delegate operator

Article • 04/08/2023

The `delegate` operator creates an anonymous method that can be converted to a delegate type. An anonymous method can be converted to types such as System.Action and System.Func<TResult> types used as arguments to many methods.

C#

```
Func<int, int, int> sum = delegate (int a, int b) { return a + b; };
Console.WriteLine(sum(3, 4));  // output: 7
```

Note

Lambda expressions provide a more concise and expressive way to create an anonymous function. Use the **=> operator** to construct a lambda expression:

C#

```
Func<int, int, int> sum = (a, b) => a + b;
Console.WriteLine(sum(3, 4));  // output: 7
```

For more information about features of lambda expressions, for example, capturing outer variables, see **Lambda expressions**.

When you use the `delegate` operator, you might omit the parameter list. If you do that, the created anonymous method can be converted to a delegate type with any list of parameters, as the following example shows:

C#

```
Action greet = delegate { Console.WriteLine("Hello!"); };
greet();

Action<int, double> introduce = delegate { Console.WriteLine("This is
world!"); };
introduce(42, 2.7);

// Output:
// Hello!
// This is world!
```

That's the only functionality of anonymous methods that isn't supported by lambda expressions. In all other cases, a lambda expression is a preferred way to write inline code.

Beginning with C# 9.0, you can use discards to specify two or more input parameters of an anonymous method that aren't used by the method:

```
C#

Func<int, int, int> constant = delegate (int _, int _) { return 42;
};
Console.WriteLine(constant(3, 4));  // output: 42
```

For backwards compatibility, if only a single parameter is named `_`, `_` is treated as the name of that parameter within an anonymous method.

Also beginning with C# 9.0, you can use the `static` modifier at the declaration of an anonymous method:

```
C#

Func<int, int, int> sum = static delegate (int a, int b) { return a +
b; };
Console.WriteLine(sum(10, 4));  // output: 14
```

A static anonymous method can't capture local variables or instance state from enclosing scopes.

You also use the `delegate` keyword to declare a delegate type.

Beginning with C# 11, the compiler may cache the delegate object created from a method group. Consider the following method:

```
C#

static void StaticFunction() { }
```

When you assign the method group to a delegate, the compiler will cache the delegate:

```
C#

Action a = StaticFunction;
```

Before C# 11, you'd need to use a lambda expression to reuse a single delegate object:

C#

```
Action a = () => StaticFunction();
```

C# language specification

For more information, see the Anonymous function expressions section of the C# language specification.

See also

- C# reference
- C# operators and expressions
- => operator