.NET C# - LINQ

Petr Novák / novakpe@fel.cvut.cz / 12.02.2021

Obsah

1 Úvod

LINQ je mocný nástroj pro procházení a úpravu dat. Jedná se o obdobu SQL pro databáze. Na rozdíl od SQL je LINQ použitelné skoro na všechny data, tedy přesněji řečeno jejich seznamy. LINQ tedy pracuje v podstatě se vším co je nějaký seznam nebo kolekce čehokoli. Aby mohlo být LINQ použito je nutno na začátek zdrojového souboru přidat:

using System.LINQ

Pokud nebude toto vloženo na začátek souboru, tak se nebudou zobrazovat žádné nápovědy ohledně LINQ. Na úvod velmi jednoduché příklady, kdy je „foreach“ nahrazen pomocí „LINQ“.

```
// seznam cisel
int[] cisla = new int[] { 1, 2, 3, 4, 5 };
int soucet = 0;
// soucet pomoci 'foreach'
foreach(int cislo in cisla) { soucet += cislo; }
// soucet pomoci 'LINQ'
cisla.All(item => { soucet += item; return true; });
```

```
// seznam cisel
int[] cisla = new int[] { 1, 2, 3, 4, 5 };
bool yes = false;
// hledani pomoci 'foreach' (yes = true – nalezeno / yes = true – nenalezeno)
foreach (int cislo in cisla) { if (cislo == 3) { yes = true; break; } }
// hledani pomoci 'LINQ' (0 – nenalezeno / cislo - nalezeno)
int hledane = cisla.FirstOrDefault(item => item == 3);
```

Uvedené hledání není zrovna ideální příklad pro LINQ, protože se při nenalezení požadovaného čísla vrátí „0“, takže nelze hledat číslo „0“. Avšak jako stručný příklad je toto postačující.

2 Procházení seznamu pomocí All a Any

Metody „All“ a „Any“ jsou tak trochu jiné a velmi často nahrazují „foreach“. Jejich význam je následující:

All – (pro všechny) Postupně prochází všechny prvky. U každého prvky vyhodnocuje zadanou podmínku. Pokud je plněna, tak se pokračuje dalším prvkem, pokud není splněna, tak se procházení seznamu končí. Může tedy sloužit pro kontrolu, zda všechny prvky v seznamu splňují zadanou podmínku. Výstupem je tedy:

- **True** – Podmínka je splněna pro všechny prvky v seznamu (došlo se až do konce seznamu)
- **False** – Podmínka není splněna pro všechny prvky seznamu (nedošlo se až do konce seznamu)

Any – (pro nějaký) Postupně prochází všechny prvky. U každého prvky vyhodnocuje zadanou podmínku. Pokud není plněna, tak se pokračuje dalším prvkem, pokud je splněna, tak se procházení seznamu končí. Může tedy sloužit pro kontrolu, zda aspoň nějaký prvek v seznamu splňují zadanou podmínku (první nalezený ukončí procházení seznamu). Výstupem je tedy:

- **True** – Podmínka je splněna aspoň pro jeden prvek v seznamu (nedošlo se až do konce seznamu)
- **False** – Podmínka není splněna pro žádný prvek seznamu (došlo se až do konce seznamu)

```
// test zda jsou vsecha cisla vetsi nez „0“
// „True“ – ano / „False“ – ne
bool yes = cisla.All(item => item > 0);
// vystup je „True“, všechna cisla jsou vetsi nez „0“
// (prochazeni doslo az do konce, byla celkova kontrola)

// test je aspoň jedno cislo vetsi nez „3“
// „True“ – ano / „False“ – ne
bool yes = cisla.Any(item => item > 3);
// vystup je „True“, existuje cislo vetsi nez „3“
// (prochazeni nedoslo az do konce, bylo ukonceno u prvku „4“)
```

Metodu „All“ lze vhodně využít pro projití celého seznamu. Jedinou podmínkou je vracet „True“ po každém prvku. Pokud by se vrátilo „False“, tak procházení konci. Příkladem je právě součet všech čísel:

```
// celkovy soucet
cisla.All(item => { soucet += item; return true; });
```

Pro každý prvek se tedy vykoná tělo, zde aktualizace součtu a pak se vyhodnocuje podmínky, která je vždy „True“. Projdou / zpracují se tedy všechny prvky seznamu. Samozřejmě podmínku lze využít například i takto:

```
// zda je celkovy soucet aspon „5“ formou „All“
cisla.All(item => { soucet += item; return !(soucet > 5); });
// zda je celkovy soucet aspon „5“ formou „Any“
cisla.Any(item => { soucet += item; return (soucet > 5); });
```

V podstatě je vidět, že „All“ a „Any“ jsou tak trochu komplementární.

3 Hledání pomocí First (FirstOrDefault) a Last (LastOrDefault)

First a **FirstOrDefault** – Nejjednodušší forma hledání je pomocí metod „First“ a „FirstOrDefault“. Obě metody jsou v podstatě stejné s rozdílem neúspěchu. Metoda „FirstOrDefaul“ při nenalezení prvku vrací nějakou defaultní hodnotu pro typ prvku a metoda „First“ při nenalezení prvku vyvolá výjimku.
U metody „FirstOrDefaul“ může být občas problémem defaultní návratová hodnota. Při hledání a nenalezení objektu je vráceno „null“ což je velmi výstižný indikátor, že hledaný objekt neexistuje. Při hledání a nenalezení řetězce („String“) je vráceno rovněž „null“ což je rovněž velmi výstižný indikátor, že hledaný řetězec neexistuje. Ovšem při hledání a nenalezení čísla typu „int“ je vrácena „0“ což nemusí být zcela výstižný indikátor nenalezení požadovaného čísla. Defaultní hodnotu pro jakýkoli typ lze získat pomocí „default(…)“. Při hledání libovolného čísla je vhodnější použít „First“ obaleného výjimkou než „FirstOrDefault“. Příklady hledání mohu být následující:

```
// hledani prvního cisla většího nez „3“
var result = cisla.FirstOrDefault(item => item > 3);

// hledani prvního zaznamu / objektu s „Name = 3“
var result = cisla.FirstOrDefault(item => item.Name == "Petr");

// podmínka muze byt i slozitejsi, nalezeni prvního cisla splňující …
int result = cisla.FirstOrDefault(item => ((item > 3) && (item < 5)));
```

Pokud je prvek nalezen, tak metody „First“ a „FirstOrDefault“ vrátí pouze první nalezený prvek i když je takovýchto prvků v seznamu více. Hledání se tedy ukončí po prvním nalezeném prvku. Obdobou k „First“ a „FirstOrDefault“ jsou metody „Last“ a „LastOrDefault“ hledající poslední výskyt požadovaného prvku (opět je vrácen pouze jeden, tedy ten poslední nalezený prvek). Hledání tedy probíhá vždy až do konce.

4 Hledání pomocí Where

Where – Metoda slouží pro hledání i více prvků splňující nějakou podmínku.

```
// nalezeni všech prvku splnujici … (prvku muze byt nekolik, nemusi byt pouze jeden)
var result = cisla.Where(item => ((item > 3) && (item < 5)));
```

Metoda „Where“ vrací „IEnumerable“ tedy nikoli pole nebo seznam. Pokud je potřeba výstupní „IEnumerable“ převést na pole nebo seznam tak lze využít následující:

```
int[] result = cisla.Where(...).ToArray();
List<int> result = cisla.Where(...).ToList();
```

5 Metody Take a Skip

Metody „Take“ a „Skip“ nejsou často používané, ale občas se opravdu shodnou.

Take – Vezme pouze několik prvků.
Skip – Přeskočí / vynechá několik prvků.

Nejjednodušší použití může být následující:

```
// vezme první tri prvky ze seznamu
int[] result = cisla.Take(3).ToArray();
```

```
// preskoci první dva prvky a vezme hned tri nasledujici
int[] result = cisla.Skip(2).Take(3).ToArray();
// preskoci první dva prvky a ve zbytku hleda ty co splnuji …
int[] result = cisla.Skip(2).Where(...).ToArray();
```

K „Take“ existují varianty „TakeLast“ a „TakeWhile“. Ke „Skip“ existují varianty „SkipLast“ a „SkipWhile“. Některé příklady použití:

TakeLast – Bere několik posledních prvků ze seznamu.
TakeWhile – Bere prvky dokud je podmínka splněna.
SkipLast – Bere seznam bez několika posledních prvků.
SkipWhile – Přeskakuje prvky dokud je podmínka splněna.

```
// preskoc ze zacatku prvky mensi nez „3“ a pak vem další první tri prvky
// (první prvek s hodnotou 3 a více konci preskakovani)
int[] result = cisla.SkipWhile(item => item < 3).Take(3).ToArray();
```

6 Obrácení seznamu pomocí Reverse

Pokud je potřeba ve vstupním seznam obrátit pořadí lze využít metodu „Reverse“. Příklad je následující:

```
// otoceni seznamu prvku (první je poslední, poslední je prvni)
int[] reversed = cisla.Reverse().ToArray();
```

7 Vytvořeni seznamu pomoci Range, Repeat, Empty

Range – Vytvoří seznam hodnot v zadaném rozsahu.
Repeat – Vytvoří seznam zadaného počtu stejných hodnot.
Empty – Vytvoří seznam prázdných / výchozích hodnot.

Některé příklady použití:

```
// vytvoreni seznamu 8 hodnot pocinaje hodnotou 10 (10,11,12,13,14,15,16,17)
int[] result = Enumerable.Range(10, 8).ToArray();
// vytvoreni mocnin 5 hodnot pocinaje hodnotou 1 (1,4,9,16,25)
int[] result = Enumerable.Range(1, 5).Select(item => item * item).ToArray();

// vytvoreni seznamu 10 cisel s hodnotou 5 (kazda hodnota je „5“)
int[] result = Enumerable.Repeat<int>(5, 10).ToArray();

// prazdny seznam retezcu (??? zatím moc nevim vyuziti ???)
var emptystringCollection = Enumerable.Empty<string>();
```

8 Indexy v seznamu

Místo použití „Skip“, „Take“ a podobných lze využít indexy.

...