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#

Rappel : délégué

Un **délégué** est un type qui référence une méthode (signature donnée).

C#

Contexte et motivation

- **C#** : langage purement orienté-objet
- Besoin fréquent : passer un **comportement** (une opération) en paramètre
- Intégrer les expressions Lambda $\Rightarrow$ ajouter un aspect fonctionnel (fonctions et procédures) au langage **C#**
  - Séparation **paramètres** / **corps** via l'opérateur `=>`
  - Cible principale : `Action`, `Func`, `Predicate`

C#

Contexte et motivation

- **C#** : langage purement orienté-objet
- Besoin fréquent : passer un **comportement** (une opération) en paramètre
- Intégrer les expressions Lambda ⇒ ajouter un aspect fonctionnel (fonctions et procédures) au langage **C#**
  - Séparation **paramètres** / **corps** via l'opérateur =>
  - Cible principale : `Action`, `Func`, `Predicate`

Expressions Lambda [**C# 3.0**]

Une expression lambda est une syntaxe concise pour définir une **fonction anonyme** pouvant être affectée à un **délégué**.

C#

Syntaxe : forme expression

```
(parametres) => expression;
```

C#

Syntaxe : forme expression

```
(parametres) => expression;
```

Ou la forme bloc

```
(parametres) => { instructions; };
```

C#

Syntaxe : forme expression

```
(parametres) => expression;
```

Ou la forme bloc

```
(parametres) => { instructions; };
```


Expressions Lambda [**C# 3.0**]

- La forme **expression** retourne implicitement la valeur de l'expression
- La forme **bloc** nécessite généralement un `return` si la lambda retourne une valeur

C#

Délégués génériques standards

- `Action<T>` : prend des paramètres, **ne retourne rien**
- `Func<T, TResult>` : prend des paramètres, **retourne une valeur**
- `Predicate<T>` : prend un `T`, **retourne bool**

C#

Règle de lecture de `Func`

`Func<A, B, C>` signifie : paramètres `A, B` et retour `C`.

C#

Exemple avec forme d'expression

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

C#

Exemple avec forme d'expression

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

Bloc

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

C#

Exemple avec forme d'expression

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

Bloc

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

Pour appeler la fonction `Carre` quelle que soit la forme

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

C#

Explications

- Un seul paramètre : parenthèses optionnelles
- Une seule instruction : accolades optionnelles
- Le type peut souvent être inféré

C#

Explications

- Un seul paramètre : parenthèses optionnelles
- Une seule instruction : accolades optionnelles
- Le type peut souvent être inféré

Les trois écritures suivantes sont équivalentes

```
Func<int,int> f1 = (int x) => x * x;
Func<int,int> f2 = (x) => x * x;
Func<int,int> f3 = x => x * x;
```

C#

Exemple avec `Action` : pas de valeur de retour

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

C#

Exemple avec `Action` : pas de valeur de retour

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

Pour appeler l'action `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#

Capture de variables (closure)

- Une lambda peut **capturer** une variable du contexte englobant.
- Capture par **référence** (la variable, pas seulement sa valeur).

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 peut capturer et modifier une variable du contexte englobant, mais ne peut pas redéclarer une variable portant le même nom

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

C#

Bonnes pratiques

- Préférer des lambdas courtes et lisibles
- Éviter la logique complexe dans une lambda (extraire en méthode)
- Limiter les effets de bord (modification d'état externe)

C#

Considérons le tableau suivant

```
List<int> numbers = [1, 2, 3, 4, 5];
```

C#

Considérons le tableau suivant

```
List<int> numbers = [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 = [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 (`method group`)

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

C#

On peut aussi utiliser le raccourci suivant (`method group`)

```
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 (`method group`)

```
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#

`Print` **peut aussi être définie par une lambda**

```
var Print = (int i) => Console.WriteLine(i);
```

C#

`Print` **peut aussi être définie par une lambda**

```
var Print = (int i) => Console.WriteLine(i);
```

L'appel ne change pas

```
numbers.ForEach(Print);
```

C#

Depuis C# 12, une Expression Lambda peut accepter une valeur par défaut

```
var SayHello = (string nom = "Doe") => Console.Write($"Hello {nom}");
```

C#

Depuis C# 12, une Expression Lambda peut accepter une valeur par défaut

```
var SayHello = (string nom = "Doe") => Console.Write($"Hello {nom}");
```

L'appel ne change pas

```
SayHello();
// Hello Doe

SayHello("Wick");
// Hello Wick
```

C#

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

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

C#

Exemple avec `Find`

```
List<int> numbers = [1, 2, 3, 4, 5];
int resultat = numbers.Find(n => n > 3);

Console.WriteLine(resultat);
// 4
```

C#

Exemple avec `Find`

```
List<int> numbers = [1, 2, 3, 4, 5];
int resultat = numbers.Find(n => n > 3);

Console.WriteLine(resultat);
// 4
```

Exemple avec `FindIndex`

```
List<int> numbers = [1, 2, 3, 4, 5];
var resultat = numbers.FindIndex(n => n > 3);

Console.WriteLine(resultat);
//  3
```

C#

Exemple avec `Exists`

```
List<int> numbers = [1, 2, 3, 4, 5];
var resultat = numbers.Exists(n => n > 3);

Console.WriteLine(resultat);
// True
```

C#

Exemple avec `Exists`

```
List<int> numbers = [1, 2, 3, 4, 5];
var resultat = numbers.Exists(n => n > 3);

Console.WriteLine(resultat);
// True
```

Exemple avec `RemoveAll`

```
List<int> numbers = [1, 2, 3, 4, 5];
numbers.RemoveAll(n => n > 3);

numbers.ForEach(n => Console.WriteLine(n));
// 1 2 3
```

C#

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

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

Exemple avec `Find`

```
int[] nombres = [1, 2, 3, 4, 5];
var resultat = Array.Find(nombres, n => n > 30);

Console.WriteLine(resultat);
// 4
```

C#

Exemple avec `Find`

```
int[] nombres = [1, 2, 3, 4, 5];
var resultat = Array.Find(nombres, n => n > 30);

Console.WriteLine(resultat);
// 4
```

Exemple avec `FindIndex`

```
int[] nombres = [1, 2, 3, 4, 5];
var resultat = Array.FindIndex(nombres, n => n > 3);

Console.WriteLine(resultat);
// 3
```

C#

Exemple avec `FindAll`

```
int[] nombres = [1, 2, 3, 4, 5];
var resultat = Array.FindAll(nombres, n => n > 3);

resultat.ToList().ForEach(n => Console.WriteLine(n));
// 4 5
```

C#

Exemple avec `FindAll`

```
int[] nombres = [1, 2, 3, 4, 5];
var resultat = Array.FindAll(nombres, n => n > 3);

resultat.ToList().ForEach(n => Console.WriteLine(n));
// 4 5
```

Exemple avec `Exists`

```
int[] nombres = [1, 2, 3, 4, 5];
var resultat = Array.Exists(nombres, n => n > 3);

Console.WriteLine(resultat);
// True
```

Remarque

Les expressions Lambda sont très utilisées par **Linq** (chapitre ultérieur).