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

Библиотека параллельных задач TPL (Task Parallel Library)

позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах (для создания многопоточных приложений).

Task описывает отдельную продолжительную операцию, которая запускается асинхронно в одном из потоков из пула потоков (можно запускать синхронно в текущем потоке) – подобна потокам, но абстракция более высокого уровня.

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

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

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

1) Создание объекта Task и вызова у него метода Start

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

task.Start();

2) использовании статического метода Task.Factory.StartNew()

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

3) использование статического метода **Task.Run()**

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

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

WaitAll Ожидает завершения выполнения всех указанных объектов [Task](https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task?view=netcore-1.1)

Wait Ожидает завершения выполнения задачи [Task](https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.task?view=netcore-3.0)(конкретной).

WaitAny ожидает завершения выполнения любой задачи Task

6. Приведите пример синхронного запуска Task?

Task task = new Task(Display);

            task.Start();

            Console.WriteLine("Завершение метода Main");

            Console.ReadLine();

        }

        static void Display()

        {

            Console.WriteLine("Начало работы метода Display");

            Console.WriteLine("Завершение работы метода Display");

        }

7. Как создать задачу с возвратом результата?

Использовать асинхронную операцию Task<T>, которая возвращает значение

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

 помощью метода **Handle()**можно выборочно обработать исключения, которые мы знаем, как обработать, остальное отправиться вверх по стеку.

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

Класс CancellationToken предназначен для прерыщения выполнения задач

Для отмены операции нам надо создать и использовать токен. Вначале создается объект CancellationTokenSource:

CancellationTokenSource cancelTokenSource = new CancellationTokenSource();

Затем из него получаем сам токен:

CancellationToken token = cancelTokenSource.Token;

Чтобы отменить операцию, необходимо вызвать метод Cancel() у объекта CancellationTokenSource

cancelTokenSource.Cancel();

10. Как организовать задачу продолжения (continuation task) ?

Первая задача задается с помощью лямбда-выражения. Вторая задача - задача продолжения задачется с помощью метода **ContinueWith**, который в качестве параметра принимает делегат Action<Task>. То есть метод Display, который передается в данный метод в качестве значения параметра, должен принимать параметр типа Task.

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

Ждет завершения 1 задачи перед выполнением второй

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

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

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?

Для отмены выполнения параллельных операций, запущенных с помощью методов **Parallel.For()** и **Parallel.ForEach()**, можно использовать перегруженные версии данных методов, которые принимают в качестве параметра объект **ParallelOptions**. Данный объект позволяет установить токен

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

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?

**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.WriteLine("Конец метода FactorialAsync");

        }