1. Что такое TPL? Как и для чего используется тип Task

Библиотека параллельных задач (TPL) представляет собой **набор открытых типов и API-интерфейсов в пространствах имен System.** **Threading и System**

В основе библиотеки TPL лежит концепция задач, каждая из которых описывает отдельную продолжительную операцию. В библиотеке классов .NET задача представлена специальным классом - классом **Task**, который находится в пространстве имен **System.Threading.Tasks**.

2. Почему эффект от распараллеливания наблюдается на большом

количестве элементов?

Рост эффективности распараллеливания при увеличении размера решаемой системы уравнений объясняется очень просто: **при увеличении размерности решаемой системы уравнений объем вычислительной работы растет пропорционально n3, а объем обменов между процессорами пропорционально n2**

3. В чем основные достоинства работы с задачами по сравнению с

потокми?

<https://habr.com/ru/post/337528/>

**Производительность Task выше**

**Отдельное асинхронное выполнение**

1. Приведите три способа создания и/или запуска Task?

Task tasks = new Task;

tasks.Start()

Task<Person> Task = new Task<Person>(() => new Person("Tom", 37))

var outer = Task.Factory.StartNew(() => {})

outer.Wait();

5. Как и для чего используют методы Wait(), WaitAll() и WaitAny()?

Метод wait() — **переводит вызывающий поток в состояние ожидания**

|  |
| --- |
|  |

**[WaitAll(Task[])](https://learn.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task.waitall?view=net-7.0" \l "system-threading-tasks-task-waitall(system-threading-tasks-task())) Ожидает завершения выполнения всех указанных объектов [Task](https://learn.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task?view=net-7.0).**

**WaitAny() Ожидает завершения выполнения любого из указанных отменяемых объектов [Task](https://learn.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task?view=net-7.0) в течение указанного временного интервала.**

6. Приведите пример синхронного запуска Task?

7. Как создать задачу с возвратом результата?

Task<int> task1 = new Task<int>(() => Sum(4, 5));

task1.Start();

task1.Wait();

Console.WriteLine(task1.GetAwaiter().GetResult());

8. Как обработать исключение, если оно произошло при выполнении

Task?

Try{

}

Catch(Exception ex){}

9. Что такое CancellationToken и как с его помощью отменить

выполнение задач?

Параллельное выполнение задач может занимать много времени. И иногда может возникнуть необходимость прервать выполняемую задачу. Для этого платформа .NET предоставляет структуру **CancellationToken** из пространства имен System.Threading.

CancellationTokenSource cancelTokenSource = new CancellationTokenSource();

CancellationToken token = cancelTokenSource.Token;

Task task2 = new Task(Function, token);

Console.WriteLine("Begin");

task2.Start();

cancelTokenSource.Cancel()

10.Как организовать задачу продолжения (continuation task) ?

int Sum(int a, int b) { return a + b; }

Task<int> task1 = new Task<int>(() => Sum(4, 5));

Task task2 = task1.ContinueWith(task => Console.WriteLine(task1.Result));

task1.Start();

task2.Wait();

11.Как и для чего используется объект ожидания при создании задач

продолжения?

12.Поясните назначение класса System.Threading.Tasks.Parallel?

Парралельный запус выполнения задач

13.Приведите пример задачи с Parallel.For(int, int, Action<int>)

Parallel.For(1, 100, Square);

// вычисляем квадрат числа

void Square(int n)

{

Console.WriteLine($"Выполняется задача {Task.CurrentId}");

Console.WriteLine($"Квадрат числа {n} равен {n \* n}");

Thread.Sleep(2000);

}

14.Приведите пример задачи с Parallel.ForEach

ParallelLoopResult result = Parallel.ForEach<int>(

new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 },

Square

);

// вычисляем квадрат числа

void Square(int n)

{

Console.WriteLine($"Выполняется задача {Task.CurrentId}");

Console.WriteLine($"Квадрат числа {n} равен {n \* n}");

Thread.Sleep(2000);

}

15.Приведите пример с Parallel.Invoke()

int Sum(int a, int b) { Console.WriteLine(a + b); return a + b; }

int Subtraction(int a, int b){ Console.WriteLine(a - b); return a - b; }

int Division(int a, int b) { Console.WriteLine(a / b); return a / b; }

Parallel.Invoke(() => Sum(1, 2), () => Subtraction(4, 2), () => Division(6, 3));

16.Как с использованием CancellationToken отменить параллельные

операции?

17.Для чего используют BlockingCollection<T>, в чем ее особенность?

Предоставляет возможности блокировки и ограничения для потокобезопасных коллекций

https://learn.microsoft.com/ru-ru/dotnet/api/system.collections.concurrent.blockingcollection-1?view=net-7.0

18.Как используя async и await организовать асинхронное выполенение

метода?