Delegates & its 5 Modern Flavors (Func, Action, Predicate, Converter, Comparison

Introduction

Delegate is a very powerful feature available in the .NET Framework. In this article, we will explore delegate & its new features which are being introduced in the framework. I will be using Visual Studio 2013 for coding sample code, assuming that you do have basic knowledge of .NET C# code.

I will be explaining the following flavors in this articles:

1. Func
2. Action
3. Predicate
4. Converter
5. Comparison

I would like to get feedback on this article. Please feel to share your comments or you can also email me at shettyashwin@outlook.com. If you like this article, please don't forget to rate it.

Background

As part of the Interview panel, I have been asked to interview a lot of candidates for the opening we had. At the time of interview when questions were asked on delegates & its features, very few candidates were able to answer this question. Even one of my team members was surprised when I optimized his code & started using action delegate. Hence, I decided to write this article which shares some details on delegates & some of its features.

What are Delegates?

If I can put it into simple words, Delegate is a pointer to a method. Delegate can be passed as a parameter to a method. We can change the method implementation dynamically at run-time, only thing we need to follow doing so would be to maintain the parameter type & return type.

For example: If I declare delegate with return type int & two parameters as string and int respectively, all the references which are set using this delegate should follow this signature.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

internal class Program

{

protected delegate **int** tempFunctionPointer(**string** strParameter, **int** intParamater);

public static void Main()

{

DelegateSample tempObj = new DelegateSample();

tempFunctionPointer funcPointer = tempObj.FirstTestFunction;

funcPointer("hello", 1);

Console.ReadKey();

funcPointer = tempObj.SecondTestFunction;

funcPointer("hello", 1);

Console.ReadKey();

}

}

public class Employee

{

public string Name { get; set; }

public int Age { get; set; }

}

public class XEmployee

{

public string Name { get; set; }

public int Age { get; set; }

public bool IsExEmployee {

get { return true; }

}

}

public class DelegateSample

{

public int FirstTestFunction(string strParameter, int intParamater)

{

Console.WriteLine("First Test Function Execution");

Console.WriteLine(strParameter);

return intParamater;

}

public int SecondTestFunction(string strParameter, int intParamater)

{

Console.WriteLine("Second Test Function Execution");

Console.WriteLine(strParameter);

return intParamater;

}

public void ThirdTestFunction(string strParameter, int intParamater)

{

Console.WriteLine("Third Test Function Execution");

Console.WriteLine(strParameter);

}

public bool FourthTestFunction(Employee employee)

{

return employee.Age < 27;

}

public XEmployee FifthTestFunction(Employee employee)

{

return new XEmployee() {Name = employee.Name, Age = employee.Age};

}

public int SixTestFunction(Employee strParameter1, Employee strParamater2)

{

return strParameter1.Name.CompareTo(strParamater2.Name);

}

}

Delegates can be executed synchronous or asynchronously. Code sample mentioned above is an example of synchronous processing. To make it an Asynchronous process, we need to use BeginInvoke method.

Hide   Copy Code

funcPointer.BeginInvoke("Hello", 1, null, null);

First two parameters are the inputs for the function. 3rd parameter can be set for getting call back after execution of process. Detailed explanation of making Asynchronous call is available [here.](http://msdn.microsoft.com/en-us/library/2e08f6yc(v=vs.110).aspx)

When Do I Use Delegate?

Looking at the sample above, a lot of us might think this can be also achieved using Interface or abstract class, then why do we need delegate.

Delegate can be used in the following scenarios:

1. If you don’t want to pass your interface or abstract class dependence to internal class or layers.
2. If the code doesn't need access to any other attributes or method of the class from which logic needs to be processed.
3. Event driven implementation needs to be done.

Different Flavors of Delegate

As .NET Framework evolved over a period of time, new flavors have been added to keep implementation simple & optimized. By default, you get all the features & functionality with flavors which you get with delegate. Let’s have a look at Func delegate.

Func<TParameter, TOutput>

Func is logically similar to base delegate implementation. The difference is in the way we declare. At the time of declaration, we need to provide the signature parameter & its return type.

Hide   Copy Code

Func<string, int, int> tempFuncPointer;

First two parameters are the method input parameters. 3rd parameter (always the last parameter) is the outparameter which should be the output return type of the method.

Hide   Copy Code

Func<string, int, int> tempFuncPointer = tempObj.FirstTestFunction;

int value = tempFuncPointer("hello", 3);

Console.ReadKey();

Func is always used when you have return object or type from method. If you have void method, you should be using Action.

Action<TParameter>

Action is used when we do not have any return type from method. Method with void signature is being used with Action delegate.

Hide   Copy Code

Action<string, int> tempActionPointer;

Similar to Func delegate, the first two parameters are the method input parameters. Since we do not have return object or type, all the parameters are considered as input parameters.

Hide   Copy Code

Action<string, int> tempActionPointer = tempObj.ThirdTestFunction;

tempActionPointer("hello", 4);

Console.ReadKey();

Predicate<in T>

Predicate is a function pointer for method which returns boolean value. They are commonly used for iterating a collection or to verify if the value does already exist. Declaration for the same looks like this:

Hide   Copy Code

Predicate<Employee> tempPredicatePointer;

For sample, I have created an Array which holds a list of Employees. Predicate is used to get employeebelow age of 27:

Hide   Copy Code

**Predicate<Employee> tempPredicatePointer** = tempObj.**FourthTestFunction**;

Employee[] lstEmployee = (new Employee[]

{

new Employee(){ Name = "Ashwin", Age = 31},

new Employee(){ Name = "Akil", Age = 25},

new Employee(){ Name = "Amit", Age = 28},

new Employee(){ Name = "Ajay", Age = 29},

});

Employee tempEmployee = Array.Find(lstEmployee, **tempPredicatePointer**);

Console.WriteLine("Person below 27 age :" + tempEmployee.Name);

Console.ReadKey();

<pre lang="cs">*//Code block which gets executed while iteration*

public bool **FourthTestFunction**(Employee employee)

{

return employee.Age < 27;

}

Converter<TInput, TOutput>

Convertor delegate is used when you need to migrate / convert one collection into another by using some algorithm. Object A gets converted into Object B.

Hide   Copy Code

Converter<Employee, XEmployee> tempConvertorPointer

= new Converter<Employee, XEmployee>(tempObj.FifthTestFunction);

For sample, I have created XEmployee entity. All the Employees in the collection get migrated to XEmployeeEntity.

Hide   Copy Code

Employee[] lstEmployee = (new Employee[]

{

new Employee(){ Name = "Ashwin", Age = 31},

new Employee(){ Name = "Akil", Age = 25},

new Employee(){ Name = "Amit", Age = 28},

new Employee(){ Name = "Ajay", Age = 29},

});

**Converter<Employee, XEmployee> tempConvertorPointer**

= new Converter<Employee, XEmployee>(**tempObj.FifthTestFunction**);

XEmployee[] xEmployee = Array.ConvertAll(lstEmployee, **tempConvertorPointer**);

Console.ReadKey();

*//Code block which get executed while iteration*

public XEmployee **FifthTestFunction**(Employee employee)

{

return new XEmployee() {Name = employee.Name, Age = employee.Age};

}

Comparison<T>

Comparison delegate is used to sort or order the data inside a collection. It takes two parameters as generic input type and return type should always be int. This is how we can declare Comparison delegate.

Hide   Copy Code

Comparison<string> **tempComparison** = new Comparison<string>(**tempObj.SixTestFunction**);

In this sample, Employee Name is used to Sort the order. All the entity inside the collection will be processed using SixthTestFunction which contains the logic for processing / sorting the data as per our requirement.

Hide   Copy Code

Comparison<Employee> **tempComparisonPointer**

= new Comparison<Employee>(**tempObj.SixTestFunction**);

Array.Sort(lstEmployee, **tempComparisonPointer**);

Console.ReadKey();

public int **SixTestFunction**(Employee strParameter1, Employee strParamater2)

{

return strParameter1.Name.CompareTo(strParamater2.Name);

}

Points of Interest

Using the new delegates, we can achieve better level of abstraction & performance can also been gained by avoiding unnecessary iteration or conversion logic. All three delegates (Predicate, Converter, Comparison) do have optimized internal logic for iteration.