LINQ (Language-Integrated Query) (C#)

Mme Ben bouzid NOURHENE

Introduction

- LINQ (Language-Integrated Query) est le nom d'un ensemble de technologies basé sur l'intégration de fonctions de requête directement dans le langage C#.
- Les requêtes de données sont exprimées comme de simples chaînes, sans vérification de type au moment de l'exécution.
- Vous pouvez écrire des requêtes LINQ en C# pour des bases de données SQL Server, des documents XML, des jeux de données ADO.NET et toute collection d'objets prenant en charge IEnumerable ou l'interface générique IEnumerable<T>.

Introduction

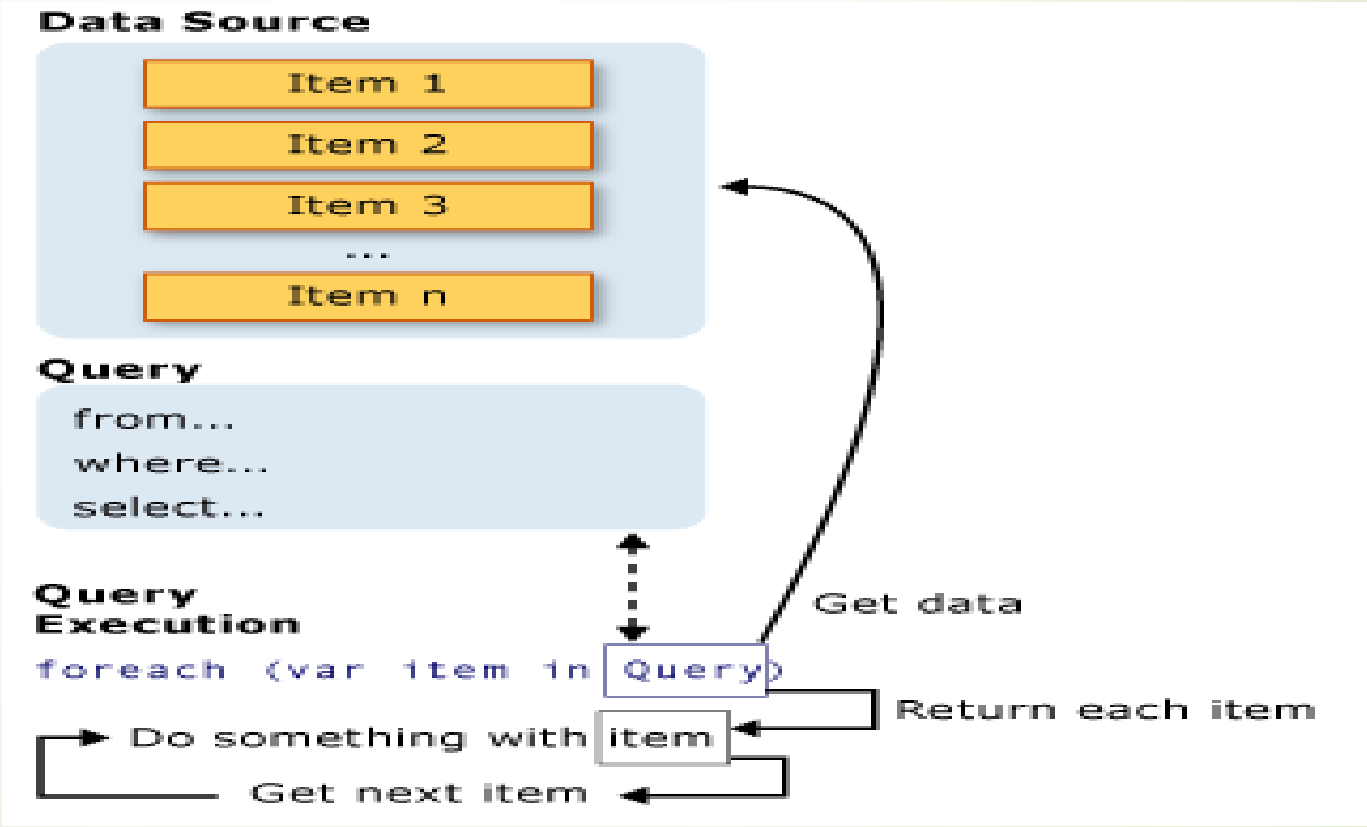
- Les expressions de requête peuvent être utilisées pour interroger et transformer des données à partir de n'importe quelle source de données compatible LINQ.
- Toutes les opérations de requête LINQ se composent de trois actions distinctes :
  - Obtention de la source de données
  - Création de la requête
  - Exécution de la requête.

Exemple:

```
class LINQQueryExpressions
{
    static void Main()
    {

        // Specify the data source.
        int[] scores = new int[] { 97, 92, 81, 60 };

        // Define the query expression.
        IEnumerable<int> scoreQuery =
            from score in scores
            where score > 80
            select score;

        // Execute the query.
        foreach (int i in scoreQuery)
        {
            Console.Write(i + " ");
        }
    }
}
// Output: 97 92 81
```

Illustration:

- Dans LINQ, l'exécution de la requête est distincte de la requête elle-même. En d'autres termes, vous n'avez récupéré aucune donnée en créant simplement une variable de requête



LINQ:

- L’expression de requête contient trois clauses : from, where et select.
- La **from** clause spécifie la source de données, la **where** clause applique le filtre et la **select** clause spécifie le type des éléments retournés.

.

LINQ et les types génériques (C#)

- Les requêtes LINQ sont basées sur des types génériques, qui ont été introduits dans la version 2,0 de .NET Framework.
- **List<T>:** Quand vous créez une instance d'une classe de collection générique comme List<T>, vous remplacez le « T » par le type des objets contenus dans la liste.Par exemple, une liste de chaînes est exprimée sous la forme List<string>, et une liste d'objets Customer est exprimée sous la forme List<Customer>.
- **IEnumerable<T>:** l'interface qui permet l'énumération des classes de collection génériques avec l'instruction foreach. Les classes de collection génériques prennent en charge IEnumerable<T> de la même façon que les classes de collection non génériques telles que ArrayList prennent en charge IEnumerable.

LINQ et les types génériques (C#)

```
IEnumerable<Customer> customerQuery =
    from cust in customers
    where cust.City == "London"
    select cust;

foreach (Customer customer in customerQuery)
{
    Console.WriteLine(customer.LastName + ", " + customer.FirstName);
}
```

C#

```
List<int> numQuery2 =
    (from num in numbers
     where (num % 2) == 0
     select num).ToList();

// or like this:
// numQuery3 is still an int[]

var numQuery3 =
    (from num in numbers
     where (num % 2) == 0
     select num).ToArray();
```

LINQ et les types génériques (C#)

- On peut éviter l'utilisation des types générique avec le mot clé **var**.
- Le mot clé var indique au compilateur d'inférer le type d'une variable de requête en examinant la source de données spécifiée dans la clause from.

```
var customerQuery2 =
    from cust in customers
    where cust.City == "London"
    select cust;

foreach(var customer in customerQuery2)
{
    Console.WriteLine(customer.LastName + ", " + customer.FirstName);
}
```

Combinaison de plusieurs entrées en une séquence de sortie

- On peut utiliser une requête LINQ pour créer une séquence de sortie qui contient des éléments issus de plusieurs séquences d'entrée.

```
List<Student> students = new List<Student>()
{
    new Student { First="Svetlana",
        Last="Omelchenko",
        ID=111,
        Street="123 Main Street",
        City="Seattle",
        Scores= new List<int> { 97, 92, 81, 60 } },
```

```
// Create the second data source.
List<Teacher> teachers = new List<Teacher>()
{
    new Teacher { First="Ann", Last="Beebe", ID=945, City="Seattle" },
    new Teacher { First="Alex", Last="Robinson", ID=956, City="Redmond"
    new Teacher { First="Michiyo", Last="Sato", ID=972, City="Tacoma" }
};
```

Combinaison de plusieurs entrées en une séquence de sortie

```
    // Create the query.
    var peopleInSeattle = (from student in students
                  where student.City == "Seattle"
                  select student.Last)
                  .Concat(from teacher in teachers
                          where teacher.City == "Seattle"
                          select teacher.Last);

    Console.WriteLine("The following students and teachers live in Seattle:");
    // Execute the query.
    foreach (var person in peopleInSeattle)
    {
        Console.WriteLine(person);
    }

    Console.WriteLine("Press any key to exit.");
    Console.ReadKey();
}
```

Les Clauses facultatives du LINQ:

- Entre la clause from initiale et la clause select ou group on peut utiliser les clauses facultatives comme where, join, orderby..

```
IEnumerable<Country> querySortedCountries =
    from country in countries
    orderby country.Area, country.Population descending
    select country;
```

```
var categoryQuery =
    from cat in categories
    join prod in products on cat equals prod.Category
    select new { Category = cat, Name = prod.Name };
```

```
var queryCountryGroups =
    from country in countries
    group country by country.Name[0];
```

Les Clauses facultatives du LINQ:

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            orderby word.Length
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    the
    fox
    quick
    brown
    jumps
*/
```

- Tri secondaire

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            orderby word.Length, word.Substring(0, 1)
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    fox
    the
    brown
    jumps
    quick
*/
```

Les Clauses facultatives du LINQ:

```
var studentQuery6 =
    from student in students
    let totalScore = student.Scores[0] + student.Scores[1] +
        student.Scores[2] + student.Scores[3]
    select totalScore;

double averageScore = studentQuery6.Average();
Console.WriteLine("Class average score = {0}", averageScore);
```

```
// Group students by the first letter of their last name
// Query variable is an IEnumerable<IGrouping<char, Student>>
var studentQuery2 =
    from student in students
    group student by student.Last[0] into g
    orderby g.Key
    select g;
```

Merci pour votre attention