

# Základy LINQu

2. dubna 2019

LINQ neboli Language Integrated Query je sada jazykových funkcí pro psaní strukturovaných typově bezpečných dotazů přes lokální kolekce objektů a vzdálená data. LINQ byl představen v C # 3.0 a frameworku 3.5.

Základní jednotky dat v LINQ jsou sekvence a prvky. Sekvence je libovolný objekt, který implementuje obecné IEnumerable rozhraní a prvek je každá položka v sekvenci. V následujícím příkladu jsou názvy sekvencemi a Tom, Ríša a Jindra jsou prvky:

```
string [] names = {"Tom", "Ríša", "Jindra"};
```

---

Sekvence, jako je tato, nazýváme lokální sekvencí , protože reprezentuje lokální sbírku objektů v paměti. Operátor dotazu je metoda, která transformuje sekvenci. Typický dotazovací operátor přijme vstupní sekvenci a vydá transformovanou výstupní sekvenci. Ve třídě Enumerable v System.Linq existuje kolem 40 operátorů dotazů, všechny implementovány jako metody statického rozšíření. Ty se nazývají standardní operátory dotazů.

# Jednoduchý dotaz

Dotaz je výraz, který transformuje sekvence jedním nebo více operátory dotazů. Nejjednodušší dotaz obsahuje jednu vstupní sekvenci a jeden operátor. Například můžeme použít operátor *Where* na jednoduchém poli, abychom mohli extrahovat jména, jejichž délka je nejméně čtyři znaky, takto:

```
string[] names = { "Tom", "Ríša", "Jindra" };

IEnumerable<string> filteredNames =
    System.Linq.Enumerable.Where ( names, n => n.Length >= 4 );
foreach (string n in filteredNames) Console.Write (n + "|");
// Ríša|Jindra|
```

---

Protože standardní operátory dotazů jsou implementovány jako metody rozšíření, můžeme volat *Where* přímo na *names* jakoby to byla metoda instance:

```
var filteredNames = names.Where ( n => n.Length >= 4 );
```

---

(Chcete-li kompilovat, musíte importovat *System.Linq* pomocí direktivy *using*.) Metoda *Where* v *System.Linq.Enumerable* má následující deklaraci:

```
static IEnumerable<TSource> Where<TSource> ( this IEnumerable<TSource> source, Func
    <TSource, bool> predicate)
```

---

Kde *source* je vstupní sekvence; *predicate* je delegát, který je vyvolán na každém vstupním prvku. Metoda *Where* vybere všechny prvky, pro které delegát vrací hodnotu true. Interně je generován kód pro iterátor:

```
foreach (TSource element in source)
    if (predicate (element))
        yield return element;
```

---

## Transformace (Select)

```
string[] names = { "Tom", "Ríša", "Jindra" };

IEnumerable<string> upperNames = names.Select (n => n.ToUpper());
foreach (string n in upperNames)
    Console.Write (n + "|");
// TOM|RÍŠA|JINDRA|
```

---

S použitím anonymního typu

```
var query = names.Select (n => new { Name = n, Length = n.Length });
foreach (var row in query)

    Console.WriteLine (row);
// { Name = Tom, Length = 3 }
```

```
// { Name = Ríša, Length = 4 }  
// { Name = Jindra, Length = 5 }
```

---

## (SelectMany)

```
double[] t = {1, 2, 3};  
var coordinates = t.SelectMany(x=>new double[] { x, x*x, Pow(x,3)});  
// coordinates { 1, 1, 1, 2, 4, 8, 3, 9, 27}
```

---

## Filtry

```
int[] numbers = { 10, 9, 8, 7, 6 };  
IEnumerable<int> firstThree = numbers.Take (3);  
// firstThree je { 10, 9, 8 }  
IEnumerable<int> lastTwo = numbers.Skip (3);  
// lastTwo je { 7, 6 }  
var whileGreater7 = numbers.TakeWhile(n=>n>7);  
// whileGreater7 je { 10, 9, 8 }  
var skipGreater7 = numbers.SkipWhile(n=>n>7);  
// skipGreater7 je { 7, 6 }  
numbers = { 10, 9, 8, 7, 6, 8, 10 };  
var distinct = numbers.Distinct();  
// distinct je { 10, 9, 8, 7, 6 }
```

---

# Elementy

```
int[] numbers = { 10, 9, 8, 7, 6 };  
int firstNumber = numbers.First(); // 10  
int lastNumber = numbers.Last(); // 6  
int secondNumber = numbers.ElementAt (2); // 8  
int firstOddNum = numbers.First (n => n%2 == 1); // 9
```

---

# Agregace

```
int[] numbers = { 10, 9, 8, 7, 6 };  
int count = numbers.Count(); // 5  
int min = numbers.Min(); // 6  
int max = numbers.Max(); // 10  
double avg = numbers.Average(); // 8  
int evenNums = numbers.Count (n => n % 2 == 0); // 3  
int maxRemainderAfterDivBy5 = numbers.Max (n => n % 5); // 4
```

---

# Kvantifikátory

```
int[] numbers = { 10, 9, 8, 7, 6 };  
bool hasTheNumberNine = numbers.Contains (9); // true  
bool hasMoreThanZeroElements = numbers.Any(); // true
```

```
bool hasOddNum = numbers.Any (n => n % 2 == 1); // true
bool allOddNums = numbers.All (n => n % 2 == 1); // false
```

---

## Množinové operace

```
int[] seq1 = { 1, 2, 3 }, seq2 = { 3, 4, 5 };
IEnumerable<int> concat = seq1.Concat (seq2), // { 1, 2, 3, 3, 4, 5 }
union = seq1.Union (seq2); // { 1, 2, 3, 4, 5 }
IEnumerable<int> common = seq1.Intersect (seq2), // { 3 }
difference1 = seq1.Except (seq2), // { 1, 2 }
difference2 = seq2.Except (seq1); // { 4, 5 }
```

---

## Odložená aplikace dotazu

```
var numbers = new List<int> { 1 };
IEnumerable<int> query = numbers.Select (n => n * 10); numbers.Add (2); // Přidáme
další prvek
foreach (int n in query)
    Console.Write (n + "|"); // 10|20|

var numbers = new List<int>() { 1, 2 };
List<int> timesTen = numbers .Select (n => n * 10) .ToList(); // Aplikace dotazu do
typu List<int>
```

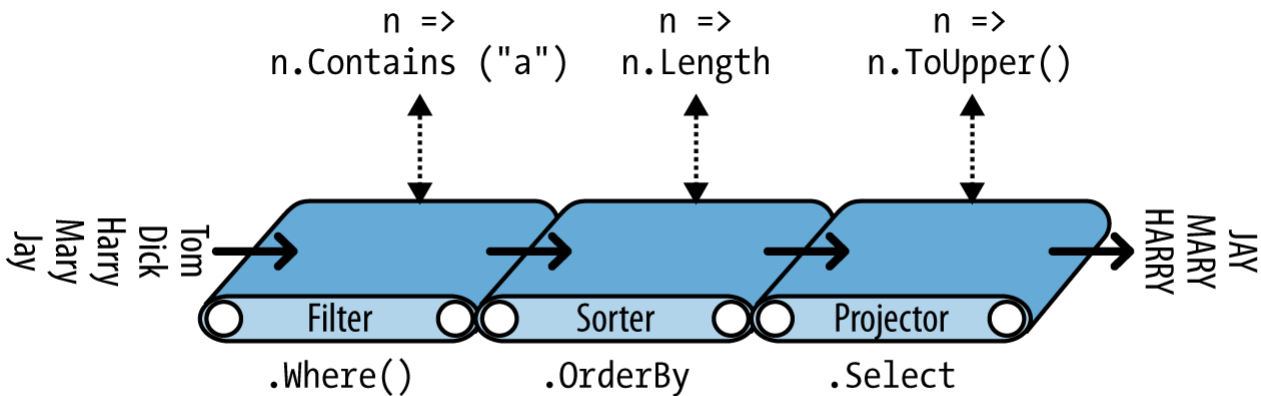
```
numbers.Clear();  
Console.WriteLine (timesTen.Count); // Stále 2, kešování
```

---

## Kompozice dotazovacích operátorů

```
string[] names = { "Tom", "Ríša", "Jindra", "Marie", "Jan" };  
IEnumerable<string> query = names  
    .Where (n => n.Contains ("a"))  
    .OrderBy (n => n.Length)  
    .Select (n => n.ToUpper());  
foreach (string name in query)  
    Console.Write (name + "|"); // JAN|RÍŠA|MARIE|JINDRA|
```

---



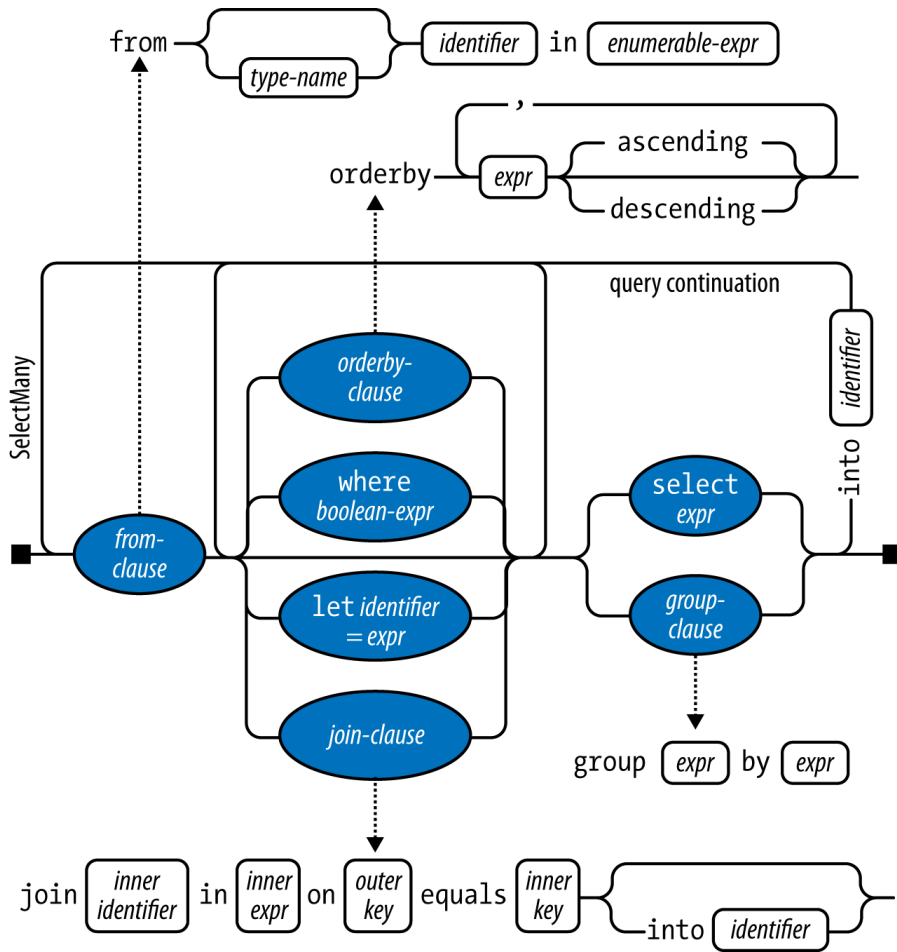
## Dotazovací jazyk LINQu

```

IEnumerable<string> query = from n in names
    where n.Contains ("a")
    orderby n.Length
    select n.ToUpper();

```





# Příklad

```
double l=1.1, r=1;
var n = 100;
var e = Enumerable.Range(0,n+1);
var h=2*l/n;
var x = from i in e select -l+i*h;
var grid = from cx in x
            from cy in x
            select new {x=cx,y=cy};
var domain = from p in grid
              where Norm(p.x,p.y) <= r
              select p;

double Norm(double x, double y) => Sqrt(Pow(x,2)+Pow(y,2));
```

---

