C# 委托（Delegate）

C# 中的委托（Delegate）类似于 C 或 C++ 中函数的指针。**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（Delegate）特别用于实现事件和回调方法。所有的委托（Delegate）都派生自 System.Delegate 类。

声明委托（Delegate）

委托声明决定了可由该委托引用的方法。委托可指向一个与其具有相同标签的方法。

例如，假设有一个委托：

```
public delegate int MyDelegate (string s);
```

上面的委托可被用于引用任何一个带有一个单一的 *string* 参数的方法，并返回一个 *int* 类型变量。

声明委托的语法如下：

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

```
using System;
public class Skill
{
    public void DabaoJian()
    {
        Console.WriteLine("大宝剑");
    }
    public void NaoCanPi()
    {
        Console.WriteLine("脑残pi");
    }
    public void Shenpan()
    {
        Console.WriteLine("审判");
    }
}
public delegate void SkillDelegate();
class Hero
{

    public SkillDelegate Q;
}
```

```
public delegate int CalulateMethod(int a, int b);
namespace 委托回调
{

    class Program
    {
        public static int Plus(int a, int b)
        {
            return a + b;
        }
        public static int Minus(int a, int b)
        {
            return a - b;
        }
        static void Main(string[] args)
        {
            //声明一个委托变量 并赋值
            //CalulateMethod method = new CalulateMethod(Plus);
            //
            //让用户从控制台输入两个数字，分别赋值给x和y
            //需求
            //如果 x>y  那么结果是x-y
            //如果 x<y 那么结果是x+y

            CalulateMethod method;
            int x = Convert.ToInt32(Console.ReadLine());
            int y = Convert.ToInt32(Console.ReadLine());

            if (x<y)
            {
                method = Minus;
            }
            else
            {
                method = Plus;
            }
           int a= method(x, y);
            //使用

            Console.WriteLine("a====" + a);
```

```
            Skill skill = new Skill();
            Hero gaylun = new Hero();
            gaylun.Q = skill.DabaoJian;

            Hero manwang = new Hero();
            manwang.Q = skill.NaoCanPi;

            Hero zhaoxin = new Hero();
            zhaoxin.Q = skill.Shenpan;

            gaylun.Q();
            manwang.Q();
            zhaoxin.Q();

        }
    }
}
```

///委托的组合

连着技能

```
public class Test
{
    public void MethodA()
    {
        Console.WriteLine("MethodA");
    }
    public void MethodB()
    {
        Console.WriteLine("MethodB");
    }
    public delegate void TestDelegate();
 public void  TTTT()
    {
        TestDelegate test = MethodA;

        test += MethodB;
        test -= MethodA;
    }
```

}

实例化委托（Delegate）

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。例如：

```
public delegate void printString(string s);
...
printString ps1 = new printString(WriteToScreen);
printString ps2 = new printString(WriteToFile);
```

下面的实例演示了委托的声明、实例化和使用，该委托可用于引用带有一个整型参数的方法，并返回一个整型值。

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();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Value of Num: 35
Value of Num: 175
```

委托的多播（Multicasting of a Delegate）

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：

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 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();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Value of Num: 75
```

委托（Delegate）的用途

下面的实例演示了委托的用法。委托 *printString* 可用于引用带有一个字符串作为输入的方法，并不返回任何东西。

我们使用这个委托来调用两个方法，第一个把字符串打印到控制台，第二个把字符串打印到文件：

实例

```
using System;
using System.IO;

namespace DelegateAppl
{
```

```
    class PrintString
    {
        static FileStream fs;
        static StreamWriter sw;
        // 委托声明
        public delegate void printString(string s);

        // 该方法打印到控制台
        public static void WriteToScreen(string str)
        {
            Console.WriteLine("The String is: {0}", str);
        }
        // 该方法打印到文件
        public static void WriteToFile(string s)
        {
            fs = new FileStream("c:\\message.txt", FileMode.Append,
FileAccess.Write);
            sw = new StreamWriter(fs);
            sw.WriteLine(s);
            sw.Flush();
            sw.Close();
            fs.Close();
        }
        // 该方法把委托作为参数，并使用它调用方法
        public static void sendString(printString ps)
        {
            ps("Hello World");
        }
        static void Main(string[] args)
        {
            printString ps1 = new printString(WriteToScreen);
            printString ps2 = new printString(WriteToFile);
            sendString(ps1);
            sendString(ps2);
            Console.ReadKey();
        }
    }
```

```
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
The String is: Hello World
```