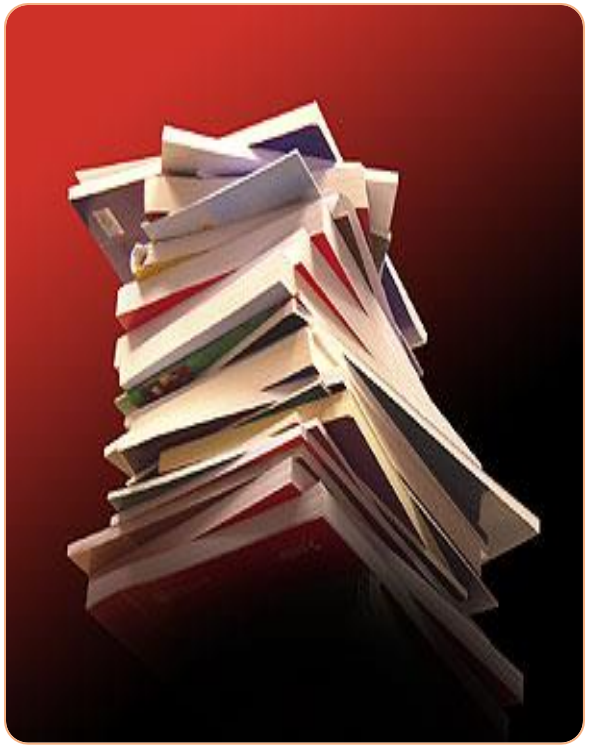
LINQ in C#

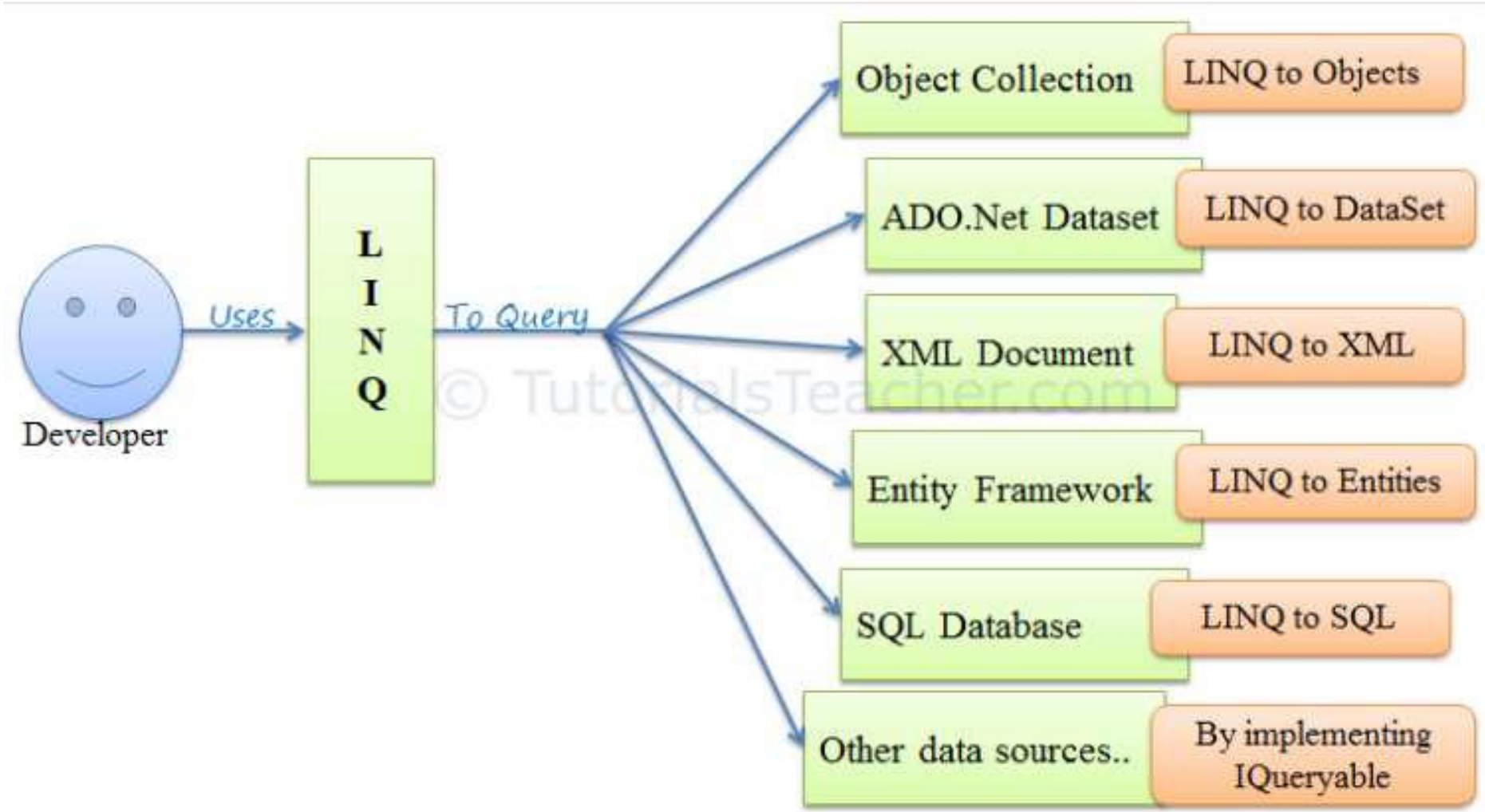
Inhoud

1. Wat is LINQ?
2. Voordelen van LINQ
3. LINQ Operators
   - Filtering Operators
   - Grouping Operators
   - Concatenation
   - Sorting Operators
   - Join Operators
   - Equality
   - Projection Operations
   - Aggregation
   - Quantifier Operations
   - Partition Operations
   - Generation Operations
   - Set Operations
   - Conversions
   - Element Operators



Wat is LINQ?

- Language Integrated Query (LINQ)
- LINQ is a query syntax dat kan gebruikt worden om gegevens te lezen en bewaren van **verschillende types van data sources:**
  - Object Collection
  - SQL server database
  - XML
  - web service
  - …


Developer
Uses
L
I
N
Q
To Query
Object Collection
LINQ to Objects
ADO.Net Dataset
LINQ to DataSet
XML Document
LINQ to XML
Entity Framework
LINQ to Entities
SQL Database
LINQ to SQL
Other data sources..
By implementing IQueryable

LINQ Usage

Voordelen van LINQ

- Syntax zeer compact
  (Query syntax vs Method syntax)
- Gemakkelijk te debuggen
- Extensible: uitbreidbaar, mogelijk LINQ op nieuw soorten datasources te gebruiken.
- Gemakkelijk om verschillende datasources te combineren (joining) in één enkele LINQ query
- Gemakkelijk om transformatie toe te passen
  (bv. transformatie van SQL data naar XML data.)

LINQ Operatoren

1. Toepassingen van Linq
2. Restriction/Filtering operators
3. Projection operators
4. Aggregate operators
5. Conversions
6. Element Operators
7. Generators
8. Grouping Operators
9. Join Operators
10. Sorting/ordering Operators
11. Partition Operations
12. Quantifier Operations
13. Sequence operations
14. Set Operations
15. Query Excecution (deferred vs immediate)

Toepassingen van Linq

- **Linq to objects**

  Linq queries op collections/arrays van objects

- **Linq to XML**

  Queries on XML data en XML documents

- **Linq to DataSet**

  Toepassen van Linq queries op ADO.NET DataSet objecten

- **Linq to Entities**

  Linq queries voor ADO.Net Entity Framework API

- **Parallel Linq (PLINQ)**

  Parallelle verwerking van data die teruggegeven door een Linq query

Linq to Objects

Linq to Objects

- Linq queries op collections/arrays van objects
- Assembly: System.Core.dll
- Gebruik namespace: **using System.Linq**

Voorbeeld:

```
public class Product {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public string Category { get; set; }
        public decimal UnitPrice { get; set; }
        public int UnitsInStock { get; set; }
         public override string ToString() => $"ProductID={ProductID} ProductName={ProductName} Category={Category}
UnitPrice={UnitPrice:C2} UnitsInStock={UnitsInStock}";    }

public static class Products{

public static List<Product> ProductList { get;  } =  new List<Product> {

new Product { ProductID = 1, ProductName = "Chai",Category = "Beverages", UnitPrice = 18.0000M, UnitsInStock = 39 },

new Product { ProductID = 2, ProductName = "Chang",Category = "Beverages", UnitPrice = 19.0000M, UnitsInStock = 17 },

new Product { ProductID=3, ProductName ="Aniseed Syrup",Category="Condiments",UnitPrice=10.0000M, UnitsInStock=13 },

new Product { ProductID = 4, ProductName="Chef Anton's Cajun Seasoning",Category= "Condiments", UnitPrice = 22.0000M,
UnitsInStock = 53 },

new Product { ProductID = 5, ProductName = "Chef Anton's Gumbo Mix", Category = "Condiments", UnitPrice = 21.3500M,
UnitsInStock = 0 }};
```

Linq to Objects

Voorbeeld (vervolg):

```
var categories =
from p in products
group p by p.Category into g
select (Category: g.Key, MostExpensivePrice: g.Max(p => p.UnitPrice));


foreach (var c in categories)
{
    Console.WriteLine($"Category: {c.Category} Most expensive product:
    {c.MostExpensivePrice}");
}
```

Linq to XML

Linq to XML

- Queries on XML data en XML documents
- Assembly: System.Xml.Linq.dll
- Gebruik namespace: **using System.Xml.Linq;**

Voorbeeld:

```
public static class Customers{

       public static List<Customer> CustomerList { get; } =

           (from e in XDocument.Parse(InputValues.CustomersXml).Root.Elements("customer")

            select new Customer{

                CustomerID = (string)e.Element("id"),  CompanyName = (string)e.Element("name"),

                Address = (string)e.Element("address"),

                City = (string)e.Element("city"), Region = (string)e.Element("region"),

                PostalCode = (string)e.Element("postalcode"),

                Country = (string)e.Element("country"), Phone = (string)e.Element("phone"),

                Orders = (

                   from o in e.Elements("orders").Elements("order")

                   select new Order{

                       OrderID = (int)o.Element("id"), OrderDate = (DateTime)o.Element("orderdate"),

                       Total = (decimal)o.Element("total")

                   }).ToArray()

           }).ToList();    }
```

Voorbeeld (vervolg):

```
List<Product> products = GetProductList();


Product product12 =
   (from p in products
   where p.ProductID == 12
   select p).First();

   Console.WriteLine(product12);
```

Linq - Restriction/Filtering operators

Where clause

Voorbeeld:

```
List<Product> producten = GetProductList();

var DureInStockProducten =
from prod in producten
where prod.UnitsInStock > 0 && prod.UnitPrice > 3.00M
select prod;

Console.WriteLine("In-stock producten duurder dan 3.00:");
foreach (var product in DureInStockProducten)
{
    Console.WriteLine($"{product.ProductName} is in en is duurder
    dan 3.00.");
}
```

Linq – Projection operators

Select clause

Voorbeeld 1:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var getallenPlus1 = from n in getallen
                    select n + 1;

Console.WriteLine("Getallen + 1:");
            foreach (var i in getallenPlus1)
{
                Console.WriteLine(i);
}
```

Linq – Projection operators (vervolg)

Select clause

Voorbeeld 2:

```
List<Product> producten = GetProductList();

var productNamen = from p in producten
                   select p.ProductName;

Console.WriteLine("Product Namen:");
foreach (var productNaam in productNamen)
{
     Console.WriteLine(productNaam);
}
```

Linq – Projection operators (vervolg 2)

Select clause

Voorbeeld 3:

```
string[] woorden = { "aPPEL", "BANaaN", "KeRS" };

var upperLowerWoorden = from w in woorden
select new{ Upper= w.ToUpper(),Lower = w.ToLower() };

foreach (var w in upperLowerWords)
{
   Console.WriteLine($"Uppercase: {w.Upper},
   Lowercase: {w.Lower}");
}
```

Linq - Aggregate operators

1. Count
2. Sum
3. Min
4. Max
5. Average
6. Aggregate

Linq - Aggregate operators – 1. Count

Count Operator

Voorbeelden:

```
int[] factors = { 2, 2, 3, 5, 5 };
int uniekeFactors = factors.Distinct().Count();
Console.WriteLine($"E zijn {uniekeFactors} unieke
factors");

int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int aantalOneven = getallen.Count(n => n % 2 == 1);
Console.WriteLine("Er zijn {0} oneven getallen in
de lijst", aantalOneven);
```

Linq - Aggregate operators – 2. Sum

Sum Operator

Voorbeelden:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
double som = getallen.Sum();
Console.WriteLine($"De som van de getallen is
{som}");

string[] woorden = { "kers", "appel", "banaan" };
double aantalLetters = woorden.Sum(w => w.Length);
Console.WriteLine($"In total zijn er {aantalLetters}
letters in alle woorden");
```

Linq - Aggregate operators – 3. Min

Min Operator

Voorbeelden:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int minGetal = getallen.Min();
Console.WriteLine($"Het minimum getal is
{minGetal}");



string[] woorden= { "kers", "appel", "banaan" };
int kortste = woorden.Min(w => w.Length);
Console.WriteLine($"Het kortste word bevat {kortste}
letters.");
```

Linq - Aggregate operators – 3. Max

Max Operator

Voorbeelden:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int MaxGetal = numbers.Max();

string[] woorden= { "kers", "appel", "banaan" };
int langste = woorden.Max(w => w.Length);
Console.WriteLine ($"Het kortste word bevat {langste}
letters.");
```

Linq - Aggregate operators – 3. Average

Average Operator

Voorbeelden:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
double gemiddelde = getallen.Average();
Console.WriteLine($"Het gemiddelde is
{gemiddelde}.");

List<Product> producten = GetProductList();
var categories = from p in producten
                 group p by p.Category into g
select (Category: g.Key, AveragePrice: g.Average(p =>
p.UnitPrice));
foreach (var c in categories){
   Console.WriteLine($"Categorie: {c.Category},
   Gemiddelde prijs: {c.AveragePrice}");}
```

Linq - Aggregate operators – 4. Aggregate

Aggregate Operator

Voorbeelden:

```
double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };
double product = doubles.Aggregate((runningProduct,
nextFactor) => runningProduct * nextFactor);
Console.WriteLine($"Totaal product van alle getallen:
{product}");
double startSaldo = 100.0;
int[] debetBedragen = { 20, 10, 40, 50, 10, 70, 30 };
double eindSaldo =
debetBedragen.Aggregate(startSaldo,
(saldo, debetbedrag) => ((debetbedrag <= saldo)
?(saldo - debetbedrag) : saldo));
Console.WriteLine($"Eindsaldo: {eindSaldo}");
```

Linq - Conversies

1. ToArray
2. ToList
3. ToDictionary
4. OfType<T>

Linq – conversies – 1. ToArray

ToArray

Voorbeeld:

```
double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };
var gesoorteerdeDoubles = from d in doubles
                          orderby d descending
                          select d;
var doublesArray = gesoorteerdeDoubles.ToArray();
```

Linq – conversies – 2. ToList

ToList

Voorbeeld:

```
 string[] words = { "cherry", "apple", "blueberry" };

var sortedWords = from w in words
                  orderby w
                  select w;
var wordList = sortedWords.ToList();
```

Linq – conversies – 3. ToDictionary

ToDictionary

Voorbeeld:

```
var scores = new[] {
new {Name = "An", Score = 50},
new {Name = "Piet"  , Score = 40},
new {Name = "Jos", Score = 45}};

var scoresDict =
scores.ToDictionary(sr => sr.Name);

Console.WriteLine("Score van Piet: {0}",
Dict["Piet"]);
```

Linq – conversies – 4. OfType<T>

OfType<T>

Voorbeeld:

```
object[] objecten = { null, 1.0, "two", 3, "four", 5,
"six", 7.0 };
var doubles = objecten.OfType<double>();
Console.WriteLine("alle doubles uit lijst:");
foreach (var d in doubles)
{
     Console.WriteLine(d);
}
```

Linq – Element Operations

1. First
2. FirstOrDefault
3. ElementAt

Linq - Element operaties – 1. First

First

Voorbeelden:

```
List<Product> producten = GetProductList();
Product product12 = (from p in producten
                     where p.ProductID == 12
                     select p).First();
Console.WriteLine(product12);

string[] strings = { "nul", "een", "twee", "drie",
"vier", "vijf", "zes", "zeven", "acht", "negen" };
string startMetT = strings.First(s => s[0] =='t');
Console.WriteLine($"Eerste die start met 't':
{startMetT}");
```

Linq - Element operaties – 2. FirstOrDefault

FirstOrDefault

Voorbeelden:

```
int[] getallen = { };
int eersteOfDefault = getallen.FirstOrDefault();
Console.WriteLine(eersteOfDefault);

List<Product> producten = GetProductList();
Product product789 = products.FirstOrDefault(p =>
p.ProductID == 789);
Console.WriteLine($"Product 789 bestaat:{product789
!= null}");
```

Linq - Element operaties – 3. ElementAt

ElementAt

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int tweedeGetalGroterDan5 = (from n in getallen
                  where n > 5
                  select n)
                  .ElementAt(1);
// het tweede element heeft index =1 (0-based
//indexing)

Console.WriteLine($"Tweede getal > 5:
{tweedeGetalGroterDan5}");
```

Linq – Generators

1. **Enumerable.Range**
2. **Enumerable.Repeat**

Linq – Generators – 1. Enumerable.Range

Enumerable.Range

Voorbeeld:

```
var getallen = from n in Enumerable.Range(100,
50)
select (Getal: n, EvenOneven: n % 2 == 1 ?
"oneven" : "even");

foreach (var n in numbers)
{
   Console.WriteLine("Het getal {0} is {1}.", n.
   Getal, n. EvenOneven);
}
```

Linq – Generators – 2. Enumerable.Repeat

Enumerable.Repeat

Voorbeeld:

```
var getallen = Enumerable.Repeat(7, 10);

foreach (var g in getallen)
{
      Console.WriteLine(g);
}
```

Lambda Expressions en LINQ

Vragen?

Referenties

- Telerik Software Academy
  - https://www.telerikacademy.com/

https://docs.microsoft.com/en-us/dotnet/api/system.linq?view=netcore-3.0