**C# 编码规范**

**目录**

[1 目标 4](#_Toc421624992)

[2 概述 4](#_Toc421624993)

[3 总体要求 4](#_Toc421624994)

[3.1 程序结构化 4](#_Toc421624995)

[3.2 代码可读性 4](#_Toc421624996)

[3.3 代码结构化 4](#_Toc421624997)

[3.4 正确性与容错性 5](#_Toc421624998)

[4 编码规范 5](#_Toc421624999)

[4.1 文件结构 5](#_Toc421625000)

[4.1.1 C# 文件 5](#_Toc421625001)

[4.1.2 目录结构 5](#_Toc421625002)

[4.2 缩进 5](#_Toc421625003)

[4.2.1 换行 5](#_Toc421625004)

[4.2.2 空格 6](#_Toc421625005)

[4.3 注释 6](#_Toc421625006)

[4.3.1 模块注释 6](#_Toc421625007)

[4.3.2 单行注释 7](#_Toc421625008)

[4.3.3 类注释 7](#_Toc421625009)

[4.3.4 方法注释 7](#_Toc421625010)

[4.4 声明 7](#_Toc421625011)

[4.4.1 单行声明变量数 7](#_Toc421625012)

[4.4.2 初始化 8](#_Toc421625013)

[4.4.3 类和接口声明 8](#_Toc421625014)

[4.5 功能语句 9](#_Toc421625015)

[4.5.1 简单逻辑 9](#_Toc421625016)

[4.5.2 if-else语句 9](#_Toc421625017)

[4.5.3 For / Foreach 语句 9](#_Toc421625018)

[4.5.4 While/do-while 语句 9](#_Toc421625019)

[4.5.5 Switch 语句 10](#_Toc421625020)

[4.5.6 Try-catch 语句 10](#_Toc421625021)

[4.6 空白 11](#_Toc421625022)

[4.6.1 空白行 11](#_Toc421625023)

[4.6.2 参数条件之间的空白 11](#_Toc421625024)

[4.6.3 表格式的样式 12](#_Toc421625025)

[4.7 命名规范 12](#_Toc421625026)

[4.7.1 大写 12](#_Toc421625027)

[4.7.1.1 Pascal 风格 12](#_Toc421625028)

[4.7.1.2 驼峰规则 12](#_Toc421625029)

[4.7.1.3 大写风格 12](#_Toc421625030)

[4.7.2 命名方法 12](#_Toc421625031)

[4.7.2.1 类命名 13](#_Toc421625032)

[4.7.2.2 接口命名 13](#_Toc421625033)

[4.7.2.3 枚举命名 13](#_Toc421625034)

[4.7.2.4 常量命名 13](#_Toc421625035)

[4.7.2.5 参数命名 13](#_Toc421625036)

[4.7.2.6 变量命名 13](#_Toc421625037)

[4.7.2.7 方法命名 13](#_Toc421625038)

[4.7.2.8 属性命名 14](#_Toc421625039)

[4.7.2.9 事件命名 14](#_Toc421625040)

[4.7.2.10 大写风格 14](#_Toc421625041)

[4.8 开发习惯 14](#_Toc421625042)

[4.8.1 可见性 14](#_Toc421625043)

[4.8.2 不要硬编码数字 15](#_Toc421625044)

[4.9 代码示例 15](#_Toc421625045)

[4.9.1 作用域（“{}”）示例 15](#_Toc421625046)

# 目标

为.Net开发的C#程序员制定一个统一的编码规范，最大限度减少不同程序员开发的代码间的差异。

# 概述

为了使应用程序的结构和编码风格标准化，便于阅读和理解编码，以提高开发效率和产品的标准化，制订一套开发规范和标准势在必行。此外，好的编码约定可使源代码严谨、可读性强且意义清楚，与其它语言约定相一致，并且尽可能的直观。希望开发人员严格遵守此套开发规范和标准，并落实到自己的程序中。

本规范主要针对C#程序员，但是其中许多规则同时适用于其他语言的程序员。

# 总体要求

## 程序结构化

* 程序结构清晰，函数功能简单易懂（单个函数的代码行数不超过100行）

## 代码可读性

* 保持注释与代码完全一致
* 每个源程序文件，都有文件头说明，详细见下节
* 每个函数，都有函数头说明，详细见下节
* 主要变量（结构、联合、类或对象）定义或引用时，注释能反映其含义
* 处理过程的每个阶段都有相关注释说明
* 在典型算法前都有注释, 同时算法在满足要求的情况下尽可能简单
* 利用缩进来显示程序的逻辑结构，缩进量一致并以Tab键为单位，定义Tab为 4个字节
* 循环、分支层次一般不应超过五层
* 代码简单的分支应该写在前面
* 不允许同行出现两个语句
* 空行和空白字符也是一种特殊注释
* 一目了然的语句不加注释
* 注释的作用范围可以为：定义、引用、条件分支以及一段代码
* 常量定义（DEFINE）有相应说明

## 代码结构化

* 禁止GOTO语句
* 用 CASE 实现多路分支
* 避免不必要的分支
* 用 IF 语句来强调只执行两组语句中的一组。尽量不使用 ELSE RETURN
* 尽量避免从循环引出多个出口

## 正确性与容错性

* 所有变量在调用前必须被初始化
* 不要比较浮点数的相等，如： 10.0 \* 0.1 == 1.0 ， 不可靠
* 访问外部资源(数据库，外部文件)时使用规范的容错语句

例如:

try

{

}

catch

{

}

finally

{

}

# 编码规范

## 文件结构

### C# 文件

尽量不要让你的类或者文件太长,一般不应超过2000行代码。请按照功能划分你的代码，使结构保持清晰。一般情况下，一个文件应当只有一个类，并且文件名应该与类名保持一致。

### 目录结构

应该为每个名称空间(namespace)建立一个目录(例如，我们可以为名称空间MyProject.TestSuite.TestTier建立这样的目录：MyProject/TestSuite/TestTier)。这样做可以让你很快定位到指定名称空间下的类文件。

## 缩进

### 换行

如果表达式太长而一行无法写下时，请按照下列规范进行换行：

* 可以在逗号后面进行换行
* 可以在操作符号后进行换行
* 尽量选择在较高层处进行换行
* 换行后的新行应当与前一行中同级别的运算符对齐

例子：

方法调用换行：

longMethodCall(expr1, expr2,  
               expr3, expr4, expr5);

算术表达式换行：

规范的:  
var = a \* b / (c - g + f) +  
      4 \* z;  
  
不规范的:  
var = a \* b / (c - g +  
      f) + 4 \* z;

上面第一个表达式的换行方式是符合规范的，它换行在括号外面（较高层）。另外请注意，换行后的新行应使用tab和空格保持与前一行的同级运算符对齐，例如:

> var = a \* b / (c - g + f) +

> ......4 \* z;

'>'表示Tab符， '.'表示空格。你可以设置你的编辑环境，使Tab和空格在编辑时是可见的，这是一个不错的习惯。

### 空格

我们选择Tab缩进作为缩进时采用的标准。

**[请不要使用空格代替Tab键!]**

## 注释

### 模块注释

在一个程序模块的开始，应用注释说明模块的名字、功能、开发者和日期和版本变更历史，如下所示：

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

\* Copyright(c) Suzsoft DotNet Group

\* Description : Tenant access class

\* CreateDate : 2006-06-02 05:03:46

\* Creater : Johnson Cao

\* LastChangeDate:

\* LastChanger :

\* Version Info : 1.0

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

### 单行注释

程序员应当在算法比较复杂的表达式前、特殊含义的变量前或者在一整段功能代码开始之前添加适当的注释。我们要求这样的单行 注释采用”//”符号，例如：

// Calculate subTotal

Decimal subTotal = 0;

### 类注释

在定义一个类之前，应用“///”注释说明类的功能、使用方法和特殊的属性，如下所示：

/// <summary>

/// This class...

/// <remarks>

/// Please note …

/// </remarks>

/// </summary>

详细的xml注释标记的使用请参见[附录](#_XML注释标记的使用)。

### 方法注释

在定义类成员方法前，应说明该过程/函数的名字、功能、输入/输出和版本变更历史，如下所示：

/// <summary>

/// This method ….

/// <param name="param1">this is a param of the method SomeMethod.</param>

/// <retvalue>a return object</retvalue>

/// </summary>

//-------------------------------------------------------------------------------------------------

// Change History:

// Date Who Changes Made

// 2000-5-1 Author1 Initial creation

// 2000-5-15 Author2 Add some code

//-------------------------------------------------------------------------------------------------

public object SomeMethod(object param1)

{

}

## 声明

### 单行声明变量数

我们推荐每行只声明一个变量，因为这样你可以在声明后面写上该变量的注释，例如：

int level; // indentation level

int size; // size of table

请不要在同一行声明不同含义的变量，比如：

int a, b; //What is 'a'? What does 'b' stand for?

上面的例子同样可以说明了没有意义的变量名称会让人很难理解，因此，在定义变量的时候，我们一定要给它们一个有含义的名字。

### 初始化

最好在一定义后就初始化变量，例如：

string name = strObject.Name;

or

int val = time.Hours;

注意：如果你想要初始化对话框变量，建议使用using 声明方式

例如：

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

...

}

### 类和接口声明

在声明类和接口的时候应当遵照下列规范：

* 方法名称和放置参数的括号”(”之间不应该有空格
* 类名声明之后应另起一行写作用域开始符”{”
* 作用域结束符”}”应当单独占一行，并与对应的开始符”{”处在同一缩进位置上

示例：

Class MySample : MyClass, IMyInterface

{

int myInt;

public MySample(int myInt)

{

this.myInt = myInt ;

}

void Inc()

{

++myInt;

}

void EmptyMethod()

{

}

}

## 功能语句

### 简单逻辑

每一行代码应当只实现一个逻辑

### if-else语句

if-else应该按照这种格式书写：

if(condition)

{

DoSomething();

...

}

else

{

DoSomethingOther();

...

}

### For / Foreach 语句

For循环语句格式：

for(int i = 0; i < 5; ++i)

{

...

}

或者，如果循环只有一个简单执行语句的话：

for (initialization; condition; update) ;

foreach循环格式:

foreach(int i in IntList)

{

...

}

**注意：即使循环中只有一句执行语句，我们也要求使用”{}”**

### While/do-while 语句

While循环格式：

while (condition)

{

...

}

对于空循环可以这样写：

while (condition) ;

do-while循环格式：

do

{

...

}while(condition);

### Switch 语句

switch格式：

switch (condition)

{

case A:

...

break;

case B:

...

break;

default:

...

break;

}

### Try-catch 语句

try-catch格式：

try

{

...

}

catch (Exception) {}

或者：

try

{

...

}

catch (Exception e)

{

...

}

或者：

try

{

...

}

catch (Exception e)

{

...

}

finally

{

...

}

## 空白

### 空白行

空白行有增强可读性的作用。它们隔离了逻辑上相互独立的模块。

* 在下列情况中我们还可以使用双空白行：
  + 源文件中的两个独立的逻辑片断之间
  + 类和接口之间（但一般情况下我们还是建议一个文件一个类或接口）
* 下列情况中我们可以使用单空白行：
  + 方法之间
  + 属性之间
  + 方法内局部变量第一出声明前
  + 方法内相对独立的逻辑之间

注意，尽量保持空白行和其他行具有同样的缩进，可以使今后插入代码能够保持相同的缩进。

### 参数条件之间的空白

通过这些空白，可以使代码更容易被阅读，例如：

TestMethod(a, b, c);

而不应该这样： TestMethod(a,b,c)

但是也不应有过多的空格，比如：

TestMethod( a, b, c );

操作符和变量或数字之间应该有空格：

a = b; // don't use a=b;

for (int i = 0; i < 10; ++i)

// don't use for (int i=0; i<10; ++i)

// or

// for(int i=0;i<10;++i)

### 表格式的样式

连续的多行付值语句可以这样子写：

string strName = "Mr. Ed";

int nValue = 5;

Test aTest = Test.TestYou;

利用空格对齐所有的“=”，使代码块看起来像表格一样。

## 命名规范

### 大写

### Pascal 风格

变量的首字母大写，如：Name

### 驼峰规则

除了首个单词，每个单词的首字母大写，如：TestName

### 大写风格

只在少于两个字母的缩写中使用大写。三个以上字母的缩写都应该使用PASCAL风格。

### 命名方法

通常我们采用匈牙利命名法来为变量命名。

匈牙利命名法通常采用变量类型的缩写作为前缀，变量含义的全拼作为后缀的方法来命名变量。这种命名方法被广泛的采用在windows程序开发中，它因由一匈牙利程序员创立而得名。

注意：好的变量命名不是突出其类型，而是突出其含义。

**对于UI控件，我们强制要求缩写前缀**，例如：

System.Windows.Forms.Button btnCancel;

System.Windows.Forms.TextBox txtName;

### 类命名

* 使用名词或名词短语命名类。
* 使用Pascal风格.
* 谨慎使用缩写命名类。.
* 不要使用任何类前缀（如C）.

### 接口命名

* 使用名词或名词短语命名接口.(例如 IComponent 或 IEnumberable)
* 使用Pascal风格
* 使用大写的I作为首字母，表示为接口

### 枚举命名

* 使用Pascal风格
* 不使用前缀
* 使用单数名词命名

### 常量命名

* 采用有意义的单词命名
* 使用全部大写字母拼写

### 参数命名

* 相对变量而言，参数更注重本身的含义作为命名
* 使用驼峰规则命名

### 变量命名

* 作为循环中的运算子的变量，更适合采用简单的命名，如i, j, k, l, m, n

### 方法命名

* 采用动词命名，或者动词词组
* 使用Pascal风格

### 属性命名

* 使用名词或名词短语命名属性
* 使用Pascal风格

### 事件命名

* 使用EventHandler作为事件句柄的后缀
* 使用两个参数：sender 和e
* 使用Pascal风格
* 使用EventArgs 作为Event Argument类型名的后缀
* 使用进行时动词和过去式动词作为事件的名称

### 大写风格

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Case | Notes |  |
| Class / Struct | Pascal Casing |  |  |
| Interface | Pascal Casing | Starts with I |  |
| Enum values | Pascal Casing |  |  |
| Enum type | Pascal Casing |  |  |
| Events | Pascal Casing |  |  |
| Exception class | Pascal Casing | End with Exception |  |
| public Fields | Pascal Casing |  |  |
| Methods | Pascal Casing |  |  |
| Namespace | Pascal Casing |  |  |
| Property | Pascal Casing |  |  |
| Protected/private Fields | Camel Casing |  |  |
| Parameters | Camel Casing |  |  |

## 开发习惯

### 可见性

请避免将类变量或一个实例声明为公共类型的变量，而应该将其声明为私有变量。

如果你确实需要公开一个类变量或者实例的话，可以使990用属性(Property)来公开它。

### 不要硬编码数字

也就是说，在你的逻辑代码中，不要采用数字直接参与计算，尤其是那些可能需要改变的数字。你应该将它付值给一个常量，使用该常量参与计算。这样做得好处是，如果这个数字改变的时候，你不必在代码中将他找出并修改，而只需修改常量的值。

public class MyMath

{

public const double PI = 3.14159...

}

## 代码示例

### 作用域（“{}”）示例

namespace ShowMeTheBracket

{

public enum Test

{

TestMe,

TestYou

}

public class TestMeClass

{

Test test;

public Test Test

{

get

{

return test;

}

set

{

test = value;

}

}

void DoSomething()

{

if (test == Test.TestMe)

{

//...stuff gets done

}

else

{

//...other stuff gets done

}

}

}

}

作用域（"{}”）应该在下列代码后另起一行开始：

* Namespace声明后
* Class/ Interface / Struct 的声明后
* Method声明后
* 循环或者判断(if…else…)后