**C#编程规范**

## 1.1 代码格式

* 所有的缩进为4个空格，使用VS.NET的默认设置。
* **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！");}

* 为了防止在阅读代码时不得不滚动源代码编辑器，每行代码或注释在1024\*800的显示频率下不得超过一显示屏
* 当一行被分为几行时，通过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。
* 每一行上放置的语句避免超过一条。
* 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;**

* 将大的复杂代码节分为较小的、易于理解的模块。

## 2.2 命名指南

### 2.2.1 大写样式

使用下面的三种大写标识符约定。

**Pascal 大小写**

将标识符的首字母和后面连接的每个单词的首字母都大写。可以对三字符或更多字符的标识符使用 Pascal 大小写。例如：

*BackColor*

**Camel 大小写**

标识符的首字母小写，而每个后面连接的单词的首字母都大写。例如：

backColor

**大写**

标识符中的所有字母都大写。仅对于由两个或者更少字母组成的标识符使用该约定。例如：

System.**IO**

System.Web.**UI**

可能还必须大写标识符以维持与现有非托管符号方案的兼容性，在该方案中所有大写字母经常用于枚举和常数值。一般情况下，在使用它们的程序集之外这些字符应当是不可见的。

下表汇总了大写规则，并提供了不同类型的标识符的示例。

|  |  |  |
| --- | --- | --- |
| **标识符** | **大小写** | **示例** |
| 类 | Pascal | **AppDomain** |
| 枚举类型 | Pascal | **ErrorLevel** |
| 枚举值 | Pascal | **FatalError** |
| 事件 | Pascal | **ValueChange** |
| 异常类 | Pascal | WebException 注意 总是以 Exception 后缀结尾。 |
| 只读的静态字段 | Pascal | **RedValue** |
| 接口 | Pascal | IDisposable **注意** 总是以 I 前缀开始。 |
| 方法 | Pascal | ToString |
| 命名空间 | Pascal | **System.Drawing** |
| **参数** | **Camel** | **typeName** |
| 属性 | Pascal | **BackColor** |
| 受保护的实例字段 | Camel | **redValue**  **注意** 很少使用。属性优于使用受保护的实例字段。 |
| 公共实例字段 | Pascal | **RedValue**  **注意** 很少使用。属性优于使用公共实例字段。 |

### 2.2.3 缩写

为了避免混淆和保证跨语言交互操作，请遵循有关区缩写的使用的下列规则：

* **不要将缩写或缩略形式用作标识符名称的组成部分。例如，使用 GetWindow，而不要使用 GetWin。**
* 不要使用计算机领域中未被普遍接受的缩写。
* 在适当的时候，使用众所周知的缩写替换冗长的词组名称。例如，用 UI 作为 User Interface 的缩写，用 OLAP 作为 On-line Analytical Processing 的缩写。
* 在使用缩写时，对于超过两个字符长度的缩写请使用 Pascal 大小写或 Camel 大小写。例如，使用 HtmlButton 或 HTMLButton。但是，应当大写仅有两个字符的缩写，如，System.IO，而不是 System.Io。
* 不要在标识符或参数名称中使用缩写。如果必须使用缩写，对于由多于两个字符所组成的缩写请使用 Camel 大小写，虽然这和单词的标准缩写相冲突。

### 2.2.6 命名空间命名指南

**应当对命名空间使用 Pascal 大小写，并用句点分隔逻辑组件，如 Microsoft.Office.PowerPoint 中所示。**如果您的商标使用非传统的大小写，请遵循您的商标所定义的大小写，即使它与规定的 Pascal 大小写相背离。例如，命名空间 NeXT.WebObjects 和 ee.cummings 阐释了对于 Pascal 大小写规则的适当背离。

如果在语义上适当，使用复数命名空间名称。例如，使用 System.Collections而不是 System.Collection。此规则的例外是商标名称和缩写。例如，使用 System.IO而不是 System.IOs。

不要为命名空间和类使用相同的名称。例如，不要既提供 Debug 命名空间也提供 Debug 类。

最后，请注意命名空间名称不必非得与程序集名称相似。例如，如果命名程序集 MyCompany.MyTechnology.dll，它没有必要非得包含 MyCompany.MyTechnology 命名空间。

### 2.2.7 类命名指南

以下规则概述命名类的指南：

* **使用名词或名词短语命名类**。
* 使用 Pascal 大小写。
* 少用缩写。

### 2.2.9 接口命名指南

以下规则概述接口的命名指南：

* 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。名称 IPersistable 使用形容词。
* 使用 Pascal 大小写。
* 少用缩写。
* **给接口名称加上字母 I 前缀，以指示该类型为接口。**
* 在定义类/接口对（其中类是接口的标准实现）时使用相似的名称。两个名称的区别应该只是接口名称上有字母 I 前缀。
* 不要使用下划线字符 (\_)。

### 2.2.10 属性命名指南

应该总是将后缀 Attribute 添加到自定义属性类。以下是正确命名的属性类的示例。

[Visual Basic]

Public Class ObsoleteAttribute

[C#]

public class ObsoleteAttribute{}

### 2.2.11 枚举类型命名指南

枚举 (**Enum**) 值类型从 Enum 类继承。以下规则概述枚举的命名指南：

* 对于 **Enum** 类型和值名称使用 Pascal 大小写。
* 少用缩写。
* **不要在 Enum 类型名称上使用 Enum 后缀。**
* 对大多数 **Enum** 类型使用单数名称，但是对作为位域的 **Enum** 类型使用复数名称。
* 总是将 **FlagsAttribute** 添加到位域 **Enum** 类型。

### 2.2.12 静态字段命名指南

以下规则概述静态字段的命名指南：

* 使用名词、名词短语或者名词的缩写命名静态字段。
* 使用 Pascal 大小写。
* 对静态字段名称使用匈牙利语表示法前缀。
* 建议尽可能使用静态属性而不是公共静态字段。

### 2.2.13 参数命名指南

以下规则概述参数的命名指南：

* 使用描述性参数名称。参数名称应当具有足够的描述性，以便参数的名称及其类型可用于在大多数情况下确定它的含义。
* 对参数名称使用 Camel 大小写。
* 使用描述参数的含义的名称，而不要使用描述参数的类型的名称。开发工具将提供有关参数的类型的有意义的信息。因此，通过描述意义，可以更好地使用参数的名称。少用基于类型的参数名称，仅在适合使用它们的地方使用它们。
* 不要使用保留的参数。保留的参数是专用参数，如果需要，可以在未来的版本中公开它们。相反，如果在类库的未来版本中需要更多的数据，请为方法添加新的重载。
* 不要给参数名称加匈牙利语类型表示法的前缀。

以下是正确命名的参数的示例。

[Visual Basic]

GetType(**typeName** As String)As Type

Format(**format** As String, object [] **args**)As String

[C#]

Type GetType(string **typeName**)

### 2.2.14 方法命名指南

以下规则概述方法的命名指南：

* **使用动词或动词短语命名方法。**
* 使用 Pascal 大小写。

以下是正确命名的方法的实例。

RemoveAll()

### 2.2.15 属性命名指南

以下规则概述属性的命名指南：

* 使用名词或名词短语命名属性。
* 使用 Pascal 大小写。
* 不要使用匈牙利语表示法。
* 考虑用与属性的基础类型相同的名称创建属性。例如，如果声明名为 Color 的属性，则属性的类型同样应该是 Color。请参阅本主题中后面的示例。

以下代码示例阐释正确的属性命名。

[Visual Basic]

Public Class SampleClass

Public Property **BackColor** As Color

' Code for Get and Set accessors goes here.

End Property

End Class

[C#]

public class SampleClass

{

public Color **BackColor**

{

// Code for Get and Set accessors goes here.

}

}

以下代码示例阐释提供其名称与类型相同的属性。

[Visual Basic]

Public Enum Color

' Insert code for Enum here.

End Enum

Public Class Control

Public Property Color As **Color**

Get

' Insert code here.

End Get

Set

' Insert code here.

End Set

End Property

End Class

[C#]

public enum Color

{

// Insert code for Enum here.

}

public class Control

{

public **Color** Color

{

get {// Insert code here.}

set {// Insert code here.}

}

}

以下代码示例不正确，原因是 Color 属性是 Integer 类型的。

[Visual Basic]

Public Enum Color

' Insert code for Enum here.

End Enum

Public Class Control

Public Property Color As **Integer**

Get

' Insert code here.

End Get

Set

' Insert code here.

End Set

End Property

End Class

[C#]

public enum Color {// Insert code for Enum here.}

public class Control

{

public **int** Color

{

get {// Insert code here.}

set {// Insert code here.}

}

}

在不正确的示例中，不可能引用 Color 枚举的成员。Color.Xxx 将被解释为访问一个成员，该成员首先获取 **Color** 属性（在 Visual Basic 中为 **Integer** 类型，在 C# 中为 **int** 类型）的值，然后再访问该值的某个成员（该成员必须是 **System.Int32** 的实例成员）。

### 2.2.16事件命名指南

以下规则概述事件的命名指南：

* **对事件处理程序名称使用 EventHandler 后缀。**
* 指定两个名为 *sender* 和 *e* 的参数。*sender* 参数表示引发事件的对象。*sender* 参数始终是 **object** 类型的，即使在可以使用更为特定的类型时也如此。与事件相关联的状态封装在名为 *e* 的事件类的实例中。对 *e* 参数类型使用适当而特定的事件类。
* 用 EventArgs 后缀命名事件参数类。
* 考虑用动词命名事件。
* 使用动名词（动词的“ing”形式）创建表示事件前的概念的事件名称，用过去式表示事件后。例如，可以取消的 **Close** 事件应当具有 Closing 事件和 Closed 事件。不要使用 BeforeXxx/AfterXxx 命名模式。

不要在类型的事件声明上使用前缀或者后缀。例如，使用 Close，而不要使用

* OnClose。
* 通常情况下，对于可以在派生类中重写的事件，应在类型上提供一个受保护的方法（称为 OnXxx）。此方法只应具有事件参数 *e*，因为发送方总是类型的实例。

以下示例阐释具有适当名称和参数的事件处理程序。

public delegate void **MouseEventHandler**(object **sender**, MouseEventArgs **e**); [Visual Basic]

Public Delegate Sub **MouseEventHandler**(**sender** As Object, **e** As MouseEventArgs)

[C#]

x

以下示例阐释正确命名的事件参数类。

[Visual Basic]

Public Class MouseEventArgs

Inherits EventArgs

Dim x As Integer

Dim y As Integer

Public Sub New MouseEventArgs(x As Integer, y As Integer)

me.x = x

me.y = y

End Sub

Public Property X As Integer

Get

Return x

End Get

End Property

Public Property Y As Integer

Get

Return y

End Get

End Property

End Class

[C#]

public class MouseEventArgs : EventArgs

{

int x;

int y;

public MouseEventArgs(int x, int y)

{ this.x = x; this.y = y; }

public int X { get { return x; } }

public int Y { get { return y; } }

}

## 2.3    注释（Comment）规范

注释规范包括：模块（类）注释规范、类的属性、方法注释规范、代码间注释

### 2.3.1 模块（类）注释规范

模块开始必须以以下形式书写模块注释：

///<summary>

///模块编号：<模块编号，可以引用系统设计中的模块编号>

///作用：<对此类的描述，可以引用系统设计中的描述>

///作者：作者中文名

///编写日期：<模块创建日期，格式：YYYY-MM-DD>

///</summary>

如果模块有修改，则每次修改必须添加以下注释：

///<summary>

///Log编号：<Log编号,从1开始一次增加>

///修改描述：<对此修改的描述>

///作者：修改者中文名

///修改日期：<模块修改日期，格式：YYYY-MM-DD>

///</summary>

### 2.3.2 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>

### 2.3.3 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>

/// 说明：<对该方法的说明>

/// </summary>

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

/// <returns>

///<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>

/// </returns>

### 2.3.4 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>**

**多行注释：**

**/\*多行注释1**

**多行注释2**

**多行注释3\*/**

代码中遇到语句块时必须添加注释（if,for,foreach,……）,添加的注释必须能够说明此语句块的作用和实现手段（所用算法等等）。