

Употреба на PLINQ

PLINQ е паралелна имплементация на Language Integrate Query (LINQ). Преди добавянето на LINQ към C#, извличането на данни от различни източници на данни като XML, БД и др. е било трудоемка задача. LINQ използва делегатите в .NET и вградените методи за извличане или модифициране на данни без нуждата от извършване на трудни задачи.

Тук ще разберем какво представлява PLINQ и как да го ползваме. Ще разгледаме следните неща:

- LINQ провайдъри в .NET
- Писане на PLINQ заявки
- Запазване на реда на данните при използване на PLINQ
- Опции за сливане в PLINQ
- Работа с изключения PLINQ
- Комбиниране на паралелни и последователни заявки
- Недостатъци на PLINQ

LINQ провайдъри в .NET

LINQ може да работи с XML, обекти и бази данни. LINQ позволява това с помощта на провайдъри на данни:

- LINQ към обекти: LINQ към обекти позволява да обработвате обекти в паметта, като масиви, колекции, шаблонни типове и др. Връща `IEnumerable` и поддържа операции от типа на сортиране, филтриране, групиране и агрегиране (сума, средно аритметично, мин, макс и др.). Използва се с `System.Linq` неймспейса.
- LINQ към XML: LINQ към XML, още XLINQ, помага на разработчиците да обработват и модифицират източници на данни в XML формат. Използва се с `System.Xml.Linq` неймспейса.
- LINQ към entities: Това е най-напредналата технология. Тя позволява на разработчиците да работят с коя да е релационна база данни, включително SQL Server, Oracle, IBM Db2, и MySQL. LINQ към entities поддържа ORM подхода.
- PLINQ: PLINQ е паралелна имплементация на LINQ към обекти. LINQ заявките се изпълняват последователно и могат да бъдат наистина бавни за по-тежки изчислителни операции. PLINQ поддържа паралелно изпълнение на заявки с помощта на множество нишки.

.NET поддържа лесно преобразуване на стандартните LINQ заявки към PLINQ заявки с помощта на `AsParallel()` метода. PLINQ е добър избор за тежки изчислителни операции.

Писане на PLINQ заявки

Клас ParallelEnumerable

`ParallelEnumerable` класът е част от `System.Linq` и `System.Core`.

Освен, че поддържа основните операции познати от LINQ, класът

`ParallelEnumerable` има още няколко метода:

- `AsParallel()`: Извикването на този метод е нужно за поддържане на паралелизъм
- `AsSequential()`: Извикването на този метод води до последователна обработка на паралелна заявка, като променя поведението ѝ.
- `AsOrdered()`: По подразбиране, PLINQ не запазва реда на данните, за да се запази реда е наложително използването на този метод.
- `Aggregate()`: Този метод може да се използва за агрегиране на резултатите от различни части в паралелна заявка.
- `WithDegreesOfParallelism()`: Този метод може да задава максималния брой на нишките, които да обработват заявката

Първа PLINQ заявка

Да се намерят всички числа от 1 до 100000, които се делят на 3.

Използвайте представяне на числата като реда:

```
var range = Enumerable.Range(1, 100000);
```

Решението с обичайна LINQ заявка изглежда по подобен начин:


```
var resultList = range.Where(number => number % 3 == 0).ToList();
```

С помощта на **AsParallel** метода ще направим така че заявката да работи паралелно:

```
var resultList = range.AsParallel().Where(i => i % 3 == 0).ToList();
```

Примерен програмен фрагмент:

```
var range = Enumerable.Range(1, 100000);  
//Here is sequential version  
var resultList = range.Where(i => i % 3 == 0).ToList();  
Console.WriteLine($"Sequential: Total items are {resultList.Count}");  
//Here is Parallel Version using .AsParallel method  
resultList = range.AsParallel().Where(i => i % 3 == 0).ToList();  
resultList = (from i in range.AsParallel()  
where i % 3 == 0  
select i).ToList();  
Console.WriteLine($"Parallel: Total items are {resultList.Count}" );  
Console.WriteLine($"Parallel: Total items are {resultList.Count}");  
The output of this will be as follows:
```

 C:\Program Files\dotnet\dotnet.exe

```
Sequential: Total items are 33333  
Parallel: Total items are 33333
```