

# 零度 C#编码规范

2017年09月01日 修订版

本文档由零度编制，精准、全面、规范、细致入微。

更多编码规范参阅零度官网：[www.xcode.me](http://www.xcode.me)

# 零度 C#编码规范

第一章 概述 .....	4
1.1 术语定义 .....	4
1.1.1 Pascal 大小写 .....	4
1.1.2 Camel 大小写 .....	5
第二章 代码外观 .....	5
2.1 列宽 .....	5
2.2 换行 .....	5
2.3 缩进 .....	5
2.4 空行 .....	5
2.5 空格 .....	6
2.6 括号 - () .....	6
2.7 花括号 - {} .....	7
第三章 程序注释 .....	7
3.1 注释概述 .....	7
3.2 文件注释 .....	8
3.3 文档型注释 .....	9
3.3 单行注释 .....	11
3.4 JavaScript 注释 .....	12
3.5 注释标签 .....	14

第四章 申明 .....	16
4.1 每行声明数 .....	16
4.2 初始化.....	17
4.3 位置.....	17
4.4 类和接口的声明.....	18
4.5 字段的声明 .....	18
第五章 命名规范.....	18
5.1 命名概述 .....	18
5.2 大小写规则 .....	19
5.3 缩写.....	20
5.4 命名空间 .....	20
5.5 文件命名 .....	21
5.6 类.....	21
5.7 接口.....	21
5.8 枚举 (Enum) .....	22
5.9 参数.....	22
5.10 方法.....	23
5.11 属性 (property).....	23
5.12 事件.....	25
5.13 常量 (const) .....	26
5.14 字段.....	26

5.15 静态字段.....	27
5.16 集合.....	27
5.17 措词.....	27
第六章 语句 .....	29
6.1 每行一个语句.....	29
6.2 复合语句 .....	29
6.3 return 语句 .....	30
6.4 if、 if-else、 if else-if 语句.....	30
6.4 for、 foreach 语句.....	31
6.5 while 语句 .....	31
6.7. do - while 语句.....	32
6.8. switch - case 语句 .....	32
6.9. try - catch 语句 .....	32
6.10. using 块语句.....	33
6.11. goto 语句 .....	33
第七章 控件命名规则.....	33
7.1 命名方法 .....	33
7.2 主要控件名简写对照表 .....	34
第八章 代码格式检查.....	34
8.1 检查分类 .....	35
8.2 安装及使用 .....	35

8.3 文档规则(Documentation Rules).....	36
8.4 布局规则(LayoutRules) .....	43
8.5 可维护性规则(Maintanability Rules)“ .....	44
第九章 代码分析.....	45
9.1.安全性规则 .....	46
9.2.可靠性规则 .....	46
9.3.可维护性规则 .....	46
9.4.命名规则 .....	46
9.5.性能规则 .....	47
9.6.用法规则 .....	47

# 第一章 概述

## 1.1 术语定义

### 1.1.1 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`

## 1.1.2 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`

# 第二章 代码外观

## 2.1 列宽

代码列宽控制在 110 字符左右。

## 2.2 换行

当表达式超出或即将超出规定的列宽，遵循以下规则进行换行

- 1、在逗号后换行;
- 2、在操作符前换行;
- 3、规则 1 优先于规则 2。

## 2.3 缩进

缩进应该是每行一个 Tab(4 个空格)，不要在代码中使用 Tab 字符。

## 2.4 空行

空行是为了将逻辑上相关联的代码分块，以便提高代码的可阅读性。  
在代码中，不能包含多个空行。

在以下情况下使用一个空行

- 1、方法与方法、属性与属性之间。
- 2、方法中变量声明与语句之间。
- 3、方法与方法之间。
- 4、方法中不同的逻辑块之间。
- 5、方法中的返回语句与其他的语句之间。
- 6、属性与方法、属性与字段、方法与字段之间。
- 7、注释与它注释的语句间不空行，但与其他的语句间空一行。

## 2.5 空格

在以下情况中要使用到空格

- 1、关键字和左括号“(”应该用空格隔开。如 `while (true)`

**注意:**在方法名和左括号“(”之间不要使用空格，这样有助于辨认代码中的方法调用与关键字。

- 2、多个参数用逗号隔开，每个逗号后都应加一个空格。
- 3、除了 `.` 之外，所有的二元操作符都应用空格与它们的操作数隔开。一元操作符、`++`及`--`与操作数间不需要空格。如

```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++)
{
    n++;
}
PrintSize( "size is " + size + "\n" );
```

- 4、语句中的表达式之间用空格隔开。如

```
for (expr1; expr2; expr3)
```

## 2.6 括号 - ()

- 1、左括号“(”不要紧靠关键字，中间用一个空格隔开。
- 2、左括号“(”与方法名之间不要添加任何空格。
- 3、没有必要的话不要在返回语句中使用`()`。如

```
if (condition)
```

```
Array.Remove(1)
```

```
return 1
```

## 2.7 花括号 - {}

- 1、左花括号 “{” 放于关键字或方法名的下一行并与之对齐。如

```
if (condition)
{
}

public int Add(int x, int y)
{
}
```

- 2、左花括号 “{” 要与相应的右花括号 “}” 对齐。
- 3、通常情况下左花括号 “{” 单独成行，不与任何语句并列一行。
- 4、if、while、do 语句后一定要使用{}，即使{}号为空或只有一条语句。如

```
if (somevalue == 1)
{
    somevalue = 2;
}
```

- 5、右花括号 “}” 后建议加一个注释以便于方便的找到与之相应的 {。如

```
while (1)
{
    if (valid)
    {
    }
    else
    {
    } // if
} // while
```

# 第三章 程序注释

## 3.1 注释概述

- 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，如幽默的不主要的备注。

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
```

## 3.2 文件注释

在每个文件头必须包含以下注释说明

```
// <copyright file="文件名.cs" company="HP">
```

```
// Copyright (c) HP. All rights reserved.
```

```
// </copyright>
```

```
// <author>×××</author>
```

```
// <date> yyyy-mm-dd </date>
```

```
// <summary>文件功能描述</summary>
```

```
// <modify>

//     修改人 : xxx

//     修改时间 : yyyy-mm-dd

//     修改描述 : xxx

//     版本 : 1.0

//</modify>
```

注意：文件功能描述只需简述，具体详情在类的注释中描述。创建标识和修改标识由创建或修改人员的拼音或英文名。如：Zhangsan。一天内有多个修改的只需做一个在注释说明中做一个修改标识就够了。在所有的代码修改处加上修改标识的注释。

### 3.3 文档型注释

该类注释采用.Net 已定义好的 Xml 标签来标记，在声明接口、类、方法、属性、字段都应该使用该类注释，以便代码完成后直接生成代码文档，让别人更好的了解代码的实现和接口。如

#### 1、类、接口注释

```
/// <summary>

/// 类功能的说明

/// </summary>

/// <see cref=""></see>

/// <remarks>

/// 创建人 : Zhangsan

/// 创建日期 : yyyy-mm-dd

/// 修改人 : Lisi

/// 修改日期 : yyyy-mm-dd

/// 修改备注 : 无
```

```
/// 版本 : 1.0
```

```
/// </remarks>
```

```
public class CountersModuleInitializer : ModuleInitializer
```

```
{
```

```
}
```

注意 : <see cref=""></see> 标签根据具体情况选择有无

## 2、方法、事件注释

```
/// <summary>
```

```
/// 根据员工编号获得员工信息
```

```
/// </summary>
```

```
/// <param name="employeeId">员工编号</param>
```

```
/// <exception cref="System.Exception">系统异常</exception>
```

```
/// <returns>员工姓名</returns>
```

```
/// <remarks>
```

```
/// 创建人 : Zhangsan
```

```
/// 创建日期 : yyyy-mm-dd
```

```
/// 修改人 : Lisi
```

```
/// 修改日期 : yyyy-mm-dd
```

```
/// 修改备注 : 无
```

```
/// 版本 : 1.1
```

```
/// </remarks>
```

```
public string GetEmployeeNameById(int employeeId)
{
    try
    {
        return "ddd";
    }
    catch (System.Exception)
    {
        throw;
    }
}
```

注意：该方法注释中的<param></param>、<exception cref=" "></exception>、<returns></returns>等标签根据具体情况选择有无，方法初始版本为 1.0，每次改动增加 0.1。

### 3、属性、常量注释

```
/// <summary>
/// session id
/// </summary>
public const string SESSION_ID = "";
```

## 3.3 单行注释

该类注释用于

1 方法内的代码注释。如变量的声明、代码或代码段的解释。注释示例：

```
// 注释语句  
private int number;
```

2 方法内变量的声明或花括号后的注释, 注释示例：

```
if ( 1 == 1)    // always true  
{  
    statement;  
}  
else          // always false  
{  
}
```

## 3.4 JavaScript 注释

a) 注释符号

```
'''
```

不允许使用 '/\*' 作注释符。

b) 函数注释

每个函数都应该描述该函数的名称、功能、作用范围、入口参数的类型和传值方式

及参数含义、返回值类型及返回值的含义。格式如下：

```
//  
  
//Function: 函数名  
  
//Purpose: 用途  
  
//Scope: 作用范围  
  
//Args: 入口参数(列表) 类型传值方式含义  
  
//Returns: 返回值类型 (可确定值列表) 含义  
  
//
```

c) 非函数注释

注明该模块的作用

格式如下：

```
//
```

```
//功能：
```

```
//
```

d) 程序行间注释

在程序行的每一个处理单元前作注释

格式如下：

```
//注释
```

e) 注释举例

```
//
```

```
//Function: F_FindObject
```

```
//Purpose: 按照空间名在可视化主对象中查找住对象内的可视化控件
```

```
//Scope: Public
```

```
//Args: is_name String value : 要查找的空间名
```

```
// ipbo_object Object value: 可视化主对象
```

```
//Returns: Boolean True 表示找到该控件
```

```
// False 表示没有找到该控件
```

```
//
```

```
function F_FindObject(is_name, ipbo_object) {
```

```

//获得显示学生信息的GreeView控件

var gv_student = document.getElementById("GVStudent");

*****

//返回true

return true;

}

```

### 3.5 注释标签

标签	用法	作用
<c>	<code>&lt;c&gt; text&lt;/c&gt;</code> <i>text</i> 希望将其指示为代码的文本。	为您提供了一种将说明中的文本标记为代码的方法。使用 <code> 将多行指示为代码
<para>	<code>&lt;para&gt; content&lt;/para&gt;</code> content 段落文本。	用于诸如 <remarks> 或 <returns> 等标记内,使您得以将结构添加到文本中。
<param>	<code>&lt;param name=' name'&gt; description&lt;/param&gt;</code> name 为方法参数名。将此名称用单引号括起来 ('')。	应当用于方法声明的注释中,以描述方法的一个参数。
<paramref>	<code>&lt;paramref name=" name"/&gt;</code> <i>name</i> 要引用的参数名。将此名称用双引号括起来 ("")。	<paramref> 标记为您提供了一种指示词为参数的方法。可以处理 XML 文件,从而用某种独特的方法格式化该参数。
<see>	<code>&lt;see cref=" member"/&gt;</code>  <i>cref = " member"</i> 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定	使您得以从文本内指定链接。使用 <seealso> 指示希望在“请参阅”一节中出现的文本。

	代码元素存在后，将 <i>member</i> 传递给输出 XML 中的元素名。必须将 <i>member</i> 括在双引号 ( " ") 中。	
<seealso>	<seealso cref="member"/> cref = "member" 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 <i>member</i> 传递给输出 XML 中的元素名。必须将 <i>member</i> 括在双引号 ( " ") 中	使您得以指定希望在“请参阅”一节中出现的文本。使用 <see> 从文本
<example>	<example> <i>description</i> </example> <i>description</i> 代码示例的说明。	使用 <example> 标记可以指定使用方法或其他库成员的示例。一般情况下，这将涉及到 <code> 标记的使用。
<code>	<code> content </code> content 为希望将其标记为代码的文本。	记为您提供了一种将多行指示为代码的方法。使用 <c> 指示应将说明中的文本标记为代码
<summary>	<summary> <i>description</i> </summary> 此处 <i>description</i> 为对象的摘要。	应当用于描述类型成员。使用 <remarks> 以提供有关类型本身的信息。
<exception>	<exception cref="member"> <i>description</i> </exception> cref = "member" 对可从当前编译环境中获取的异常的引用。编译器检查到给定异常存在后，将 <i>member</i> 转换为输出 XML 中的规范化元素名。必须将 <i>member</i> 括在双引号 ( " ") 中。 <i>description</i> 说明。	<exception> 标记使您可以指定类能够引发的异常。
<include>	<include file='filename' path='tagpath[@name="id"]' /> filename 包含文档的文件名。该文件名可用路径加以限定。将 <i>filename</i> 括在单引号中 ( ' ')。 <i>Tagpath</i> : <i>filename</i> 中指向标记名的标记路径。将此路径括在单引号中 ( ' ')。 <i>name</i> 注释前边的标记中的名称说明符；名称具有一个 <i>id</i> 。 <i>id</i> 位于注释之前的标记的 <i>id</i> 。将此 <i>id</i> 括在双引号中 ( " ")。	<include> 标记使您得以引用描述源代码中类型和成员的另一文件中的注释。这是除了将文档注释直接置于源代码文件中之外的另一种可选方法。 <include> 标记使用 XML XPath 语法。有关自定义 <include> 使用的方法，请参阅 XPath 文档。
<list>	<list type="bullet"   "number"   "table"> <listheader> <term> <i>term</i> </term> <description> <i>description</i> </description>	<listheader> 块用于定义表或定义列表中的标题行。定义表时，只需为标题中的项提供一个项。 列表中的每一项用 <item> 块指定。 创建定义列表时，既需要指定 <i>term</i>



	<pre> &lt;/listheader&gt; &lt;item&gt;   &lt;term&gt; term&lt;/term&gt;  &lt;description&gt; description&lt;/description&gt; &lt;/item&gt; &lt;/list&gt; </pre> <p><i>term</i> 定义的项，该项将在 <i>text</i> 中定义。  <i>description</i> 目符号列表或编号列表中的项或者 <i>term</i> 的定义。</p>	<p>也需要指定 <i>text</i>。但是，对于表、项目符号列表或编号列表，只需为 <i>text</i> 提供一个项。  列表或表所拥有的 <code>&lt;item&gt;</code> 块数可以根据需要而定。</p>
<code>&lt;permission&gt;</code>	<pre> &lt;permission   cref="member"&gt; description&lt;/permission&gt; </pre> <p><code>cref = "member"</code> 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 <i>member</i> 转换为输出 XML 中的规范化元素名。必须将 <i>member</i> 括在双引号 (" ") 中。  <i>description</i> 成员的访问的说明。</p>	<p><code>&lt;permission&gt;</code> 标记使您得以将成员的访问记入文档。  <b>System.Security.PermissionSet</b> 使您得以指定对成员的访问。</p>
<code>&lt;remarks&gt;</code>	<pre> &lt;remarks&gt; description&lt;/remarks&gt; </pre> <p><i>description</i> 成员的说明。</p>	<p><code>&lt;remarks&gt;</code> 标记是可以指定有关类或其他类型的概述信息的位置。  <code>&lt;summary&gt;</code> 是可以描述该类型的成员的位置。</p>
<code>&lt;returns&gt;</code>	<pre> &lt;returns&gt; description&lt;/returns&gt; </pre> <p><i>description</i> 返回值的说明。</p>	<p><code>&lt;returns&gt;</code> 标记应当用于方法声明的注释，以描述返回值。</p>
<code>&lt;value&gt;</code>	<pre> &lt;value&gt; property-description&lt;/value&gt; </pre> <p><i>property-description</i> 属性的说明。</p>	<p><code>&lt;value&gt;</code> 标记使您得以描述属性。请注意，当在 Visual Studio .NET 开发环境中通过代码向导添加属性时，它将会为新属性添加 <code>&lt;summary&gt;</code> 标记。然后，应该手动添加 <code>&lt;value&gt;</code> 标记以描述该属性所表示的值。</p>

## 第四章 申明

### 4.1 每行声明数

一行只作一个声明,如

```
int level; //推荐
int size; //推荐
int x, y; //不推荐
```

## 4.2 初始化

建议在变量声明时就对其做初始化。

## 4.3 位置

变量建议置于块的开始处，不要总是在第一次使用它们的地方做声明。如

```
void MyMethod()
{
    int int1 = 0; // beginning of method block

    if (condition)
    {
        int int2 = 0; // beginning of "if" block
        ...
    }
}
```

不过也有一个例外

```
for (int i = 0; i < maxLoops; i++)
{
    ...
}
```

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
        ...
    }
}
```

```
    }  
    ...  
}
```

## 4.4 类和接口的声明

- 1 在方法名与其后的左括号间没有任何空格。
- 2 左花括号 “{” 出现在声明的下行并与之对齐，单独成行。
- 3 方法间用一个空行隔开。

## 4.5 字段的声明

不要使用是 `public` 或 `protected` 的实例字段。如果避免将字段直接公开给开发人员，可以更轻松地地对类进行版本控制，原因是在维护二进制兼容性时字段不能被更改为属性。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 Control: Component  
{  
    private int handle;  
    public int Handle  
    {  
        get  
        {  
            return handle;  
        }  
    }  
}
```

# 第五章 命名规范

## 5.1 命名概述

名称应该说明“什么”而不是“如何”。通过避免使用公开基础实现（它们会发生改变）的名称，可以保留简化复杂性的抽象层。例如，可以使用 `GetNextStudent()`，而不是 `GetNextArrayElement()`。  
命名原则是：

选择正确名称时的困难可能表明需要进一步分析或定义项的目的。使名称足够长以便有一定的意义，并且足够短以避免冗长。唯一名称在编程上仅用于将各项区分开。表现力强的名称是为了帮助人们阅读；因此，提供人们可以理解的名称是有意义的。不过，请确保选择的名称符合适用语言的规则 and 标准。

以下是推荐的命名方法。

- 1、避免容易被主观解释的难懂的名称，如方面名 `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 `Book.BookTitle`。而是应该使用 `Book.Title`。
- 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、在变量名中使用互补对，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 `fileIsFound`。
- 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`。可能的情况下，尽量不要使用原义数字或原义字符串，如 `For i = 1 To 7`。而是使用命名常数，如 `For i = 1 To NUM_DAYS_IN_WEEK` 以便于维护和理解。
- 8、用于事件处理的委托添加“**EventHandler**”后缀
- 9、用于事件处理之外的那些委托添加“**Callback**”后缀
- 10、不要给委托添加“**Delegate**”后缀
- 11、用名词或名词词组来给类型命名，在少数情况下也可以用形容词词组来给类型命名
- 12、用动词或动词词组来命名方法
- 13、用名词、名词词组或形容词来命名属性
- 14、要用动词或动词短语来命名事件
- 15、要用名词或名词短语来命名字段

## 5.2 大小写规则

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`

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	样例
名字空间	Pascal	namespace <b>System.Security</b> { ... }
类型	Pascal	public class <b>StreamReader</b> { ... }

接口	Pascal	public interface <b>IEnumerable</b> { ... }
方法	Pascal	public class Object { public virtual string <b>ToString</b> (); }
属性	Pascal	public class String { public int <b>Length</b> { get; } }
事件	Pascal	public class Process{ public event EventHandler <b>Exited</b> ; }
字段(私有实例)	Camel	private string <b>userName</b> ;
字段(公共静态)	Pascal	public static readonly string <b>UserId</b> ;
枚举	Pascal	enum FileMode { <b>Append</b> , ... }
参数	Camel	public class Convert { public static int ToInt32(string <b>userId</b> ); }

## 5.3 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：

- 1 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`。
- 2 不要使用计算机领域中未被普遍接受的缩写。
- 3 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` 缩写，用 `OLAP` 作为 `On-line Analytical Processing` 的缩写。
- 4 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`。
- 5 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。

## 5.4 命名空间

- 1、给命名空间命名时的一般性规则是使用公司名称，后跟技术名称和可选的功能与设计，如下所示。

`CompanyName.TechnologyName[.Feature][.Design]`

例如：

```
namespace JadeBird.StudentManager           //学员系统
namespace JadeBird.StudentManager.Register   //北学员注册模块
```

- 2、命名空间使用 Pascal 大小写，用点号分隔。
- 3、TechnologyName 指的是该项目的英文缩写，或软件名。
- 4、命名空间和类不能使用同样的名字。例如，有一个类被命名为 Debug 后，就不要再使用 Debug 作为一个名称空间名。

## 5.5 文件命名

- 1 文件名遵从 Pascal 命名法，无特殊情况，扩展名小写。
- 2 使用统一而又通用的文件扩展名：

文件类型	扩展名
C#类	.cs
Asp.net 页面	.aspx
...	使用默认扩展名

## 5.6 类

- 1、使用 Pascal 大小写。
- 2、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、有时候需要提供以字母 I 开始的类名称，虽然该类不是接口。只要 I 是作为类名称组成部分的整个单词的第一个字母，这便是适当的。例如，类名称 IdentityStore 是适当的。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，ApplicationException 对于从名为 Exception 的类派生的类是适当的名称，原因 ApplicationException 是一种 Exception。请在应用该规则时进行合理的判断。例如，Button 对于从 Control 派生的类是适当的名称。尽管按钮是一种控件，但是将 Control 作为类名称的一部分将使名称不必要地加长。

```
public class FileStream
public class Button
public class String
```

## 5.7 接口

以下规则概述接口的命名指南：

- 1、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。

- 2、使用 Pascal 大小写。
- 3、少用缩写。
- 4、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。
- 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。

以下是正确命名的接口的示例。

```
public interface IServiceProvider
public interface IFormatable
```

以下代码示例阐释如何定义 IComponent 接口及其标准实现 Component 类。

```
public interface IComponent
{
    // Implementation code goes here.
}

public class Component: IComponent
{
    // Implementation code goes here.
}
```

## 5.8 枚举 (Enum)

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

- 1 对于 Enum 类型和值名称使用 Pascal 大小写。
- 2 少用缩写。
- 3 不要在 Enum 类型名称上使用 Enum 后缀。
- 4 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。
- 5 总是将 FlagsAttribute 添加到位域 Enum 类型。
- 6 不要命名“Reserved”枚举值。

## 5.9 参数

以下规则概述参数的命名指南：

- 1、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。

- 2、对参数名称使用 Camel 大小写。
- 3、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。
- 4、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。
- 5、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。

```
Type GetType(string typeName)
string Format(string format, args() As object)
```

## 5.10 方法

以下规则概述方法的命名指南：

- 1 使用动词或动词短语命名方法。
- 2 使用 Pascal 大小写。
- 3 以下是正确命名的方法的实例。

```
RemoveAll()
GetCharArray()
Invoke()
```

## 5.11 属性 (property)

以下规则概述属性的命名指南：

- 1 使用名词或名词短语命名属性。
- 2 使用 Pascal 大小写。
- 3 不要使用匈牙利语表示法。
- 4 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。

```
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。

```
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 Int 类型的。

```
public enum Color  
{  
    // Insert code for Enum here.  
}  
  
public class Control  
{  
    public int Color  
    {  
        // 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（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）。

## 5.12 事件

以下规则概述事件的命名指南：

- 1、对事件处理程序名称使用 EventHandler 后缀。
- 2、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。
- 3、用 EventArgs 后缀命名事件参数类。
- 4、考虑用动词命名事件。
- 5、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。
- 6、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。
- 7、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);
```

以下示例阐释正确命名的事件参数类。

```
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;
        }
    }
}
```

## 5.13 常量 (const)

以下规则概述常量的命名指南：

所有单词大写，多个单词之间用 "\_" 隔开。 如

```
public const string PAGE_TITLE = "Welcome";
```

## 5.14 字段

以下规则概述字段的命名指南：

- 1、private、protected 使用 Camel 大小写。
- 2、public 使用 Pascal 大小写。
- 3、拼写出字段名称中使用的所有单词。仅在开发人员一般都能理解时使用缩写。字段名称不要使用大写字母。下面是正确命名的字段的示例。

```
class SampleClass
{
    string url;
    string destinationUrl;
}
```

- 4、不要对字段名使用匈牙利语表示法。好的名称描述语义，而非类型。
- 5、不要对字段名或静态字段名应用前缀。具体说来，不要对字段名称应用前缀来区分静态和非静态字段。例如，应用 g\_ 或 s\_ 前缀是不正确的。
- 6、对预定义对象实例使用公共静态只读字段。如果存在对象的预定义实例，则将它们声明为

对象本身的公共静态只读字段。使用 Pascal 大小写，原因是字段是公共的。下面的代码示例阐释公共静态只读字段的正确使用。

```
public struct Color
{
    public static readonly Color Red = new Color(0x0000FF);

    public Color(int rgb)
    {
        // Insert code here.
    }

    public Color(byte r, byte g, byte b)
    {
        // Insert code here.
    }

    public byte RedValue
    {
        get
        {
            return Color;
        }
    }
}
```

## 5.15 静态字段

以下规则概述静态字段的命名指南：

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

## 5.16 集合

集合是一组组合在一起的类似的类型化对象，如哈希表、查询、堆栈、字典和列表，集合的命名建议用复数。

## 5.17 措词

避免使用与常用的 .NET 框架命名空间重复的类名称。例如，不要将以下任何名称用作类名称：System、Collections、Forms 或 UI。有关 .NET 框架命名空间的列表，请参阅类库。

另外，避免使用和以下关键字冲突的标识符。

AddHandler	AddressOf	Alias	And	Ansi
As	Assembly	Auto	Base	Boolean
ByRef	Byte	ByVal	Call	Case
Catch	CBool	CByte	Cchar	CDate
CDec	Cdbl	Char	Cint	Class
CLng	CObj	Const	Cshort	CSng
CStr	CType	Date	Decimal	Declare
Default	Delegate	Dim	Do	Double
Each	Else	Elseif	End	Enum
Erase	Error	Event	Exit	ExternalSource
False	Finalize	Finally	Float	For
Friend	Function	Get	GetType	Goto
Handles	If	Implements	Imports	In
Inherits	Integer	Interface	Is	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On
Option	Optional	Or	Overloads	Overridable

Overrides	ParamArray	Preserve	Private	Property
Protected	Public	RaiseEvent	ReadOnly	ReDim
Region	REM	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared	Short
Single	Static	Step	Stop	String
Structure	Sub	SyncLock	Then	Throw
To	True	Try	TypeOf	Unicode
Until	volatile	When	While	With
WithEvents	WriteOnly	Xor	Eval	extends
instanceof	package	var		

## 第六章 语句

### 6.1 每行一个语句

每行最多包含一个语句。如

```
a++;      //推荐
b--;      //推荐
a++; b--; //不推荐
```

### 6.2 复合语句

复合语句是指包含"父语句{子语句;子语句;}"的语句，使用复合语句应遵循以下几点

- 1 子语句要缩进。
- 2 左花括号 "{" 在复合语句父语句的下一行并与之对齐，单独成行。

3 即使只有一条子语句要不要省略花括号 “{}”。如

```
while (d + = s++)  
{  
    n++;  
}
```

## 6.3 return 语句

return 语句中不使用括号，除非它能使返回值更加清晰。如

```
return;  
return myDisk.size();  
  
return (size ? size : defaultSize);
```

## 6.4 if、if-else、if else-if 语句

if、if-else、if else-if 语句使用格式

```
if (condition)  
{  
    statements;  
}  
if (condition)  
{  
    statements;  
}  
else  
{  
    statements;  
}  
  
if (condition)  
{  
    statements;  
}  
else if (condition)  
{  
    statements;  
}
```

```
else
{
    statements;
}
```

## 6.4 for、foreach 语句

for 语句使用格式

```
for (initialization; condition; update)
{
    statements;
}
```

空的 for 语句（所有的操作都在 initialization、condition 或 update 中实现）使用格式

```
for (initialization; condition; update); // update user id
```

foreach 语句使用格式

```
foreach (object obj in array)
{
    statements;
}
```

- 注意** 1 在循环过程中不要修改循环计数器。  
2 对每个空循环体给出确认性注释。

## 6.5 while 语句

while 语句使用格式

```
while (condition)
{
    statements;
}
```

空的 while 语句使用格式



```
while (condition);
```

## 6.7. do - while 语句

do - while 语句使用格式

```
do
{
    statements;
} while (condition);
```

## 6.8. switch - case 语句

switch - case 语句使用格式

```
switch (condition)
{
    case 1:
        statements;
        break;

    case 2:
        statements;
        break;

    default:
        statements;
        break;
}
```

注意：

- 1、语句 switch 中的每个 case 各占一行。
- 2、语句 switch 中的 case 按字母顺序排列。
- 3、为所有 switch 语句提供 default 分支。
- 4、所有的非空 case 语句必须用 break; 语句结束。

## 6.9. try - catch 语句

try - catch 语句使用格式

```
try
{
    statements;
}
catch (ExceptionClass e)
{
    statements;
}
finally
{
    statements;
}
```

## 6.10. using 块语句

using 块语句使用格式

```
using (object)
{
    statements;
}
```

## 6.11. goto 语句

goto 语句使用格式

```
goto Label1:
    statements;
Label1:
    statements;
```

# 第七章 控件命名规则

## 7.1 命名方法

控件名简写+英文描述，英文描述首字母大写

## 7.2 主要控件名简写对照表

控件名	简写	控件名	简写
Label	lbl	TextBox	txt
Button	btn	LinkButton	Inkbtn
ImageButton	imgbtn	DropDownList	ddl
ListBox	lst	DataGrid	dg
DataList	dl	CheckBox	chk
CheckBoxList	chkls	RadioButton	rdo
RadioButtonList	rdolt	Image	img
Panel	pnl	Calender	cld
AdRotator	ar	Table	tbl
RequiredFieldValidator	rfv	CompareValidator	cv
RangeValidator	rv	RegularExpressionValidator	rev
ValidatorSummary	vs	CrystalReportViewer	rptview

## 第八章 代码格式检查

代码格式检查使用微软内部代码检查工具 StyleCop 版本 4.3.2.1，它会根据预定义的 C#代码格式的最佳实践，对源代码进行检查，并给出不符合编码风格的错误提示（版本语言英文）。

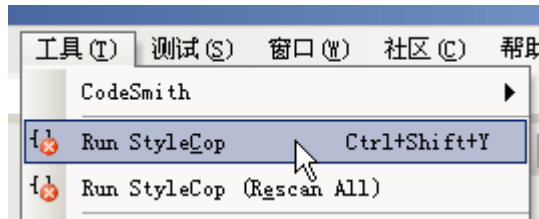
## 8.1 检查分类

检查规则分为 7 个部分，分别是

- “文档规则(Documentation Rules)”
- “布局规则(LayoutRules)”
- “可维护性规则(Maintanability Rules)”
- “命名规则(Naming Rules)”
- “代码顺序规则(Ordering Rules)”
- “可读性规则(Readability Rules)”
- “间距规则(Spacing Rules)”

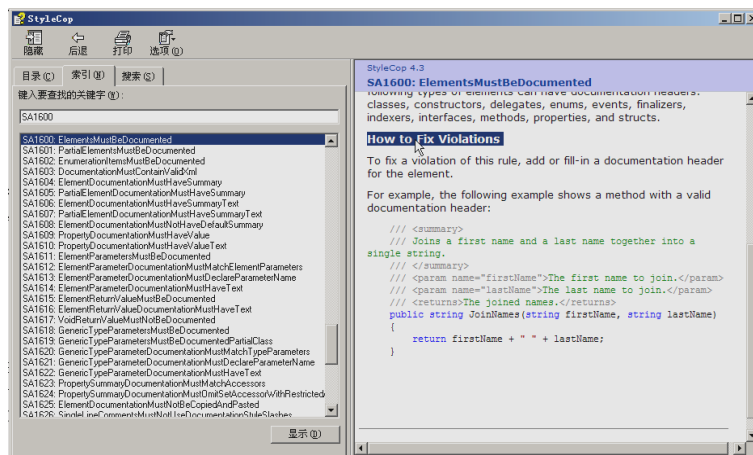
## 8.2 安装及使用

安装：安装程序位于附件，Install 文件夹中。该软件属于 Visual Studio 插件，安装后在工具菜单栏中。



使用 运行该功能后，可以根据“错误列表”中的警告信息的信息 Id 从帮助文件（位于附件，Document 文件夹中，文件名 StyleCop.chm）找到相应编码格式，进行修复。

说明	文件
9 SA1633: The file has no header, the header Xml is invalid, or the header is not located at the top of the file.	AssemblyInfo.cs
10 SA1600: The class must have a documentation header.	Class1.cs
11 SA1630: The documentation text within the summary tag does not contain any whitespace between words.	Class1.cs



开发文档：位于附件，Document 文件夹中，文件名 StyleCopSDK.chm）

对于以下情况 “是否检查” 一项中为 “√” 的内容不能违反。

### 8.3 文档规则(Documentation Rules)

标识及提示	中文及示例	是否检查
SA1600: ElementsMustBeDocumented	<b>元素必须被注释</b> <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; } </pre>	√
SA1601: PartialElementsMustBeDocu mented	<b>部分元素必须被注释</b> <pre> /// &lt;summary&gt; /// Documentation for the first part of Class1. /// &lt;/summary&gt; public partial class Class1 { }  /// &lt;summary&gt; /// Documentation for the second part of Class1. /// &lt;/summary&gt; public partial class Class1 { } </pre>	√
SA1602: EnumerationItemsMustBeDoc umented	<b>枚举项必须被注释</b> <pre> /// &lt;summary&gt; /// Types of animals. /// &lt;/summary&gt; public enum Animals {     /// &lt;summary&gt;     /// Represents a dog.     /// &lt;/summary&gt;     Dog,      /// &lt;summary&gt;     /// Represents a cat.     /// &lt;/summary&gt;     Cat,      /// &lt;summary&gt;     /// Represents a horse.     /// &lt;/summary&gt;     Horse } </pre>	√
SA1603: DocumentationMustContainV alidXml	<b>文档必须包含有效的 XML 注释</b> <pre> /// &lt;summary&gt; /// An example of badly formed Xml. /// &lt;/summary&gt; public class Example { } </pre>	√
SA1604: ElementDocumentationMust HaveSummary	<b>文档注释必须包含 Summary</b> <pre> /// &lt;summary&gt; /// Represents a customer in the database. /// &lt;/summary&gt; public class Customer { } </pre>	√
SA1605: PartialElementDocumentation	<b>部分文档注释必须包含 Summary</b> <pre> /// &lt;summary&gt; /// Documentation for the first part of Class1. </pre>	√

MustHaveSummary	<pre> /// &lt;/summary&gt; public partial class Class1 { }  /// &lt;summary&gt; /// Documentation for the second part of Class1. /// &lt;/summary&gt; public partial class Class1 { } </pre>	
SA1606: ElementDocumentationMust HaveSummaryText	<p>文档注释 Summary 必须有内容</p> <pre> /// &lt;summary&gt; &lt;/summary&gt; /// &lt;param name="customerId"&gt;The ID of the customer to find.&lt;/param&gt; /// &lt;returns&gt;The customer, or null if the customer could not be /// found.&lt;/returns&gt; public Customer FindCustomer(int customerId) { // ... finds the customer ... } </pre>	√
SA1607: PartialElementDocumentation MustHaveSummaryText	<p>部分文档注释 Summary 必须有内容</p> <pre> /// &lt;summary&gt; &lt;/summary&gt; /// &lt;param name="customerId"&gt;The ID of the customer to find.&lt;/param&gt; /// &lt;returns&gt;The customer, or null if the customer could not be found.&lt;/returns&gt; public Customer FindCustomer(int customerId) { // ... finds the customer ... }  修复代码 /// &lt;summary&gt;Attempts to locate a record for the customer with the given ID.&lt;/summary&gt; /// &lt;param name="customerId"&gt;The ID of the customer to find.&lt;/param&gt; /// &lt;returns&gt;The customer, or null if the customer could not be found.&lt;/returns&gt; public Customer FindCustomer(int customerId) { // ... finds the customer ... } </pre>	√
SA1608: ElementDocumentationMust NotHaveDefaultSummary	<p>文档注释不能有默认的 Summary</p> <pre> /// &lt;summary&gt; /// Summary description for the Example class. /// &lt;/summary&gt; public class Example { } </pre>	√
SA1609: PropertyDocumentationMust HaveValue	<p>属性注释必须有值</p> <pre> /// &lt;summary&gt; /// Gets the name of the customer. /// &lt;/summary&gt; /// &lt;value&gt;The name of the customer.&lt;/value&gt; public bool Name { get { return this.name; } } </pre>	√
SA1610: PropertyDocumentationMust HaveValueText	<p>属性注释必须有值内容</p> <pre> /// &lt;summary&gt; /// Gets the name of the customer. /// &lt;/summary&gt; /// &lt;value&gt;The name of the customer.&lt;/value&gt; public bool Name { get { return this.name; } } </pre>	√
SA1611: ElementParametersMustBeDo cumented	<p>元素的参数必须注释</p> <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) { return firstName + " " + lastName; } </pre>	√

SA1612: ElementParameterDocumentationMustMatchElementParameters	元素的参数注释必须与元素参数成对 <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; } </pre>	√
SA1613: ElementParameterDocumentationMustDeclareParameterName	元素的参数注释必须定义参数名 <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; } </pre>	√
SA1614: ElementParameterDocumentationMustHaveText	元素的参数注释必须有值 <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; } </pre>	√
SA1615: ElementReturnValueMustBeDocumented	元素的返回值必须被注释 <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; } </pre>	√
SA1616: ElementReturnValueDocumentationMustHaveValue	元素的返回值注释必须有值 <pre> /// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; } </pre>	√
SA1617: VoidReturnValueMustNotBeDocumented	元素空返回值不能注释 <pre> /// &lt;summary&gt; /// Prints the given name. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name.&lt;/param&gt; public void PrintNames(string firstName, string lastName) {     Console.WriteLine(firstName + " " + lastName); } </pre>	√
SA1618: GenericTypeParametersMustBeDocumented	泛型参数必须被注释 <pre> /// &lt;summary&gt; /// A sample generic class. /// &lt;/summary&gt; /// &lt;typeparam name="S"&gt;The first generic type parameter.&lt;/typeparam&gt; /// &lt;typeparam name="T"&gt;The second generic type parameter.&lt;/typeparam&gt; public class Class1&lt;S, T&gt; { </pre>	√

	} }	
SA1619: GenericTypeParametersMustBeDocumentedPartialClass	泛型参数在部分类中必须被注释  /// <summary> /// A sample generic class. /// </summary> /// <typeparam name="S">The first generic type parameter.</typeparam> /// <typeparam name="T">The second generic type parameter.</typeparam> public class Class1 <S, T> { }	√
SA1620: GenericTypeParameterDocumentationMustMatchTypeParameters	泛型参数注释必须与参数类型对应  /// <summary> /// A sample generic class. /// </summary> /// <typeparam name="S">The first generic type parameter.</typeparam> /// <typeparam name="T">The second generic type parameter.</typeparam> public class Class1 <S, T> { }	√
SA1621: GenericTypeParameterDocumentationMustDeclareParameterName	泛型参数注释必须定义参数名  /// <summary> /// A sample generic class. /// </summary> /// <typeparam name="S">The first generic type parameter.</typeparam> /// <typeparam name="T">The second generic type parameter.</typeparam> public class Class1 <S, T> { }	√
SA1622: GenericTypeParameterDocumentationMustHaveText	泛型参数注释必须有内容  /// <summary> /// A sample generic class. /// </summary> /// <typeparam name="S">The first generic type parameter.</typeparam> /// <typeparam name="T">The second generic type parameter.</typeparam> public class Class1 <S, T> { }	√
SA1623: PropertySummaryDocumentationMustMatchAccessors	属性摘要文档必须和访问者对应 (参考帮助文档)	√
SA1624: PropertySummaryDocumentationMustOmitSetAccessorWithRestrictedAccess	属性摘要文档必须必须省略设置访问器约束访问 (参考帮助文档)	√
SA1625: ElementDocumentationMustNotBeCopiedAndPasted	元素注释不能被拷贝和粘贴  /// <summary> /// Joins a first name and a last name together into a single string. /// </summary> /// <param name="firstName">The first name to join.</param> /// <param name="lastName">The last name to join.</param> /// <returns>The joined names.</returns> public string JoinNames(string firstName, string lastName) { return firstName + " " + lastName; }	√
SA1626: SingleLineCommentsMustNotUseDocumentationStyleSlashes	单行注释不能使用斜线样式  /// <summary> /// Joins a first name and a last name together into a single string. /// </summary> /// <param name="firstName">Part of the name.</param> /// <param name="lastName">Part of the name.</param> /// <returns>The joined names.</returns> public string JoinNames(string firstName, string lastName) { return firstName + " " + lastName; }	√
	A legal comment beginning with two slashes:	



	<pre>// Join the names together. string fullName = firstName + " " + lastName;  An illegal comment beginning with three slashes: /// Trim the name. fullName = fullName.Trim();  A line of commented-out code beginning with four slashes: ////fullName = asfd;  return fullName; }</pre>	
SA1627: DocumentationTextMustNotBeEmpty	<p>注释内容不能为空</p> <pre>/// &lt;summary&gt; /// Joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt; &lt;/param&gt; /// &lt;param name="lastName"&gt; Part of the name.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     ... }</pre>	√
SA1628: DocumentationTextMustBeginWithACapitalLetter	<p>注释内容首字母必须大写</p> <pre>/// &lt;summary&gt; /// joins a first name and a last name together into a single string. /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     ... }</pre>	√
SA1629: DocumentationTextMustEndWithAPeriod	<p>注释内容必须用句号结尾</p> <pre>/// &lt;summary&gt; /// Joins a first name and a last name together into a single string /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     ... }</pre>	√
SA1630: DocumentationTextMustContainWhitespace	<p>注释内容单词之间必须包含空白符</p> <pre>/// &lt;summary&gt; /// Joinsnames /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;First&lt;/param&gt; /// &lt;param name="lastName"&gt;Last&lt;/param&gt; /// &lt;returns&gt;Name&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     ... }</pre>	
SA1631: DocumentationTextMustMeetCharacterPercentage	<p>注释内容必须满足字符比例特殊字符不能过多</p> <pre>/// &lt;summary&gt; /// @)\$(^A name----- /// &lt;/summary&gt; public class Name {     ... }</pre>	
SA1632: DocumentationTextMustMeetMinimumCharacterLength	<p>注释内容必须满足小写字符长度</p> <pre>/// &lt;summary&gt; /// A name /// &lt;/summary&gt; public class Name {     ... }</pre>	√
SA1633: FileMustHaveHeader	<p>文件必须有文件头部注释</p> <pre>//-----</pre>	√

	<pre>// &lt;copyright file="NameOfFile.cs" company="CompanyName" &gt; //   Company copyright tag. // &lt;/copyright&gt; //-----  For example, a file called Widget.cs from a fictional company called Sprocket Enterprises should contain a file header similar to the following: //----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----  The dashed lines at the top and bottom of the header are not strictly necessary, so the header could be written as: // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt;  It is possible to add additional tags, although they will not be checked or enforced by StyleCop: //----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; // &lt;author&gt;John Doe&lt;/author&gt; //-----</pre>	
<p>SA1634: FileHeaderMustShowCopyright</p>	<p>文件头部注释必须显示版权信息标识</p> <pre>//----- // &lt;Tag&gt; A file header which does not contain a copyright tag &lt;/Tag&gt; //-----  A file header should include a copyright tag, as follows: //----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----</pre>	√
<p>SA1635: FileHeaderMustHaveCopyrightText</p>	<p>文件头部注释必须有版权信息表示内容</p> <pre>//----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; // &lt;/copyright&gt; //-----  A file header should include copyright text, as follows: //----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----</pre>	√
<p>SA1636: FileHeaderMustContainFileName</p>	<p>文件头部注释必须包含文件名</p> <pre>//----- // &lt;copyright file="Widget.cs" company="My Company" &gt; //   Custom company copyright tag. // &lt;/copyright&gt; //-----</pre>	√
<p>SA1637: FileHeaderMustContainFileName</p>	<p>文件头部注释必须包含文件名</p> <pre>//----- // &lt;copyright company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----  //----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----</pre>	√
<p>SA1638: FileHeaderFileNameDocumentationMustMatchFileName</p>	<p>文件头部注释文件名注释必须与文件名对应</p> <pre>//----- // &lt;copyright file="File2.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----  A violation of this rule would occur, since the file tag does not contain the name of</pre>	√

	<p>the file. The header should be written as:</p> <pre>//----- // &lt;copyright file="File1.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----</pre>	
SA1639: FileHeaderMustHaveSummary	<p>文件头部注释不许有摘要</p> <pre>//----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----</pre> <p>If this rule is enabled, the file header should contain a summary tag. For example:</p> <pre>//----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; // &lt;summary&gt;Defines the Widget class.&lt;/summary&gt; //-----</pre>	√
SA1640: FileHeaderMustHaveValidCompanyText	<p>文件头部注释必须有正确的公司信息</p> <pre>//----- // &lt;copyright file="Widget.cs" company="Sprocket Enterprises"&gt; //   Copyright (c) Sprocket Enterprises. All rights reserved. // &lt;/copyright&gt; //-----</pre>	√
SA1641: FileHeaderCompanyNameTextMustMatch	<p>文件头部注释公司内容必须对应</p> <pre>//----- // &lt;copyright file="Widget.cs" company="My Company"&gt; //   Custom company copyright tag. // &lt;/copyright&gt; //-----</pre>	√
SA1642: ConstructorSummaryDocumentationMustBeginWithStandardText	<p>构造器的 Summary 注释必须由标准内容开始 (参考帮助文档)</p>	√
SA1643: DestructorSummaryDocumentationMustBeginWithStandardText	<p>析构器的 Summary 注释必须由标准内容开始</p> <pre>/// &lt;summary&gt; /// Initializes a new instance of the &lt;see cref="Customer1"/&gt; class. /// &lt;/summary&gt; public Customer() { }  /// &lt;summary&gt; /// Initializes a new instance of the &lt;see cref="Customer{T}"/&gt; class. /// &lt;/summary&gt; public Customer() { }</pre>	√
SA1644: DocumentationHeadersMustNotContainBlankLines	<p>注释头部不能包含空白行</p> <pre>/// &lt;summary&gt; /// &lt;para&gt; /// Joins a first name and a last name together into a single string. /// &lt;/para&gt; &lt;para&gt; /// Uses a simple form of string concatenation. /// &lt;/para&gt; /// &lt;/summary&gt; /// &lt;param name="firstName"&gt;The first name to join.&lt;/param&gt; /// &lt;param name="lastName"&gt;The last name to join.&lt;/param&gt; /// &lt;returns&gt;The joined names.&lt;/returns&gt; public string JoinNames(string firstName, string lastName) {     return firstName + " " + lastName; }</pre>	√
SA1645: IncludedDocumentationFilesDoesNotExist	<p>导入的文档注释文件不存在</p> <pre>///&lt;include file="IncludedDocumentation.xml" path="root/EnabledMethodDocs" /&gt; public bool Enabled(bool true) { }</pre>	√

SA1646: IncludedDocumentationXPath DoesNotExist	导入的文档注释的 XPath 不存在 <pre>///&lt;include file="IncludedDocumentation.xml" path="root/EnabledMethodDocs" /&gt; public bool Enabled(bool true) { }</pre>	√
SA1647: IncludeNodeDoesNotContain ValidFileAndPath	导入的结点不存在正确的文件和路径 <pre>///&lt;include file="IncludedDocumentation.xml" path="root/EnabledMethodDocs" /&gt; public bool Enabled(bool true) { }</pre>	√

#### 8.4 布局规则(LayoutRules)

标识及提示	中文及示例	是否检查
SA1500: CurlyBracketsForMultiLineStat ementsMustNotShareLine	当多行代码使用{}时 "{" 与 "}" 必须单独占一行	√
SA1501: StatementMustNotBeOnSingl eLine	{ } 不能与代码写在同一行	√
SA1502: ElementMustNotBeOnSingleL ine	元素定义不能在同一行	√
SA1503: CurlyBracketsMustNotBeOmit ted	{ } 符号不能被省略	√
SA1504: AllAccessorMustBeMultiLineO rSingleLine	所有的访问器代码，必须是多行或一行(不能 Get 多行，Set 单行)	√
SA1505: OpeningCurlyBracketsMustN otBeFollowedByBlankLine	{ 符号下面不能跟空行	√
SA1506: ElementDocumentationHead ersMustNotBeFollowedByBla nkLine	元素注释头部不能跟空行	√
SA1507: CodeMustNotContainMultipl eBlankLinesInARow	代码不能包含连续的多个空行	√
SA1508: ClosingCurlyBracketsMustNot BePrecededByBlankLine	} 符号上方不能有空行	√

SA1509: OpeningCurlyBracketsMustNotBePrecededByBlankLine	{ 符号与代码之间不能有空行	√
SA1510: ChainedStatementBlocksMustNotBePrecededByBlankLine	使用 {} 符号并且连接的代码之间不能有空行	√
SA1511: WhileDoFooterMustNotBePrecededByBlankLine	Do While 代码之间不能有空行	√
SA1512: SingleLineCommentsMustNotBeFollowedByBlankLine	单行注释与代码之间不能有空行	√
SA1513: ClosingCurlyBracketMustBeFollowedByBlankLine	} 符号后面必须有空行	√
SA1514: ElementDocumentationHeaderMustBePrecededByBlankLine	元素注释头部必须加空行	√
SA1515: SingleLineCommentMustBePrecededByBlankLine	单行注释前面必须加空行	√
SA1516: ElementsMustBeSeparatedByBlankLine	元素必须用空行分隔	√

### 8.5 可维护性规则(Maintanability Rules)”

标识及提示	中文及示例	是否检查
SA1300: ElementMustBeginWithUpperCaseLetter	元素必须首字母大写	√
SA1301: ElementMustBeginWithLowerCaseLetter	元素必须首字母小写	√
SA1302: InterfaceNamesMustBeginWithI	命名接口必须以 I 开头	√
SA1303: ConstFieldNamesMustBeginWithUpperCaseLetter	常量字段命名必须首字母大写	√

SA1304: NonPrivateReadOnlyFieldsMustBeginWithUpperCaseLetter	非私有只读字段必须首字母大写	√
SA1305: FieldNamesMustNotUseHungarianNotation	字段名不能使用匈牙利命名法	√
SA1306: FieldNamesMustBeginWithLowerCaseLetter	字段名必须首字母小写	√
SA1307: AccessibleFieldsMustBeginWithUpperCaseLetter	可访问字段必须首字母大写	√
SA1308: VariableNamesMustNotBePrefixed	变量名不能加前缀	√
SA1309: FieldNamesMustNotBeginWithUnderscore	字段名不能以"_"开头	√
SA1310: FieldNamesMustNotContainUnderscore	字段名不能包含"_"	√

## 第九章 代码分析

使用 Visual Studio 自身的代码分析功能，检查内容如下表，分为

- 1.安全规则。
- 2.互操作性规则
- 3.可维护性规则
- 4.可以只性规则
- 5.命名规则
- 6.全球化规则
- 7.设计规则
- 8.性能规则
- 9.移动性规则

## 10.用法规则

其中“是否检查”一项中为“√”的内容不能违反。需在 Visual Studio 中设置为错误

### 9.1.安全性规则

标识	详细信息	是否检查
CA2100	检查 Sql 查询中是否有安全漏洞	√
CA2104	不要声明只读可变引用类型	√
CA2105	数组字段不应为只读	√
CA2121	静态构造函数应为私有	√

### 9.2.可靠性规则

标识	详细信息	是否检查
CA2000	超出范围前释放对象	√

### 9.3.可维护性规则

标识	详细信息	是否检查
CA1500	变量名不应与字段名相同	√
CA1501	避免过度继承	√
CA1502	避免过度复杂	√

### 9.4.命名规则

标识	详细信息	是否检查
CA1700	不要将枚举值命名为“Reserved”	√
CA1705	较长的首字母缩略词应采用 Pascal 大小写格式。	√
CA1706	较短的首字母缩略词应全部大写	√
CA1707	标识符不应包含下划线	√
CA1709	标识符的大小写应该正确	√
CA1710	标识符应具有正确的后缀	√
CA1711	标识符应采用正确的后缀	√

CA1712	不要将类型名用作枚举值的前缀	√
CA1713	事件不应具有 before 或 after 前缀	√
CA1715	标识符应具有正确的前缀	√
CA1716	标识符不应与关键字冲突	√
CA1718	避免在参数中使用特定于语言的类型名	√
CA1719	参数名不应与成员名冲突	√
CA1720	标识符不应包含类型名	√
CA1721	属性名不应与 get 方法冲突	√
CA1722	标识符应采用正确的前缀	√
CA1724	类型名不应与命名空间冲突	√
CA1725	参数名应与基方法中的声明保持一致	√

### 9.5.性能规则

标识	详细信息	是否检查
CA1800	避免进行不必要的强制转换	√
CA1804	移除未使用的局部变量	√
CA1805	避免进行不必要的初始化	√
CA1809	避免过多的局部变量	√
CA1812	避免未实例化的内部类	√
CA1813	避免未密封的属性	√
CA1819	属性不应返回数组	√
CA1823	避免未使用的私有字段	√

### 9.6.用法规则

标识	详细信息	是否检查
CA1801	检查未使用的参数	√
CA2202	不要多次释放对象	√
CA2211	非常数字段不应该是可见的	√
CA2218	重写 Equals 时重写 GetHashCode	√
CA2219	不要在异常子句中引发异常	√
CA2222	不要降低继承成员的可见性	√
CA2230	对个数可变的参数使用 params	√
CA2233	运算不应溢出	√