西南石油大学图像处理与并行计算实验室

C#编程规范

Version 1.0 版

起草日期：2018-12-12

目录

1. 概述

1.1 基本原则

1. 程序结构清晰，简单易懂。
2. 代码精简，避免垃圾程序。
3. 尽量使用标准库函数和公共函数。
4. 谨慎使用全局变量，尽量使用局部变量。
5. 重视代码注释，对于关键代码必须进行注释说明，具体注释要求见规范。
6. 除约定俗成外，尽量不用缩写。
7. 除用于循环外，不要使用单个字母的变量。
8. 类模块必须明确构造函数应完成什么工作。
9. 必须设置足够的错误陷阱，防止进程的异常终止。
10. 重复使用的完成相对独立功能的算法或代码应抽象为公共控件或类。
11. 公共控件或类应考虑 OOP 思想，减少外界联系，考虑独立性或封装性。做到高内聚，低耦合。
12. 尽量保证一个文件只定义一个类。
13. 类中所有的字段都应该时私有的（private），利用 get，set 访问器，为字段设置属性，提供外部访问。

1.2 排版

1. 代码列宽控制在 110 字符左右。
2. 单个函数的程序行数原则上不超过 100 行。
3. 在一段相对独立的功能代码、或者函数、属性之间插入空行。
4. 较长的语句（超出列宽）要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。
5. 若函数或过程中的参数较长，则要进行适当的划分。
6. 程序块要采用缩进风格编写，缩进为一个TAB（统一为4 个空格），不要在代码中使用Tab 字符。Visual Studio.Net 设置：工具->选项->文本编辑器

->C#->制表符->插入空格。

7. 除非密切相关，否则不允许把多个短语句写在一行中，即一行只能写一条语句。

8. if、for、do、while、case、switch、default 等语句自占一行，且if、for、do、while 等语句后一定要使用{}，即使{}号中为空或只有一条语句。

9. 函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case 语句下的情况处理语句也要遵从语句缩进要求。

2.命名规范

2.1 命名方法

命名方式主要有：匈牙利命名法、Pascal 命名法 、和 Camel 命名法 三种。

匈牙利命名法：变量名=属性+类型+对象描述，标识符的名字以一个或者多个小写字母开头作为前缀，用以表示该数据的类型；前缀之后的是首字母大写的一个单词或多个单词组合，该单词要指明变量的用途。例如 List<CasingPipe> lstCasingPipe, String strName, Button btnOK 等。

Pascal：每个单词的首字母大写，例如 TubeWell；

Camel：首个单词的首字母小写，其余单词的首字母大写，例如 computePressure。

以下几点是推荐的命名方法。

1. 避免容易被主观解释的难懂的名称，如方法名 AnalyzeThis()，或者属性名 xxK8，这样的名称会导致多义性。
2. 在类属性的名称中包含类名是多余的，如

```
class Tubular
{
    double tubularID;      //管柱内径
    double tubularOD;    //管柱外径
}
```

则调用上述字段的时候，会出现冗余。例如 tubular. tubularID。

而是应该使用如下的定义方法：

```
class Tubular
{
    double id;          //内径
    double od;          //外径
}
```

则调用上述字段的时候，不会出现信息冗余现象，tubular.id, tubular.od。

3. 只要合适，在变量名的末尾或开头加计算限定符（Avg、Sum、Min、Max、Index）。

4. 在命名时，使用互补词，如 min/max、begin/end 和 open/close。
5. 布尔变量名应该包含 is，这意味着 Yes/No 或 True/False 值，如 isFileFound。
6. 在命名状态变量时，避免使用诸如 Flag 的术语。状态变量不同于布尔变量的地方是它可以具有两个以上的可能值。不是使用 documentFlag，而是使用更具描述性的名称，如 documentFormatType。 （此项只供参考）
7. 即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环索引使用单字母变量名，如 i 或 j。

2.2 控件命名

用小写字母的前缀表示变量的类型，前缀的下一个字母用大写。

变量类型	缩写格式	标准命名举例
Array	arr	arrShoppingList
Byte	byt	bytPixelValue
Char	chr	chrDelimiter
DateTime	dtm	dtmStartDate
String	str	strFirstName
Object	obj	objReturnValue

常用 WinForm 控件命名方法如下，用小写前缀表示控件类型，具体命名方法如下表所示。

控件名	控件名简写	标准命名举例
Button	btn	btnSubmit
Label	lbl	lblResults
TextBox	txt	txtFirstName
CheckBox	chk	chkBlue
RadioButton	rad	radFemale
ListBox	lst	lstCountries
CheckBoxList	chkl	chklstFavColors
RadioButtonList	radl	radlGender
DropDownList	drop	dropCountries
Image	img	imgAuntBetty
Calendar	cal	calMettingDates
CompareValidator	valc	valcValidAge
CustomValidator	valx	valxDBCheck
DataGrid	dgrd	dgrdTitles
DataList	dlst	dlstTitles
HyperLink	lnk	lnkDetails
ImageButton	ibtn	ibtnSubmit
LinkButton	lbtn	lbtnSubmit
Panel	pnl	pnlForm2
RangeValidator	valg	valgAge
RegularExpression	vale	valeEmail_Validator
Repeater	rpt	rptQueryResults
RequiredFieldValidator	valr	valrFirstName
Table	tbl	tblCountryCodes

2.4 类命名

1. 类的命名使用 Pascal 命名法。

2. 由于类表示现实世界中的一个实体或者概念，用名词或名词短语命名类。

3. 使用全称避免缩写，除非缩写已是一种公认的约定，如 URL、HTML。

4. 不要使用类型前缀，如在类名称上对类使用C 前缀。例如，使用类名称 FileStream，而不是CFileStream。

5. 不要使用下划线字符(_)。

6. 在适当的地方，使用复合单词命名派生的类。派生类名称的第二个部分应当是基类的名称。例如，基类Exception， 派生类ApplicationException，IOException，DriverException。

2.5 接口命名

1. 接口的命名使用 Pascal 命名法。

2. 用名词或名词短语，或者描述行为的形容词命名接口。例如，接口名称IComponent使用描述性名词。接口名称ICustomAttributeProvider 使用名词短语。

3. 使用全称避免缩写。

4. 给接口名称加上字母I 前缀，以指示该类型为接口。例如IServiceProvider，IFormatable。

5.不要使用下划线字符(_)。

6.当类是接口的标准执行时，定义这一对类/接口组合就要使用相似的名称。两个名称的不同之处只是接口名前有一个 I 前缀。

2.6 变量命名

1. 局部变量/类中的字段/方法中的形参的命名使用 Camel 命名法。例如 double casingPressure。

2. 使用全称避免缩写。

3. 不要使用下划线字符(_)。

2.7 方法命名

1. 方法使用 Pascal 命名法。例如

```
public void PrintDocument(sting fileName);
public void OpenFile(sting fileName);
public void ComputeTemperature(double depth);
```

2. 使用全称避免缩写。
3. 不要使用下划线字符(_)。

3.注释

1. 修改代码时，总是使代码周围的注释保持最新。
2. 在每个例程的开始，提供标准的注释样本以指示例程的用途、假设和限制很有帮助。注释样本应该是解释它为什么存在和可以做什么的简短介绍.
3. 避免在代码行的末尾添加注释；行尾注释使代码更难阅读。不过在批注变量声明时，行尾注释是合适的；在这种情况下，将所有行尾注释在公共制表位处对齐。
4. 避免杂乱的注释，如一整行星号。而是应该使用空白将注释同代码分开。
5. 避免在块注释的周围加上印刷框。这样看起来可能很漂亮，但是难于维护。
6. 在部署发布之前，移除所有临时或无关的注释，以避免在日后的维护工作中产生混乱。
7. 如果需要用注释来解释复杂的代码节，请检查此代码以确定是否应该重写它。尽一切可能不注释难以理解的代码，而应该重写它。尽管一般不应该为了使代码更简单以便于人们使用而牺牲性能，但必须保持性能和可维护性之间的平衡。
8. 在编写注释时使用完整的句子。注释应该阐明代码，而不应该增加多义性。
9. 在编写代码时就注释，因为以后很可能没有时间这样做。另外，如果有机会复查已编写的代码，在今天看来很明显的东西六周以后或许就不明显了。
10. 避免多余的或不适当的注释，如幽默的不重要的备注。
11. 使用注释来解释代码的意图。它们不应作为代码的联机翻译。
12. 注释代码中不十分明显的任何内容。
13. 为了防止问题反复出现，对错误修复和解决方法代码总是使用注释，尤其是在团队环境中。
14. 对由循环和逻辑分支组成的代码使用注释。这些是帮助源代码读者的主要方面。
15. 在整个应用程序中，使用具有一致的标点和结构的统一样式来构造注释。
16. 用空白将注释同注释分隔符分开。在没有颜色提示的情况下查看注释时，这样做会使注释很明显且容易被找到。
17. 在所有的代码修改处加上修改标识的注释。
18. 为了是层次清晰，在闭合的右花括号后注释该闭合所对应的起点。

```
namespace Langchao.Procument.Web
{
} // namespace Langchao.Procument.Web
```

3.1 文件注释

在每个文件头必须包含以下注释说明

```
/*----------------------------------------------------------------
// Copyright (C) 2018 图像处理与并行计算实验室 版权所有。
//
// 文件功能描述：
//作者：XXX
//时间：2018-12-12
//----------------------------------------------------------------*/
```

文件功能描述只需简述，具体详情在类的注释中描述。

3.2 接口等注释

接口、类、方法、属性、字段等注释使用///三斜线注释，这种注释是基于XML 的，不仅能导出XML 制作帮助文档，而且在各个接口、类、方法、属性、字段等的使用中，编辑环境会自动带出注释，方便你的开发。如：

```
        /// <summary>
        /// 测试管柱测试参数
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void barButtonItem6_ItemClick(object sender,
DevExpress.XtraBars.ItemClickEventArgs e)
        {
            TubeTestdata tubeTestdata = new TubeTestdata();
            tubeTestdata.ShowDialog();        }
```

4.声明

1. 尽可能在变量声明时就对其做初始化（就近原则）。

2. 变量尽可能置于块的开始处，不要总是在第一次使用它们的地方做声明。如

```
void MyMethod()
{
   int rowNumber = 0; // beginning of method block
   if (condition)
   {
      Double totPressure = 0; // beginning of "if" block
      ...
   }
}
```

3. 应避免不同层次间的变量重名，如：

```
int count;
...
void MyMethod()
{
   if (condition)
   {
      int count = 0; // 避免
      ...
   }
   ...
}
```

3. 不要使用public 或protected 的实例字段。考虑为字段提供get 和set 属性访问器，而不是使它们成为公共的。get 和set 属性访问器中可执行代码的存在使得可以进行后续改进，如在使用属性或者得到属性更改通知时根据需要创建对象。下面的代码示例阐释带有get 和set 属性访问器的私有实例字段的正确使用。示例：

```
public class Tubular
{
   private double maxPressure;
   public double MaxPressure
   {
       get
       {
           return maxPressure;
       }
   }
}
```