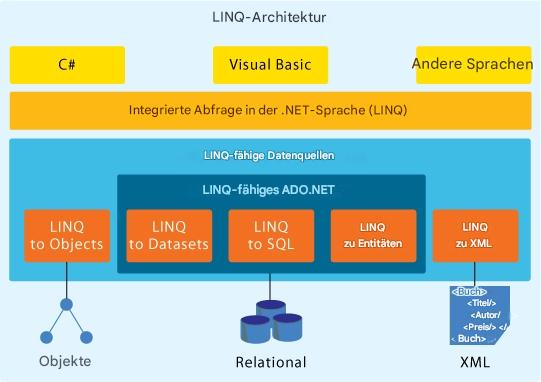
**LINQ to Objects (1)**

**Schneller Start**

LINQ (Language Integrated Query = In (einer .NET-Sprache integrierte Abfrage) ist eine Abfragesprache in C# und anderen .NET-Sprachen, mit der sich unterschiedlichste Datenquellen (z. B. Listen, Arrays, Datenbanken, XML, …) auf eine einheitliche Weise **abfragen** und **verarbeiten** lassen.



**Vorteil:** Man muss keine spezielle Abfragesprache oder Technologie für das Abfragen der jeweiligen Datenquelle lernen (wie, z.B.: SQL, XmlDocument, Anwendung von Schleifen, …), denn LINQ ermöglicht alle die **absturzsicher** und effizient abzufragen. So kann man statt fehleranfälliger SQL-Codierung oder verwickelten, ineinander verschachtelten Schleifen, mit gutem Gewissen LINQ anwenden.

Hinweis: LINQ ist offene Technologie, der jederzeit neue „Provider“ hinzugefügt werden können.

**LINQ to Objects: die erste Probe, wie in SQL**

var namen = new List<string>() { "Anna", "Roman", "Tobias", "Matthias" };

// Definition der Linq-Abfrage

var result = from name in namen

where name.Contains("ias")

select name;

// Erst hier wird die Linq-Abfrage ausgeführt:

foreach (var item in result)

Console.WriteLine(item);

**Ohne IEnumerable kein LINQ**

**IEnumerable** stellt die Kernfunktionalität einer Collection bereit und macht einen Enumerator verfügbar, der eine einfache Iteration durch die Auflistung unterstützt

**Unterschied zwischen IEnumerable<T> und List<T>**

|  |  |  |
| --- | --- | --- |
| Kriterium | IEnumerable<T> | List<T> |
| Typ | Schnittstelle (Abstraktion) | Konkrete Klasse (Datenstruktur) |
| Speicher | **Lazy Loading** (Daten werden bei Bedarf generiert) | **Eager Loading** (Daten vollständig im Speicher) |
| Zugriff | Nur sequentiell (foreach) | Direkter Zugriff per Index (list[i]) |
| Funktionen | Nur Iteration (GetEnumerator(), LINQ) | Hinzufügen (Add), Entfernen (Remove), Sortieren (Sort) |
| Leistung | Speicherfreundlich (on-demand) | Schneller Zugriff, aber mehr Speicher |
| Verwendung | Große Datenströme, **LINQ-Abfragen** | Temporäre Speicherstruktur, **Bearbeitung** von Listen |

**Wann verwendet man was?**

**IEnumerable<T>:** Wenn Sie nur durch eine Sammlung iterieren oder LINQ-Abfragen durchführen möchten.

**List<T>:** Wenn Sie Elemente hinzufügen, entfernen oder ändern oder direkt per Index zugreifen möchten.

**Was ist IEnumerable?**

* + Eine Schnittstelle, die eine **sequentielle Iteration** über eine Sammlung ermöglicht.
  + Unterstützt die foreach-Schleife.

**Beispiel**

IEnumerable<int> enumerableZahlen = [1, 2, 3, 4, 5];

foreach (int zahl in enumerableZahlen)

{

Console.WriteLine(zahl);

}

Oder:

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

foreach (int zahl in list.**AsEnumerable())**

{

Console.WriteLine(zahl);

}

**Erklärung:**

IEnumerable<int> erlaubt die Iteration mit foreach.

**Hinweis:**

Fast alle Listen und Arrays implementieren **IEnumerable**.

**Syntax von LINQ (Query-Syntax vs. Methodensyntax)**

**Query-Syntax** (ähnlich SQL):

var ergebnis = from zahl in zahlen

where zahl % 2 == 0

select zahl;

**Methoden-Syntax** (mit Lambda-Ausdrücken):

var ergebnis = zahlen.Where(zahl => zahl % 2 == 0);

**Übung**

Erstellen Sie eine **LINQ-Abfrage,** die aus einer Liste von Zahlen nur die **ungeraden Zahlen** auswählt. Setzen Sie es jeweils in **Query und**  **Methodensyntax** um**.**

**Einführung anonyme Typen (anonyme Objekte)**

**Was sind anonyme Objekte?**

Anonyme Objekte sind Objekte **ohne explizite Klassen-Definition**, die vom Compiler automatisch erzeugt werden. „Namenslos“ bedeutet, dass uns der Name der Klasse nicht bekannt ist und man deshalb keinen direkten Zugriff auf die Klasse hat. Lediglich eine Instanz steht zur Verfügung, die man ausschließlich lokal, d.h. im Bereich der Deklaration verwenden kann.

Praktisch, **wenn man Daten „on-the-fly“ benötigt**.

**Beispiel:**

var person = new { Name = "Anna", Alter = 25 };

Console.WriteLine($"Name: {person.Name}, Alter: {person.Alter}");

**Erklärung:**

Anonyme Objekte verwenden die new { ... }-Syntax.

Sie sind **besonders nützlich in Kombination mit LINQ**.

**Übung:**

Erstellen Sie ein anonymes Objekt für eine „Buch“-Entität mit den Eigenschaften Titel, Autor und Jahr

**LINQ to Objects mit anonymen Objekten**

**Beispiel: LINQ mit anonymen Objekten**

var personen = new List<string> { "Anna", "Bernd", "Clemens", "Diana" };

var ergebnis = from name in personen

where name.StartsWith("A")

select new { Name = name, Laenge = name.Length };

foreach (var person in ergebnis)

{

Console.WriteLine($"Name: {person.Name}, Länge: {person.Laenge}");

}

**Erklärung**

select erzeugt ein neues anonymes Objekt mit den Feldern Name und Laenge.

**Übung:**

Schreiben Sie eine LINQ-Abfrage, die eine Liste von Städten filtert (z. B. nur Städte, die mit "B" beginnen) und die Länge des Stadtnamens zurückgibt.

**Erklärung der gängigen Linq-Methoden**

|  |  |  |
| --- | --- | --- |
| Methode | Beschreibung | Beispiel |
| Where | Filtert Elemente nach einer Bedingung. | x => x % 2 == 0 (gerade Zahlen) |
| First | Erstes Element, wirft Fehler bei leerer Liste. | First(x => x > 5) |
| FirstOrDefault | Erstes Element oder Standardwert (null). | FirstOrDefault(x => x > 10) |
| OrderBy | Sortieren (aufsteigend). | OrderBy(x => x.Age) |
| ThenBy | Weitere Sortierbedingung. | ThenBy(x => x.Name) |
| Any | Mindestens ein Element erfüllt Bedingung. | Any(x => x > 10) |
| All | Alle Elemente erfüllen Bedingung. | All(x => x > 0) |
| Sum | Summe der Elemente. | Sum(x => x.Price) |
| Average | Durchschnitt der Elemente. | Average(x => x.Price) |
| Min | Kleinstes Element. | Min(x => x.Age) |
| Max | Größtes Element. | Max(x => x.Age) |