Класс Task

1. Основные отличия класса Task (Task<>)
2. Примеры применения класса Task

Класс Task обеспечивает **запуск асинхронных вычислительных операций**, а также содержит встроенный **механизм, позволяющий узнать о завершении операции и получить возвращаемое значение**.

Пространство имен **System.Threading.Tasks**

По сравнению с потоком (**Thread**) задача (Task) является абстракцией более высокого уровня - она представляет собой асинхронную операцию, которая может быть или не быть подкреплена потоком.

Типы Task появились в версии **.NET Framework 4.0** как часть библиотеки параллельного программирования (**TPL**). Однако с тех пор они были усовершенствованы (за счет применения объектов ожидания (**awaiter**)), чтобы функционировать столь же эффективно в более универсальных сценариях реализации параллелизма, и имеют поддерживающие типы для **асинхронных функций C#.**

1. Элементарная единица исполнения инкапсулируется в TPL средствами класса Task.
2. Класс Task отличается от класса Thread тем, что он является абстракцией, представляющей асинхронную операцию, а в классе Thread инкапсулируется поток исполнения.
3. Исполнением задач класса Task управляет планировщик задач, который работает с пулом потоков.

в **.NET Framework 4.5** простейший способ запуска задачи, подкреплённой потоком, - метод **Task.Run()**

```
Task.Run(() => Console.WriteLine("Go"));
```

в **.NET Framework 4.0** – метод **Task.Factory.StartNew()**

```
Task.Factory.StartNew(() => Console.WriteLine("Go"));
```

!!! По умолчанию задачи используют потоки из пула потоков, которые являются фоновыми потоками.

Класс **Task** имеет обобщенный подкласс по имени **Task<TResult>**, который позволяет задаче выдавать возвращаемое значение.

```
Task<int> task = Task.Run(() =>
{
  Console.WriteLine("Go");
  return 3;
});
```

Класс Task

```
public class Task : IThreadPoolWorkItem, IAsyncResult, IDisposable
{
  public Task(Action action);
  public Task(Action<object> action, object state);
  public Task(Action action, CancellationToken cancellationToken);
  public Task(Action action, TaskCreationOptions creationOptions);
  public Task(Action<object> action, object state,
                                          CancellationToken cancellationToken);
  public Task(Action<object> action, object state,
                                          TaskCreationOptions creationOptions);
  public Task(Action action, CancellationToken cancellationToken,
                                          TaskCreationOptions creationOptions);
  public Task(Action<object> action, object state,
    CancellationToken cancellationToken, TaskCreationOptions creationOptions);
...
}
```

```
delegate void Action();

delegate void Action<in T>(T obj)
```

Класс Task

```
[Flags, Serializable]
public enum TaskCreationOptions
{
  None = 0x0000, // По умолчанию

  // Сообщает планировщику, что задание должно быть поставлено
  // на выполнение по возможности скорее
  PreferFairness = 0x0001,

  // Сообщает планировщику, что ему следует более активно
  // создавать потоки в пуле потоков.
  LongRunning = 0x0002,

  // Всегда учитывается: присоединяет задание к его родителю
  AttachedToParent = 0x0004,

  // Если задача пытается присоединиться к родительской задаче,
  // она интерпретируется как обычная, а не как дочерняя задача.
  DenyChildAttach = 0x0008,

  // Заставляет дочерние задачи использовать планировщик по умолчанию
  // вместо родительского планировщика.
  HideScheduler = 0x0010
}
```

Класс Task<>

```
public class Task<TResult> : Task
{
 public Task(Func<TResult> function);
 public Task(Func<object, TResult> function, object state);
 public Task(Func<TResult> function, CancellationToken cancellationToken);
 public Task(Func<TResult> function, TaskCreationOptions creationOptions);
 public Task(Func<object, TResult> function, object state,
                                              CancellationToken cancellationToken);
 public Task(Func<object, TResult> function, object state,
                                              TaskCreationOptions creationOptions);
 public Task(Func<TResult> function,
    CancellationToken cancellationToken, TaskCreationOptions creationOptions);
 public Task(Func<object, TResult> function, object state,
    CancellationToken cancellationToken, TaskCreationOptions creationOptions);
 ...
}
```

```
delegate TResult Func<out TResult>();

delegate TResult Func<in T, out TResult>(T arg);
```

Класс Task. Метод Run()

```
public class Task : IThreadPoolWorkItem, IAsyncResult, IDisposable
{
  ...
  public static Task Run(Action action);
  public static Task<TResult> Run<TResult>(Func<Task<TResult>> function);
  public static Task Run(Func<Task> function);
  public static Task<TResult> Run<TResult>(Func<TResult> function);
  public static Task Run(Action action, CancellationToken cancellationToken);
  public static Task<TResult> Run<TResult>(Func<Task<TResult>> function,
                                           CancellationToken cancellationToken);
  public static Task Run(Func<Task> function,
                                           CancellationToken cancellationToken);
  public static Task<TResult> Run<TResult>(Func<TResult> function,
                                           CancellationToken cancellationToken);
  ...
}
```

Класс Task

```
public enum TaskStatus
{
  // Флаги, обозначающие состояние задания:
  Created,     // Задание создано в явном виде и может быть запущено вручную
  WaitingForActivation, // Задание создано неявно и запускается автоматически
  WaitingToRun,         // Задание запланировано, но еще не запущено
  Running,              // Задание выполняется

                //Задание ждет завершения дочерних заданий, чтобы завершиться
  WaitingForChildrenToComplete,

  // Возможные окончательные состояния задания:
  RanToCompletion,
  Canceled,
  Faulted
}
```