C# 编码约定（C# 编程指南）

[C# 语言规范](http://go.microsoft.com/fwlink/?LinkId=199552)没有定义编码标准。 但是，Microsoft 使用本主题中的这些指南开发示例和文档。

编码约定可实现以下目的：

* 它们创建一致的代码外观，从而使读者可以关注内容而非布局。
* 它们使读者能够根据以前的经验作出假设，从而更加快速地理解代码。
* 有利于复制、更改和维护代码。
* 演示 C# 最佳做法。

命名约定

* 命名指南在 [名称准则](https://msdn.microsoft.com/zh-cn/library/ms229002(v=vs.100).aspx) 中介绍。
* 您无需更改由 Visual Studio 设计器工具创建的对象的名称，就可以让它们符合指南。
* 在不包括 [using 语句](https://msdn.microsoft.com/zh-cn/library/sf0df423(v=vs.100).aspx) 的简短示例中，使用命名空间限定。 如果您知道默认情况下会将某命名空间导入项目中，则无需完全限定来自该命名空间的名称。 如果限定名称太长无法放入一行，则可在点 (.) 后截断它，如以下示例所示。

C#

var currentPerformanceCounterCategory = new System.Diagnostics.

PerformanceCounterCategory();

布局约定

好的布局使用格式设置来强调代码的结构，并使该代码更易于阅读。 Microsoft 示例符合以下约定：

* 使用默认代码编辑器设置（智能缩进、四字符缩进、将制表符保存为空格）。 有关更多信息，请参见[“选项”对话框 ->“文本编辑器”->“C#”->“格式设置”](https://msdn.microsoft.com/zh-cn/library/03864tbz(v=vs.100).aspx)。
* 每行仅编写一个语句。
* 每行仅编写一个声明。
* 如果续行不自动缩进，将它们缩进一个制表位（四个空格）。
* 在方法定义和属性定义之间添加至少一个空白行。
* 使用括号突显表达式中的子句，如下面的代码所示。

C#

if ((val1 > val2) && (val1 > val3))

{

// Take appropriate action.

}

注释约定

* 将注释放到另一行，而不要放在代码行的末尾。
* 以大写字母作为注释文本的开头。
* 以句点结束注释文本。
* 在注释分隔符 (//) 和注释文本之间插入一个空格，如以下示例所示。

C#

// The following declaration creates a query. It does not run

// the query.

* 请勿在注释周围创建已设置格式的星号块。

语言指南

以下各部分描述了 C# 团队准备代码示例时遵循的做法。

String 数据类型

* 使用 **+** 运算符来连接短字符串，如以下代码所示。

C#

string displayName = nameList[n].LastName + ", " + nameList[n].FirstName;

* 若要在循环中附加字符串，尤其是在您处理大量文本时，请使用 [StringBuilder](https://msdn.microsoft.com/zh-cn/library/system.text.stringbuilder(v=vs.100).aspx) 对象。

C#

var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";

var manyPhrases = new StringBuilder();

for (var i = 0; i < 10000; i++)

{

manyPhrases.Append(phrase);

}

//Console.WriteLine("tra" + manyPhrases);

隐式类型化局部变量

* 如果赋值语句右侧的变量类型十分明显，或在精确的类型并不重要时，使用 [隐式类型](https://msdn.microsoft.com/zh-cn/library/bb384061(v=vs.100).aspx) 的局部变量。

C#

// When the type of a variable is clear from the context, use var

// in the declaration.

var var1 = "This is clearly a string.";

var var2 = 27;

var var3 = Convert.ToInt32(Console.ReadLine());

* 赋值语句右侧的类型不明显时，不要使用 [var](https://msdn.microsoft.com/zh-cn/library/bb383973(v=vs.100).aspx)。

C#

// When the type of a variable is not clear from the context, use an

// explicit type.

int var4 = ExampleClass.ResultSoFar();

* 不要依赖变量名来指定变量的类型。 这可能是不正确的。

C#

// Naming the following variable inputInt is misleading.

// It is a string.

var inputInt = Console.ReadLine();

Console.WriteLine(inputInt);

* 避免使用 **var** 取代 [dynamic](https://msdn.microsoft.com/zh-cn/library/dd264741(v=vs.100).aspx)。
* 使用隐式类型在 [for](https://msdn.microsoft.com/zh-cn/library/ch45axte(v=vs.100).aspx) 和 [foreach](https://msdn.microsoft.com/zh-cn/library/ttw7t8t6(v=vs.100).aspx) 循环中确定循环变量的类型。

下面的示例在 **for** 语句中使用了隐式类型。

C#

var syllable = "ha";

var laugh = "";

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

{

laugh += syllable;

Console.WriteLine(laugh);

}

下面的示例在 **foreach** 语句中使用了隐式类型。

C#

foreach (var ch in laugh)

{

if (ch == 'h')

Console.Write("H");

else

Console.Write(ch);

}

Console.WriteLine();

无符号数据类型

* 一般情况下，使用 **int** 而不是无符号类型。 在整个 C# 中，使用 **int** 很常见，并且在您使用 **int** 时很容易与其他库进行交互。

数组

* 在您初始化声明行上的数组时，请使用简洁的语法。

C#

// Preferred syntax. Note that you cannot use var here instead of string[].

string[] vowels1 = { "a", "e", "i", "o", "u" };

// If you use explicit instantiation, you can use var.

var vowels2 = new string[] { "a", "e", "i", "o", "u" };

// If you specify an array size, you must initialize the elements one at a time.

var vowels3 = new string[5];

vowels3[0] = "a";

vowels3[1] = "e";

// And so on.

委托

* 请使用简洁的语法来创建委托类型的实例。

C#

// First, in class Program, define the delegate type and a method that

// has a matching signature.

// Define the type.

public delegate void Del(string message);

// Define a method that has a matching signature.

public static void DelMethod(string str)

{

Console.WriteLine("DelMethod argument: {0}", str);

}

C#

// In the Main method, create an instance of Del.

// Preferred: Create an instance of Del by using condensed syntax.

Del exampleDel2 = DelMethod;

// The following declaration uses the full syntax.

Del exampleDel1 = new Del(DelMethod);

异常处理中的 try-catch 和 using 语句

* 为大多数异常处理使用 [try-catch](https://msdn.microsoft.com/zh-cn/library/0yd65esw(v=vs.100).aspx) 语句。

C#

static string GetValueFromArray(string[] array, int index)

{

try

{

return array[index];

}

catch (System.IndexOutOfRangeException ex)

{

Console.WriteLine("Index is out of range: {0}", index);

throw;

}

}

* 使用 C# [using 语句](https://msdn.microsoft.com/zh-cn/library/yh598w02(v=vs.100).aspx)简化代码。 如果有一个 [try-finally](https://msdn.microsoft.com/zh-cn/library/zwc8s4fz(v=vs.100).aspx) 语句，其中 **finally** 块中的唯一代码为对 [Dispose](https://msdn.microsoft.com/zh-cn/library/system.idisposable.dispose(v=vs.100).aspx) 方法的调用，请改用 **using** 语句。

C#

// This try-finally statement only calls Dispose in the finally block.

Font font1 = new Font("Arial", 10.0f);

try

{

byte charset = font1.GdiCharSet;

}

finally

{

if (font1 != null)

{

((IDisposable)font1).Dispose();

}

}

// You can do the same thing with a using statement.

using (Font font2 = new Font("Arial", 10.0f))

{

byte charset = font2.GdiCharSet;

}

&& 和 || 运算符

* 若要跳过不必要的比较来避免出现异常和提高性能，在执行比较时，请使用 [&&](https://msdn.microsoft.com/zh-cn/library/2a723cdk(v=vs.100).aspx) 代替 [&](https://msdn.microsoft.com/zh-cn/library/sbf85k1c(v=vs.100).aspx)，使用 [||](https://msdn.microsoft.com/zh-cn/library/6373h346(v=vs.100).aspx) 代替[|](https://msdn.microsoft.com/zh-cn/library/kxszd0kx(v=vs.100).aspx) ，如以下示例所示。

C#

Console.Write("Enter a dividend: ");

var dividend = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter a divisor: ");

var divisor = Convert.ToInt32(Console.ReadLine());

// If the divisor is 0, the second clause in the following condition

// causes a run-time error. The && operator short circuits when the

// first expression is false. That is, it does not evaluate the

// second expression. The & operator evaluates both, and causes

// a run-time error when divisor is 0.

if ((divisor != 0) && (dividend / divisor > 0))

{

Console.WriteLine("Quotient: {0}", dividend / divisor);

}

else

{

Console.WriteLine("Attempted division by 0 ends up here.");

}

New 运算符

* 通过隐式类型使用简洁形式的对象实例，如以下声明所示。

C#

var instance1 = new ExampleClass();

上面的行与以下声明等效。

C#

ExampleClass instance2 = new ExampleClass();

* 使用对象初始值设定项来简化对象创建。

C#

// Object initializer.

var instance3 = new ExampleClass { Name = "Desktop", ID = 37414,

Location = "Redmond", Age = 2.3 };

// Default constructor and assignment statements.

var instance4 = new ExampleClass();

instance4.Name = "Desktop";

instance4.ID = 37414;

instance4.Location = "Redmond";

instance4.Age = 2.3;

事件处理

* 如果您正在定义无需日后移除的事件处理程序，请使用 lambda 表达式。

C#

public Form2()

{

// You can use a lambda expression to define an event handler.

this.Click += (s, e) =>

{

MessageBox.Show(

((MouseEventArgs)e).Location.ToString());

};

}

C#

// Using a lambda expression shortens the following traditional definition.

public Form1()

{

this.Click += new EventHandler(Form1\_Click);

}

void Form1\_Click(object sender, EventArgs e)

{

MessageBox.Show(((MouseEventArgs)e).Location.ToString());

}

静态成员

* 使用类名称调用 [static](https://msdn.microsoft.com/zh-cn/library/98f28cdx(v=vs.100).aspx) 成员：*ClassName.StaticMember*。 不要从派生类访问在基类中定义的静态成员。

LINQ 查询

* 对于查询变量使用有意义的名称。 下面的示例使用 seattleCustomers 代表地处西雅图的客户。

C#

var seattleCustomers = from cust in customers

where cust.City == "Seattle"

select cust.Name;

* 使用别名以确保通过 Pascal 大小写格式正确设置匿名类型的属性名称的大小写。

C#

var localDistributors =

from customer in customers

join distributor in distributors on customer.City equals distributor.City

select new { Customer = customer, Distributor = distributor };

* 在结果中的属性名称不明确时重命名属性。 例如，如果查询返回一个客户名称和一个分销商 ID，不要在结果中将它们保留为 Name 和 ID，而是将它们重命名，以阐明 Name 是客户的名称，ID 是分销商的 ID。

C#

var localDistributors2 =

from cust in customers

join dist in distributors on cust.City equals dist.City

select new { CustomerName = cust.Name, DistributorID = dist.ID };

* 使用隐式类型来声明查询变量和范围变量。

C#

var seattleCustomers = from cust in customers

where cust.City == "Seattle"

select cust.Name;

* 将查询子句对齐在 [from](https://msdn.microsoft.com/zh-cn/library/bb383978(v=vs.100).aspx)子句下方，如前面的示例所示。
* 在其他查询子句前面使用 [where](https://msdn.microsoft.com/zh-cn/library/bb311043(v=vs.100).aspx) 子句，以确保对一组经过简化和筛选的数据执行后面的查询子句。

C#

var seattleCustomers2 = from cust in customers

where cust.City == "Seattle"

orderby cust.Name

select cust;

* 使用多个 **from** 子句而不是一个 [join](https://msdn.microsoft.com/zh-cn/library/bb311040(v=vs.100).aspx) 子句来访问内部集合。 例如，在 Student 对象集合中，每个对象可能都包含一个考试成绩集合。 在执行下面的查询时，它将返回每一个超过 90 的分数，以及获得该分数的学生的姓氏。

C#

// Use a compound from to access the inner sequence within each element.

var scoreQuery = from student in students

from score in student.Scores

where score > 90

select new { Last = student.LastName, score };