Chapitre 4

Deep Dive Into LINQ

Introduction

- LINQ est un feature introduit avec le .Net 3.5.
- Il nous permet de travailler avec plusieurs types de données.
- Fournit la possibilité d'écrire des requêtes sur les données.

1. Understand LINQ
2. Understand LINQ Operators
3. Understand LINQ Syntaxes
4. Working with LINQ Queries
5. Working with LINQ to XML

Introduction

LINQ (Language Integrated Query) is a way to query different types of data sources that support IEnumerable<T> or IQueryable<T>.
It offers an easy and elegant way to access or manipulate data from a database object, XML document, and in-memory objects.

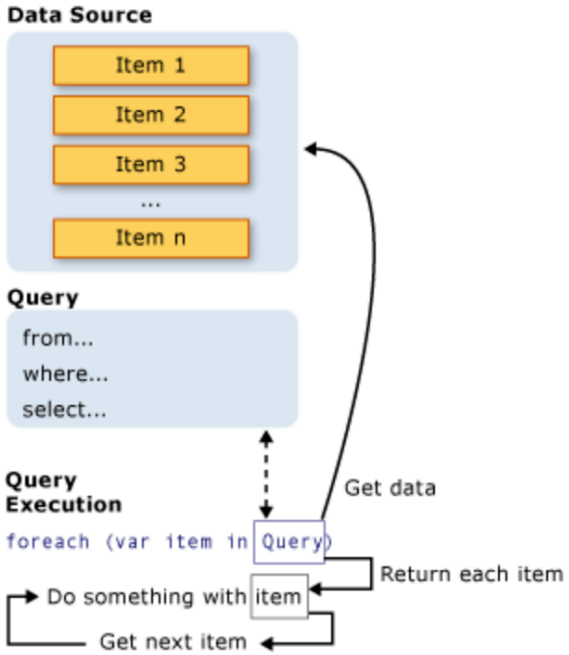
Why we use LINQ

LINQ usually is more important than other query structures due to its way of working with different data sources.

Introduction

Toutes les opérations de requête LINQ se composent de trois actions distinctes :

1. Obtention de la source de données
2. Création de la requête
3. exécutez la requête.




```
class IntroToLINQ
{
    static void Main()
    {
        // The Three Parts of a LINQ Query:
        // 1. Data source.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Query creation.
        // numQuery is an IEnumerable<int>
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        // 3. Query execution.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

Types de LINQ

LINQ opère avec plusieurs sources de données, en les classifiant :

LINQ to XML

LINQ to XML charge un document XML dans un type requêtable XElement

```
// using System.Xml.Linq;
XElement contacts = XElement.Load(@"c:\myContactList.xml")
```

LINQ to SQL

Créer un mappage relationnel d'objet au moment de la conception.

Ecrire les requêtes sur les objets, et à l'exécution ;

LINQ to SQL gère la communication avec la base de données.

```
Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// Query for customers in London.
IQueryable<Customer> custQuery =
    from cust in db.Customers
    where cust.City == "London"
    select cust;
```

Types de LINQ

LINQ opère avec plusieurs sources de données, en les classifiant :

LINQ to Entities
« LINQ to Objects » fait référence à l'utilisation directe de requêtes LINQ avec n'importe quelle collection
IEnumerable ou IEnumerable<T>.
Pas besoin d'API intermédiaire comme LINQ to SQL ou LINQ to XML.

```
void QueryHighScores(int exam, int score)
{
    var highScores =
        from student in students
        where student.ExamScores[exam] > score
        select new
        {
            Name = student.FirstName,
            Score = student.ExamScores[exam]
        };

    foreach (var item in highScores)
    {
        Console.WriteLine($"{item.Name,-15}{item.Score}");
    }
}

QueryHighScores(1, 90);
```

Que retourne la requête suivante?

Les opérateurs LINQ

1. Opérateur de projection : Utilisé lorsque l'objet change de forme en se basant sur une condition.

Operator	Description	Syntax
Select	Select an obtained result from a data source	Select

```
    IEnumerable<string> result = from p in persons
                                 where p.Name.Length > 4
                                 select p.Name;
foreach (var name in result)
{
    Console.WriteLine(name);
}
```

SelectMany

Listing 6-8. SelectMany Operator

```
    var result = (from p in persons
                  where p.Name.Length > 4
                  select new
                  {
                      PersonID = p.ID,
                      PersonName = p.Name,
                           PersonAddress=p.Address
                  });

foreach (var item in result)
{
    Console.WriteLine(item.PersonID + "\t" + item.PersonName );
}
```

Les opérateurs LINQ

2. Opérateur de jointure : Joindre deux collections en se basant sur une clé

Operator	Description	Syntax
Join	Join sequence on the basis of a matching key	join..in..on.equals

```
class Class
        {
            public int ClassID { get; set; }
            public string ClassName { get; set; }
        }

        class Student
        {
            public int StudentID { get; set; }
            public string StudentName { get; set; }
            public int ClassID { get; set; }
        }

    List<Class> classes = new List<Class>();
    classes.Add(new Class { ClassID = 1, ClassName = "BSCS" });
    classes.Add(new Class { ClassID = 2, ClassName = "BSSE" });
    classes.Add(new Class { ClassID = 3, ClassName = "BSIT" });

    List<Student> students = new List<Student>();
    students.Add(new Student { ClassID = 1, StudentID = 1, StudentName = "Hamza" });
    students.Add(new Student { ClassID = 2, StudentID = 2, StudentName = "Zunaira" });
    students.Add(new Student { ClassID = 1, StudentID = 3, StudentName = "Zeeshan" });
```

```
var result = (from std in students
              join clas in classes on std.ClassID equals clas.ClassID
              select new
              {
                  _Student = std.StudentName,
                  _Class = clas.ClassName
              });

  foreach (var item in result)
  {
      Console.WriteLine(item._Student + "\t" + item._Class);
  }
```

Les opérateurs LINQ

3. Opérateur de Groupement : Organiser les éléments en se basant sur une clé

Operator	Description	Syntax
GroupBy	Return a sequence of items in groups as an IGroup<key,element>	group.....by <or> group...by..into <or> GroupBy(<predicate>)

```
    var result = from p in persons
                 group p by p.Address;

foreach (var student in result)
{
    Console.WriteLine("Address:" + student.Key);
    foreach (var st in student)
    {
        Console.WriteLine(st.ID + "\t" + st.Name);
    }
}
```

4. Opérateur de Partition : Créer des partitions à partir d'une collection

Operator	Description	Syntax
Skip	Skip the supplied number of records and return the remaining ones.	Skip<T>(<count>)
Take	Take the supplied number of records and skip the remaining ones.	Take<T>(<count>)

Les opérateurs LINQ

4. Opérateur de Partition : Créer des partitions à partir d'une collection

Operator	Description	Syntax
Skip	Skip the supplied number of records and return the remaining ones.	Skip<T>(<count>)
Take	Take the supplied number of records and skip the remaining ones.	Take<T>(<count>)

```
var result = (from p in persons
              where p.Address.StartsWith("P")
              select p).Take(2);

foreach (var item in result)
{
    Console.WriteLine(item.ID + "\t" + item.Name);
}
```

Skip

Listing 6-12. Skip Operator

```
var result = (from p in persons
              where p.Address.StartsWith("P")
              select p).Skip(2);

foreach (var item in result)
{
    Console.WriteLine(item.ID + "\t" + item.Name);
}
```

Les opérateurs LINQ

5. Agrégation: Appliquer des fonctions d'agrégation sur LINQ
Une fonction d'agrégation opère les requêtes et retourne une valeur unique

Operator	Description	Syntax
Average	Take the average of a numeric collection.	Average<T>(<param>)
Count	Count the number of elements in a collection.	Count<T>(<param>)
Max	Return the highest value from the collection of numeric values.	Max<T>(<param>)
Min	Return the highest value from the collection of numeric values.	Min<T>(<param>)
Sum	Compute the sum of numeric values in a collection.	Sum<T>(<param>)

Les opérateurs LINQ

6. Opérateur de filtrage: Appliquer des fonctions d'agrégation sur LINQ
Une fonction d'agrégation opère les requêtes et retourne une valeur unique

```
IEnumerable<Person> result = from p in persons
                             where p.Name.Length > 4
                             select p;
      foreach (var item in result)
      {
          Console.WriteLine(item.ID + "\t" + item.Name + "\t" + item.Address);
      }
```

Syntaxe LINQ

Il y a deux manières d'écrire les requêtes LINQ:

1. Query syntaxe
2. DotNotation syntaxe

```
var queue = from q in dc.SomeTable
        where q.SomeDate <= DateTime.Now && q.Locked != true
        orderby (q.Priority, q.TimeCreated)
        select q;

var queue2 = dc.SomeTable
        .Where( q => q.SomeDate <= DateTime.Now && q.Locked != true )
        .OrderBy(q => q.Priority)
        .ThenBy(q => q.TimeCreated);
```

Mode d’exécution

Immédiat

L’exécution immédiate signifie que la source de données est lue et que l’opération est effectuée une seule fois.

Vous pouvez forcer l’exécution d’une requête immédiatement à l’aide des méthodes Enumerable.ToList ou Enumerable.ToArray.

L’exécution immédiate permet la réutilisation des résultats de la requête.

Les résultats sont récupérés une fois, puis stockés pour une utilisation ultérieure.

Listing 6-4. Force Execution

```
List<Person> persons = new List<Person>()
{
    new Person() { ID=1,Name="Ali Asad"},
    new Person() { ID=5,Name="Hamza Ali"},

};

var query = (from p in persons
             select p).ToList();

persons.Add(new Person() { ID = 3, Name = "John Snow" });

foreach (var item in query)
{
    Console.WriteLine(item.ID + "\t" + item.Name);
}
```

Mode d’exécution

Différé

L’exécution différée signifie que l’opération n’est pas effectuée au point où la requête est déclarée dans le code.

L’opération est effectuée uniquement quand la variable de requête est énumérée, par exemple à l’aide d’une instruction `foreach`.

Cela signifie que les résultats de l’exécution de la requête dépendent du contenu de la source de données au moment de l’exécution de la requête, et non au moment de sa définition.

```
class Person
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public decimal Salary { get; set; }
}

    List<Person> persons = new List<Person>()
    {
        new Person() { ID=1,Name="Ali Asad"},
        new Person() { ID=5,Name="Hamza Ali"},

    };

    var query = from p in persons
                select p;

    int count = 0;
    count = query.Count();//Counts 2 records

    persons.Add(new Person() { ID = 3, Name = "John Snow" });

    count = query.Count();//Count 3 records

    Console.WriteLine(query);
```

Extensions LINQ

Extension Method	Description	Deferred
`All`	Returns `true` if all the items in the source data match the predicate	No
`Any`	Returns `true` if at least one of the items in the source data matches the predicate	No
`Contains`	Returns `true` if the data source contains a specific item or value	No
`Count`	Returns the number of items in the data source	No
`First`	Returns the first item from the data source	No
`FirstOrDefault`	Returns the first item from the data source or the default value if there are no items	No
`Last`	Returns the last item in the data source	No
`LastOrDefault`	Returns the last item in the data source or the default value if there are no items	No
`Max` `Min`	Returns the largest or smallest value specified by a lambda expression	No
`OrderBy` `OrderByDescending`	Sorts the source data based on the value returned by the lambda expression	Yes
`Reverse`	Reverses the order of the items in the data source	Yes
`Select`	Projects a result from a query	Yes
`SelectMany`	Projects each data item into a sequence of items and then concatenates all of those resulting sequences into a single sequence	Yes
`Single`	Returns the first item from the data source or throws an exception if there are multiple matches	No
`SingleOrDefault`	Returns the first item from the data source or the default value if there are no items, or throws an exception if there are multiple matches	No
`Skip` `SkipWhile`	Skips over a specified number of elements, or skips while the predicate matches	Yes
`Sum`	Totals the values selected by the predicate	No
`Take` `TakeWhile`	Selects a specified number of elements from the start of the data source or selects items while the predicate matches	Yes

`ToArray`
`ToDictionary`
`ToList`
	Converts the data source to an array or other collection type	No

`Where`	Filters items from the data source that do not match the predicate	Yes