

LINQ – Teil 3

Ingo Köster

Diplom-Informatiker (FH)

LINQ Abfrageoperatoren Teil 1

Operatortyp	Operator
Aggregatoperatoren	Aggregate, Average, Count, LongCount, Min, Max, Sum
Casting-Operatoren	Cast, OfType, ToArray, ToDictionary, ToList, ToLookup, ToSequence
Elementoperatoren	DefaultIfEmpty, ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Gleichheitsoperatoren	EqualAll
Sequenzoperatoren	Empty, Range, Repeat
Gruppierungsoperatoren	GroupBy
Join-Operatoren	Join, GroupJoin

LINQ Abfrageoperatoren Teil 2

Operatortyp	Operator
Sortieroperatoren	OrderBy, ThenBy, OrderByDescending, ThenByDescending, Reverse
Aufteilungsoperatoren	Skip, SkipWhile, Take, TakeWhile
Quantifizierungsoperatoren	All, Any, Contains
Restriktionsoperatoren	Where
Projektionsoperatoren	Select, SelectMany
Set-Operatoren	Concat, Distinct, Except, Intersect, Union

Casting-Operatoren

```
› int[] zahlen = { 14, 42, 23, 19, 18, 47 };
```

```
› // in Array casten
```

```
› int[] sortiertesArray = zahlen.OrderBy(x => x).ToArray();
```

```
› // in List<T> casten
```

```
› List<int> sortierteListe = zahlen.OrderBy(x =>  
    x).ToList();
```

Casting-Operatoren

```
class Daten {  
    public int Id { get; set; }  
    public string Name { get; set; }  
}
```

```
› List<Daten> daten = new List<Daten> { ... };
```

```
› // Id wird zum Schlüssel
```

```
› Dictionary<int, Daten> datenDict = daten.ToDictionary(d =>  
    d.Id);
```

Mengenoperationen

› `int[] teil1 = { 1, 2, 3 }, teil2 = { 3, 4, 5 };`

› `teil1.Concat(teil2);` `// 1,2,3,3,4,5`

› `teil1.Union(teil2);` `// 1,2,3,4,5`

› `teil1.Intersect(teil2);` `// 3`

› `teil1.Except(teil2);` `// 1,2`

› `teil2.Except(teil1);` `// 4,5`

Mengenoperationen

- › Unterschiedliche Elemente aus einer Sequenz ermitteln
 - › `int[] werte = { 21, 46, 46, 55, 17, 21, 55, 55 };`
 - › `werte.Distinct();` `// 21, 46, 55, 17`
- › Für benutzerdefinierte Datentypen muss die generische Schnittstelle `IEnumerable<T>` in der Klasse oder Struktur implementiert werden

Gruppierung - Beispieldaten

```
class Daten {  
    public int Id { get; set; }  
    public string Name { get; set; }  
}
```

```
List<Daten> alleDaten = new List<Daten>  
{  
    new Daten { Id=1, Name="Rot"},  
    new Daten { Id=1, Name="Grün"},  
    new Daten { Id=2, Name="Leicht"},  
    new Daten { Id=2, Name="Schwer"}  
};
```


Gruppierungsoperatoren

› Gruppiert Elemente einer Sequenz entsprechend einer angegebenen Schlüsselauswahlfunktion

// Nach ID gruppieren

```
var idGruppen = alleDaten.GroupBy(d => d.Id);
```

```
foreach (var gruppe in idGruppen) {
```

```
    // Daten.Id ist in Key gespeichert
```

```
    Console.WriteLine("ID '{0}':", gruppe.Key);
```

```
    // gruppe -> Element-Collection mit gleicher ID
```

```
    foreach (Daten eintrag in gruppe)
```

```
    {
```

```
        Console.WriteLine(eintrag);
```

```
    }
```

```
}
```

1. Gruppe
Rot, Grün

2. Gruppe
Leicht, Schwer

Gruppierungsoperatoren - Beispieldaten

```
List<Daten> alleDaten = new List<Daten> {  
    new Daten { Id=1, Name="Rot"},  
    new Daten { Id=1, Name="Grün"},  
    new Daten { Id=2, Name="Leicht"},  
    new Daten { Id=2, Name="Schwer"} };
```

› Gruppiert:

ID '1':

Rot

Grün

ID '2':

Leicht

Schwer

Gruppierungsoperatoren - IGrouping

- › Die Ergebnismenge einer GroupBy Klausel ist vom Typ IGrouping
- › Ist eine Auflistung von Objekten, welche über einen gemeinsamen Schlüssel verfügen

```
var iDGruppen = alleDaten.GroupBy(d => d.Id);
```
- › Anschließend ist iDGruppen als <int,Daten> typisiert
- › Jedes Element von iDGruppen verfügt über das Property Key
- › Jedes Element von iDGruppen ist eine Collection aus Objekten (in diesem Beispiel Objekte der Klasse Daten)

IGrouping

› Die IGrouping Objekte einer GroupBy Klausel müssen nicht zwingend von einer Schleife durchlaufen werden

› Beispiele:

```
IGrouping<int, Daten> ersteGruppe = iDGruppen.First();
```

```
IGrouping<int, Daten> mitIdZwei = iDGruppen.Single(g => g.Key == 2);
```

Gruppierungsoperatoren

› Ein Key kann auch aus den zu gruppierenden Daten gebildet werden, falls dieser nicht vorhanden ist

```
string[] alleWörter = { "blueberry", "chimpanzee", "abacus", "banana",  
"apple", "cheese" };
```

```
var g = alleWörter.GroupBy(wort => wort[0]);
```

```
foreach (var gruppe in g) {  
    Console.WriteLine("'{0}':", gruppe.Key);  
    foreach (var wort in gruppe)  
    {  
        Console.WriteLine("{0}", wort);  
    }  
}
```

ElementSelector

- › Beim Gruppieren müssen nicht zwingend ganze Objekt gruppiert werden
- › Mittels des ElementSelectors können Teile eines Objektes ausgewählt werden

```
List<Person> allePersonen = new List<Person> {  
    new Person { Vorname = "Tim", Nachname = "Schulz", Alter = 42,  
    Ort = "Dortmund" },  
    new Person { Vorname = "Pia", Nachname = "Müller", Alter = 38,  
    Ort = "Bochum" },  
};
```

- › Gruppiert „ganze“ Objekte nach Ort

```
allePersonen.GroupBy(person => person.Ort)
```

ElementSelector

› `List<Person> allePersonen = ...`

1. Gruppiert „ganze“ Objekte


`allePersonen.GroupBy(person => person.Ort)`

2. Gruppiert mit **ElementSelector**

`allePersonen.GroupBy(p => p.Ort, p => p.Nachname);`

3. Gruppiert mit **ElementSelector**, jedoch wie 1.

`allePersonen.GroupBy(p => p.Ort, p => p);`



```
'Dortmund'  
Schulz  
Schmidt  
Klein  
  
'Bochum'  
Müller  
Meyer  
  
'Essen'  
Scholz  
Wulf
```

Gruppierungsoperatoren

- › Aus den zu gruppierenden Daten kann eine gruppierte Menge aus anonymen Typen gebildet werden
- › Im nächsten Beispiel wird ein anonymer Typ aus dem Anfangsbuchstaben eines Wortes und den Wörtern mit den gleichen Anfangsbuchstaben erstellt
- › Die Strings werden nach dem Anfangsbuchstaben gruppiert

Gruppierungsoperatoren

```
string[] alleWörter = { "blueberry", "chimpanzee", "abacus", "banana",  
"apple", "cheese" };  
  
var wörterGruppen =  
    alleWörter.GroupBy(  
        wort => wort[0],           // Key (Anfangsbuchstabe)  
        wort => wort,             // Value (Wörter mit gleichem Key)  
        //(Key, Value)  
        (buchstabe, dieWörter) => new  
        {  
            Anfangsbuchstabe = buchstabe, // Dient als Key (z.B. a)  
            Wörter = dieWörter           // Elemente der Gruppe (abacus, apple)  
        });
```

Gruppierungsoperatoren

```
string[] alleWörter = { "blueberry", "chimpanzee", "abacus", "banana",  
"apple", "cheese" };
```

```
foreach (var gruppe in wörterGruppen) {  
    Console.WriteLine("'{0}':", gruppe.Anfangsbuchstabe);  
    foreach (var wort in gruppe.Wörter) {  
        Console.WriteLine("{0}", wort);  
    }  
}
```

'b':

Blueberry

Banana

'c':

chimpanzee

cheese

'a':

abacus

apple

Gruppierungsoperatoren - into

```
var query = from einWort in alleWörter
    // Gruppieren nach der Wortlänge
    group einWort by einWort.Length
    // Bezeichner für die Gruppe
    into wortGruppe
    // Sortieren nach der Wortlänge der Gruppen
    orderby wortGruppe.Key
    select wortGruppe;
```

- › Das into Schlüsselwort kann zum Erstellen eines temporären Bezeichners verwendet werden, um die Ergebnisse der group Klausel in einem neuen Bezeichner zu speichern

Linq Beispiele von Microsoft

- › **101 LINQ Samples**
- › <https://github.com/dotnet/try-samples/tree/master/101-linq-samples>