Делегаты :: часть 22

**Тэги:** [делегаты C#](http://blog.nguen.net/index.php?mod=search&db=no&stroka=%E4%E5%EB%E5%E3%E0%F2%FB+C%23), [delegte](http://blog.nguen.net/index.php?mod=search&db=no&stroka=delegate),

**»** 10 ноября 2007 года

Данная статья будет посвящена делегатам. Делегатами пользуются достаточно редко, но все же это достаточно интересная возможность, чтобы с ней познакомиться, тем более что не во всех языках программирования она есть.

Что же такое делегат? Делегат - это специализированный класс, производный от класса**System.MulticastDelegate**, либо от **System.Delegate**. По сути, делегат - это ссылка на метод. Отличие **System.MulticastDelegate** от **System.Delegate** в том, что первый содержит несколько ссылок на методы, а второй всего одну. Обычно используют класс **System.MulticastDelegate**, так как, по сути, он более функциональный. В **C++** есть подобная возможность, но там используются обычные указатели, что не безопасно. Делегат - это более удобное средство.

Зачем они нужны? В основном их используют для того, чтобы делать более гибкие методы и для того, чтобы разделить работу между классами. Надо заметить, что использование делегатов достаточно усложняет код, из-за чего их часто боятся.

С помощью делегатов мы можем передавать одну функцию в качестве параметра другой функции. Использовать это можно по-разному, например: представим, что у нас есть функция сортирующая массив любых объектов. Для того чтобы отсортировать объекты, нужно уметь их сравнивать. Т.е. необходимо в каждом объекте реализовывать интерфейс **IComparable**, либо просто передавать в нашу функцию сортировки еще одну функцию, которая будет сравнивать объекты. В принципе одно и то же, но в первом случае нам придется изменять класс, а это может быть достаточно неудобно, если вообще возможно, поэтому использование делегатов в данном случае является вполне оправданным решением.

Конечно же, без делегатов можно обойтись, но в некоторых случаях они способны облегчить вам жизнь.

Для того чтобы создать делегат используется ключевое слово **delegate**. Создать класс наследник от**System.MulticastDelegate** или от **System.Delegate** не получится.

Давайте разберемся, как работают делегаты на практике. Допустим, что у нас есть класс **Money**(деньги), у данного класса есть метод **Send** (перевод). Предположим, что все банки осуществляют перевод только в определенной валюте, поэтому необходимо перед переводом перевести деньги из одной валюты в другую. Мы можем делать всю работу прямо в методе **Send**, а можем использовать одну из функций: **rub2usd** (из рублей в доллары), **usd2rub** (из долларов в рубли). Таких функций может быть много, но для примера нам хватит двух. Это выглядит более логично, т.к. мы разграничиваем две разные операции (перевод денег, и покупка/продажа валюты).

Итак, осуществим задуманное. Начнем с класса **Money**:

class Money

{

public double count; //кол-во денег

public string type; //вид валюты

public Money(double m)

{

count = m;

type = "rub";

}

public delegate void MoneyDelegate(Money m1);

}

Класс весьма примитивный, но обратите внимание на последнюю запись. Мы создаем делегат**MoneyDelegate**, который будет ссылаться на функцию, принимающую в качестве параметра объект типа **Money**. Т.е. мы заранее прописываем прототип функции. Разумеется, что подобных функций может быть огромное количество. В нашем примере таких функций будет 2: **rub2usd** и **usd2rub**.

А сейчас напишем класс **MoneyOperation**, в котором будут содержаться методы для работы с деньгами:

class MoneyOperation

{

public void rub2usd(Money m1)

{

m1.type = "usd";

m1.count /= 24;

}

public void usd2rub(Money m1)

{

m1.type = "rub";

m1.count \*= 24;

}

public void send(Money m1, string bank, Money.MoneyDelegate proc)

{

if (proc.Target != null)

{

proc(m1);

}

Console.WriteLine(m1.count + " " + m1.type + " успешно переведены в " + bank);

}

}

Давайте разбираться. У нас есть 2 почти одинаковых метода: **usd2rub** и **rub2usd**. Думаю, что назначение этих методов понятно. Разумеется, что жестко прописывать курсы валют это ошибка, но сейчас для нас это не важно, это лишь пример. Сейчас для нас наибольший интерес представляет функция **send**. Данная функция принимает в качестве аргументов 3 параметра: объект **Money** (деньги которые нужно перевести), строковую переменную **bank** (название банка) и делегат типа**Money.MoneyDelegate**. Напомню, что делегат - это, по сути, ссылка на метод. В данный момент мы не знаем, на какой именно метод будет ссылаться делегат. Что же происходит внутри метода **Send**? Сперва мы проверяем, не пустой ли делегат. Делегат - это объект и у него есть ряд полезных методов и свойств, target - как раз такое свойство. Если делегат не указывает ни на какой метод, то **target**равен **null**, в противном случае он содержит имя класса. Если делегат не пустой, то мы вызываем связанный с ним метод. Напомню, что в качестве аргумента передается один параметр, это было прописано при объявлении делегата.

Настало время проверить, как это все работает. Для примера напишем простую программу:

static void Main(string[] args)

{

Money m = new Money(456);

MoneyOperation oper = new MoneyOperation();

Money.MoneyDelegate del = new Money.MoneyDelegate(oper.rub2usd);

oper.send(m, "Сбербанк", del);

Console.ReadLine();

}

Разберем все по полочкам. Сперва мы создаем экземпляр класса **Money**. С помощью конструктора "перечисляем" 456 рублей на счет. После этого создаем объект класса **MoneyOperation**. А дальше идет самое интересное, мы создаем делегат, который ссылается на метод **rub2usd** объекта **oper**. Ну а дальше мы переводим наши денежки в "Сбербанк", при этом переведя их в доллары. если бы мы вместо**rub2usd** написали **usd2rub**, то в программе возникла бы логическая ошибка, т.к. мы не сделали необходимых проверок.

Если мы захотим узнать на какой метод ссылается делегат, то можем воспользоваться свойством**Method**.

Напомню, что делегат может ссылаться на несколько методов, которые будут вызываться по цепочке. Это делает делегаты мощным инструментом, но на практике он применяется не часто.

Надеюсь, что вы поняли общий принцип работы делегатов. В следующей статье мы познакомимся с событиями. На этом все.

Хочешь изучить C#? Купи книгу, не жалей денег!  
  
[Основы ASP.NET с примерами на C#](http://www.books.ru/shop/books/143757?partner=ledworm)

Автор: Нгуен Павел (LedWorm)  
Блог автора: [nguen.net](http://nguen.net/)  
Оригинал статьи: <http://nguen.net/post208-delegate.html>