LINQ의 개요

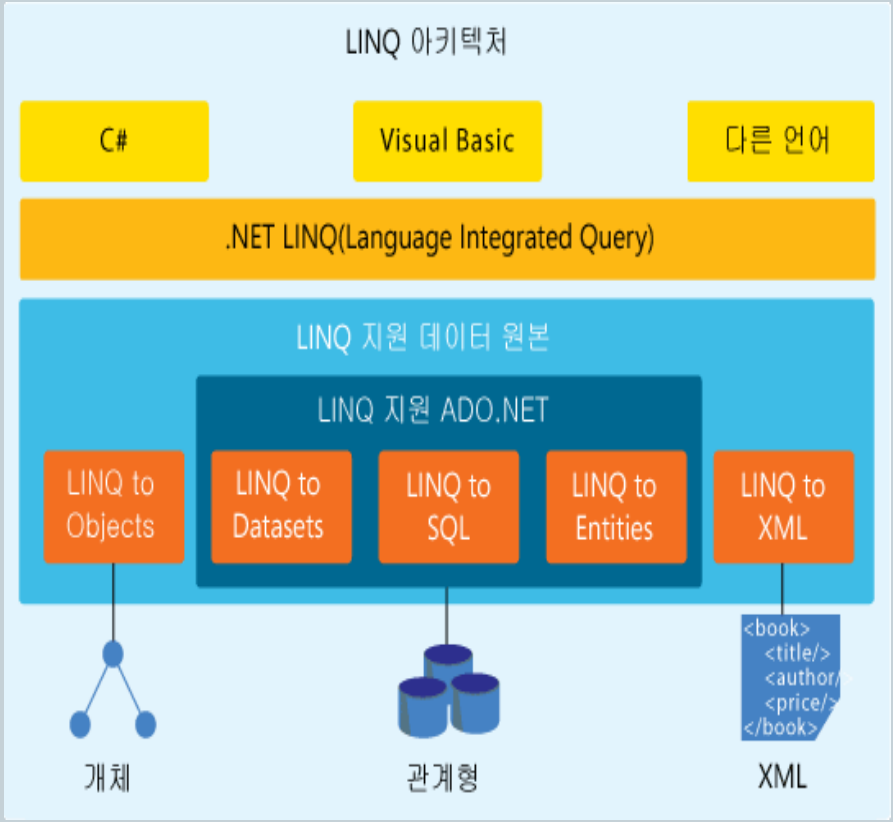
이 수 겸
@올랩컨설팅

KENIALLEE_AT_GMAIL.COM
HTTP://KENIAL.TISTORY.COM

2007.10.6

LINQ

- .NET Language-INtegrated Query
- .NET 프레임워크 3.5, Visual Studio 2008(Orcas) 적용
- 관계형 데이터베이스나 XML 기반의 데이터를 다루는 기능을 프로그래밍 언어나 런타임에 추가하는 대신에, 모든 정보의 소스에 접근할 수 있는 범용의 조회 기능을 .NET 프레임워크에 추가하는 프로젝트
- 일종의 언어 확장(Language-Extension)이며, XML(LINQ to XML), 데이터베이스(LINQ to SQL, LINQ to Dataset 및 LINQ to Entities를 포함하는 LINQ 지원 ADO.NET), 개체(LINQ to Objects) 등의 타입을 다룸



표준 쿼리 연산자

- Standard Query Operators
- Array의 경우 :

```
static void Main() {
   string[] names = { "Burke", "Connor", "Frank",
                      "Everett", "Albert", "George",
                      "Harris", "David" };

   IEnumerable<string> query = from s in names
                               where s.Length == 5
                               orderby s
                               select s.ToUpper();

   foreach (string item in query)
     Console.WriteLine(item);
```

  - query 변수는 쿼리 표현식(query expression)으로 초기화된다. where, orderby, select가 나타나는 것을 볼 수 있다.

표준 쿼리 연산자(cont.)

- VB 코드 :
  - 언어가 다름에도 기본적으로 별 차이가 없다.

```
Dim query As IEnumerable(Of String) =
                From s in names _
                Where s.Length = 5 _
                Order By s _
                Select s.ToUpper()
```

- 람다 식을 사용한 경우:

```
IEnumerable<string> query = names
                        .Where(s => s.Length == 5)
                        .OrderBy(s => s)
                        .Select(s => s.ToUpper());
```

람다 식(Lambda Expressions)

- 기본적으로 C# 2.0의 delegate를 이용한 익명 메서드와 비슷한 성격

```
Func<string, bool>   filter  = s => s.Length == 5;
Func<string, string> extract = s => s;
Func<string, string> project = s => s.ToUpper();

IEnumerable<string> query = names.Where(filter)
                            .OrderBy(extract)
                            .Select(project);
```

- 만약 C# 2.0의 delegate, 익명 메서드를 사용해서 구현한다면 :

```
Func<string, bool>   filter  = delegate (string s) {
                                 return s.Length == 5;
                               };
Func<string, string> extract = delegate (string s) {
                                 return s;
                               };
Func<string, string> project = delegate (string s) {
                                 return s.ToUpper();
                               };
IEnumerable<string> query = names.Where(filter)
                            .OrderBy(extract)
                            .Select(project);
```

확장 메서드(Extension Methods)

- 표준 쿼리 연산자를 직접 확장할 경우 사용

```
namespace System.Linq {
  using System;
  using System.Collections.Generic;

  public static class Enumerable {
    public static IEnumerable<T> Where<T>(
          this IEnumerable<T> source,
          Func<T, bool> predicate) {

      foreach (T item in source)
        if (predicate(item))
          yield return item;
    }
  }
}
```

LINQ to SQL: SQL Integration

- 테이블 생성 쿼리

```
create table People (
    Name nvarchar(32) primary key not null,
    Age int not null,
    CanCode bit not null )

create table Orders (
    OrderID nvarchar(32) primary key not null,
    Customer nvarchar(32) not null,
    Amount int )
```

- 클래스 정의

```
[Table(Name="People")]
public class Person {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string Name;
  [Column]
  public int Age;
  [Column]
  public bool CanCode;  }

[Table(Name="Orders")]
public class Order {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string OrderID;
  [Column(DbType="nvarchar(32) not null")]
  public string Customer;
  [Column]
  public int Amount;   }
```

LINQ to SQL (cont.)

- 구현 코드
  - DataContext는 LINQ와 SQL을 통합할 수 있게 해주는 중간 개체이다.

```
// establish a query context over ADO.NET sql connection
DataContext context = new DataContext(
    "Initial Catalog=petdb;Integrated Security=sspi");

// grab variables that represent the remote tables that
// correspond to the Person and Order CLR types
Table<Person> custs = context.GetTable<Person>();
Table<Order> orders   = context.GetTable<Order>();

// build the query
var query = from c in custs
            from o in orders
            where o.Customer == c.Name
            select new {
                    c.Name,
                    o.OrderID,
                    o.Amount,
                    c.Age
            };

// execute the query
foreach (var item in query)
   Console.WriteLine("{0} {1} {2} {3}",
                item.Name, item.OrderID,
                item.Amount, item.Age);
```

LINQ to SQL (cont.)

- 앞장의 구현 코드는 다음 SQL 구문과 동일 :

```
SELECT [to].[Age], [t1].[Amount],
    [to].[Name], [t1].[OrderID]
FROM [Customers] AS [to], [Orders] AS [t1]
WHERE [t1].[Customer] = [to].[Name]
```

- 암시적으로 형식이 지정된 지역 변수(implicitly typed local variables)
  - 컴파일러가 지역 변수의 타입을 추론함 (Jscript의 var와 비슷)
  - 이런 코드는 에러가 발생 :
  - var 변수는 특정 메서드의 경계를 벗어날 수 없다. (필드, 혹은 매개 변수로 사용 불가)

```
var integer = 1;
integer = "hello";
```

LINQ to XML: XML Integration

- XML 데이터 구현 코드 :

```
var e = new XElement("Person",
                new XAttribute("CanCode", true),
                new XElement("Name", "Loren David"),
                new XElement("Age", 31));
```

- 혹은 이런 코드도 사용할 수 있다 :

```
var e2 = XElement.Load(xmlReader);
var e1 = XElement.Parse(
@"<Person CanCode='true'>
  <Name>Loren David</Name>
  <Age>31</Age>
</Person>");
```

- 실제 XML 데이터

```
<Person CanCode="true">
  <Name>Loren David</Name>
  <Age>31</Age>
</Person>
```

LINQ to XML (cont.)

- LINQ-XML 쿼리 코드 1

```
var query = from p in people
            where p.CanCode
            select new XElement("Person",
                   new XAttribute("Age", p.Age), p.Name);
```

- LINQ-XML 쿼리 코드 2

```
var x = new XElement("People",
                from p in people
                where p.CanCode
                select
                  new XElement("Person",
                  new XAttribute("Age", p.Age), p.Name));
```

표준 쿼리 연산자 목록

Operator	Description
Where	Restriction operator based on predicate function
Select/SelectMany	Projection operators based on selector function
Take/Skip/ TakeWhile/SkipWhile	Partitioning operators based on position or predicate function
Join/GroupJoin	Join operators based on key selector functions
Concat	Concatenation operator
OrderBy/ThenBy/OrderByDescending/ThenByDescending	Sorting operators sorting in ascending or descending order based on optional key selector and comparer functions
Reverse	Sorting operator reversing the order of a sequence
GroupBy	Grouping operator based on optional key selector and comparer functions
Distinct	Set operator removing duplicates
Union/Intersect	Set operators returning set union or intersection
Except	Set operator returning set difference
AsEnumerable	Conversion operator to **IEnumerable<T>**
ToArray/ToList	Conversion operator to **array** or **List<T>**
ToDictionary/ToLookup	Conversion operators to **Dictionary<K,T>** or **Lookup<K,T>** (multi-dictionary) based on key selector function

Operator	Description
OfType/Cast	Conversion operators to **IEnumerable<T>** based on filtering by or conversion to type argument
SequenceEqual	Equality operator checking pairwise element equality
First/FirstOrDefault/Last/LastOrDefault/Single/SingleOrDefault	Element operators returning initial/final/only element based on optional predicate function
ElementAt/ElementAtOrDefault	Element operators returning element based on position
DefaultIfEmpty	Element operator replacing empty sequence with default-valued singleton sequence
Range	Generation operator returning numbers in a range
Repeat	Generation operator returning multiple occurrences of a given value
Empty	Generation operator returning an empty sequence
Any/All	Quantifier checking for existential or universal satisfaction of predicate function
Contains	Quantifier checking for presence of a given element
Count/LongCount	Aggregate operators counting elements based on optional predicate function
Sum/Min/Max/Average	Aggregate operators based on optional selector functions
Aggregate	Aggregate operator accumulating multiple values based on accumulation function and optional seed

Thank you!