**使用委托（C# 编程指南）**

[委托](https://msdn.microsoft.com/zh-cn/library/900fyy8e.aspx)是安全封装方法的类型，类似于 C 和 C++ 中的函数指针。与 C 函数指针不同的是，委托是面向对象的、类型安全的和可靠的。委托的类型由委托的名称确定。以下示例声明名为 Del 的委托，该委托可以封装采用[字符串](https://msdn.microsoft.com/zh-cn/library/362314fe.aspx)作为参数并返回 [void](https://msdn.microsoft.com/zh-cn/library/yah0tteb.aspx) 的方法：

public delegate void Del(string message);

委托对象通常通过提供委托将封装的方法的名称或使用[匿名方法](https://msdn.microsoft.com/zh-cn/library/0yw3tz5k.aspx)构造。对委托进行实例化后，委托会将对其进行的方法调用传递到该方法。调用方传递到委托的参数将传递到该方法，并且委托会将方法的返回值（如果有）返回到调用方。这被称为调用委托。实例化的委托可以按封装的方法本身进行调用。例如：

// Create a method for a delegate.

public static void DelegateMethod(string message)

{

System.Console.WriteLine(message);

}

// Instantiate the delegate.

Del handler = DelegateMethod;

// Call the delegate.

handler("Hello World");

委托类型派生自 .NET Framework 中的 [Delegate](https://msdn.microsoft.com/zh-cn/library/system.delegate.aspx) 类。委托类型是[封装的](https://msdn.microsoft.com/zh-cn/library/88c54tsw.aspx)，它们不能派生出其他类，也不能从 [Delegate](https://msdn.microsoft.com/zh-cn/library/system.delegate.aspx) 派生出自定义类。由于实例化的委托是一个对象，因此可以作为参数传递或分配给一个属性。这允许方法作为参数接受委托并在稍后调用委托。这被称为异步回调，是在长进程完成时通知调用方的常用方法。当以这种方式使用委托时，使用委托的代码不需要知道要使用的实现方法。功能类似于封装接口提供的功能。

回调的另一个常见用途是定义自定义比较方法并将该委托传递到短方法。它允许调用方的代码成为排序算法的一部分。以下示例方法使用 Del 类型作为参数：

public void MethodWithCallback(int param1, int param2, Del callback)

{

callback("The number is: " + (param1 + param2).ToString());

}

然后，你可以将上面创建的委托传递到该方法：

MethodWithCallback(1, 2, handler);

并将以下输出接收到控制台：

The number is: 3

以抽象方式使用委托时，MethodWithCallback 不需要直接调用控制台，记住，其不必设计为具有控制台。 MethodWithCallback 的作用是简单准备字符串并将字符串传递到其他方法。由于委托的方法可以使用任意数量的参数，此功能特别强大。

当委托构造为封装实例方法时，委托将同时引用实例和方法。委托不知道除其所封装方法以外的实例类型，因此委托可以引用任何类型的对象，只要该对象上有与委托签名匹配的方法。当委托构造为封装静态方法时，委托仅引用方法。请考虑以下声明：

public class MethodClass

{

public void Method1(string message) { }

public void Method2(string message) { }

}

加上之前显示的静态 DelegateMethod，我们现在已有三个 Del 实例可以封装的方法。

调用时，委托可以调用多个方法。这被称为多播。若要向委托的方法列表（调用列表）添加其他方法，只需使用加法运算符或加法赋值运算符（“+”或“+=”）添加两个委托。例如：

MethodClass obj = new MethodClass();

Del d1 = obj.Method1;

Del d2 = obj.Method2;

Del d3 = DelegateMethod;

//Both types of assignment are valid.

Del allMethodsDelegate = d1 + d2;

allMethodsDelegate += d3;

此时，allMethodsDelegate 的调用列表中包含三个方法，分别为 Method1、Method2 和 DelegateMethod。原有的三个委托（d1、d2 和 d3）保持不变。调用 allMethodsDelegate 时，将按顺序调用所有三个方法。如果委托使用引用参数，引用将按相反的顺序传递到所有这三个方法，并且一种方法进行的任何更改都将在另一种方法上见到。当方法引发未在方法内捕获到的异常时，该异常将传递到委托的调用方，并且不会调用调用列表中的后续方法。如果委托具有返回值和/或输出参数，它将返回上次调用方法的返回值和参数。若要删除调用列表中的方法，请使用减法运算符或减法赋值运算符（“+”或“+=”）。例如：

//remove Method1

allMethodsDelegate -= d1;

// copy AllMethodsDelegate while removing d2

Del oneMethodDelegate = allMethodsDelegate - d2;

由于委托类型派生自 **System.Delegate**，因此可以在委托上调用该类定义的方法和属性。例如，若要查询委托调用列表中方法的数量，你可以编写：

int invocationCount = d1.GetInvocationList().GetLength(0);

调用列表中具有多个方法的委托派生自 [MulticastDelegate](https://msdn.microsoft.com/zh-cn/library/system.multicastdelegate.aspx)，该类属于 **System.Delegate** 的子类。由于这两个类都支持 **GetInvocationList**，因此在其他情况下，上述代码也将产生作用。

多播委托广泛用于事件处理中。事件源对象将事件通知发送到已注册接收该事件的接收方对象。若要注册一个事件，接收方需要创建用于处理该事件的方法，然后为该方法创建委托并将委托传递到事件源。事件发生时，源调用委托。然后，委托将对接收方调用事件处理方法，从而提供事件数据。给定事件的委托类型由事件源确定。有关详细信息，请参阅[事件（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/awbftdfh.aspx)。

在编译时比较分配的两个不同类型的委托将导致编译错误。如果委托实例是静态的 **System.Delegate** 类型，则允许比较，但在运行时将返回 false。例如：

delegate void Delegate1();

delegate void Delegate2();

static void method(Delegate1 d, Delegate2 e, System.Delegate f)

{

// Compile-time error.

//Console.WriteLine(d == e);

// OK at compile-time. False if the run-time type of f

// is not the same as that of d.

System.Console.WriteLine(d == f);

}

**请参阅**

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

[委托（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173171.aspx)

[在委托中使用变体（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-cn/library/ms173174.aspx)

[委托中的变体（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-cn/library/dd233060.aspx)

[对 Func 和 Action 泛型委托使用变体（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-cn/library/dd465122.aspx)

[事件（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/awbftdfh.aspx)