方法是包含一系列语句的代码块。程序通过调用该方法并指定任何所需的方法参数使语句得以执行。在 C# 中，每个执行的指令均在方法的上下文中执行。Main 方法是每个 C# 应用程序的入口点，并在启动程序时由公共语言运行时 (CLR) 调用。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 本主题讨论命名的方法。有关匿名函数的信息，请参阅[匿名函数（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/bb882516.aspx)。 |

**方法签名**

通过指定访问级别（如 **public** 或 **private**）、可选修饰符（如 **abstract** 或 **sealed**）、返回值、方法的名称以及任何方法参数，在[类](https://msdn.microsoft.com/zh-CN/library/0b0thckt.aspx)或[结构](https://msdn.microsoft.com/zh-CN/library/ah19swz4.aspx)中声明方法。这些部件一起构成方法的签名。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 出于方法重载的目的，方法的返回类型不是方法签名的一部分。但是在确定委托和它所指向的方法之间的兼容性时，它是方法签名的一部分。 |

方法参数在括号内，并且用逗号分隔。空括号指示方法不需要任何参数。此类包含三种方法：

abstract class Motorcycle

{

// Anyone can call this.

public void StartEngine() {/\* Method statements here \*/ }

// Only derived classes can call this.

protected void AddGas(int gallons) { /\* Method statements here \*/ }

// Derived classes can override the base class implementation.

public virtual int Drive(int miles, int speed) { /\* Method statements here \*/ return 1; }

// Derived classes must implement this.

public abstract double GetTopSpeed();

}

**方法访问**

调用对象上的方法就像访问字段。在对象名之后添加一个句点、方法名和括号。参数列在括号里，并且用逗号分隔。因此，可在以下示例中调用 Motorcycle 类的方法：

class TestMotorcycle : Motorcycle

{

public override double GetTopSpeed()

{

return 108.4;

}

static void Main()

{

TestMotorcycle moto = new TestMotorcycle();

moto.StartEngine();

moto.AddGas(15);

moto.Drive(5, 20);

double speed = moto.GetTopSpeed();

Console.WriteLine("My top speed is {0}", speed);

}

}

**方法参数与参数**

该方法定义指定任何所需参数的名称和类型。调用代码调用该方法时，它为每个参数提供了称为参数的具体值。参数必须与参数类型兼容，但调用代码中使用的参数名（如果有）不需要与方法中定义的参数名相同。例如:

public void Caller()

{

int numA = 4;

// Call with an int variable.

int productA = Square(numA);

int numB = 32;

// Call with another int variable.

int productB = Square(numB);

// Call with an integer literal.

int productC = Square(12);

// Call with an expression that evaulates to int.

productC = Square(productA \* 3);

}

int Square(int i)

{

// Store input argument in a local variable.

int input = i;

return input \* input;

}

**按引用传递与按值传递**

默认情况下，值类型传递给方法时，传递的是副本而不是对象本身。因此，对参数的更改不会影响调用方法中的原始副本。可以使用 ref 关键字按引用传递值类型。有关详细信息，请参阅[传递值类型参数（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/9t0za5es.aspx)。有关内置值类型的列表，请参阅[值类型表（C# 参考）](https://msdn.microsoft.com/zh-CN/library/bfft1t3c.aspx)。

引用类型的对象传递到方法中时，将传递对对象的引用。也就是说，该方法接收的不是对象本身，而是指示该对象位置的参数。如果通过使用此引用更改对象的成员，即使是按值传递该对象，此更改也会反映在调用方法的参数中。

通过使用 **class** 关键字创建引用类型，如以下示例所示。

public class SampleRefType

{

public int value;

}

现在，如果将基于此类型的对象传递到方法，则将传递对对象的引用。下面的示例将 SampleRefType 类型的对象传递到 ModifyObject 方法。

public static void TestRefType()

{

SampleRefType rt = new SampleRefType();

rt.value = 44;

ModifyObject(rt);

Console.WriteLine(rt.value);

}

static void ModifyObject(SampleRefType obj)

{

obj.value = 33;

}

该示例执行的内容实质上与先前示例相同，均按值将参数传递到方法。但是因为使用了引用类型，结果有所不同。 ModifyObject 中所做的对形参 obj 的 value 字段的修改，也会更改 TestRefType 方法中实参 rt 的 value 字段。 TestRefType 方法显示 33 作为输出。

有关如何按引用和按值传递引用类型的详细信息，请参阅[传递引用类型参数（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/s6938f28.aspx)和[引用类型（C# 参考）](https://msdn.microsoft.com/zh-CN/library/490f96s2.aspx)。

**返回值**

方法可以将值返回到调用方。如果列在方法名之前的返回类型不是 **void**，则该方法可通过使用 **return** 关键字返回值。带 **return** 关键字，后跟与返回类型匹配的值的语句将该值返回到方法调用方。 **return** 关键字还会停止执行该方法。如果返回类型为 **void**，没有值的 **return** 语句仍可用于停止执行该方法。没有 **return** 关键字，当方法到达代码块结尾时，将停止执行。具有非空的返回类型的方法都需要使用 **return** 关键字来返回值。例如，这两种方法都使用 **return** 关键字来返回整数：

class SimpleMath

{

public int AddTwoNumbers(int number1, int number2)

{

return number1 + number2;

}

public int SquareANumber(int number)

{

return number \* number;

}

}

若要使用从方法返回的值，调用方法可以在相同类型的值足够的地方使用该方法调用本身。也可以将返回值分配给变量。例如，以下两个代码示例实现了相同的目标：

int result = obj.AddTwoNumbers(1, 2);

result = obj.SquareANumber(result);

// The result is 9.

Console.WriteLine(result);

result = obj.SquareANumber(obj.AddTwoNumbers(1, 2));

// The result is 9.

Console.WriteLine(result);

在这种情况下，使用本地变量 result 存储值是可选的。此步骤可以帮助提高代码的可读性，或者如果需要存储该方法整个范围内参数的原始值，则此步骤可能很有必要。

有关详细信息，请参阅[return（C# 参考）](https://msdn.microsoft.com/zh-CN/library/1h3swy84.aspx)。

**异步方法**

通过使用异步功能，你可以调用异步方法而无需使用显式回调，也不需要跨多个方法或 lambda 表达式来手动拆分代码。Visual Studio 2012 中已引入异步功能。

如果用 [async](https://msdn.microsoft.com/zh-CN/library/hh156513.aspx) 修饰符标记方法，则可以使用该方法中的 [await](https://msdn.microsoft.com/zh-CN/library/hh156528.aspx) 运算符。当控件到达异步方法中的 await 表达式时，控件将返回到调用方，并在等待任务完成前，方法中进度将一直处于挂起状态。任务完成后，可以在方法中恢复执行。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 异步方法在遇到第一个尚未完成的 awaited 对象或到达异步方法的末尾时（以先发生者为准），将返回到调用方。 |

异步方法可以具有 [Task<TResult>](https://msdn.microsoft.com/zh-CN/library/dd321424.aspx)、[Task](https://msdn.microsoft.com/zh-CN/library/system.threading.tasks.task.aspx) 或 void 返回类型。Void 返回类型主要用于定义需要 void 返回类型的事件处理程序。无法等待返回 void 的异步方法，并且返回 void 方法的调用方无法捕获该方法引发的异常。

在以下示例中，DelayAsync 是具有 [Task<TResult>](https://msdn.microsoft.com/zh-CN/library/dd321424.aspx) 返回类型的异步方法。 DelayAsync 具有返回整数的 **return** 语句。因此，DelayAsync 的方法声明必须具有 **Task<int>** 的返回类型。因为返回类型是 **Task<int>**，DoSomethingAsync 中 **await** 表达式的计算如以下语句所示得出整数：int result = await delayTask。

startButton\_Click 方法是具有 void 返回类型的异步方法的示例。因为 DoSomethingAsync 是异步方法，调用 DoSomethingAsync 的任务必须等待，如以下语句所示：await DoSomethingAsync();。 startButton\_Click 方法必须使用 **async** 修饰符进行定义，因为该方法具有 **await** 表达式。

// using System.Diagnostics;

// using System.Threading.Tasks;

// This Click event is marked with the async modifier.

private async void startButton\_Click(object sender, RoutedEventArgs e)

{

await DoSomethingAsync();

}

private async Task DoSomethingAsync()

{

Task<int> delayTask = DelayAsync();

int result = await delayTask;

// The previous two statements may be combined into

// the following statement.

//int result = await DelayAsync();

Debug.WriteLine("Result: " + result);

}

private async Task<int> DelayAsync()

{

await Task.Delay(100);

return 5;

}

// Output:

// Result: 5

异步方法不能声明任何 [ref](https://msdn.microsoft.com/zh-CN/library/14akc2c7.aspx) 或 [out](https://msdn.microsoft.com/zh-CN/library/t3c3bfhx.aspx) 参数，但是可以调用具有这类参数的方法。

有关异步方法的详细信息，请参阅[使用 Async 和 Await 的异步编程（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-CN/library/hh191443.aspx)、[异步程序中的控制流（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-CN/library/hh873191.aspx)和[异步返回类型（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-CN/library/hh524395.aspx)。

**表达式主体定义**

具有立即仅返回表达式结果，或单个语句作为方法主题的方法定义很常见。以下是使用 **=>** 定义此类方法的语法快捷方式：

public Point Move(int dx, int dy) => new Point(x + dx, y + dy);

public void Print() => Console.WriteLine(First + " " + Last);

// Works with operators, properties, and indexers too.

public static Complex operator +(Complex a, Complex b) => a.Add(b);

public string Name => First + " " + Last;

public Customer this[long id] => store.LookupCustomer(id);

如果该方法返回 **void** 或是异步方法，则该方法的主体必须是语句表达式（与 lambda 相同）。对于属性和索引器，两者必须是只读，并且不使用 **get** 访问器关键字。

**迭代器**

迭代器对集合执行自定义迭代，如列表或数组。迭代器使用 [yield return](https://msdn.microsoft.com/zh-CN/library/9k7k7cf0.aspx) 语句返回元素，每次返回一个。当 [yield return](https://msdn.microsoft.com/zh-CN/library/9k7k7cf0.aspx) 语句到达时，将记住当前在代码中的位置。下次调用迭代器时，将从该位置重新开始执行。

通过使用 [foreach](https://msdn.microsoft.com/zh-CN/library/ttw7t8t6.aspx) 语句从客户端代码调用迭代器。

迭代器的返回类型可以是 [IEnumerable](https://msdn.microsoft.com/zh-CN/library/system.collections.ienumerable.aspx)、[IEnumerable<T>](https://msdn.microsoft.com/zh-CN/library/9eekhta0.aspx)、[IEnumerator](https://msdn.microsoft.com/zh-CN/library/system.collections.ienumerator.aspx) 或 [IEnumerator<T>](https://msdn.microsoft.com/zh-CN/library/78dfe2yb.aspx)。

有关详细信息，请参阅[迭代器（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-CN/library/dscyy5s0.aspx)。

## 请参阅

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

[类和结构（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173109.aspx)

[访问修饰符（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173121.aspx)

[静态类和静态类成员（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/79b3xss3.aspx)

[继承（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173149.aspx)

[抽象类、密封类及类成员（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173150.aspx)

[params（C# 参考）](https://msdn.microsoft.com/zh-CN/library/w5zay9db.aspx)

[return（C# 参考）](https://msdn.microsoft.com/zh-CN/library/1h3swy84.aspx)

[out（C# 参考）](https://msdn.microsoft.com/zh-CN/library/t3c3bfhx.aspx)

[ref（C# 参考）](https://msdn.microsoft.com/zh-CN/library/14akc2c7.aspx)

[传递参数（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/0f66670z.aspx)