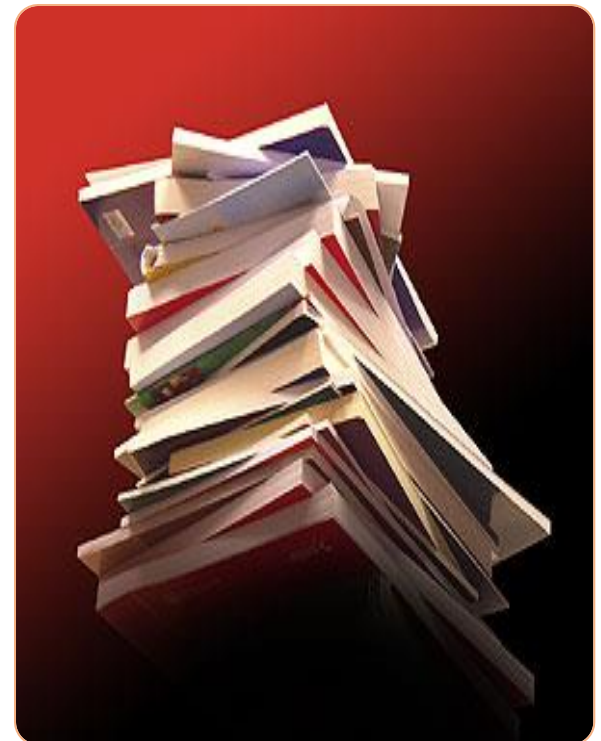


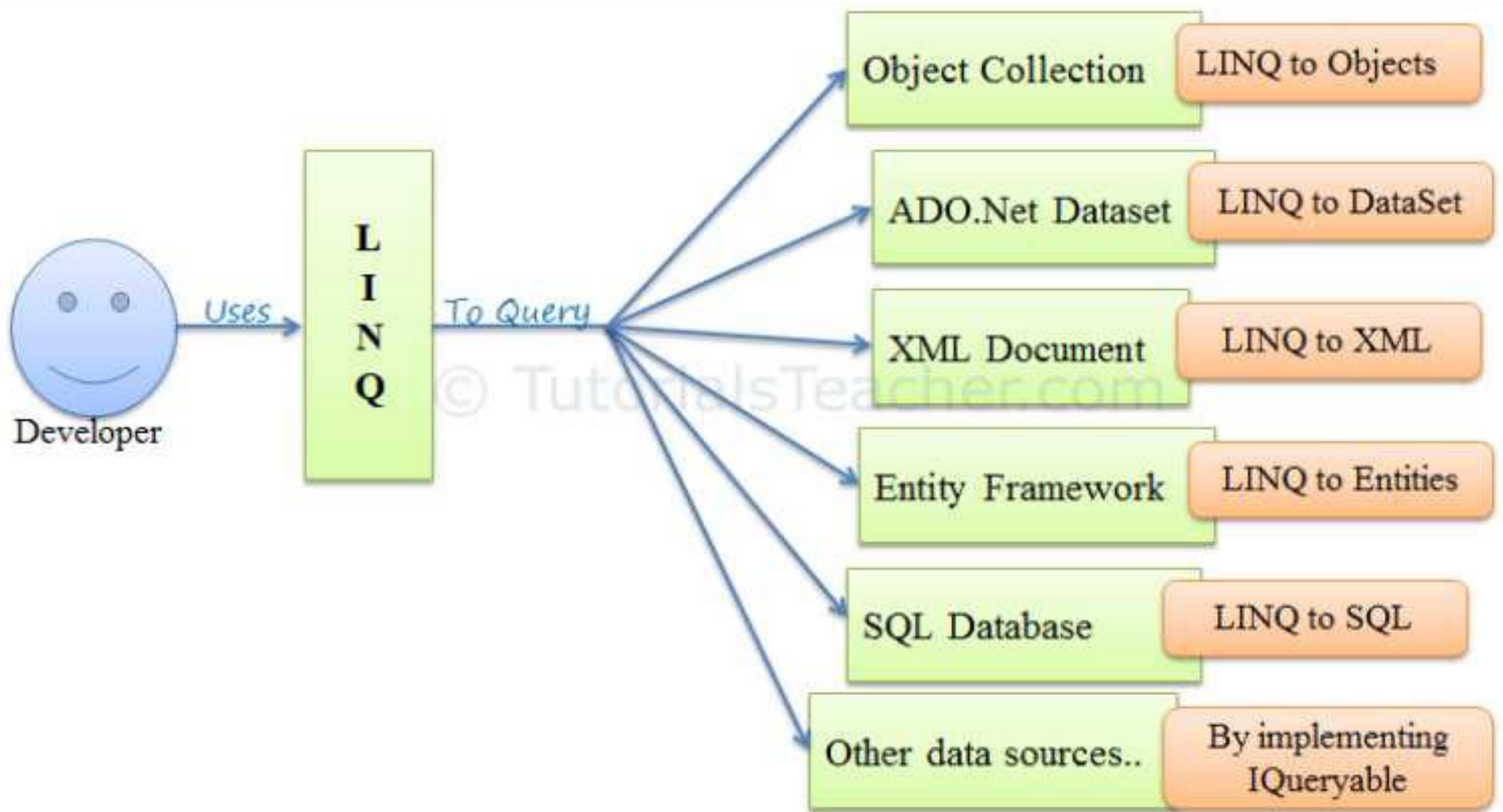
LINQ in C#



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





LINQ Usage

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)**

Parallele verwerking van data die teruggegeven door
een Linq query

Linq – Sorteerooperatoren

1. Orderby ...
2. Orderby ...descending
3. Orderby...ThenBy
4. Reverse

Linq – sorteeroperators- 1.OrderBy

Voorbeeld:

```
string[] woorden = { "kers", "appel", "banaan" };
```

```
var GesorteerdeWoorden = from woord in woorden  
                          orderby woord.Length  
                          select woord;
```

```
List<Product> products = GetProductList();  
var sortedProducts = from prod in products  
                     orderby prod.ProductName  
                     select prod;
```

Linq – sorteeroperators-

2. Orderby ...descending

Voorbeeld:

```
double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };
```

```
var gesorteerdeDoubles = from d in doubles  
                          orderby d descending  
                          select d;
```

```
List<Product> products = GetProductList();  
var sortedProducts = from prod in products  
                     orderby prod.UnitsInStock descending  
                     select prod;
```

Linq - sorteeroperatoren- 3. Orderby

...ThenBy

```
string[] cijfers = { "nul", "een", "twee", "drie", "vier",  
"vijf", "zes", "zeven", "acht", "negen" };  
var gesorteerdOpLengte= from cijfer in cijfers  
                        orderby cijfer.Length, cijfer  
                        select cijfer;  
  
// Custom comparer voor sorteeroperator kan zelf gedefinieerd  
//worden:  
public class CaseInsensitiveComparer : IComparer<string>{  
    public int Compare(string x, string y) =>  
        string.Compare(x, y, StringComparison.OrdinalIgnoreCase);}  
// gebruik van eigen Custom comparer:  
string[] woorden = { "aPPEL", "AbAcUs", "bRaNcH", "BosBes",  
"CiTRoEN", "keRSen" };  
  
var gesorteerdeWoorden = woorden  
                        .OrderBy(w => w.Length)  
                        .ThenBy(w => w, new CaseInsensitiveComparer());
```


Linq – sorteer- operatoren- 4. Reverse

Voorbeeld:

```
string[] cijfers = { "nul", "een", "twee", "drie",  
"vier", "vijf", "zes", "zeven", "acht", "negen" };  
  
var cijfersOmgekeerd = (  
    from cijfer in cijfers  
    where cijfers[1] == 'e'  
    select cijfers)  
    .Reverse();  
Console.WriteLine("Cijfers met tweede karakter =  
'e' in omgekeerde volgorde:");  
foreach (var c in cijfersOmgekeerd){  
    Console.WriteLine(c);  
}
```

Linq – Partitie operatoren

1. Take
2. Skip
3. TakeWhile
4. SkipWhile

Linq - partitie operators- 1.Take

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
var eerste3 = getallen.Take(3);  
Console.WriteLine("Eerste 3 getallen:");  
foreach (var g in eerste3){  
    Console.WriteLine(g);  
}
```

```
List<Customer> customers = GetCustomerList();  
var eerste2USbestellingen = ( from cust in customers  
    from order in cust.Orders  
    where cust.Region == "US"  
    select (cust.CustomerID, order.OrderID,  
order.OrderDate)).Take(3);
```

Linq - partitie operators- 2. Skip

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var alleBehalveEerste4 = getallen.Skip(4);
Console.WriteLine("Alle behalve de eerste 4 getallen:");
foreach (var n in alleBehalveEerste4){
    Console.WriteLine(n);
}

List<Customer> customers = GetCustomerList();
var usOrders = from cust in customers
               from order in cust.Orders
               where cust.Region == "US"
               select (cust.CustomerID, order.OrderID,
                       order.OrderDate);
var alleBehalveEerste2 = usOrders.Skip(2);
Console.WriteLine("Alle bestellingen behalve de eerste 2
orders in US regio:");
foreach (var order in alleBehalveEerste2) {
    Console.WriteLine(order); }
```

Linq - partitie operators- 3. TakeWhile

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
Console.WriteLine("Eerste getallen Kleiner dan 6:");  
var eersteKleinerDan6 = getallen.TakeWhile(n => n < 6);
```

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
Console.WriteLine("Eerste getallen niet Kleiner dan hun  
positie(index):");
```

```
var eersteKleinerDan6 =  
getallen.TakeWhile((n, index) => n >= index);
```

Linq - partitie operators- 4. SkipWhile

Voorbeeld:

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

```
Console.WriteLine("Alle elementen startend van het eerste  
drievoud:");
```

```
var allButFirst3Numbers = getallen.SkipWhile(n => n % 3 != 0);
```

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

```
Console.WriteLine(" Alle elementen startend van het eerste dat  
Kleiner is dan zijn positie(index):");
```

```
var laterNumbers = getallen.SkipWhile((n, index) => n >= index);
```

Linq – Quantificeerders

1. Any
2. All

Linq – Quantificeerder – 1. Any

Voorbeeld:

```
string[] woorden = { "eileider", "plezier", "reis" };  
bool iNaE = woorden.Any(w => w.Contains("ei"));
```

```
List<Product> products = GetProductList();  
var productGroups = from p in products  
                     group p by p.Category into g  
                     where g.Any(p => p.UnitsInStock == 0)  
                     select (Category: g.Key, Products: g);
```

```
foreach(var group in productGroups){  
    Console.WriteLine(group.Category);  
    foreach(var product in group.Products){  
        Console.WriteLine($"{t{product}");  
    }  
}
```


Linq – Quantificeerder – 2. All

Voorbeeld:

```
int[] getallen = { 1, 11, 3, 19, 41, 65, 19 };  
bool enkelOnevenGetallen = getallen.All(n => n % 2 == 1);  
Console.WriteLine($"De lijst bevat al de oneven getallen:  
{enkelOnevenGetallen}");  
  
List<Product> products = ProductList;  
bool areAllproductsInStock = products.All(p => p.UnitsInStock > 0);  
Console.WriteLine("Are all products in stock ? " +  
areAllproductsInStock);
```

Linq – Query Executie

1. Uitgestelde (Deferred)
2. Onmiddellijke (Immediate) (Eager)

Linq – Query executie - Uitgestelde

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int i = 0;
var q = from n in getallen
        select ++i;
// de locale variabele 'i' wordt niet
//geïncrementeed totdat elk element is
//geëvalueerd (hier in de foreach) :
foreach (var v in q)
{
    Console.WriteLine($"v = {v}, i = {i}");
}
```

Linq – Query executie - Onmiddellijke

Voorbeeld:

```
// Methoden zoals ToList(), ToArray() zorgen
//ervoor dat de query onmiddellijk wordt
//uitgevoerd en de resultaten worden gecached.
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0
};
int i = 0;
var q = (from n in getallen
        select ++i).ToList();
// De locale variabele i wordt reeds volledig
//geïncrementeed voordat we de resultaten
//itereren via foreach:
foreach (var v in q){
    Console.WriteLine($"v = {v}, i = {i}");
}
```

Linq – Groepeer-operators

1. GroupBy
2. GroupBy **x** by **y** into **z**

Linq - groepeeroperators –

2. Group x by y into z

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var getallenGroepen = from getal in getallen
    group getal by getal % 5 into groep
    select (RestBijDeling: groep.Key, Getallen:
        groep);

foreach (var groep in getallenGroepen){
    Console.WriteLine($"Getallen met rest {groep.
RestBijDeling} bij deling door 5:");
    foreach (var getal in groep. Getallen){
        Console.WriteLine(getal);
    }
}
```

Linq - groepeeroperators – 1. groupby

GroupBy

```
var queryNotatie = from dier in dieren
group dier by Math.Floor(dier.Leeftijd) into groep
select new
{
    DierKey = groep.Key,
    Count = groep.Count(),
    Min = groep.Min(d => d.Leeftijd),
    Max = groep.Max(d => d.Leeftijd)
};
```

```
var methodNotatie = dieren.GroupBy(dier =>
Math.Floor(dier.Leeftijd),
    dier => dier.Leeftijd,
    (LeeftijdKey, leeftijden) =>
new { Key = LeeftijdKey,
    Count = leeftijden.Count(),
    Min = leeftijden.Min(),
    Max = leeftijden.Max() });
```

//vervolg: zie volgende slide

Linq - groepeeroperators – 1. groupby

GroupBy (vervolg 1)

```
class Dier:IComparable<Dier>
{
    public string Name { get; set; }
    public double Leeftijd { get; set; }

    public int CompareTo([AllowNull] Dier other)
    {
        if (Leeftijd == other.Leeftijd)
            return 0;
        else if (Leeftijd < other.Leeftijd)
            return -1;
        else
            return 1;
    }
}
```

//vervolg: zie volgende slide

Linq - groepeeroperators – 1. groupby

GroupBy (vervolg 2)

```
foreach (var groep in queryNotatie)
{
    Console.WriteLine(groep.DierKey + ":");
    Console.WriteLine("\taantal: " + groep.Count);
    Console.WriteLine("\tMin: " + groep.Min);
    Console.WriteLine("\tMax: " + groep.Max);
}
```

Linq – Quantificeerder – 2. All

Voorbeeld:

```
List<Product> products = ProductList;
    var productGroups = from p in products
                        group p by p.Category into g
                        where g.All(p => p.UnitsInStock > 0)
                        select (Category: g.Key, Products: g);

//Tuple, geen anonymous type!

    foreach (var group in productGroups)
    {
        Console.WriteLine(group.Category);
        foreach (var product in group.Products)
        {
            Console.WriteLine($"{\t{product}");
        }
    }
```

Linq – Join operator – group-join

Voorbeeld:

```
string[] categories = {"Beverages", "Condiments",  
                        "Vegetables", "Dairy Products",  
                        "Seafood"};
```

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

```
var q = from c in categories
```

```
    join p in products on
```

```
    c equals p.Category into ps
```

```
select (Category: c, Products: ps);
```

```
foreach (var v in q){
```

```
    Console.WriteLine(v.Category + ":");
```

```
    foreach (var p in v.Products){
```

```
        Console.WriteLine("        " + p.ProductName);} }
```

Linq – Join operator

1. Inner join
2. Group join

Linq – Join operator – inner join

Join

Voorbeeld:

```
string[] categories = {  
    "Beverages",  
    "Condiments",  
    "Vegetables",  
    "Dairy Products",  
    "Seafood"};
```

```
List<Product> products = GetProductList();  
var q = from c in categories  
        join p in products on c equals p.Category  
        select (Category: c, p.ProductName);
```

Linq – Join operator – group-join

Voorbeeld:

```
string[] categories = {"Beverages", "Condiments",  
                        "Vegetables", "Dairy Products",  
                        "Seafood"};
```

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

```
var q = from c in categories
```

```
    join p in products on
```

```
    c equals p.Category into ps
```

```
select (Category: c, Products: ps);
```

```
foreach (var v in q){
```

```
    Console.WriteLine(v.Category + ":");
```

```
    foreach (var p in v.Products){
```

```
        Console.WriteLine("        " + p.ProductName);} }
```

Linq – Join operator – group-join

Voorbeeld:

```
//GroupJoin met Method syntax (zelfde resultaat als  
//query van vorige slide):
```

```
string[] categories = {"Beverages", "Condiments",  
                       "Vegetables", "Dairy Products",  
                       "Seafood"};
```

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

```
var q = categories.GroupJoin(products, //inner sequence  
    cat => cat, //outerKeySelector  
    prod => prod.Category, //innerKeySelector  
    (cat, prodGroep) => new // resultSelector  
    {  
        Products = prodGroep,  
        Naam = cat  
    }  
));
```

Linq – Set Operatie - Union

Voorbeeld:

```
ICollection<int> getallen1 = new List<int>() { 1,2,3,4};  
ICollection<int> getallen2 = new List<int>() { 2,3,4,5 };  
  
var resultaat = getallen1.Union(getallen2);  
Console.WriteLine("De unieke elementen uit de 2  
lijsten getallen:");  
foreach (int getal in resultaat)  
    Console.Write(getal + " ");
```


Linq – Set Operatie - Intersect

Voorbeeld:

```
IList<int> getallen1 = new List<int>() { 1, 2, 3, 4 };  
IList<int> getallen2 = new List<int>() { 2, 3, 4, 5 };  
var resultaat = getallen1.Intersect(getallen2);
```

```
Console.WriteLine("De gemeenschappelijke elementen uit de 2  
lijsten getallen:");  
foreach (int getal in resultaat)  
    Console.Write(getal + " ");
```

Linq – Set Operatie - Distinct

Voorbeeld:

```

IList<string> fruit =
new List<string>() { "appel", "citroen", "banaan", "appel",
"banaan" };

IList<int> getallen =
    new List<int>() { 1, 2, 3, 2, 4, 4, 3, 5 };

var distinctFruit = fruit.Distinct();

foreach (var str in distinctFruit)
    Console.WriteLine(str);

var distinctgetallen = getallen.Distinct();

foreach (var i in distinctgetallen)
    Console.WriteLine(i);

```

Linq – Set Operatie - Intersect

Voorbeeld:

```
ICollection<int> getallen1 = new List<int>() { 1, 2, 3, 4 };  
ICollection<int> getallen2 = new List<int>() { 2, 3, 4, 5 };  
  
var resultaat = getallen1.Intersect(getallen2);  
Console.WriteLine("Getallen die in de 1ste lijst voorkomen én  
niet in de 2de");  
foreach (int getal in resultaat)  
    Console.WriteLine(getal);
```

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>

