C# : Expression Lambda

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

C#

Expressions Lambda [**C# 3.0**]

- **C#** : langage purement orienté-objet
- Intégrer les expressions Lambda $\Rightarrow$ ajouter un aspect fonctionnel (fonctions et procédures) au langage **C#**
- Pour séparer la partie paramètres et la partie instructions : on utilise =>

C#

Syntaxe

```
([arguments]) => { instructions };
```

C#

Syntaxe

```
([arguments]) => { instructions };
```

Ou

```
([arguments]) =>  instruction;
```

C#

Trois types d'expression Lambda

- Une première qui ne retourne pas de valeur : peut être affectée à un objet (délégué) générique de type `Action`.
- Une deuxième qui accepte un seul paramètre et retourne un booléen : peut être affectée à un objet générique de type `Predicate`.
- Une troisième qui accepte zéro ou plusieurs paramètres et retourne une valeur : peut être affectée à un objet générique de type `Func`.

C#

Exemple avec valeur de retour

```
Func<int, int> Carre = x => x * x;
```

C#

Exemple avec valeur de retour

```
Func<int, int> Carre = x => x * x;
```

Explications

- Si l'expression a un seul paramètre, les parenthèses peuvent être supprimées.
- Si l'expression a une seule instruction, les accolades peuvent être supprimées.

C#

Exemple avec valeur de retour

```
Func<int, int> Carre = x => x * x;
```

Explications

- Si l'expression a un seul paramètre, les parenthèses peuvent être supprimées.
- Si l'expression a une seule instruction, les accolades peuvent être supprimées.

Pour appeler la fonction `Carre`

```
Console.WriteLine(Carre(3));
// affiche 9
```

C#

La fonction `Carre` peut être écrite ainsi

```
Func<int, int> Carre = (x) =>
{
    return x * x;
};
```

C#

Exemple avec `Action`

```
Action<string> DireBonjour = (nom) => Console.
   WriteLine($"Bonjour { nom }");
```

C#

Exemple avec `Action`

```
Action<string> DireBonjour = (nom) => Console.
   WriteLine($"Bonjour { nom }");
```

Pour appeler la fonction `DireBonjour`

```
DireBonjour("wick");
// Bonjour wick
```

C#

Exemple avec `Predicate`

```
Predicate<int> EstPair = (int a) => a % 2 == 0;
```

C#

Exemple avec `Predicate`

```
Predicate<int> EstPair = (int a) => a % 2 == 0;
```

Pour appeler la fonction `EstPair`

```
Console.WriteLine($"4 est pair : { EstPair(4) }");
// affiche 4 est pair : True

Console.WriteLine($"5 est pair : { EstPair(5) }");
// affiche 5 est pair : False
```

C#

Une expression lambda peut utiliser une variable (ou un attribut) définie dans le contexte englobant (et modifier sa valeur)

```
int i = 2, j = 3;
Func<int, int, int> DoSomething = (x, y) =>
{
  i++;
  j++;
  return x * i + y * j;
};

Console.WriteLine(DoSomething(1, 1));
// affiche 7
```

C#

Une expression lambda ne peut redéfinir une variable (ou un attribut) définie dans le contexte englobant (Ceci est faux car `i` a été déclaré et initialisé juste avant)

```
int i = 2, j = 3;
Func<int, int, int> DoSomething = (x, y) =>
{
  int i = 0;
  j++;
  return x * i + y * j;
};
```

C#

Considérons le tableau suivant

```
List <int> numbers = new List<int>{ 1, 2, 3, 4, 5 };
```

C#

Considérons le tableau suivant

```
List <int> numbers = new List<int>{ 1, 2, 3, 4, 5 };
```

Pour afficher le tableau précédent, on peut utiliser `foreach`

```
foreach (var elt in numbers)
{
    Console.WriteLine(elt);
}
```

C#

Considérons le tableau suivant

```
List <int> numbers = new List<int>{ 1, 2, 3, 4, 5 };
```

Pour afficher le tableau précédent, on peut utiliser `foreach`

```
foreach (var elt in numbers)
{
    Console.WriteLine(elt);
}
```

On peut aussi utiliser la méthode `ForEach` qui prend en paramètre une expression Lambda

```
numbers.ForEach(elt => Console.WriteLine(elt));
```

C#

On peut aussi utiliser le raccourci suivant

```
numbers.ForEach(Console.WriteLine);
```

C#

On peut aussi utiliser le raccourci suivant

```
numbers.ForEach(Console.WriteLine);
```

On peut aussi définir une méthode `print`

```
public static void print(int n)
{
    Console.WriteLine(n);
}
```

C#

On peut aussi utiliser le raccourci suivant

```
numbers.ForEach(Console.WriteLine);
```

On peut aussi définir une méthode `print`

```
public static void print(int n)
{
    Console.WriteLine(n);
}
```

Et ensuite l'appeler

```
numbers.ForEach(print);
```

C#

Quelques méthodes de la classe `List` prenant comme paramètre une Expression Lambda

- `Find`
- `FindAll`
- `FindIndex`
- `RemoveAll`
- `Exists`
- ...

C#

Remarque

Les expressions Lambda sont très utilisées par **Linq**.