# Асинхронный шаблон программирования
# Ключевые слова async await. Техническая реализация.

**№ урока:** 4 **Курс:** C# Асинхронное программирование

**Средства обучения:** Компьютер с установленной Visual Studio

## Обзор, цель и назначение урока

Урок познакомит вас с работой ключевых слов async await. Будут подробно рассмотрены правила использования каждого из этих ключевых слов. Для более глубокого понимания работы async await, будет рассмотрена их внутренняя реализация, которая обеспечивается с помощью специальных типов и некоторой работы компилятора.

## Изучив материал данного занятия, учащийся сможет:

- Понимать, что делает ключевое слово async.
- Понимать, что делает ключевое слово await.
- Создавать асинхронные методы, используя модификатор async.
- Понимать внутреннюю реализацию ключевых слов async await.
- Понимать предназначение внутренних типов для обеспечения инфраструктуры ключевых слов async await.
- Создавать и использовать асинхронный метод Main.

## Содержание урока

1. Рассмотрение ключевого слова async
2. Рассмотрение ключевого слова await
3. Асинхронные методы
4. Правила использования ключевых слов async await
5. Ожидаемые методы
6. Конечный автомат async await
7. Метод-заглушка
8. Задача-марионетка
9. Строители асинхронных методов
10. Интерфейс IAsyncStateMachine
11. Метод MoveNext()
12. Объект ожидания TaskAwaiter
13. Метод AwaitUnsafeOnCompleted
14. Асинхронный метод Main
15. Результат асинхронной операции

## Резюме

- Ключевое слово **async** – является модификатором для методов. Указывает, что метод является асинхронным. Модификатор async позволяет использовать в асинхронном методе ключевое слово await и указывает компилятору на необходимость создания конечного автомата для обеспечения работы асинхронного метода.
- Ключевое слово **await** – является унарным оператором, операнд которого располагается справа от самого оператора. Применение оператора await означает, что необходимо дождаться завершения выполнения асинхронной операции. При этом если ожидание будет произведено, то вызывающий поток будет освобожден для своих дальнейших действий, а код, находящейся после оператора await, по завершению асинхронной операции будет выполнен в виде продолжения.



CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- Работу ключевых слов async await обеспечивает компилятор, поэтому без его поддержки не будет работать функционал ключевых слов.
- **Асинхронные методы** – это методы, которые используют ключевые слова async/await и один из специальных типов возвращаемого значения. В имени метода у них указан суффикс Async или TaskAsync для быстрой узнаваемости.
- Наличие ключевого слова async не означает, что метод будет выполняться во вторичном/фоновом потоке.
- Доступные стандартные типы возвращаемых значений для асинхронных методов (с модификатором async):
  - void – для асинхронных обработчиков событий;
  - Task – для асинхронной операции, которая не возвращает значение;
  - Task<TResult> - для асинхронной операции, которая возвращает значение;
  - ValueTask – для асинхронной операции, которая не возвращает значение;
  - ValueTask<TResult> – для асинхронной операции, которая возвращает значение.
- Для применения оператора await к типу, данный тип должен иметь доступный метод GetAwaiter. Возвращаемый объект метода должен иметь реализацию одного из интерфейсов (ICriticalNotifyCompletion или INotifyCompletion), свойство bool IsCompleted { get; } и метод GetResult. Тип возвращаемого значения метода GetResult зависит от того, должна ли возвращать асинхронная операция результат. Если да, то тип метода GetResult должен совпадать с результатом асинхронной операции. Если нет – тип возвращаемого значения должен быть void.
- **Ожидаемые методы** – это асинхронные методы, завершение которых можно подождать, если необходим результат их работы в данный момент.
- Ожидать завершения асинхронных методов и задач необходимо с помощью оператора await. Есть и другие способы ожидания, но самый эффективный – оператор await.
- **Конечный автомат** – это модель вычислений, которая позволяет объекту изменить свое поведение, в зависимости от своего внутреннего состояния.
- Работу ключевых слов async await обслуживает конечный автомат. Он создается компилятором с помощью интерфейса IAsyncStateMachine и специальных строителей асинхронных методов.
- Асинхронный метод будет превращен в метод-заглушку, который занимается созданием, настройкой и запуском конечного автомата. Тело асинхронного метода переносится в метод MoveNext конечного автомата с оптимизациями и изменениями от компилятора для его удобства.
- Метод-заглушка может возвращать задачу-марионетку, если его тип возвращаемого значения отличный от void.
- **Задача-марионетка** – это задача, жизненным циклом которой управляет программист. Результат выполнения такой задачи указывается программистом в любое время, когда решит программист. Результат может быть как успешным, так и провальным (исключение).
- **Строители асинхронных методов** – это структуры, которые владеют функционалом для внедрения инфраструктуры фиксации результатов или исключений из асинхронных операций, а также для реализации функционала оператора await в асинхронных методах. Эти же типы используются компилятором для создания задач-марионеток в асинхронных методах.
- Стандартные разновидности строителей:
  - AsyncVoidMethodBuilder
  - AsyncTaskMethodBuilder
  - AsyncTaskMethodBuilder<TResult>
  - AsyncValueTaskMethodBuilder
  - AsyncValueTaskMethodBuilder<TResult>
- Не рекомендуется использовать строители асинхронных методов напрямую. Для создания своих задач-марионеток необходимо использовать класс TaskCompletionSource.
- Конечный автомат для async await – это объект, способный представить состояние асинхронного метода, которое можно сохранить при достижении оператора await и восстановить позже, для дальнейшего продолжения выполнения асинхронного метода.
- Конечный автомат выступает в роли типа, который сохраняет состояние и локальные переменные метода в виде полей. При сохранении объекта такого типа мы полноценно сохраняем состояние асинхронного метода в любой точке, для дальнейшего возобновления его работы позже.



CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- Конечный автомат для повышения производительности описывается структурой, ведь при синхронном завершении асинхронного метода не придется выделять память для объекта кучи.
- Конечный автомат в сборке DEBUG – представлен классом.
- Конечный автомат в сборке RELEASE – представлен структурой.
- Конечный автомат async await может иметь достаточно много полей. Их можно классифицировать следующим образом:
  - Состояние конечного автомата;
  - Строитель асинхронных методов;
  - Объекты ожидания;
  - Параметры асинхронного метода;
  - Локальные переменные асинхронного метода;
  - Временные переменные стека;
  - Внешний тип.
- Конечный автомат может иметь три разных состояния:
  - «-1» - начальное состояние или состояние выполнения;
  - «-2» - конечное состояние. Указывает, что работа завершена (успешно или с ошибкой);
  - Любое другое значение – означает приостановку через использование оператора await.
- Интерфейс IAsyncStateMachine имеет два метода:
  - void MoveNext();
  - void SetStateMachine(IAsyncStateMachine stateMachine).
- Метод MoveNext – занимается выполнением тела асинхронного метода и перемещением конечного автомата в следующее состояние.
- Метод SetStateMachine – упаковывает конечный автомат из стека на кучу. Тело этого метода всегда будет одинаковым и состоит оно из одной строки (если конечный автомат представлен структурой):

  this.builder.SetStateMachine(stateMachine);
- Метод SetStateMachine будет вызван в случае первого срабатывания оператора await. Он упаковывает конечный автомат один раз. Все дальнейшие действия будут происходить на упакованном конечном автомате.
- Метод MoveNext запускается через вызов метода Start на строителе асинхронных методов. Каждый последующий запуск происходит в виде продолжения, когда ему необходимо возобновить выполнение после приостановки, вызванной оператором await.
- Метод MoveNext выполняет тело асинхронного метода, которое было перемещено компилятором в этот метод с оптимизациями и изменениями.
- Метод MoveNext выполнится полностью синхронно в одном вызывающем потоке, если ни один из операторов await не запросит ожидание завершения асинхронной операции.
- Тело асинхронного метода помещается в тело блока try метода MoveNext, чтобы при возникновении ошибки во время выполнения асинхронной операции поймать ее в блоке catch и зафиксировать с помощью строителя асинхронных методов.
- **Объект ожидания** – это специальный объект, который способен взаимодействовать с оператором await. Этот объект поддерживает ожидание завершения асинхронной операции. Для получения такого объекта оператор await использует метод GetAwaiter.
- У задач есть реализация объектов ожидания – это структура TaskAwaiter для типа Task и TaskAwaiter<TResult> для типа Task<TResult>.
- Объект ожидания должен реализовать свойство bool IsCompleted { get; }. С помощью этого свойства оператор await будет проверять, завершена ли асинхронная операция.
- Объект ожидания должен реализовать метод void/TResult GetResult(). Тип возвращаемого значения метода зависит от задачи:
  - void - Task
  - TResult - Task<TResult>
- Объект ожидания должен реализовать один из следующих интерфейсов: INotifyCompletion или ICriticalNotifyCompletion.
- Интерфейс INotifyCompletion имеет один метод:
  - void OnCompleted(Action continuation);
- Интерфейс ICriticalNotifyCompletion наследуется от интерфейса INotifyCompletion и имеет два метода:
  - void OnCompleted(**Action continuation**);



CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

    - void UnsafeOnCompleted(**Action continuation**).
- Рекомендуется, чтобы объект ожидания реализовывал интерфейс ICriticalNotifyCompletion. Потому что компилятор предпочитает его для выбора.
- Отличие метода OnCompleted от UnsafeOnCompleted в том, что метод UnsafeOnCompleted является небезопасным, поэтому он не захватывает контекст выполнения (ExecutionContext) для выполнения в нем делегата продолжения. Захватом контекста выполнения также занимается строитель асинхронных методов. Он помещает продолжение на выполнение в этот контекст. Компилятор предпочитает метод UnsafeOnCompleted, чтобы не захватывать контекст выполнения дважды. Так производится оптимизация.
- Работа оператора await:
    - Получение объекта ожидания от асинхронной операции (вызов метода GetAwaiter).
    - Проверка на завершение асинхронной операции (вызов свойства IsCompleted).
    - Если асинхронная операция завершена – вызвать метод GetResult для завершения ожидания и, возможно, получения результата операции. После, продолжить далее синхронно выполнять код метода MoveNext.
    - Если асинхронная операция не завершена:
        - Сохранить необходимые значения локальных переменных метода. Изменить состояние конечного автомата.
        - Вызвать метод AwaitUnsafeOnCompleted/AwaitOnCompleted (в зависимости от наличия реализованного интерфейса объекта ожидания).
        - Освободить вызывающий поток.
        - По завершению работы асинхронной операции будет вызвано продолжение. Оно начнет выполнять код метода с точки останова.
- Метод AwaitUnsafeOnCompleted/AwaitOnCompleted – занимается планированием конечного автомата, чтобы перейти к следующему действию по завершению заданного объекта типа awaiter (Объект ожидания). В этом методе происходит проверка на упаковку конечного автомата. Если он не упакован, то будет выполнена команда упаковки.
- После версии C# 7.1 нам доступна возможность создавать асинхронный метод Main. Необходимо указать модификатор async и использовать один из следующих типов возвращаемых значений: Task, Task<int>. Таким образом, метод Main позволит использовать внутри себя оператор await и при этом останется точкой входа.
- У асинхронной операции может быть результат. Для его извлечения можно использовать:
    - Свойство Result, вызванное на экземпляре класса Task<TResult>;
    - Метод GetResult, вызванный на объекте ожидания;
    - Оператор await.
- Рекомендуется использовать оператор await для получения результата асинхронной операции.
- Если ваш асинхронный метод имеет тип возвращаемого значения Task<TResult> и модификатор async, то вам потребуется вернуть результат, но в виде значения TResult, а не Task<TResult>.

## Закрепление материала

- Дайте определение ключевого слова async.
- Дайте определение ключевого слова await.
- Какие типы возвращаемых значений можно использовать для асинхронного метода?
- Что требует от типа для своей работы оператор await?
- Означает ли ключевое слово async, что метод выполнится в контексте вторичного потока?
- Что на самом деле представляют собой ключевые слова async await?
- Что такое конечный автомат?
- Что такое «метод-заглушка» при работе с async await?
- Что такое задача-марионетка?
- Что такое строитель асинхронных методов?
- Зачем нужен строитель асинхронных методов?
- Как компилятор выбирает строителя асинхронных методов?
- Как компилятор реализует объект конечного автомата async await?
- Какие виды полей могут быть у конечного автомата async await?



CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- Сколько методов есть в интерфейсе IAsyncStateMachine? Какие у них типы возвращаемых значений и сигнатура?
- В чем цель метода SetStateMachine?
- Как компилятор реализует метод SetStateMachine?
- Всегда ли метод SetStateMachine реализован?
- Сколько раз может быть упакован конечный автомат?
- Конечный автомат обязательно упаковывается?
- В чем цель метода MoveNext?
- Кем вызывается метод MoveNext?
- Какое состояние означает выполнение конечного автомата?
- Когда оператор await освобождает вызывающий поток?
- Всегда ли оператор await освобождает вызывающий поток?
- Может ли метод MoveNext завершится синхронно (без изменений потока)?
- Что произойдет с методом MoveNext, если при выполнении асинхронной операции будет получено исключение?
- Какие состояния бывают у конечного автомата async await?
- Что такое объект ожидания?
- В чем отличия интерфейса INotifyCompletion и ICriticalNotifyCompletion?
- Какими способами можно получить результат асинхронной операции?
- Какое состояние конечного автомата означает завершение его работы?
- Какой способ является самым оптимальным и эффективным для получения результата асинхронной операции?
- Как создать асинхронный метод Main?
- Сколько перегрузок асинхронного метода Main доступно?

## Дополнительное задание

Задание
Создайте приложение по шаблону Windows Form. Переместите из элементов управления (ToolBox) на форму текстовое поле и кнопку. Создайте закрытый метод GenerateAnswer, который ничего не принимает и возвращает string. В теле метода вызовите метод Sleep, передав туда значение 10000, после с помощью оператора return верните строку «Привет мир!». Создайте обработчик события для добавленной вами кнопки. Сделайте обработчик события асинхронным, добавив модификатор async к методу. Далее, создайте следующий код в указанном порядке:

1. Сделайте кнопку неактивной на время асинхронной операции.
2. Вызовите через задачу (Task.Run) метод GenerateAnswer. Примените оператор await к запущенной задаче, чтобы не блокировать UI интерфейс. Возвращаемое значение оператора await запишите в переменную.
3. Значение переменной запишите в текстовое поле, добавив к строке $« Выполнено в потоке {Thread.CurrentThread.ManagedThreadId}».
4. Сделайте кнопку вновь активной для нажатий.

Посмотрите на результаты работы. Каковы отличия работы ключевых слов async await здесь от консольных приложений.

## Самостоятельная деятельность учащегося

Задание 1
Выучите основные конструкции и понятия, рассмотренные на уроке.

Задание 2
Создайте приложение по шаблону Console Application. Создайте метод с названием CalculateFactorial, который считает факториал переданного в параметрах числа и возвращает результат. Создайте асинхронный метод с названием CalculateFactorialAsync, который асинхронно, в контексте задачи выполняет метод CalculateFactorial, но не возвращает значение факториала, а выводит его на экран консоли. Вызовите из метода Main асинхронный метод CalculateFactorialAsync, не блокируя работу



CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

метода Main. Для проверки, в методе Main выводите в цикле на экран консоли какие-то символы, пока асинхронный метод выполняется.

Задание 3
Перепишите предыдущий пример, используя асинхронный метод Main. При вызове метода CalculateFactorialAsync дождитесь завершения асинхронного метода. Для этого используйте оператор await.

Задание 4
Создайте приложение по шаблону Console Application. Создайте метод с названием ParseAsync, он должен возвращать Task<IList<string>> и принимать строковой параметр с названием inputData. В теле метода разбейте все содержимое строки inputData на отдельные слова по разделителям: пробел, запятая, точка. Полученные слова запишите в строковой массив. Создайте коллекцию List<string>, в которую запишите слова из массива без повторений. В методе Main считайте текст из файла или введите несколько десятков слов через клавиатуру (на ваше усмотрение). Запишите это в строковую переменную. Вызовите метод ParseAsync, куда передайте строковую переменную. Возвращаемое значение метода в виде задачи запишите в переменную. Пока выполняется метод ParseAsync, сделайте следующее:

- Выведите на экран консоли строку «Введите свое имя».
- Примите ввод данных пользователя с клавиатуры в строковую переменную name.
- Создайте экземпляр класса FileStream. На его основе создайте экземпляр класса StreamWriter (НЕ ИСПОЛЬЗОВАТЬ АСИНХРОННЫЕ МЕТОДЫ ДЛЯ ЗАПИСИ В ФАЙЛ!).

На этом моменте вызовите ожидание завершения полученной задачи от вызова метода ParseAsync, используя оператор await! Результат оператора await запишите в переменную с названием parseResult. Запишите в файл первую строку: «{name} нашел {parseResult.Count} уникальных слов. Перечисление слов: ». После, запишите через запятую (исключая последнее слово, там необходимо поставить точку) все найденные слова методом ParseAsync.

## Рекомендуемые ресурсы

MSDN: Task-based Asynchronous Pattern
https://docs.microsoft.com/ru-ru/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap

MSDN: Task Parallel Library
https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl

MSDN: Task
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=netframework-4.7.2

MSDN: Task<TResult>
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task-1?view=netframework-4.7.2

MSDN: TaskStatus
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskstatus?view=netframework-4.7.2

MSDN: TaskCreationOptions
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcreationoptions?view=netframework-4.7.2

MSDN: Continuations
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task.continuewith?view=netframework-4.7.2

MSDN: TaskContinuationOptions
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcontinuationoptions?view=netframework-4.7.2



CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

MSDN: TaskFactory
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskfactory?view=netframework-4.7.2

MSDN: TaskFactory<TResult>
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskfactory-1?view=netframework-4.7.2

MSDN: ValueTask
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.valuetask?view=netcore-2.2

MSDN: ValueTask<TResult>
https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.valuetask-1?view=netcore-2.2



CyberBionic systematics

CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com