**分部类和方法（C# 编程指南）**

可以将[类](https://msdn.microsoft.com/zh-CN/library/0b0thckt.aspx)或[结构](https://msdn.microsoft.com/zh-CN/library/ah19swz4.aspx)、[接口](https://msdn.microsoft.com/zh-CN/library/87d83y5b.aspx)或方法的定义拆分到两个或多个源文件中。每个源文件包含类型或方法定义的一部分，编译应用程序时将把所有部分组合起来。

**分部类**

在以下几种情况下需要拆分类定义：

* 处理大型项目时，使一个类分布于多个独立文件中可以让多位程序员同时对该类进行处理。
* 使用自动生成的源时，无需重新创建源文件便可将代码添加到类中。Visual Studio 在创建 Windows 窗体、Web 服务包装代码等时都使用此方法。无需修改 Visual Studio 创建的文件，就可创建使用这些类的代码。
* 若要拆分类定义，请使用 [partial](https://msdn.microsoft.com/zh-CN/library/wbx7zzdd.aspx) 关键字修饰符，如下所示：

public partial class Employee

{

public void DoWork()

{

}

}

public partial class Employee

{

public void GoToLunch()

{

}

}

**partial** 关键字指示可在命名空间中定义该类、结构或接口的其他部分。所有部分都必须使用 **partial** 关键字。在编译时，各个部分都必须可用来形成最终的类型。各个部分必须具有相同的可访问性，如 **public**、**private** 等。

如果将任意部分声明为抽象的，则整个类型都被视为抽象的。如果将任意部分声明为密封的，则整个类型都被视为密封的。如果任意部分声明基类型，则整个类型都将继承该类。

指定基类的所有部分必须一致，但忽略基类的部分仍继承该基类型。各个部分可以指定不同的基接口，最终类型将实现所有分部声明所列出的全部接口。在某一分部定义中声明的任何类、结构或接口成员可供所有其他部分使用。最终类型是所有部分在编译时的组合。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| **partial** 修饰符不可用于委托或枚举声明中。 |

下面的示例演示嵌套类型可以是分部的，即使它们所嵌套于的类型本身并不是分部的也如此。

class Container

{

partial class Nested

{

void Test() { }

}

partial class Nested

{

void Test2() { }

}

}

编译时将对分部类型定义的特性进行合并。例如，请考虑下列声明：

[SerializableAttribute]

partial class Moon { }

[ObsoleteAttribute]

partial class Moon { }

它们等效于以下声明：

[SerializableAttribute]

[ObsoleteAttribute]

class Moon { }

将从所有分部类型定义中对以下内容进行合并：

* XML 注释
* 接口
* 泛型类型参数特性
* 类特性
* members

例如，请考虑下列声明：

partial class Earth : Planet, IRotate { }

partial class Earth : IRevolve { }

它们等效于以下声明：

class Earth : Planet, IRotate, IRevolve { }

**限制**

处理分部类定义时需遵循下面的几个规则：

* 要作为同一类型的各个部分的所有分部类型定义都必须使用 **partial** 进行修饰。例如，下面的类声明将生成错误：
* public partial class A { }
* //public class A { } // Error, must also be marked partial
* **partial** 修饰符只能出现在紧靠关键字 **class**、**struct** 或 **interface** 前面的位置。
* 分部类型定义中允许使用嵌套的分部类型，如下面的示例中所示：
* partial class ClassWithNestedClass
* {
* partial class NestedClass { }
* }
* partial class ClassWithNestedClass
* {
* partial class NestedClass { }
* }
* 要成为同一类型的各个部分的所有分部类型定义都必须在同一程序集和同一模块（.exe 或 .dll 文件）中进行定义。分部定义不能跨越多个模块。
* 类名和泛型类型参数在所有的分部类型定义中都必须匹配。泛型类型可以是分部的。每个分部声明都必须以相同的顺序使用相同的参数名。
* 下面的用于分部类型定义中的关键字是可选的，但是如果某关键字出现在一个分部类型定义中，则该关键字不能与在同一类型的其他分部定义中指定的关键字冲突：
  + [public](https://msdn.microsoft.com/zh-CN/library/yzh058ae.aspx)
  + [private](https://msdn.microsoft.com/zh-CN/library/st6sy9xe.aspx)
  + [protected](https://msdn.microsoft.com/zh-CN/library/bcd5672a.aspx)
  + [internal](https://msdn.microsoft.com/zh-CN/library/7c5ka91b.aspx)
  + [abstract](https://msdn.microsoft.com/zh-CN/library/sf985hc5.aspx)
  + [sealed](https://msdn.microsoft.com/zh-CN/library/88c54tsw.aspx)
  + 基类
  + [new](https://msdn.microsoft.com/zh-CN/library/51y09td4.aspx) 修饰符（嵌套部分）
  + 泛型约束

有关更多信息，请参见 [类型参数的约束（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/d5x73970.aspx)。

**示例 1**

**说明**

下面的示例在一个分部类定义中声明 CoOrds 类的字段和构造函数，在另一个分部类定义中声明成员 PrintCoOrds。

**代码**

public partial class CoOrds

{

private int x;

private int y;

public CoOrds(int x, int y)

{

this.x = x;

this.y = y;

}

}

public partial class CoOrds

{

public void PrintCoOrds()

{

Console.WriteLine("CoOrds: {0},{1}", x, y);

}

}

class TestCoOrds

{

static void Main()

{

CoOrds myCoOrds = new CoOrds(10, 15);

myCoOrds.PrintCoOrds();

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

// Output: CoOrds: 10,15

**示例 2**

**说明**

从下面的示例可以看出，您也可以开发分部结构和接口。

**代码**

partial interface ITest

{

void Interface\_Test();

}

partial interface ITest

{

void Interface\_Test2();

}

partial struct S1

{

void Struct\_Test() { }

}

partial struct S1

{

void Struct\_Test2() { }

}

**分部方法**

分部类或结构可以包含分部方法。类的一个部分包含方法的签名。可以在同一部分或另一个部分中定义可选实现。如果未提供该实现，则会在编译时移除方法以及对方法的所有调用。

分部方法使类的某个部分的实施者能够定义方法（类似于事件）。类的另一部分的实施者可以决定是否实现该方法。如果未实现该方法，编译器将移除方法签名以及对该方法的所有调用。调用该方法，包括调用中的任何计算结果，在运行时没有任何影响。因此，分部类中的任何代码都可以随意地使用分部方法，即使未提供实现也是如此。如果调用了未实现的方法，将不会导致编译时错误或运行时错误。

在自定义生成的代码时，分部方法特别有用。这些方法允许保留方法名称和签名，因此生成的代码可以调用方法，而开发人员可以决定是否实现方法。与分部类非常类似，分部方法使代码生成器创建的代码和开发人员创建的代码能够协同工作，而不会产生运行时开销。

分部方法声明由两个部分组成：定义和实现。它们可以位于分部类的不同部分中，也可以位于同一部分中。如果不存在实现声明，则编译器将优化定义声明和对方法的所有调用。

// Definition in file1.cs

partial void onNameChanged();

// Implementation in file2.cs

partial void onNameChanged()

{

// method body

}

* 分部方法声明必须以上下文关键字 [partial](https://msdn.microsoft.com/zh-CN/library/wbx7zzdd.aspx) 开头，并且方法必须返回 [void](https://msdn.microsoft.com/zh-CN/library/yah0tteb.aspx)。
* 分部方法可以有 [ref](https://msdn.microsoft.com/zh-CN/library/14akc2c7.aspx) 参数，但不能有 [out](https://msdn.microsoft.com/zh-CN/library/t3c3bfhx.aspx) 参数。
* 分部方法为隐式 [private](https://msdn.microsoft.com/zh-CN/library/st6sy9xe.aspx) 方法，因此不能为 [virtual](https://msdn.microsoft.com/zh-CN/library/9fkccyh4.aspx) 方法。
* 分部方法不能为 [extern](https://msdn.microsoft.com/zh-CN/library/e59b22c5.aspx) 方法，因为主体的存在确定了方法是在定义还是在实现。
* 分部方法可以有 [static](https://msdn.microsoft.com/zh-CN/library/98f28cdx.aspx) 和 [unsafe](https://msdn.microsoft.com/zh-CN/library/chfa2zb8.aspx) 修饰符。
* 分部方法可以为泛型的。约束将放在定义分部方法声明上，但也可以选择重复放在实现声明上。参数和类型参数名称在实现声明和定义声明中不必相同。
* 您可以为已定义并实现的分部方法生成[委托](https://msdn.microsoft.com/zh-CN/library/900fyy8e.aspx)，但不能为已经定义但未实现的分部方法生成委托

## 请参阅

[C# 编程指南](https://msdn.microsoft.com/zh-CN/library/67ef8sbd.aspx)

[类（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/x9afc042.aspx)

[结构（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/saxz13w4.aspx)

[接口（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173156.aspx)

[分部（类型）（C# 参考）](https://msdn.microsoft.com/zh-CN/library/wbx7zzdd.aspx)