Делегаты

- Конструкторы
- Деструкторы
- Константы
- Поля
- Методы
- Свойства
- Индексаторы
- Операторы
- События
- Делегаты
- Классы
- Интерфейсы
- Структуры

Делегат — эти объект, который безопасно инкапсулирует метод, его действие схоже с указателем функции в C и C++. Делегаты используются для передачи методов в качестве аргументов. Все делегаты являются объектами типа System.Delegate. Поскольку созданный экземпляр делегата является объектом, его можно передавать как параметр или назначать свойству.

Делегаты имеют следующие свойства.

- Делегаты похожи на указатели функций в C++, но являются типобезопасными.
- Делегаты позволяют производить передачу методов подобно обычным параметрам.
- Делегаты можно использовать для задания функций обратного вызова.
- Делегаты можно связывать друг с другом; например, несколько методов можно вызвать по одному событию

В отличие от указателей функций в C делегаты объектно-ориентированы, строго типизированы и безопасны.

Объявление типа делегата аналогично сигнатуре метода. Оно имеет возвращаемое значение и некоторое число параметров какого-либо типа.

В следующем примере объявляется делегат с именем **Del**, который может инкапсулировать метод, использующий в качестве аргумента значение **string** и возвращающий значение **void**:

```
public delegate void Del(string message);
```

Ключевое слово **delegate** используется для объявления ссылочного типа, который может быть использован для инкапсуляции именованного или анонимного метода.

Объект делегата обычно создается указанием имени метода, для которого делегат будет служить оболочкой, или с помощью анонимного метода.

```
public static void DelegateMethod(string message)
{
    System.Console.WriteLine(message);
}
```

После создания экземпляра делегата вызов метода, выполненный в делегате, передается делегатом в этот метод. Параметры, передаваемые делегату вызывающим объектом, передаются в метод, а возвращаемое методом значение (при его наличии) возвращается делегатом в вызывающий объект.

Эта процедура называется вызовом делегата. Делегат, для которого создан экземпляр, можно вызвать, как если бы это был метод, для которого создается оболочка. Например:

```
Del handler = DelegateMethod; // Instantiate the delegate.

handler("Hello World");       // Call the delegate.
```

Все методы в среде .NET можно разделить на две группы: статические (**static**) и экземплярные (**instance**).

Если делегат ссылается на **статический** метод, то все действительно просто. Так как в этом случае есть вся необходимая для вызова метода информация: адрес метода и параметры.

Если же делегат ссылается на экземплярный метод, то задача усложняется. Для экземплярного метода каждый документ получает частную копию данных.

Чтобы вызвать экземплярный метод, делегату необходимо знать ссылку на объект, к которому привязан данный конкретный метод. Оказывается, что эта ссылка хранится в самом объекте делегата и указывается при его создании.

На протяжении всей жизни объекта делегата данная ссылка не изменяет своего значения, она всегда постоянна и может быть задана только при его создании.

Таким образом, вне зависимости от того, ссылается ли делегат на статическую функцию или на экземплярный метод, обращение к нему извне ничем отличаться не будет.

Всю необходимую функциональность обеспечивает сам делегат, вкупе со средой исполнения. Это очень удобно, поскольку множество разных делегатов можно привязывать к одному событию.

В следующем примере один делегат сопоставлен как со статическим методом, так и с методом экземпляра и возвращает из каждого метода определенные сведения.

```
delegate void Del(); // Declare a delegate

class SampleClass
{
    public void InstanceMethod()
       { System.Console.WriteLine("A message from the instance method."); }
    static public void StaticMethod()
       { System.Console.WriteLine("A message from the static method.");   }
}

class TestSampleClass
{
  static void Main()
  {
     SampleClass sc = new SampleClass();
     Del d = sc.InstanceMethod;
     d();                                  // Map the delegate to the instance method:
     d = SampleClass.StaticMethod;         // Map to the static method:
     d();
   }
}

/* Output:
 A message from the instance method.
 A message from the static method.
*/
```

Метод, который передается как параметр делегата, должен иметь ***такую же сигнатуру***, как и объявление делегата.

Обратите внимание на то что в следующем примере, делегат **Del**, так и сопоставленный метод **MultiplyNumbers** имеют одинаковую сигнатуру

```
delegate void Del(int i, double j); // Declare a delegate


class MathClass
{
   static void Main()
    {
      MathClass m = new MathClass();
      Del d = m.MultiplyNumbers; // Delegate instantiation using "MultiplyNumbers"
      System.Console.WriteLine("Invoking the delegate using 'MultiplyNumbers':");

      for (int i = 1; i <= 5; i++) { d(i, 2); }
      System.Console.WriteLine("Press any key to exit.");
      System.Console.ReadKey();
    }

    void MultiplyNumbers(int m, double n) // Declare the associated method.
    {
       System.Console.Write(m * n + " ");
    }
}
 /* Output:
      Invoking the delegate using 'MultiplyNumbers':
      2 4 6 8 10
*/
```

Это использование ***именованного метода***. Делегаты, созданные с помощью именованного метода, могут инкапсулировать **статический** метод или метод экземпляра.

Именованные методы являются единственным способом создания экземпляра делегата в ранних версиях C#. Анонимные методы были представлены в C# 2.0, а в версиях C# 3.0 и более поздних лямбда-выражения заменяют эти методы и являются предпочтительным способом написания встроенного кода.

Однако в ситуациях, когда создание нового метода является нежелательным, C# позволяет создать экземпляр делегата и сразу же указать блок кода, обрабатываемый делегатом при вызове.

Анонимные методы позволяют отказаться от использования списка параметров. Это означает, что ***анонимный метод может быть преобразован в делегаты с различными сигнатурами***. Это невозможно в ситуации с лямбда-выражениями.

Блок может содержать либо ***лямбда-выражение***, либо ***анонимный метод***.

Создание анонимных методов является, по существу, способом передачи блока кода в качестве параметра делегата.
Два примера:

```
// Create a handler for a click event.

button1.Click += delegate(System.Object o, System.EventArgs e)
                    { System.Windows.Forms.MessageBox.Show("Click!"); };
```

```
// Create a delegate.
delegate void Del(int x);

// Instantiate the delegate using an anonymous method.
Del d = delegate(int k) { /* ... */ };
```

Использование анонимных методов позволяет сократить издержки на кодирование при создании делегатов, поскольку не требуется создавать отдельный метод.

Лямбда-выражение — это анонимная функция, которую можно использовать для создания таких типов как делегаты или деревья выражений.

С помощью лямбда-выражений можно написать локальные функции, которые затем можно передавать в другие функции в качестве аргументов или возвращать из них в качестве значения. Лямбда-выражения особенно полезны для составления выражений запросов LINQ.

В приложениях часто используются данные из баз данных SQL или XML-документов. Традиционно, разработчики должны были изучить как основной язык программирования, такой как C#, так и дополнительный, например SQL или XQuery.

LINQ (Language-Integrated Query) предоставляет возможность осуществлять запросы на самом языке C#. Теперь, вместо изучения отдельного языка запросов, можно выполнять запросы к базам данных SQL, наборам данных ADO.NET, XML-документам и любым классам коллекций .NET Framework, реализующих интерфейс IEnumerable, используя знание C# и нескольких дополнительных ключевых слов и основных понятий.

Для создания лямбда-выражения можно определить входные параметры (если таковые имеются) слева лямбда-оператора => и поместить блок выражений или выписки на другую сторону.

Например, лямбда-выражение x => x * x принимает параметр с именем x и возвращает значение x, возведённое в квадрат. Это выражение можно присвоить типу делегата, как показано в следующем примере.

Для использования с анонимными методами и lamda-выражениями делегат и код, который должен быть связан с ним, должны быть объявлены вместе.

```
delegate double MathAction(double num); // Определение delegate – задается сигнатура:

class DelegateTest
{
   static double Double(double input)  // Метод с той же сигнатурой
          { return input * 2; }

    static void Main()
    {
     MathAction ma = Double;          // Создание delegate
     double multByTwo = ma(4.5);      // Вызов delegate ma:
     Console.WriteLine("multByTwo: {0}", multByTwo);

     MathAction ma2 = delegate(double input) // Создание delegate с анонимным методом:
               { return input * input; };
     double square = ma2(5);
     Console.WriteLine("square: {0}", square);

     MathAction ma3 = s => s * s * s; // Создание delegate с помощью lambda выражения
     double cube = ma3(4.375);
     Console.WriteLine("cube: {0}", cube);
     }

 // Output:
 // multByTwo: 9
 // square: 25
 // cube: 83.740234375
 }
```

Выражения-лямбды

Лямбда-выражение, правая часть которого представляет собой выражение, называется лямбдой выражения или выражением-лямбдой.

Выражения-лямбды возвращают результат выражения и имеют следующую основную форму:

```
(input parameters) => expression
```

Если лямбда имеет только один входной параметр, скобки можно не ставить, во всех остальных случаях они обязательны. Если входных параметров два и более, то они разделяются запятыми и заключаются в скобки:

```
(x, y) => x == y
```

Иногда компилятору бывает трудно или даже невозможно вывести типы входных параметров. В этом случае типы можно указать в явном виде, как показано в следующем примере:

```
(int x, string s) => s.Length > x
```

Отсутствие входных параметров задаётся пустыми скобками.

```
() => SomeMethod()
```

Обратите внимание на предыдущий пример: тело выражения-лямбды может состоять из вызова метода.

Операторная лямбда

Лямбда операторов (или операторная лямбда) напоминает выражение-лямбду, за исключением того, что оператор (или операторы) заключается в фигурные скобки.

```
(input parameters) => {statement;}
```

Тело лямбды операторов может состоять из любого количества операторов; однако на практике обычно используется не больше двух-трёх.

```
delegate void TestDelegate(string s);
 …
TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); };
myDel("Hello");
```

Лямбда-выражения могут ссылаться на внешние переменные, попадающие в область действия метода или типа, в котором определено это лямбда-выражение.

Переменные, захваченные таким способом, сохраняются для использования в лямбда-выражениях даже в том случае, если эти переменные иначе попадают за границы области действия и уничтожаются сборщиком мусора.

Внешней переменной должно быть присвоено определённое значение, прежде чем она сможет использоваться в лямбда-выражениях.

```
delegate bool D();
delegate bool D2(int i);

class Test
{
   D del;
   D2 del2;
   public void TestMethod(int input)
   {
      int j = 0;

      del = () => { j = 10; return j > input; };
      del2 = (x) => {return x == j; };

      Console.WriteLine("j = {0}", j);

      bool boolResult = del(); // Invoke the delegate.

      Console.WriteLine("j = {0}. b = {1}", j, boolResult); }
      static void Main()
       {
         Test test = new Test();
         test.TestMethod(5);
         bool result = test.del2(10);
         Console.WriteLine(result); // Output: True

         Console.ReadKey();
        }
}
```

Делегаты являются основой событий.