiseSVN](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=4225739&ss_c=ssc.citiao.link" \t "_blank)进行版本控制。

附录B  C#程序编码规范

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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=398075&ss_c=ssc.citiao.link)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) | 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) | 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，但在.[NET框架](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=584747&ss_c=ssc.citiao.link)中不提倡使用，.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。

尽量不要提供public和[protected](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7667016&ss_c=ssc.citiao.link" \t "_blank)的成员变量，使用属性代替它们。

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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=29490967&ss_c=ssc.citiao.link) | dgv |
| ListBox | lb |
| Timer | tmr |
| CheckBox | ckbox |
| RichTextBox | rtbox |
| CheckedListBox | clbox |
| RadioButton | rbtn |
| NumericUpDown | nudown |
| Panel | pl |
| GroupBox | gbox |
| ListView | lv |
| TreeView | tv |
| PictrueBox | pbox |
| …… | …… |

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](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55257294&ss_c=ssc.citiao.link)、[比较运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8011293&ss_c=ssc.citiao.link)、[算术运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=56555579&ss_c=ssc.citiao.link)、[逻辑运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=54086220&ss_c=ssc.citiao.link)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。

B.4  数据库编程命名规范

B.4.1  数据库命名规范

数据库命名以字母"db"开头（小写），后面加数据库相关英文单词或缩写。下面举例说明，如表B.4所示。

表B.4  数据库命名

|  |  |
| --- | --- |
| 数据库名称 | 描    述 |
| db\_PWMS | [企业人事管理系统](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10117693&ss_c=ssc.citiao.link)数据库 |

B.4.2  数据表命名规范

数据表以字母"tb"开头（小写），后面加数据表相关英文单词或缩写。下面举例说明，如表B.5所示。

表B.5  数据表命名

|  |  |
| --- | --- |
| 数据表名称 | 描    述 |
| tb\_Login | 登录信息表 |

B.4.3  字段命名规范

字段一率采用英文单词或词组（可利用[翻译软件](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8479185&ss_c=ssc.citiao.link" \t "_blank)）命名，如找不到专业的英文单词或词组，可以用相同意义的英文单词或词组代替。下面举例说明，如表B.6所示。

表B.6  字段命名

|  |  |
| --- | --- |
| 字 段 名 称 | 描    述 |
| name | 名字 |
| pwd | 密码 |

B.4.4  视图命名规范

视图命名以字母"view"开头（小写），后面加表示该视图作用的相关英文单词或缩写。下面举例说明，如表B.7所示。

表B.7  视图命名

|  |  |
| --- | --- |
| 视 图 名 称 | 描    述 |
| view\_AdminInfo | 视图全名 |

view：视图。

AdminInfo：查看管理员信息。

B.4.5  存储过程命名规范

存储过程命名以字母"proc"开头（小写），后面加表示该存储过程作用的相关英文单词或缩写。下面举例说明，如表B.8所示。

表B.8  存储过程命名

|  |  |
| --- | --- |
| 存储过程名称 | 描    述 |
| proc\_Login | 存储过程全名 |

proc：存储过程。

Login：实现登录功能。

B.4.6  触发器命名规范

触发器命名以字母"trig"开头（小写），后面加表示该触发器作用的相关英文单词或缩写。下面举例说明，如表B.9所示。

表B.9  触发器命名

|  |  |
| --- | --- |
| 触发器名称 | 描    述 |
| trig\_inAdmin | 触发器全名 |

trig：触发器。

inAdmin：添加管理员信息。

**代码示例**

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

{

double sum = 0;

for (int i = 1; i < sdr.Length; i++)//从第二行开始循环计算 坐标方位角、观测角度累加值

{

cr[i] = cr[i - 1] + sdr[i] - Math.PI;//计算坐标方位角/左角

if (cr[i] >= Math.PI \* 2)//判断坐标方位角是否在0到2PI之间

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

else if (cr[i] < 0.0)

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

sum += sdr[i];

}

return sum;

}

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

{

dataGridView1.DataSource = null; //清除数据源

dataGridView1.Rows.Clear(); //清空数据表格的行列

dataGridView1.Columns.Clear();

OpenFileDialog file = new OpenFileDialog(); //声明 打开文件对话框 file

file.Filter = "Excel文件|\*.xls|Excel文件|\*.xlsx"; //文件过滤器，只显示Excel文件

if (file.ShowDialog() == DialogResult.OK) //如果文件正常打开

{

string fname = file.FileName; //获取打开的文件名称

string strSource = "provider=Microsoft.ACE.OLEDB.12.0;" + "Data Source=" + fname + ";Extended Properties='Excel 8.0; HDR=Yes;IMEX=1'"; //准备文件来源信息

OleDbConnection conn = new OleDbConnection(strSource); //Excel文件源放到conn中

string sqlstring = "SELECT \* FROM [Sheet1$]"; //准备选择表中的Sheet1

OleDbDataAdapter adapter = new OleDbDataAdapter(sqlstring, conn); //声明数据适配器adapter

DataSet da = new DataSet(); //声明数据集da

adapter.Fill(da); //使用adapter填充方法

dataGridView1.DataSource = da.Tables[0]; //将da.Tables[0]作为dataGridView1的数据源

}

else

return;

}

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]);

}

//坐标增量调整与计算

double[] lg = new double[cr.Length - 1];//新建一个数组存放观测长度的原始值

double[] lgx = new double[cr.Length - 1];//新建一个数组存放△x

double[] lgy = new double[cr.Length - 1];//新建一个数组存放△y

double sum1 = 0;

double sum2 = 0;

double sum3 = 0;

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

{

lg[1] = Convert.ToDouble(dataGridView1.Rows[1].Cells[5].Value);//将获取的距离导入数组

lg[2] = Convert.ToDouble(dataGridView1.Rows[2].Cells[5].Value);

lg[3] = Convert.ToDouble(dataGridView1.Rows[3].Cells[5].Value);

lg[4] = Convert.ToDouble(dataGridView1.Rows[4].Cells[5].Value);

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

lgx[i] = lg[i] \* Math.Cos(cr[i]);//计算坐标增量

lgy[i] = lg[i] \* Math.Sin(cr[i]);

lgx[i] = Math.Round(lgx[i], 4);

lgy[i] = Math.Round(lgy[i], 4);

dataGridView1.Rows[i].Cells[6].Value = lgx[i];//

dataGridView1.Rows[i].Cells[7].Value = lgy[i];

sum1 = lg[1]+lg[2]+lg[3]+lg[4]+lg[5];//计算距离总和

sum2 += Math.Round(lgx[i], 4);//计算△x总和

sum3 += Math.Round(lgy[i], 4);//计算△y总和

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[5].Value = sum1;

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[6].Value = sum2;

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[7].Value = sum3;

}

double zuobiaoyA, zuobiaoyC;//新建一个数组存放x坐标值

double zbzlx, zbzly, qcbhc, xdbhc;

double[] x = new double[cr.Length - 1];

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

x[1] = Convert.ToDouble(dataGridView1.Rows[1].Cells[12].Value);//获取x坐标

zuobiaoyA = Convert.ToDouble(dataGridView1.Rows[6].Cells[12].Value);//获取y坐标

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

zuobiaoyC = Convert.ToDouble(dataGridView1.Rows[6].Cells[13].Value);

zbzlx = sum2 - (zuobiaoyA - x[1]);//计算坐标增量闭合差

zbzly = sum3 - (zuobiaoyC - y[1]);

qcbhc = Math.Sqrt(zbzlx \* zbzlx + zbzly \* zbzly);//计算导线全长闭合差

xdbhc = sum1 / qcbhc;

dataGridView1.Rows[9].Cells[7].Value = Math.Round(zbzlx, 4);//坐标增量放入表格

dataGridView1.Rows[10].Cells[7].Value = Math.Round(zbzly, 4);

dataGridView1.Rows[9].Cells[10].Value = Math.Round(qcbhc, 4);

dataGridView1.Rows[10].Cells[11].Value = Math.Round(xdbhc, 4);

double[] gzsx = new double[cr.Length - 1];//定义数组存放改正数

double[] gzsy = new double[cr.Length - 1];

double sum4 = 0;

double sum5 = 0;

double[] gzhx = new double[cr.Length - 1];//定义改正后数组

double[] gzhy = new double[cr.Length - 1];

double sum6 = 0;

double sum7 = 0;

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

{

gzsx[k] = -zbzlx \* lg[k] / sum1;//计算改正数

gzsy[k] = -zbzly \* lg[k] / sum1;

sum4 += gzsx[k];

sum5 += gzsy[k];

dataGridView1.Rows[k].Cells[8].Value = Math.Round(gzsx[k], 4);

dataGridView1.Rows[k].Cells[9].Value = Math.Round(gzsy[k], 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[8].Value = Math.Round(sum4, 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[9].Value = Math.Round(sum5, 4);

gzhx[k] = lgx[k] + gzsx[k];//计算改正后坐标增量

gzhy[k] = lgy[k] + gzsy[k];

sum6 += gzhx[k];

sum7 += gzhy[k];

dataGridView1.Rows[k].Cells[10].Value = Math.Round(gzhx[k], 4);

dataGridView1.Rows[k].Cells[11].Value = Math.Round(gzhy[k], 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[10].Value = Math.Round(sum6, 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[11].Value = Math.Round(sum7, 4);

}

for (int j = 2; j < lg.Length; j++)

{

x[j] = x[j - 1] + gzhx[j];

y[j] = y[j - 1] + gzhy[j];

dataGridView1.Rows[j].Cells[12].Value = Math.Round(x[j], 4);

dataGridView1.Rows[j].Cells[13].Value = Math.Round(y[j], 4);

}

}

}

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

{

Application.Exit();

}

}

}