**XML 文档注释（C# 编程指南）**

在 Visual C# 中，你可以通过以下方式为代码创建文档：将特殊注释字段中的 XML 元素包含在源代码中注释引用的代码块的前面，例如：

/// <summary>

/// This class performs an important function.

/// </summary>

public class MyClass{}

使用 [/doc](https://msdn.microsoft.com/zh-cn/library/3260k4x7.aspx) 选项进行编译时，编译器将在源代码中搜索所有 XML 标记，并创建一个 XML 文档文件。若要基于编译器生成的文件创建最终文档，你可以创建一个自定义工具，也可以使用 [Sandcastle](http://go.microsoft.com/fwlink/?LinkId=124061) 等工具。

若要引用 XML 元素（例如，你的函数将处理你要在 XML 文档注释中描述的特定 XML 元素），你可使用标准引用机制（< 和 >）。若要引用代码引用 (**cref**) 元素中的通用标识符，则可使用转义字符（例如，cref=”List<T>”）或大括号 (cref=”List{T}”)。作为特例，编译器会将大括号解析为尖括号以在引用通用标识符时使作者能够更轻松地进行文档注释。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| XML 文档注释不是元数据；它们不包括在编译的程序集中，因此无法通过反射对其进行访问。 |

# 建议的文档注释标记（C# 编程指南）

C# 编译器处理代码中的文档注释，将它们设置为 XML 文件格式，其文件名在 **/doc** 命令行选项中指定。若要创建基于编译器生成的文件的最终文档，您可以创建自定义工具，也可以使用诸如等工具 [约定](http://shfb.codeplex.com/)。

在代码构造（如类型和类型成员）上处理标记。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 文档注释不能应用于命名空间。 |

编译器将处理任何为有效 XML 的标记。下列标记提供了用户文档中常用的功能。

## 标记

| [**<c>**](https://msdn.microsoft.com/zh-cn/library/te6h7cxs.aspx) | [**<para>**](https://msdn.microsoft.com/zh-cn/library/x640hcd2.aspx) | [**<see>**](https://msdn.microsoft.com/zh-cn/library/acd0tfbe.aspx)**\*** |
| --- | --- | --- |
| [<code>](https://msdn.microsoft.com/zh-cn/library/f8hahtxf.aspx) | [<param>](https://msdn.microsoft.com/zh-cn/library/8cw818w8.aspx)\* | [<seealso>](https://msdn.microsoft.com/zh-cn/library/xhd7ehkk.aspx)\* |
| [<example>](https://msdn.microsoft.com/zh-cn/library/9w4cf933.aspx) | [<paramref>](https://msdn.microsoft.com/zh-cn/library/wb7x2fhw.aspx) | [<summary>](https://msdn.microsoft.com/zh-cn/library/2d6dt3kf.aspx) |
| [<exception>](https://msdn.microsoft.com/zh-cn/library/w1htk11d.aspx)\* | [<permission>](https://msdn.microsoft.com/zh-cn/library/h9df2kfb.aspx)\* | [<typeparam>](https://msdn.microsoft.com/zh-cn/library/ms173191.aspx)\* |
| [<include>](https://msdn.microsoft.com/zh-cn/library/9h8dy30z.aspx)\* | [<remarks>](https://msdn.microsoft.com/zh-cn/library/3zw4z1ys.aspx) | [<typeparamref>](https://msdn.microsoft.com/zh-cn/library/ms173192.aspx) |
| [<list>](https://msdn.microsoft.com/zh-cn/library/y3ww3c7e.aspx) | [<returns>](https://msdn.microsoft.com/zh-cn/library/4dcfdeds.aspx) | [<value>](https://msdn.microsoft.com/zh-cn/library/azda5z79.aspx) |

（\* 表示编译器验证语法。）

如果想要尖括号出现在文档注释文本中，使用 < 和 >。如下面示例所示。

XML

/// <summary cref="C<T>"

/// </summary>

# 处理 XML 文件（C# 编程指南）

编译器为代码中被标记为生成文档的每一个构造生成一个 ID 字符串。（有关如何标记代码的信息，请参见[建议的文档注释标记](https://msdn.microsoft.com/zh-cn/library/5ast78ax.aspx)。）ID 字符串唯一地标识构造。处理 XML 文件的程序可以使用 ID 字符串标识文档应用于的相应 .NET Framework 元数据/反射项。

XML 文件不是代码的分层表示形式，它是一个平面列表，其中的每一个元素都有一个生成的 ID。

编译器在生成 ID 字符串时遵循下列规则：

* 字符串中没有空白。
* ID 字符串的第一部分通过单个字符后跟一个冒号来标识所标识成员的种类。使用下列成员类型：

| **字符** | **说明** |
| --- | --- |
| N | 命名空间不可将文档注释添加到命名空间中，但是可以对它们进行 cref 引用（在受支持的位置）。 |
| T | 类型：类、接口、结构、枚举、委托 |
| F | 字段 |
| P | 属性（包括索引程序或其他索引属性） |
| M | 方法（包括一些特殊方法，例如构造函数、运算符等） |
| E | event |
| ! | 错误字符串字符串的其余部分会提供有关此错误的信息。C# 编译器为无法解析的链接生成错误信息。 |

* 字符串的第二部分是项的完全限定名，从命名空间的根开始。项的名称、其封闭类型和命名空间以句号分隔。如果项的名称本身包含句号，则用哈希符号 ('#') 替换这些句号。假定任何项的名称中都不直接存在哈希符号。例如，String 构造函数的完全限定名是“System.String.#ctor”。
* 对于属性和方法，如果该方法带参数，则将其后的参数列表括在括号中。如果没有参数，则没有括号。多个参数以逗号分隔。每个参数的编码都直接遵循它在 .NET Framework 签名中的编码方法：
  + 基类型。常规类型（ELEMENT\_TYPE\_CLASS 或 ELEMENT\_TYPE\_VALUETYPE）表示为类型的完全限定名。
  + 内部类型（例如，ELEMENT\_TYPE\_I4、ELEMENT\_TYPE\_OBJECT、ELEMENT\_TYPE\_STRING、ELEMENT\_TYPE\_TYPEDBYREF。和 ELEMENT\_TYPE\_VOID）表示为相应完全类型的完全限定名。例如，System.Int32 或 System.TypedReference。
  + ELEMENT\_TYPE\_PTR 表示为“\*”，在修改的类型之后。
  + ELEMENT\_TYPE\_BYREF 表示为“@”，在修改的类型之后。
  + ELEMENT\_TYPE\_PINNED 表示为“^”，在修改的类型之后。C# 编译器永远不会生成此结果。
  + ELEMENT\_TYPE\_CMOD\_REQ 表示为“|”和修饰符类的完全限定名，在修改的类型之后。C# 编译器永远不会生成此结果。
  + ELEMENT\_TYPE\_CMOD\_OPT 表示为“!”和修饰符类的完全限定名，在修改的类型之后。
  + ELEMENT\_TYPE\_SZARRAY 表示为“[]”，在数组的元素类型之后。
  + ELEMENT\_TYPE\_GENERICARRAY 表示为“[?]”，在数组的元素类型之后。C# 编译器永远不会生成此结果。
  + ELEMENTTYPE\_ARRAY 表示为 [\_lowerbound:size,lowerbound:size]，其中逗号数为秩 - 1，每一维的下限和大小（如果已知）用十进制表示。如果未指定下限及大小，它将完全被省略。如果省略了特定维的下限及大小，则“:”也将被省略。例如，以 1 作为下限并且未指定大小的二维数组是 [1:,1:]。
  + ELEMENTTYPE\_FNPTR 表示为“=FUNC:\_type(signature)”，其中 type 是返回类型，signature 是方法的参数。如果没有参数，则将省略括号。C# 编译器永远不会生成此结果。

不表示下列签名组件，因为从不使用它们来区分重载方法：

* + 调用约定
  + 返回类型
  + ELEMENT\_TYPE\_SENTINEL
* 仅对于转换运算符（op\_Implicit 和 op\_Explicit），方法的返回值才被编码为“~”，后跟返回类型，如上述编码所示。
* 对于泛型类型，类型名称后跟反勾号，再跟一个数字，指示泛型类型参数的个数。例如，

对于定义为 public class SampleClass<T, U> 的类型，<member name="T:SampleClass`2"> 是其标记。

对于接受泛型类型作为参数的方法，泛型类型参数被指定为数字前面加上反勾号（如 0、1）。表示类型泛型参数的数组表示法（从零开始）的每个数字。

## 示例

下列示例显示了如何生成类及其成员的 ID 字符串：

namespace N

{

/// <summary>

/// Enter description here for class X.

/// ID string generated is "T:N.X".

/// </summary>

public unsafe class X

{

/// <summary>

/// Enter description here for the first constructor.

/// ID string generated is "M:N.X.#ctor".

/// </summary>

public X() { }

/// <summary>

/// Enter description here for the second constructor.

/// ID string generated is "M:N.X.#ctor(System.Int32)".

/// </summary>

/// <param name="i">Describe parameter.</param>

public X(int i) { }

/// <summary>

/// Enter description here for field q.

/// ID string generated is "F:N.X.q".

/// </summary>

public string q;

/// <summary>

/// Enter description for constant PI.

/// ID string generated is "F:N.X.PI".

/// </summary>

public const double PI = 3.14;

/// <summary>

/// Enter description for method f.

/// ID string generated is "M:N.X.f".

/// </summary>

/// <returns>Describe return value.</returns>

public int f() { return 1; }

/// <summary>

/// Enter description for method bb.

/// ID string generated is "M:N.X.bb(System.String,System.Int32@,System.Void\*)".

/// </summary>

/// <param name="s">Describe parameter.</param>

/// <param name="y">Describe parameter.</param>

/// <param name="z">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public int bb(string s, ref int y, void\* z) { return 1; }

/// <summary>

/// Enter description for method gg.

/// ID string generated is "M:N.X.gg(System.Int16[],System.Int32[0:,0:])".

/// </summary>

/// <param name="array1">Describe parameter.</param>

/// <param name="array">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public int gg(short[] array1, int[,] array) { return 0; }

/// <summary>

/// Enter description for operator.

/// ID string generated is "M:N.X.op\_Addition(N.X,N.X)".

/// </summary>

/// <param name="x">Describe parameter.</param>

/// <param name="xx">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public static X operator +(X x, X xx) { return x; }

/// <summary>

/// Enter description for property.

/// ID string generated is "P:N.X.prop".

/// </summary>

public int prop { get { return 1; } set { } }

/// <summary>

/// Enter description for event.

/// ID string generated is "E:N.X.d".

/// </summary>

public event D d;

/// <summary>

/// Enter description for property.

/// ID string generated is "P:N.X.Item(System.String)".

/// </summary>

/// <param name="s">Describe parameter.</param>

/// <returns></returns>

public int this[string s] { get { return 1; } }

/// <summary>

/// Enter description for class Nested.

/// ID string generated is "T:N.X.Nested".

/// </summary>

public class Nested { }

/// <summary>

/// Enter description for delegate.

/// ID string generated is "T:N.X.D".

/// </summary>

/// <param name="i">Describe parameter.</param>

public delegate void D(int i);

/// <summary>

/// Enter description for operator.

/// ID string generated is "M:N.X.op\_Explicit(N.X)~System.Int32".

/// </summary>

/// <param name="x">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public static explicit operator int(X x) { return 1; }

}

}

# 文档标记的分隔符

XML 文档注释需要使用分隔符，这些分隔符指示文档注释开始和结束的位置。 可以将以下分隔符用于 XML 文档标记：

* /// 单行分隔符：文档示例和 C# 项目模板使用此形式。 如果分隔符后面有空格，则它不会包含在 XML 输出中。

**备注**

Visual Studio 会自动插入 <summary> 和 </summary> 标记，你在代码编辑器中键入 /// 分隔符后，光标将置于这些标记中。 可以在[**“选项”对话框**](https://learn.microsoft.com/zh-cn/visualstudio/ide/reference/options-text-editor-csharp-advanced)中打开/关闭此功能。

* /\*\* \*/ 多行分隔符：/\*\* \*/ 分隔符具有以下格式规则：
  + 在包含 /\*\* 分隔符的行上，如果行的其余部分为空格，则不将此行作为注释处理。 如果 /\*\* 分隔符后面的第一个字符为空格，则忽略此空格字符，并处理行的其余部分。 否则，将 /\*\* 分隔符后面的行的所有文本作为注释的一部分进行处理。
  + 在包含 \*/ 分隔符的行中，如果 \*/ 分隔符前面只有空格，此行将被忽略。 否则，将 \*/ 分隔符之前的行的文本作为注释的一部分进行处理。
  + 对于以 /\*\* 分隔符开头的行后面的行，编译器在各行的开头寻找共同模式。 此模式可以包含空格和星号 (\*)，后面跟更多空格。 如果编译器在不以 /\*\* 分隔符开头或不以 \*/ 分隔符结尾的各行开头找到共同模式，则忽略此每个行的模式。
  + 以下注释中将被处理的唯一部分是以 <summary> 开头的行。 三种标记格式产生的注释相同。

C#复制

/\*\* <summary>text</summary> \*/

/\*\*

<summary>text</summary>

\*/

/\*\*

\* <summary>text</summary>

\*/

* + 编译器识别出第二和第三行开头的共同模式“\*”。 此模式不包括在输出中。

C#复制

/\*\*

\* <summary>

\* text </summary>\*/

* + 编译器在下面的注释中未找到共同模式，因为第三行的第二个字符不是一个星号。 第二和第三行上的所有文本将处理为注释的一部分。

C#复制

/\*\*

\* <summary>

text </summary>

\*/

* + 编译器在以下注释中未找到模式，原因有两个。 首先，星号前的空格数不一致。 其次，第 5 行以制表符开头，这与空格不匹配。 第二到第五行的所有文本都作为注释的一部分进行处理。

C#复制

/\*\*

\* <summary>

\* text

\* text2

\* </summary>

\*/

若要引用 XML 元素（例如，你的函数将处理你要在 XML 文档注释中描述的特定 XML 元素），你可使用标准引用机制（&lt; 和 &gt;）。 若要引用代码引用 (cref) 元素中的通用标识符，可使用转义字符（例如，cref="List&lt;T&gt;"）或大括号 (cref="List{T}")。 作为特例，编译器会将大括号解析为尖括号以在引用通用标识符时使作者能够更轻松地进行文档注释。

**备注**

XML 文档注释不是元数据；它们不包括在编译的程序集中，因此无法通过反射对其进行访问。

# 使用 XML 文档功能（C# 编程指南）

下面的示例提供文档类型的基本概述。

// If compiling from the command line, compile with: /doc:YourFileName.xml

/// <summary>

/// Class level summary documentation goes here.</summary>

/// <remarks>

/// Longer comments can be associated with a type or member through

/// the remarks tag.</remarks>

public class TestClass : TestInterface

{

/// <summary>

/// Store for the name property.</summary>

private string \_name = null;

/// <summary>

/// The class constructor. </summary>

public TestClass()

{

// TODO: Add Constructor Logic here.

}

/// <summary>

/// Name property. </summary>

/// <value>

/// A value tag is used to describe the property value.</value>

public string Name

{

get

{

if (\_name == null)

{

throw new System.Exception("Name is null");

}

return \_name;

}

}

/// <summary>

/// Description for SomeMethod.</summary>

/// <param name="s"> Parameter description for s goes here.</param>

/// <seealso cref="System.String">

/// You can use the cref attribute on any tag to reference a type or member

/// and the compiler will check that the reference exists. </seealso>

public void SomeMethod(string s)

{

}

/// <summary>

/// Some other method. </summary>

/// <returns>

/// Return results are described through the returns tag.</returns>

/// <seealso cref="SomeMethod(string)">

/// Notice the use of the cref attribute to reference a specific method. </seealso>

public int SomeOtherMethod()

{

return 0;

}

public int InterfaceMethod(int n)

{

return n \* n;

}

/// <summary>

/// The entry point for the application.

/// </summary>

/// <param name="args"> A list of command line arguments.</param>

static int Main(System.String[] args)

{

// TODO: Add code to start application here.

return 0;

}

}

/// <summary>

/// Documentation that describes the interface goes here.

/// </summary>

/// <remarks>

/// Details about the interface go here.

/// </remarks>

interface TestInterface

{

/// <summary>

/// Documentation that describes the method goes here.

/// </summary>

/// <param name="n">

/// Parameter n requires an integer argument.

/// </param>

/// <returns>

/// The method returns an integer.

/// </returns>

int InterfaceMethod(int n);

}

//此 .xml 文件生成了与上面的代码示例。<?xml version="1.0"?>

<doc>

<assembly>

namexmlsampleAMP\_LT/nameAMP\_GT

</assembly>

<members>

member name="T:SomeClass"

<summary>

Class level summary documentation goes here.</summary>

<remarks>

Longer comments can be associated with a type or member

通过注释 tagAMP\_LT/remarksAMP\_GT

</member>

member name="F:SomeClass.m\_Name"

<summary>

名称的 propertyAMP\_LT/summaryAMP\_GT 存储

</member>

member name="M:SomeClass.#ctor"

summary类 constructor.AMP\_LT/summaryAMP\_GT

</member>

member name="M:SomeClass.SomeMethod(System.String)"

<summary>

Description for SomeMethod.</summary>

param name="s" 中的参数声明转到 hereAMP\_LT/paramAMP\_GT

<seealso cref="T:System.String">

You can use the cref attribute on any tag to reference a type or member

and the compiler will check that the reference exists.</seealso>

</member>

member name="M:SomeClass.SomeOtherMethod"

<summary>

Some other method.</summary>

<returns>

Return results are described through the returns tag.</returns>

seealso cref="M:SomeClass.SomeMethod(System.String)"

通知使用 cref 特性引用特定方法 AMP\_LT/seealsoAMP\_GT

</member>

member name="M:SomeClass.Main(System.String[])"

<summary>

The entry point for the application.</summary>

param name="args" 命令行 argumentsAMP\_LT/paramAMP\_GT 列表

</member>

member name="P:SomeClass.Name"

<summary>

name 属性 AMP\_LT/summaryAMP\_GT

<value>

值标记用于描述属性 valueAMP\_LT/valueAMP\_GT

</member>

</members>

</doc>

## 编译代码

若要编译该示例，请键入以下命令行:

csc XMLsample.cs /doc:XMLsample.xml

这将创建 XML 文件 XMLsample.xml，使用类型的命令，您可以在浏览器或。

## 可靠编程

XML 文档以 /// 开头。当您创建新项目时，该向导为您放置一些起始 ///行。对这些注释的处理有一些限制：

* 文档必须是格式良好的 XML。如果 XML 格式不正确，将生成警告，并且文档文件将包含添加的注释遇到错误。
* 开发人员可自由创建自己的标记集。具有建议的标记集 (请参见其他阅读材料部分)。某些建议的标记具有特殊含义：
  + <param> 标记用于描述参数。如果使用，编译器将验证参数是否存在，以及文档中是否描述了所有参数。如果验证失败，编译器会发出警告。
  + **cref** 特性可以附加到任意标记，以提供对代码元素的引用。编译器将验证该代码元素是否存在。如果验证失败，编译器会发出警告。编译器在查找 **cref** 特性中描述的类型时，会考虑所有的 **using** 语句。
  + <summary> 标记由 Visual Studio 内的“Intellisense”使用，用来显示类型或成员的其他相关信息。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| XML 文件不提供有关类型和成员的完整信息 (例如，它不包含任何类型信息)。若要获得有关类型或成员的完整信息，文档文件必须与实际类型或成员上的反射一起使用。 |