.NET und Language Integrated Query (LINQ)

W3L AG
info@W3L.de

2012

Inhaltsverzeichnis

Vergleich Schleifen und LINQ to Objects

ForEach-Schleife (imperativ)

```
List<Person> persons = new List<Person>();
foreach (Person person in allPersons)
{
    if (person.LastName.StartsWith("S") &&
        person.Gender == Gender.Male &&
        person.Age >= 25)
    {
        persons.Add(person);
    }
}

foreach (Person person in persons)
    Console.WriteLine(person);
```

LINQ-Anweisung (deklarativ)

```
List<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= 25)
    .ToList();

foreach (Person person in persons)
    Console.WriteLine(person);
```

LINQ - Syntax

Erweiterungsmethoden-Syntax (Extension Method Syntax)

```
IEnumerable<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= 25);
```

Abfrage-Syntax (Query Expression Syntax)

```
IEnumerable<Person> persons =
    from person in allPersons
    where person.LastName.StartsWith("S")
    where person.Gender == Gender.Male
    where person.Age >= 25
    select person;
```

LINQ Abfrage - Bestandteile

```
int age = GetAge();

var persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= age)
    .Select(person => new { person.FirstName, person.LastName, person.Address.City });

foreach (var person in persons)
    Console.WriteLine("{0} {1} - {2}", person.FirstName, person.LastName, person.City);
```

1. **Erweiterungsmethoden** `.Where()`
2. **Typinferenz** `var persons`
3. **Lambda-Ausdrücke** `person => person.Gender == Gender.Male`
4. **Objektinitialisierer** `{ person.FirstName, person.LastName }`
5. **Anonyme Typen** `new { }`
6. **Closures** `int age` `person => person.Age >= age`

Erweiterungsmethoden (1)

- **Methoden, die zu existierenden Typen hinzugefügt werden, ohne dass Typen geändert oder abgeleitete Typen erstellt werden**
  - Erweitern von abstrakten Klassen
  - Erweitern von nicht vererbbaren Klassen
  - Erweitern von Interfaces
- **Statische Methoden**
- **Deklariert in statischen Klassen**
- **Können wie Instanzmethoden aufgerufen werden**
- **Bei vorhandener Instanzmethode hat diese Vorrang**

Erweiterungsmethoden (2)

Beispiel – Erweiterung der Klasse String

```
public static class StringExtension
{
    public static string GetValueOrDefault(this string source, string defaultValue)
    {
        return source ?? defaultValue;
    }
}
```

Beispiel – Verwendung einer Erweiterungsmethode für String

```
const string defaultValue = "Default";

const string sampleString = "Hello World";
Console.WriteLine(sampleString.GetValueOrDefault(defaultValue));

const string nullString = null;
Console.WriteLine(nullString.GetValueOrDefault(defaultValue));
```

Ausgabe

```
Hello World
Default
```

Erweiterungsmethoden (3)

- **Warum Erweiterungsmethoden?**
- **Erweiterungsmethoden sind semantisch äquivalent zu statischen Methoden**

Beispiel – Mit Erweiterungsmethoden

```
IEnumerable<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= 25);
```

Beispiel – Ohne Erweiterungsmethoden

```
IEnumerable<Person> persons =
    Where(
        Where(
            Where(
                allPersons, person => person.LastName.StartsWith("S")),
            person => person.Gender == Gender.Male),
        person => person.Age >= 25);
```

Verzögerte Ausführung (1)

- LINQ-Abfragen werden lediglich deklariert
- Ausführung erfolgt erst, sobald über die Ergebnismenge iteriert wird
- Dieselben LINQ-Abfragen können bei einmaliger Deklaration mehrfach ausgeführt werden
- Iteration wird ebenfalls bei Aufruf einiger Erweiterungsmethoden durchgeführt, u. a. Count(), ToList()

Beispiel

```
// Deklaration
IEnumerable<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= 25);

// Ausführung
foreach (Person person in persons)
    Console.WriteLine(person);
```

Verzögerte Ausführung (2)

Beispiel – Mehrfache Ausführung derselben LINQ-Abfrage

```
var allPersons = new List<Person>() {
    new Person() { FirstName = "Peter", LastName = "Meier" },
    new Person() { FirstName = "Heike", LastName = "Schulz" }
};

IEnumerable<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"));

foreach (Person person in persons)
    Console.WriteLine(String.Format("{0}, {1}", person.FirstName, person.LastName));

Console.WriteLine(new String('-', 3));

allPersons.Add(new Person() { FirstName = "Manfred", LastName = "Schulz" });

foreach (Person person in persons)
    Console.WriteLine(String.Format("{0}, {1}", person.FirstName, person.LastName));
```

Ausgabe

```
Heike, Schulz
---
Heike, Schulz
Manfred, Schulz
```

Schlüsselwort `yield` (1)

- **Generiert jede Iteration nur bei Bedarf**
- **Kennzeichnet die Methode, in der `yield` enthalten ist, für den Compiler als Iterator-Block**
- `yield break` **beendet explizit den Iterator-Block**
- **Compiler generiert Klasse, die das Verhalten des Iterator-Blocks implementiert**

Beispiel

```
private static IEnumerable<int> RandomNumbers()
{
    const int maxValue = 5;
    Random random = new Random();
    int number = random.Next(maxValue);
    while (number > 0)
    {
        yield return number;
        number = random.Next(maxValue);
    }
}
```

Schlüsselwort yield (2)

Beispiel - Programmfluss

```
private static void Sample()
{
    foreach (int value in Numbers())
        Console.WriteLine(value);
}

private static IEnumerable<int> Numbers()
{
    foreach (int value in Enumerable.Range(0, 2))
    {
        Console.WriteLine("[Numbers] Before yield return {0}", value);
        yield return value;
        Console.WriteLine("[Numbers] After yield return {0}", value);
    }
}
```

Ausgabe

```
[Numbers] Before yield return 0
0
[Numbers] After yield return 0
[Numbers] Before yield return 1
1
[Numbers] After yield return 1
```

Schlüsselwort yield (3)

Vollständige Ausführung vs. verzögerte Ausführung

Beispiel – Vollständige Ausführung

```
public static IEnumerable<T> Where<T>(this IEnumerable<T> source, Func<T, bool> predicate)
{
    IList<T> result = new List<T>();
    foreach (T item in source)
        if (predicate(item))
            result.Add(item);
    return result;
}
```

Beispiel – Verzögerte Ausführung

```
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;
}
```

Aufruf

```
IEnumerable<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= 25);
```

Typinferenz

- Statische Typ-Ableitung aus einer Anweisung
- Nur für lokale Variablen
- Nur bei direkter Zuweisung eines Wertes
- Wert darf weder null noch ein Lambda-Ausdruck sein

Beispiel

```
var i = 42;
var myDictionary = new Dictionary<string, int>();
var someValue = person.SomeMethod();
```

Beispiel – Array-Typinferenz

```
var names = new[] { "Manfred", "Heike", "Peter" };
```

Lambda-Ausdrücke (1)

- **Anonyme Methoden**
- **Können Delegates und Expression Tree Typen erzeugen**
  - Delegates sind Objekte, die Referenzen auf Methoden mit kompatibler Signatur speichern
  - Expression Trees repräsentieren Daten über eine Lambda-Abfrage-Expression als Baum-Datenstruktur
- **Delegate-Typen**
  - Action, Action<T>, Action<T1, T2>, ...
    Referenzen auf Methoden mit Parametern vom Typ T, T1, ...
    und Rückgabetyp void
  - Func<TResult>, Func<T, TResult>, Func<T1, T2, TResult>, ...
    Referenzen auf Methoden mit Parametern vom Typ T, T1, ...
    und Rückgabetyp TResult

Lambda-Ausdrücke (2) - Action

Aufzurufende Methode

```
protected void InvokeOnUIThread(Action action)
{
    if (action == null)
        return;
    if (Dispatcher.CheckAccess())
        action();
    else
        Dispatcher.Invoke(action);
}
```

Anonyme Methode

```
InvokeOnUIThread(delegate()
{
    Items.Add(newItem);
});
```

Ausdrucks-Lambda (Expression Lambda)

```
InvokeOnUIThread(() => Items.Add(newItem));
```

Anweisungs-Lambda (Statement Lambda)

```
InvokeOnUIThread(() =>
{
    Items.Clear();
    Items.Add(newItem);
});
```

Lambda-Ausdrücke (3) - Func

Aufzurufende Methode

```
private static IEnumerable<int> Matches(
  IEnumerable<int> source, Func<int, bool> predicate)
{
    foreach (int value in source)
        if (predicate(value))
            yield return value;
}
```

Ausdrucks-Lambda

```
IEnumerable<int> result = Matches(someInts,
  (int i) => i % 2 == 0);
```

Ausdrucks-Lambda, kurz

```
IEnumerable<int> result = Matches(someInts,
  i => i % 2 == 0);
```

Anweisungs-Lambda

```
IEnumerable<int> result = Matches(someInts,
  i => { return i % 2 == 0; });
```

Lambda-Ausdrücke in LINQ-Abfragen

Beispiele

```
IEnumerable<Person> persons = allPersons
    .Where(person => person.LastName.StartsWith("S"))
    .Where(person => person.Gender == Gender.Male)
    .Where(person => person.Age >= 25);


if (allPersons
    .SkipWhile(person =>
        !String.IsNullOrEmpty(person.FirstName) &&
        !String.IsNullOrEmpty(person.LastName) &&
        Enum.IsDefined(typeof(Gender), person.Gender) &&
        person.Age >= 0 &&
        person.Age <= 200)
    .Any())
{
    return false;
}
return true;
```

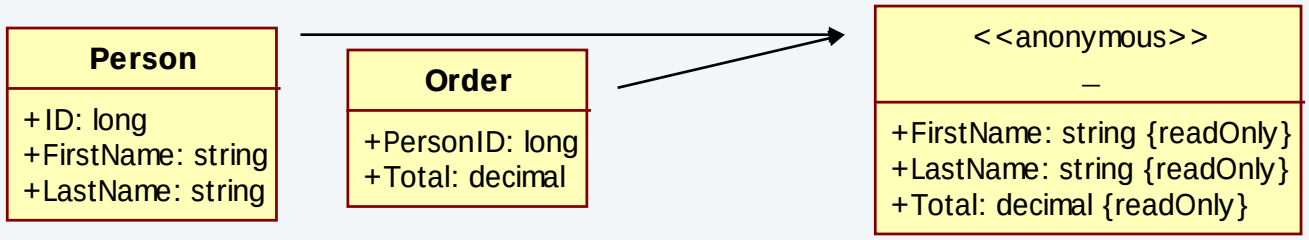
Verknüpfung von Daten mit LINQ to Objects (1)

- **Verknüpfen zweier Sequenzen von Objekten `Person` und `Order`**
- **Elemente der inneren Sequenz (erster Parameter) werden in einer Datenstruktur gruppiert**

Beispiel

```
var personsWithOrders = allPersons
    .Join(allOrders,
        person => person.ID,
        order => order.PersonID,
        (person, order) => new { person.FirstName, person.LastName, order.Total });

foreach (var personWithOrders in personsWithOrders)
    Console.WriteLine("{0} {1} - {2}",
        personWithOrders.FirstName, personWithOrders.LastName, personWithOrders.Total);
```



Verknüpfung von Daten mit LINQ to Objects (2)

- **Implementiert als Hash-Join (Equi-Join)**
- **Iteration erfolgt über die äußere Sequenz**
- Lookup **Datenstruktur entspricht einem Dictionary, in dem Schlüssel mehrfach vorkommen können**

```
public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(
  this IEnumerable<TOuter> outer,
  IEnumerable<TInner> inner,
  Func<TOuter, TKey> outerKeySelector,
  Func<TInner, TKey> innerKeySelector,
  Func<TOuter, TInner, TResult> resultSelector)
{
    var lookup = Lookup<TKey, TInner>.CreateForJoin(inner, innerKeySelector, null);
    foreach (TOuter current in outer)
    {
        Lookup<TKey, TInner>.Grouping grouping = lookup.GetGrouping(outerKeySelector(current), false);
        if (grouping != null)
        {
            for (int i = 0; i < grouping.Count; i++)
            {
                yield return resultSelector(current, grouping.elements[i]);
            }
        }
    }
    yield break;
}
```

Closures in .NET (1)

- **Referenzen auf Methoden und die Umgebungen, welche von den Methoden selbst referenziert werden**

Beispiel

```
IEnumerable<int> oneToTen = Enumerable.Range(1, 10);

int upperBound = 5;
IEnumerable<int> lowerEqualUpperBound = oneToTen.Where(i => i <= upperBound);
Console.WriteLine(lowerEqualUpperBound.Count());

upperBound = 2;
Console.WriteLine(lowerEqualUpperBound.Count());
```

Ausgabe

```
5
2
```

Closures in .NET (2)

Beispiel

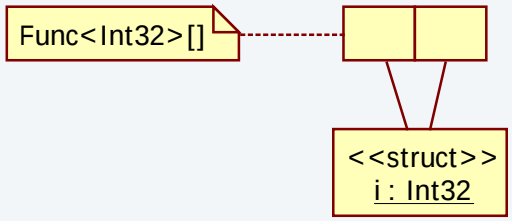
```
const int max = 2;
var funcs = new Func<int>[max];

for (int i = 0; i < max; i++)
    funcs[i] = () => i;

foreach (Func<int> func in funcs)
    Console.WriteLine(func());
```

Ausgabe

```
2
2
```



Beispiel

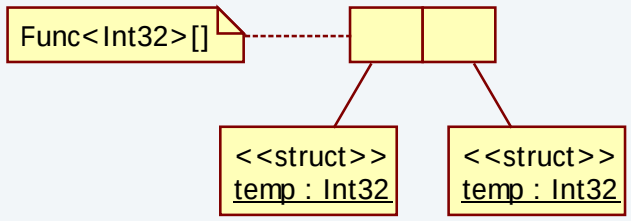
```
const int max = 2;
var funcs = new Func<int>[max];

for (int i = 0; i < max; i++)
{
    int temp = i;
    funcs[i] = () => temp;
}

foreach (Func<int> func in funcs)
    Console.WriteLine(func());
```

Ausgabe

```
0
1
```



Ausgewählte Standardabfrageoperatoren von LINQ

Art	Operatoren
Aggregation	Aggregate, Average, Count, LongCount, Min, Max, Sum
Filtern	Where, OfType
Gruppierung	GroupBy, ToLookup
Mengen	Distinct, Except, Intersect, Union
Partitionieren	Skip, SkipWhile, Take, TakeWhile
Projektion	Select, SelectMany
Quantifizierung	All, Any, Contains
Verknüpfung	Join, GroupJoin

Übernommen aus [3], gekürzt

Beispiel – Select, SelectMany

Beispiel - Select

LINQ

```
List<PersonViewModel> viewModels = allPersons
    .Select(person => new PersonViewModel(person))
    .ToList();
```

ForEach-Schleife

```
var viewModels = new List<PersonViewModel>();
foreach (Person person in allPersons)
    viewModels.Add(new PersonViewModel(person));
```

Beispiel - SelectMany

LINQ

```
List<string> phones = allPersons
    .SelectMany(person => person.Phones)
    .Where(phone => phone.StartsWith("555"))
    .ToList();
```

ForEach-Schleife

```
var phones = new List<string>();
foreach (Person person in allPersons)
    foreach (string phone in person.Phones)
        if (phone.StartsWith("555"))
            phones.Add(phone);
```

Beispiel – ToLookup

LINQ

```
ILookup<string, Person> personsByCity = allPersons
    .ToLookup(person => person.Address.City);
```

ForEach-Schleife

```
var personsByCity = new Dictionary<string, List<Person>>();
foreach (Person person in allPersons)
{
    List<Person> personsOfCurrentCity;
    if (!personsByCity.TryGetValue(person.Address.City, out personsOfCurrentCity))
    {
        personsOfCurrentCity = new List<Person>();
        personsByCity.Add(person.Address.City, personsOfCurrentCity);
    }
    personsOfCurrentCity.Add(person);
}
```

Beispiel – OrderBy, ThenBy, ToLookup in Kombination

LINQ

```
ILookup<string, Person> orderedPersonsByCity = allPersons
    .OrderBy(person => person.Address.City)
    .ThenBy(person => person.LastName)
    .ToLookup(person => person.Address.City);
```

ForEach-Schleife

```
var orderedPersons = new List<Person>(allPersons);
orderedPersons.Sort((person1, person2) =>
    {
        int cityCompare = person1.Address.City.CompareTo(person2.Address.City);
        return cityCompare != 0 ? cityCompare : person1.LastName.CompareTo(person2.LastName);
    });
var orderedPersonsByCity = new Dictionary<string, List<Person>>();
foreach (Person person in orderedPersons)
{
    List<Person> personsOfCurrentCity;
    if (!orderedPersonsByCity.TryGetValue(person.Address.City, out personsOfCurrentCity))
    {
        personsOfCurrentCity = new List<Person>();
        orderedPersonsByCity.Add(person.Address.City, personsOfCurrentCity);
    }
    personsOfCurrentCity.Add(person);
}
```

Vor- und Nachteile von LINQ

- **Vorteile**
  - Verwendbar für verschiedene Datenquellen
    - Relationale Datenbanken
    - O/R Mapper
    - XML-Dateien/-Streams
    - Objekt-Sequenzen
  - Typsicher
    - Syntaktische Prüfung durch Compiler
    - Unterstützung von Rename-Refactorings
  - Unterstützung durch Code-Vervollständigung (IntelliSense)
  - Oftmals lesbarer und wartbarer als Schleifen
  - Abfrage kann in mehreren Schritten aufgebaut werden

- **Nachteile**
  - Keine automatische Optimierung von Abfragen
  - Langsamer als Schleifen
  - Schwieriger zu debuggen als Schleifen

LINQ Provider

- LINQ to Amazon
- LINQ to Active Directory
- LINQ to Datasets
- LINQ to Entity
- LINQ to Excel
- LINQ to Google
- LINQ to NHibernate
- LINQ to Objects
- LINQ to Parallel
- LINQ to SQL
- LINQ to XML
- und weitere

Quellenverzeichnis

- **[1] MSDN**
- **[2] Pro LINQ**
  Language Integrated Query in C# 2010
  - Apress
  - ISBN: 987-1-4302-2653-6
- **[3] Bachelorarbeit**
  Objektbasierte 1:n Navigation:
  Ein Vergleich zwischen externer Iteration und LINQ
  - Oktober 2009
  - Jens Duczmal
  - http://www.fernuni-hagen.de/imperia/md/content/ps/bachelorarbiet-duczmal.pdf

Vielen Dank
für
Ihre Aufmerksamkeit.

LINQ to XML - Beispiel

```
<?xml version="1.0" encoding="utf-8" ?>
<Sample>
  <SimpleItems>
    <Item>
      <Title>C#</Title>
      <Author>Manfred Schulz</Author>
      <Year>2002</Year>
    </Item>
    <Item>
      <Title>Visual Basic 6</Title>
      <Author>Manfred Schulz</Author>
      <Year>1990</Year>
    </Item>
  </SimpleItems>
</Sample>
```

LINQ-Abfrage

```
XDocument xmlDocument = XDocument.Load(fileName);
var books = xmlDocument
    .Descendants("Item")
    .Where(item => ((int)item.Element("Year")) > 2000)
    .Select(item => new {
        Title = item.Element("Title").Value,
        Author = item.Element("Author").Value });
```

NHibernate Query Over API (angelehnt an LINQ)

Query-Over-Abfrage

```
IList<Person> persons = Session.QueryOver<Person>()
    .Where(person => person.LastName == "Schulz")
    .JoinQueryOver<Order>(person => person.Orders)
        .Where(order => order.Date >= new DateTime(2000, 1, 1, 0, 0, 0, DateTimeKind.Local))
    .List();
```

Generierte SQL-Abfrage

```
SELECT this_.Id as Id0_1_,
  this_.FIRST_NAME as FIRST2_0_1_,
  this_.LAST_NAME as LAST3_0_1_,
  order1_.Id as Id1_0_,
  order1_.Date as Date1_0_
FROM PERSON this_ inner join "ORDER" order1_ on this_.Id=order1_.ORDER_ID
WHERE this_.LAST_NAME = @p0 and
  order1_.Date >= @p1;@p0 = 'Schulz'
  [Type: String (0)], @p1 = 01.01.2000 00:00:00 [Type: DateTime (0)]
```

Optimierung einer LINQ Abfrage (1)

Negativ-Beispiel – LINQ Abfrage

```
if (!(
    allPersons.All(person => !String.IsNullOrEmpty(person.FirstName)) &&
    allPersons.All(person => !String.IsNullOrEmpty(person.LastName)) &&
    allPersons.All(person => Enum.IsDefined(typeof(Gender), person.Gender)) &&
    allPersons.All(person => person.Age >= 0) &&
    allPersons.All(person => person.Age <= 200)))
{
    return false;
}

return true;
```

Optimierung einer LINQ Abfrage (2)

Negativ-Beispiel – ForEach-Schleifen

```
foreach (Person person in allPersons)
    if (String.IsNullOrEmpty(person.FirstName))
        return false;

foreach (Person person in allPersons)
    if (String.IsNullOrEmpty(person.LastName))
        return false;

foreach (Person person in allPersons)
    if (!Enum.IsDefined(typeof(Gender), person.Gender))
        return false;

foreach (Person person in allPersons)
    if (person.Age < 0)
        return false;

foreach (Person person in allPersons)
    if (person.Age > 200)
        return false;

return true;
```

Optimierung einer LINQ Abfrage (3)

Optimiertes Beispiel – ForEach-Schleife

```
foreach (Person person in allPersons)
{
    if (String.IsNullOrEmpty(person.FirstName) ||
        String.IsNullOrEmpty(person.LastName) ||
        !Enum.IsDefined(typeof(Gender), person.Gender) ||
        person.Age < 0 ||
        person.Age > 200)
    {
        return false;
    }
}
return true;
```

Optimierung einer LINQ Abfrage (4)

Optimiertes Beispiel – LINQ Abfrage

```
if (allPersons
    .SkipWhile(person =>
        !String.IsNullOrEmpty(person.FirstName) &&
        !String.IsNullOrEmpty(person.LastName) &&
        Enum.IsDefined(typeof(Gender), person.Gender) &&
        person.Age >= 0 &&
        person.Age <= 200)
    .Any())
{
    return false;
}
return true;
```

Inhouse-Schulungen

Wir bieten Inhouse-Schulungen und Beratung durch unsere IT-Experten und –Berater.

Schulungsthemen

- Softwarearchitektur (OOD)
- Requirements Engineering (OOA)
- Nebenläufige & verteilte Programmierung

Gerne konzipieren wir auch eine individuelle Schulung zu Ihren Fragestellungen.

w3l. Software . Solutions . Services

Sprechen Sie uns an!
Tel. 0231/61 804-0, info@W3L.de

W3L-Akademie

In Zusammenarbeit mit der Fachhochschule Dortmund bieten wir

zwei Online-Studiengänge

- B.Sc. Web- und Medieninformatik
- B.Sc. Wirtschaftsinformatik

und 7 Weiterbildungen im IT-Bereich an.

w3l. web life long learning

Besuchen Sie unsere Akademie!
http://Akademie.W3L.de