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

TPL (Task Parallel Library), основной функционал которой располагается в пространстве имен System.Threading.Tasks. Данная библиотека позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах, если на целевом компьютере имеется несколько ядер.

**2. Почему эффект от распараллеливания наблюдается на большом количестве элементов?**

Потому что Parallel сначала получает очень много информации о системе(например, процессорное время), на что также затрачивается время. Поэтому разница видна лишь тогда, когда используются огромные блоки кода.

**3. В чем основные достоинства работы с задачами по сравнению с потоками?**

Автоматически занимает поток для своего выполнения из пула потоков, диспетчеризация, отмена и обработка исключений.

**4. Приведите три способа создания и/или запуска Task?**

task.Start();

Task task = Task.Factory.StartNew(() => Console.WriteLine("Hello Task!"));

Task task = Task.Run(() => Console.WriteLine("Hello Task!"));

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

Wait() ждёт завершение конкретной задачи

WaitAll() ждёт завершение всех задач из набора

WaitAny() ждёт завершение любой задачи из набора

**6. Приведите пример синхронного запуска Task?**

**7. Как создать задачу с возвратом результата?**

Объявить делегат Func<> и передать его аргументом в запускаемую задачу. Чтобы получить результат используется свойство Result

**8. Как обработать исключение, если оно произошло при выполнении Task?**

Описать исключение в теле кода задачи/блоками try - catch

**9. Что такое CancellationToken и как с его помощью отменить выполнение задач?**

CancellationToken - класс, используемый для получения токена отмены и отмены задачи/задач.

Нужно получить токен и затем вызвать метод запроса отмены. В ближайшей же итерации задача будет отменена.

**10.Как организовать задачу продолжения (continuation task)?**

Задачи продолжения или continuation task позволяют определить задачи, которые выполняются после завершения других задач.

Task task1 = new Task(()=>{

Console.WriteLine("Task: {0}", Task.CurrentId);

});

Task task2 = task1.ContinueWith(Display);

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

Объект ожидания позволяет продолжить выполнение задачи после выполнения предыдущего блока кода задачи и использовать объект ожидания, чтобы, например, выбрать действие на основе данного объекта(результата).

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

Класс Parallel позволяет распаллеливать блоки кода между потоками автоматически. Однако использовать его стоит только если код достаточно тяжёл или же при очень большом кол-ве элементов. Иначе можно наоборот нанести вред производительности приложения.

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

**14.Приведите пример задачи с Parallel.ForEach**

**15.Приведите пример с Parallel.Invoke()**

**16.Как с использованием CancellationToken отменить параллельные операции?**

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

BlockingCollection<T>

Коллекция, которая осуществляет блокировку и ожидает, пока не появится возможность выполнить действие по добавлению или извлечению элемента. BlockingCollection<T> предлагает интерфейс для добавления и извлечения элементов методами Add() и Take(). Эти методы блокируют поток и затем ожидают, пока не появится возможность выполнить задачу.

это потокобезопасный класс коллекции, обеспечивающий следующие возможности:

|  |
| --- |
|  |
|  | · реализует шаблон "производитель-получатель"; |
|  | · поддерживает параллельное добавление и извлечение элементов из нескольких потоков; |
|  | · допускает указание максимальной емкости; |
|  | · поддерживает операции вставки и удаления, блокирующиеся при опустошении или заполнении коллекции; |
|  | · поддерживает условные операции вставки и удалении, не блокирующиеся или блокирующиеся лишь на определенное время; |

**18.Как используя async и await организовать асинхронное выполенение метода?**

создать метод async и в него поместить несколько await.

async и await, цель которых - упростить написание асинхронного кода. Они используются вместе для создания асинхронного метода

|  |
| --- |
|  |
|  | static async void FactorialAsync() |
|  | { |
|  | Console.WriteLine("Начало метода FactorialAsync"); // выполняется синхронно |
|  | await Task.Run(()=>Factorial()); // выполняется асинхронно |
|  | Console.WriteLin("Конец метода FactorialAsync"); |
|  | } |