

Transformeur (aka **Map** ou **Select**)



Après **Filter** incarné par le **Where** de *Linq* qui permet de filtrer une collection, il est temps d'observer de près d'autres fonctions d'ordre supérieur telles que **Select** (aka map, transformeur ou projection) .

Transformeur | Map | Select | Projection

Inspiré du mots clé SQL de la projection ensembliste **SELECT**, LINQ propose l'outil **Select** pour faire des opérations de *mapping* dans une collection...

Avec notre filtre, nous avons retenu des éléments (objets) qui ont le contenu que l'on veut, mais leur forme n'est pas forcément adaptée à l'usage que l'on veut en faire.

Je ne veux pas transformer, je veux juste *sélectionner*

Le terme **transformer** peut sembler inadapté comparé à celui de **projection**, pourtant en sélectionnant une sous-partie d'un ensemble de départ, il y a une transformation...

Selon les langages et bibliothèques, il existe une nuance importante entre transformation et projection dans le sens où la transformation s'applique sur la source directement plutôt que de renvoyer une nouvelle structure de données.

Cet aspect touche à l'**immutabilité** qui sera présentée ultérieurement....

Select: Choix d'un attribut dans une classe

```
class Person{  
    public string Name{get;set;}
```

```

    public int Age{get;set;}
    public int Sisters{get;set;}
    public int Brothers{get;set;}
}

List<Person> cid5d = new List<Person>(){
    new Person(){Name="Paul",Age=15,Sisters=2,Brothers=1},
    new Person(){Name="Lucie",Age=18,Sisters=1,Brothers=3},
    new Person(){Name="Claude",Age=16,Sisters=0,Brothers=0}
};

IEnumerable<string> names = cid5d.Select(person =>
    person.Name); //{ "Paul", "Lucie", "Claude" }
List<string> stringNames = names.ToList();

```

Comme en SQL, on sélectionne une partie des données (ici juste le nom) pour générer une nouvelle liste contenant ceux-ci.

Select: Modification d'une valeur

Au moment de produire le résultat, on veut parfois le transformer plutôt que de conserver la valeur brute:

```

IEnumerable<int> numberOfSiblings = cid5d.Select(person =>
    person.Sisters+person.Brothers); //{ 3, 4, 0 }

```

Cette fois-ci, on génère un nouvel ensemble de valeurs modifiées selon l'ensemble de base...

Classe, Tuple et classe anonyme

En sélectionnant des attributs, on crée une nouvelle une nouvelle structure de donnée ... ou pas!

On peut en effet s'appuyer sur une classe déjà existante:

```

List<Member> members = cid5d.Select(person => new
    Member(person.Name,person.Age,person.Sisters,person.Brothers)).ToList();

```

Tuple

Pour récupérer un sous-ensemble d'attributs d'une classe, le tuple peut s'avérer utile:

```

var adults =
    cid5d.Select(person => (person.Name/*Devient Item1*/,person.Age/*Devient
    Item2*/))
        .Where(tuple=>tuple.Item2>=18); //Item2 correspond à l'age

    Console.WriteLine(adults.First().Item1); //Lucie

```

Le **Tuple** ressemble à une classe anonyme pouvant supporter n attributs *readonly* qu'on accède avec les propriétés Item1, Item2, Item3, ItemN un peu comme avec un tableau...

```
var tuple = (1,2,3);
Console.WriteLine(tuple.Item1);//1
Console.WriteLine(tuple.Item2);//2
Console.WriteLine(tuple.Item3);//3
```

Il est possible de nommer les attributs d'un tuple

```
var tuple = (first: 1, second: 2, third: 3);
Console.WriteLine(tuple.first);//1
Console.WriteLine(tuple.second);//2
Console.WriteLine(tuple.third);//3
```

ATTENTION toutefois car un **ToList()** fera disparaître cette information...

Classe anonyme

À défaut du tuple, il est aussi possible de générer une classe anonyme qui gardera ses informations sur toute la chaîne d'appel LINQ:

```
var anon = new {first= 1, second= 2, third= 3};
Console.WriteLine(anon.first);//1
Console.WriteLine(anon.second);//2
Console.WriteLine(anon.third);//3
```

En détail

Pour plus de détail, consulter [la documentation officielle](#)

Name	Access modifier	Type	Custom member name	Deconstruction support	Expression tree support
Anonymous types	<code>internal</code>	<code>class</code>	✓	✗	✓
Tuple	<code>public</code>	<code>class</code>	✗	✗	✓
ValueTuple	<code>public</code>	<code>struct</code>	✓	✓	✗

Transformeurs communs

LINQ propose quelques transformeurs utiles:

- `GroupBy`
- `ToList`
- `ToArray`
- `ToDictionary`

GroupBy

Comme son nom l'indique, il groupe selon un critère et renvoie un dictionnaire dont chaque entrée a:

- Une clé
- La collection des éléments qui satisfont le critère (sur laquelle on va pouvoir appliquer des agrégateurs tels que `Count`, `Sum`, `Max`, `Min`,)

Voir l'exemple dans la [documentation .NET](#)

ToList

Convertit l'entrée (de type `IEnumerable`) en liste (de type `List<T>`).

ToDictionary

Crée un dictionnaire selon la fonction d'affectation pour la clé et la valeur.

Cela peut s'avérer utile dans certaines situations, par exemple pour retrouver plus rapidement une information. En effet, retrouver une personne par index dans un dictionnaire comme ceci:

```
Dictionary<int, Person> dico;  
Person toto = dico[712]
```

est beaucoup plus rapide qu'une recherche LinQ comme cela

```
List<Person> people;  
Person toto = people.Where(p => p.Id == 712).First();
```

Si vous en doutez, essayez [ceci](#)...