Класс Parallel

Класс Parallel также является частью TPL и предназначен для упрощения параллельного выполнения кода. Parallel имеет ряд методов, которые позволяют распараллелить задачу.

Одним из методов, позволяющих параллельное выполнение задач, является метод `Invoke`:

```
static void Main(string[] args)
{
    Parallel.Invoke(Display,
        () => {
                    Console.WriteLine($"Выполняется задача {Task.CurrentId}");
                    Thread.Sleep(3000);
                },
        () => Factorial(5));

    Console.ReadLine();
}

static void Display()
{
    Console.WriteLine($"Выполняется задача {Task.CurrentId}");
    Thread.Sleep(3000);
}

static void Factorial(int x)
{
    int result = 1;

    for (int i = 1; i <= x; i++)
    {
        result *= i;
    }
    Console.WriteLine($"Выполняется задача {Task.CurrentId}");
    Thread.Sleep(3000);
    Console.WriteLine($"Результат {result}");
}
```

Метод `Parallel.Invoke` в качестве параметра принимает массив объектов `Action`, то есть мы можем передать в данный метод набор методов, которые будут вызываться при его выполнении. Количество методов может быть различным, но в данном случае мы определяем выполнение трех методов. Опять же как и в случае с классом Task мы можем передать либо название метода, либо лямбда-выражение.

И таким образом, при наличии нескольких ядер на целевой машине данные методы будут выполняться параллельно на различных ядрах.

Parallel.For

Метод Parallel.For позволяет выполнять итерации цикла параллельно. Он имеет следующее определение: `For(int, int, Action<int>)`, где первый параметр задает начальный индекс элемента в цикле, а второй параметр - конечный индекс. Третий параметр - делегат Action - указывает на метод, который будет выполняться один раз за итерацию:

```
static void Main(string[] args)
{
    Parallel.For(1, 10, Factorial);

    Console.ReadLine();
}

static void Factorial(int x)
{
    int result = 1;

    for (int i = 1; i <= x; i++)
    {
        result *= i;
    }
    Console.WriteLine($"Выполняется задача {Task.CurrentId}");
    Console.WriteLine($"Факториал числа {x} равен {result}");
    Thread.Sleep(3000);
}
```

В данном случае в качестве первого параметра в метод Parallel.For передается число 1, а в качестве второго - число 10. Таким образом, метод будет вести итерацию с 1 до 9 включительно. Третий параметр представляет метод, подсчитывающий факториал числа. Так как этот параметр представляет тип Action<int>, то этот метод в качестве параметра должен принимать объект int.

А в качестве значения параметра в этот метод передается счетчик, который проходит в цикле от 1 до 9 включительно. И метод Factorial, таким образом, вызывается 9 раз.

Parallel.ForEach

Метод Parallel.ForEach осуществляет итерацию по коллекции, реализующей интерфейс IEnumerable, подобно циклу foreach, только осуществляет параллельное выполнение перебора. Он имеет следующее определение: `ParallelLoopResult ForEach<TSource>(IEnumerable<TSource> source,Action<TSource> body)`, где первый параметр представляет перебираемую коллекцию, а второй параметр - делегат, выполняющийся один раз за итерацию для каждого перебираемого элемента коллекции.

На выходе метод возвращает структуру `ParallelLoopResult`, которая содержит информацию о выполнении цикла.

```
static void Main(string[] args)
{
```

```
3       ParallelLoopResult result = Parallel.ForEach<int>(new List<int>() { 1, 3, 5, 8 },
4                   Factorial);
5
6       Console.ReadLine();
7   }
8   static void Factorial(int x)
9   {
10      int result = 1;
11
12      for (int i = 1; i <= x; i++)
13      {
14          result *= i;
15      }
16      Console.WriteLine($"Выполняется задача {Task.CurrentId}");
17      Console.WriteLine($"Факториал числа {x} равен {result}");
18      Thread.Sleep(3000);
19  }
```

В данном случае поскольку мы используем коллекцию объектов int, то и метод, который мы передаем в качестве второго параметра, должен в качестве параметра принимать значение int.

Выход из цикла

В стандартных циклах for и foreach предусмотрен преждевременный выход из цикла с помощью оператора `break`. В методах Parallel.ForEach и Parallel.For мы также можем, не дожидаясь окончания цикла, выйти из него:

```
1   static void Main(string[] args)
2   {
3       ParallelLoopResult result = Parallel.For(1, 10, Factorial);
4
5       if (!result.IsCompleted)
6           Console.WriteLine($"Выполнение цикла завершено на итерации {result.LowestBrea
7       Console.ReadLine();
8   }
9   static void Factorial(int x, ParallelLoopState pls)
10  {
11      int result = 1;
12
13      for (int i = 1; i <= x; i++)
14      {
15          result *= i;
16          if (i == 5)
17              pls.Break();
18      }
19      Console.WriteLine($"Выполняется задача {Task.CurrentId}");
20      Console.WriteLine($"Факториал числа {x} равен {result}");
21  }
```

Здесь метод Factorial, обрабатывающий каждую итерацию, принимает дополнительный параметр - объект **ParallelLoopState**. И если счетчик в цикле достигнет значения 5, вызывается метод Break. Благодаря чему система осуществит выход и прекратит выполнение метода Parallel.For при первом удобном случае.

Методы Parallel.ForEach и Parallel.For возвращают объект **ParallelLoopResult**, наиболее значимыми свойствами которого являются два следующих:

- **IsCompleted**: определяет, завершилось ли полное выполнение параллельного цикла
- **LowestBreakIteration**: возвращает индекс, на котором произошло прерывание работы цикла

Так как у нас на индексе равном 5 происходит прерывание, то свойство IsCompleted будет иметь значение `false`, а `LowestBreakIteration` будет равно 5.