C# Асинхронное программирование

Исключения в асинхронном коде. Скоординированная отмена. Блокировки

C# Асинхронное программирование

Автор курса

Гнатюк Владислав

MCID:16354168

После урока обязательно

Повторите этот урок в видео формате на ITVDN.com

Проверьте как Вы усвоили данный материал на TestProvider.com

Исключения в асинхронном коде. Скоординированная отмена. Блокировки

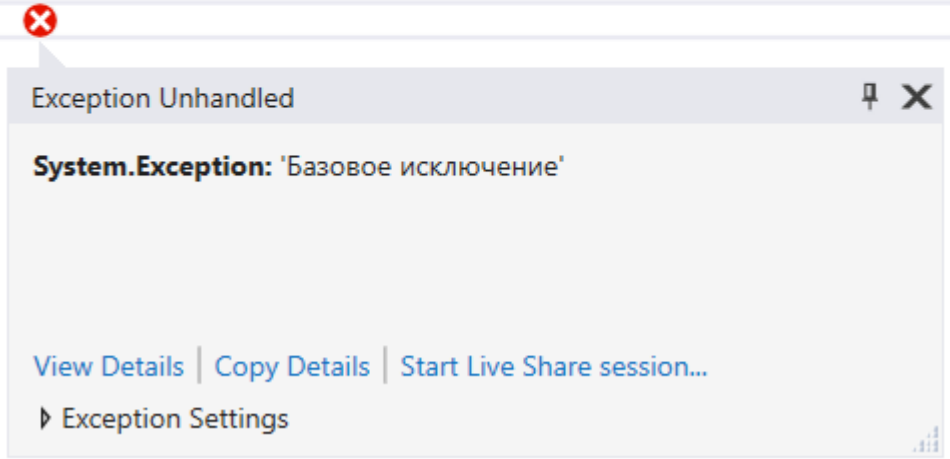
План урока

1) Обработка исключений из асинхронного кода
2) Скоординированная отмена
3) Блокировки
4) Прогресс асинхронной операции. IProgress<T>

Исключение

Исключение – ситуация, при которой продолжение выполнения кода в соответствии с базовым алгоритмом невозможно или бессмысленно.

```
namespace ExceptionExample
{
    0 references
    internal class Program
    {
        0 references
        private static void Main()
        {
            throw new Exception("Базовое исключение");
        }
    }
}
```



Виды исключений

- **Синхронные** – возникают в заранее известных, определенных точках программы. Такой вид исключения легко обрабатывается конструкцией try – catch.

- **Асинхронные** – возникают в любой момент времени в другом потоке и не зависят от того, какую конкретно инструкцию выполняет поток. Такой вид исключения тяжело обрабатывать, из-за непредсказуемости времени и места возникновения.

Исключения в контексте вторичного потока

Если исключение возникает в контексте потока (Thread) – это разрушает работу приложения. Такое исключение можно поймать и обработать только в контексте того потока, в котором оно произошло.

Опасным кодом является использование класса Thread или ThreadPool для выполнения операции в контексте вторичного потока. Если код не помещен в конструкцию try-catch – это разрушает работу приложения.

Исключения в контексте задачи

При выполнении операции в контексте задачи, поведение исключения меняется. Все исключения, возникшие в контексте задачи, будут записаны в свойство Exception. Это свойство типа AggregateException. При этом приложение не будет разрушено.

AggregateException – представляет класс, который может принять несколько исключений и записать в себя. Он содержит коллекцию всех возникших исключений асинхронной операции.

Обработка асинхронных исключений задач (Task)

Обработка исключений из асинхронного кода привычным способом через конструкцию try – catch не получится.

В Debug вы получите исключение, это помогает отладке.

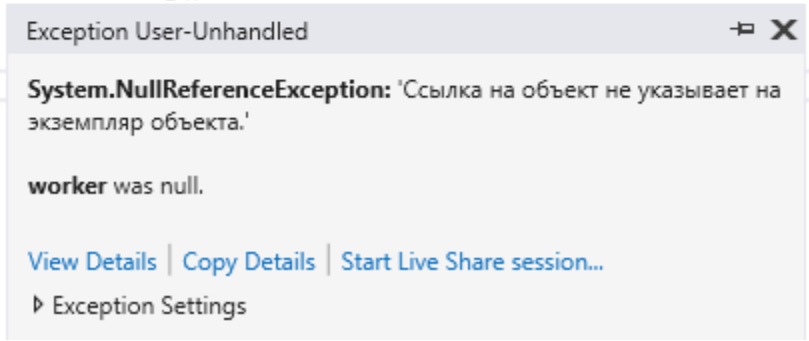
В Release исключение будет проигнорировано.

```
0 references
private static void Main()
{
    try
    {
        Task task = new Task(CalculateSomething);
        task.Start();

        Console.WriteLine("Имитация работы метода Main");

        Console.ReadKey();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Обработка ошибки : {ex?.InnerException?.Message}");
    }
}

1 reference
private static void CalculateSomething()
{
    Worker worker = null;
    worker.Do();
}
```



Обработка асинхронных исключений (Task)

Доступные API, выдающие исключения при вызове:

- Свойство Exception типа AggregateException;
- Методы ожидания – Wait, WaitAll, WaitAny;
- Получение результата операции – свойство Result;
- Оператор await;
- Асинхронные методы ожидания – WhenAll, WhenAny.

```
Task<int> task = new Task<int>(CalculateSomething);
task.Start();

// Выполнение полезного кода из основного потока...

try
{
        task.Wait();
        var result = task.Result;
        var result = await task;
}
catch(AggregateException ex)
{
        Console.WriteLine($"Обработка ошибка {ex}");
}
```

Скоординированная отмена

Скоординированная отмена – отмена, которая требует помимо команды на прерывание выполнения операции, подтверждения в месте операции, которая должна быть отменена.

Для обеспечения этого шаблона в .NET есть три специальных типа :

- CancellationTokenSource – источник токенов отмены, класс для управления отменами.
- **CancellationToken** – токен (некий талон/жетон отмены), проверяет наличие приказа на отмену, а так же подтверждает, что нужно совершить отмену.
- OperationCanceledException – исключение, представляющее в .NET отмену.

Блокировка (Deadlock)

Блокировка – ситуация, когда несколько потоков ожидают ресурсы, занятые друг другом, что не дает продолжать выполнение.

Возможна ситуация, когда в блокировке участвует один поток – он сам себя ждет и никто не дает ему сигнал для продолжения работы.

Прогресс асинхронной операции

Бывает, что запрошенная пользователем операция может занять много времени для выполнения. Поэтому, вы бы хотели предоставить пользователю какую-то возможность, которая позволит следить за ходом выполнения процесса.

Эта возможность может повлиять на решения пользователя:

1. Пользователь проинформирован, что операция была начата.
2. Пользователь видит прогресс выполнения операции.
3. Пользователь может отменить операцию, видя, что индикатор операции долго не отвечает или зависает. (Если отмена поддерживается асинхронной операцией)

Для отображения прогресса могут использоваться самые разные элементы, начиная от банальных прогресс баров, заканчивая самой разной анимацией. Этот выбор ложится на вас, что вы считаете предпочтительным.

Прогресс асинхронной операции

Мониторинг асинхронной операции – это не новое решение, но оно приветствуется в TAP подходе. Очень важно, при возможности, добавлять какое-то средство для отображения хода асинхронной операции. Ведь использование асинхронности не означает, что вы не заставите пользователя ждать результата операции.

Иногда приложение не может продолжить свою работу без результата, а цель использования асинхронности - не заблокировать интерфейс (UI) или не потреблять ресурсы сервера (если речь о серверном приложении). Индикатор может помочь пользователю спокойно дождаться окончания операции.

Интерфейс IProgress<T>

Для представления прогресса асинхронной операции был введен интерфейс IProgress<T>.

Благодаря этому интерфейсу, вы можете перегрузить свои методы с асинхронными операциями новой перегрузкой, принимающей в качестве параметра тип реализующий интерфейс IProgress<T>.

Методы интерфейса:
void Report(T item) – сообщает о прогрессе операции. Передает значение обновленного прогресса.

Базовый класс Progress<T>

Progress<T> – базовый класс для отображения прогресса асинхронной операции. Можно использовать этот тип напрямую.

Класс удобен, потому что имеет несколько вариантов оповещения. В числе вариантов присутствует как инициация события и всех его обработчиков, так и вызов установленного делегата Action<T>, когда задача сообщает о ходе выполнения.

Класс автоматически захватывает контекст синхронизации, если он доступен. Если нет, будет использован ThreadPool.

Dispose Task

Задача реализует интерфейс IDisposable для освобождения ресурсов. Она это делает из-за использования внутри себя других ресурсов, которые тоже реализуют интерфейс IDisposable.

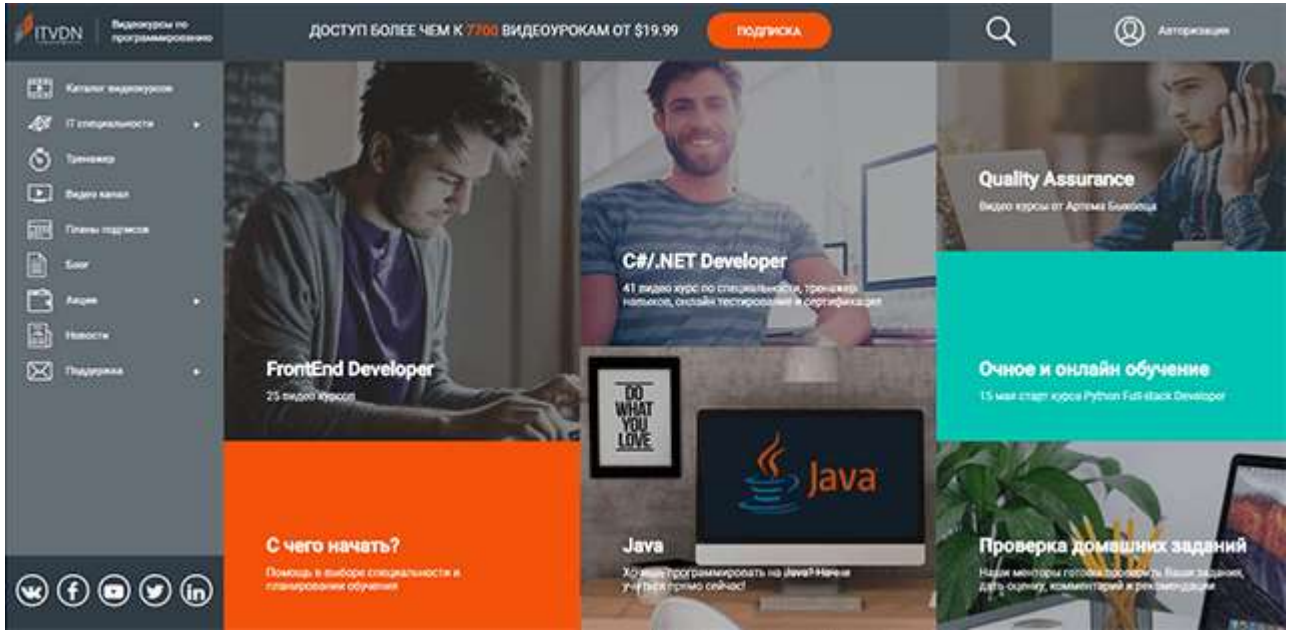
Многие задаются вопросом: «А необходимо ли мне освобождать ресурсы после завершения работы с задачей?».

Ответ: Нет необходимости **всегда** вызывать метод Dispose на задаче, если ваше приложение работает на .NET Framework 4.5 и выше.

Но, если тестирование производительности и масштабирования показывает, что общая производительность вашего приложения будет улучшена за счет явной утилизации задач, то вы можете использовать метод Dispose. При этом, вы должны быть уверены, что в момент утилизации задачи она выполнена и ее никто более не использует.

Смотрите наши уроки в видео формате

ITVDN.com



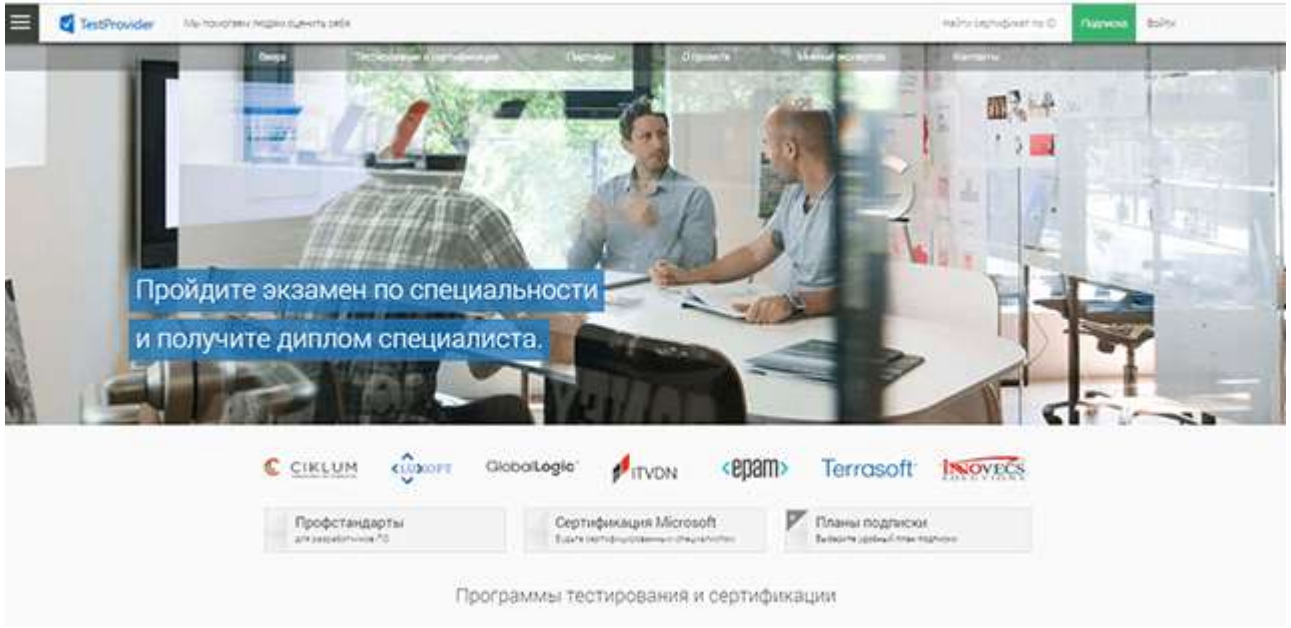
Посмотрите этот урок в видео формате на образовательном портале ITVDN.com для закрепления пройденного материала.

Курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics и другими высококвалифицированными разработчиками.

ITVDN
IT VIDEO DEVELOPERS NETWORK

Проверка знаний

TestProvider.com



TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

TestProvider

Q&A

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Windows Azure Cloud Partner EBA

Асинхронное программирование

После урока обязательно

Повторите этот урок в видео формате на ITVDN.com

Проверьте как Вы усвоили данный материал на TestProvider.com