[delegate](https://msdn.microsoft.com/zh-cn/library/900fyy8e.aspx) 是表示对具有特定参数列表和返回类型的方法的引用的类型。在实例化委托时，你可以将其实例与任何具有兼容签名和返回类型的方法相关联。你可以通过委托实例调用方法。

委托用于将方法作为参数传递给其他方法。事件处理程序就是通过委托调用的方法。你可以创建一个自定义方法，当发生特定事件时，某个类（如 Windows 控件）就可以调用你的方法。下面的示例演示了一个委托声明：

public delegate int PerformCalculation(int x, int y);

可将任何可访问类或结构中与委托类型匹配的任何方法分配给委托。该方法可以是静态方法，也可以是实例方法。这样便能通过编程方式来更改方法调用，还可以向现有类中插入新代码。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 在方法重载的上下文中，方法的签名不包括返回值。但在委托的上下文中，签名包括返回值。换句话说，方法和委托必须具有相同的返回类型。 |

将方法作为参数进行引用的能力使委托成为定义回调方法的理想选择。例如，对比较两个对象的方法的引用可以作为参数传递到排序算法中。由于比较代码在一个单独的过程中，因此可通过更常见的方式编写排序算法。

## 委托概述

委托具有以下属性：

* 委托类似于 C++ 函数指针，但它们是类型安全的。
* 委托允许将方法作为参数进行传递。
* 委托可用于定义回调方法。
* 委托可以链接在一起；例如，可以对一个事件调用多个方法。
* 方法不必与委托类型完全匹配。有关详细信息，请参阅[在委托中使用变体（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-cn/library/ms173174.aspx)。
* C# 2.0 版引入了[匿名方法](https://msdn.microsoft.com/zh-cn/library/0yw3tz5k.aspx)的概念，此类方法允许将代码块作为参数传递来代替单独定义的方法。C# 3.0 引入了 Lambda 表达式，利用它们可以更简练地编写内联代码块。匿名方法和 Lambda 表达式（在某些上下文中）都可编译为委托类型。这些功能现在统称为匿名函数。有关 lambda 表达式的更多信息，请参见 [匿名函数（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb882516.aspx)。

## 本节内容

* [使用委托（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173172.aspx)
* [When to Use Delegates Instead of Interfaces (C# Programming Guide)](https://msdn.microsoft.com/zh-cn/library/ms173173.aspx)
* [带有命名方法的委托与带有匿名方法的委托（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/98dc08ac.aspx)
* [匿名方法（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/0yw3tz5k.aspx)
* [在委托中使用变体（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-cn/library/ms173174.aspx)
* [如何：合并委托（多路广播委托）（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173175.aspx)
* [如何：声明、实例化和使用委托（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173176.aspx)

## 参考文档1:

**委托是什么？**

委托首先是一个数据类型，我们可以像[定义结构体](https://so.csdn.net/so/search?q=%E5%AE%9A%E4%B9%89%E7%BB%93%E6%9E%84%E4%BD%93&spm=1001.2101.3001.7020)一样定义一个委托类型，委托（Delegate） 是存有对某个方法的引用的一种引用类型变量。引用可在运行时被改变

委托的定义

//语法

delegate <return type> <delegate-name> <parameter list>

//实例

public delegate int MyDelegate (string s);

**实例化委托（**[Delegate](https://so.csdn.net/so/search?q=Delegate&spm=1001.2101.3001.7020)**）**

例如：

public delegate void printString(string s);

...

printString ps1 = new printString(WriteToScreen);

printString ps2 = new printString(WriteToFile);

**说白了就是类似指向函数方法的指针，我们可以类似定义指针类型一样，定义委托的类型**

在定义委托类型时要指定函数的参数类型和返回值，一旦声明了委托类型，委托对象必须使用 new 关键字来创建，且与一个特定的方法有关。当创建委托时，传递到 new 语句的参数就像方法调用一样书写，但是不带有参数。

**简单用法实例**

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl

{

class TestDelegate

{

static int num = 10;

public static int AddNum(int p)

{

num += p;

return num;

}

public static int MultNum(int q)

{

num \*= q;

return num;

}

public static int getNum()

{

return num;

}

static void Main(string[] args)

{

// 创建委托实例

NumberChanger nc1 = new NumberChanger(AddNum);

NumberChanger nc2 = new NumberChanger(MultNum);

// 使用委托对象调用方法

nc1(25);

Console.WriteLine("Value of Num: {0}", getNum());

nc2(5);

Console.WriteLine("Value of Num: {0}", getNum());

Console.ReadKey();

}

}

}

**这只是简单的用法介绍，可以看出 这和在主程序里直接调用方法没啥区别，并不是实际的用法。**

**委托的用途**

**多播**

委托对象可使用 “+” 运算符进行合并。一个合并委托调用它所合并的两个委托。只有相同类型的委托可被合并。"-" 运算符可用于从合并的委托中移除组件委托。

使用委托的这个有用的特点，您可以创建一个委托被调用时要调用的方法的调用列表。这被称为委托的 多播（multicasting），也叫组播。下面的程序演示了委托的多播

static void Main(string[] args)

{

// 创建委托实例

NumberChanger nc;

NumberChanger nc1 = new NumberChanger(AddNum);

NumberChanger nc2 = new NumberChanger(MultNum);

nc = nc1;

nc += nc2;

// 调用多播

nc(5);

Console.WriteLine("Value of Num: {0}", getNum());

Console.ReadKey();

}

}

}

**关键： 委托的使用场景**

委托，故名思意，咱们用一个形象的比喻，公司里 ，老板/调用者 ，可以不用委托我/激活委托函数，亲自去给张三/func1,李四/func2 安排工作,就是直接调用，这时或许要准备调用的参数和调用方式的代码 比如循环调用，  
老板/调用者 还可以通过我/激活委托的函数 去安排张三李四的工作，同时也可以指定张三/func1,李四/func2 一起工作/**±多播**，也可以拓展新的员工/方法。  
和类的继承多态有异曲同工之妙，不过更加灵活，我/激活委托函数 不需要知道自己要调用什么函数以及具体实现，只管用调用者给的参数就行了 ，父类不用管子类如何实现的，通过子类型的赋值，就能调用子类重写的方法。

//委托

// 委托声明

public delegate void doSomething(string s);

//定义 回调函数 ，公司里的员工 张三，李四.............

public static void ZangSan(string a) {

Console.WriteLine("zhangsan"+a);

}

public static void LiSi(string a)

{

Console.WriteLine("lisi"+a);

}

/// <summary>

/// 中介，委托处理 ,使用此方法触发委托回调

/// </summary>

/// <param name="from">开始</param>

/// <param name="to">结束</param>

/// <param name="fb">委托引用</param>

private static void handleDelegate(int from, int to, doSomething ds)

{

//一些预处理 ，这样使用委托就比直接调用方便了

for (int val = from; val <= to; val++)

{

if (ds != null)

{

ds(val.ToString());

}

//ds?.Invoke(val); 简化版本调用

}

}

//主函数，调用者 老板

public void testDelegate()

{ //老板通过我/中介 给张三安排工作，

doSomething dsZhangSan = new doSomething(ZangSan);

//给张三李四 按顺序安排工作 //按序执行

doSomething dsLisi = new doSomething(LiSi);

doSomething ds = dsZhangSan + dsLisi;

//在调用的时可以给张三,李四传参数，这里在handleDelegate里面给了 ds(val.ToString());

handleDelegate(0, 3, ds);

//方法的返回值，参数必须完全一样，符合委托类型的定义

}

**执行结构**



**回调函数的机制**

把我想象成是独立的提供服务的中介，而老板/函数实现的一方， 张三/func1,李四/func2 ~~~是一个公司的，合作时，老板指定张三或是李四… , 而我不需要关心，我只是在特点条件发生时去调用张三李四就行了

⑴定义一个回调函数；

⑵提供函数实现的一方在初始化的时候，将回调函数的函数指针注册给调用者；

⑶当特定的事件或条件发生的时候，调用者使用函数指针调用回调函数对事件进行处理。

# 参考文档2.

## 1. 委托是什么？

其实，我一直思考如何讲解委托，才能把委托说得更透彻。说实话，每个人都委托都有不同的见解，因为看问题的角度不同。个人认为，可以从以下2点来理解：

 （1） 从数据结构来讲，委托是和类一样是一种用户自定义**类型**。

 （2） 从设计模式来讲，委托（类）提供了**方法**（对象）的抽象。

既然委托是一种类型，那么它存储的是什么数据？

我们知道，委托是方法的抽象，它存储的就是一系列具有相同签名和返回回类型的方法的地址。调用委托的时候，委托包含的所有方法将被执行。

## 2. 委托类型的定义

委托是类型，就好像类是类型一样。与类一样，委托类型必须在被用来创建变量以及类型对象之前声明。

delegate void MyDel(int x);

委托类型声明：

（1） 以deleagate关键字开头。

（2）返回类型+委托类型名+参数列表。

## 3. 声明委托变量

MyDel del1,del2;

## 4. 初始化委托变量

(1) 使用new运算符

new运算符的操作数的组成如下：

* 委托类型名
* 一组圆括号，其中包含作为调用列表中的第一个成员的方法的名字。方法可以是实例方法或静态方法。

del1 = new MyDel( myInstObj.MyM1 );

del2 = new MyDel( SClass.OtherM2 );

(2)使用快捷语法

快键语法，它仅由方法说明符构成。之所以能这样，是因为在方法名称和其相应的委托类型之间有隐式转换。

del1 = myInstObj.MyM1;

del2 = SClass.OtherM2;

## 5. 赋值委托

 由于委托是引用类型，我们可以通过给它赋值来改变包含在委托变量中的方法地址引用。旧的引用会被垃圾回收器回收。

MyDel del;

del = myInstaObj.MyM1; //委托初始化

del = SClass.OtherM2;//委托重新赋值，旧的引用将被回收

## 6. 组合委托

 委托可以使用额外的运算符来组合。这个运算最终会创建一个新的委托，其调用列表是两个操作数的委托调用列表的副本的连接。

委托是恒定的，操作数委托创建后不会被改变。**委托组合拷贝的是操作数的副本**。

MyDel del1 = myObj.MyMethod;

MyDel del2 = SClass.OtherM2;

MyDel del3 = del1 + del2; //组合调用列表

## 7. 委托加减运算

可以使用+=运算符，为委托新增方法。

同样可以使用-=运算符，为委托移除方法。

MyDel del = myObj.MyMethod;

del += SClass.OtherM2; // 增加方法

del -= myObj.MyMethod; // 移除方法

## 8. 委托调用

委托调用跟方法调用类似。委托调用后，调用列表的每个方法将会被执行。

**在调用委托前，应判断委托是否为空**。调用空委托会抛出异常。

if(null != del)

{

del();//委托调用

}

## 9. 匿名方法

匿名方法是在初始化委托时内联声明的方法。

基本结构：

deleage( 参数 ) { 语句块 }

例如：

delegate int MyDel (int x); //定义一个委托   
  
MyDel del = delegate( int x){ return x; };

从上面我们可以看到，**匿名方法是不会显示声明返回值的**。

## 10. Lambda表达式

Lambda表达式主要用来简化匿名方法的语法。在匿名方法中，delegate关键字有点多余，因为编译器已经知道我们将方法赋值给委托。通过几个简单步骤，我们就可以将匿名方法转换为Lambda表达式：

* 删除delegate关键字
* 在参数列表和匿名方法主体之间防Lambda运算符=>。Lambda运算符读作"goes to"。

MyDel del = delegate( int x) { return x; };//匿名方法

MyDel del2 = (int x) => {return x;};//Lambda表达式

MyDel del3 = x => {return x};//简写的Lambda表达式