

C# Асинхронное программирование

Ключевые слова `async` `await`. Техническая реализация

C# Асинхронное программирование

Автор курса



Гнатюк Владислав



MCID:16354168

Асинхронное программирование

После урока обязательно



Повторите этот урок в видео формате на [ITVDN.com](http://itvdn.com)



Проверьте как Вы усвоили данный материал на [TestProvider.com](http://testprovider.com)

Ключевые слова `async` `await`
Техническая реализация

C# Асинхронное программирование

План урока

- 1) Ключевые слова `async` `await`
- 2) Асинхронные методы
- 3) Типы возвращаемых значений асинхронных методов
- 4) Ожидаемые методы
- 5) Внутренняя реализация `async` `await`
- 6) Типы, поддерживающие работу ключевых слов «под капотом»
- 7) Объект ожидания завершения асинхронной задачи
- 8) Асинхронный метод `Main`

C# Асинхронное программирование

Ключевые слова `async` `await`

- 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.

Работу ключевых слов `async` и `await` обеспечивает компилятор, поэтому без его поддержки будет потеряна вся «магия» ключевых слов.

C# Асинхронное программирование

Асинхронные методы

Асинхронные методы – методы, которые используют ключевые слова `async/await` и имеют специальный тип возвращаемого значения. В имени метода имеют суффикс `Async` или `TaskAsync` для быстрой узнаваемости.

```
public async void Method1Async()
{
    await Task.Run(() => Console.WriteLine(nameof(Method1Async)));
}

public async Task Method2Async()
{
    await Task.Run(() => Console.WriteLine(nameof(Method2Async)));
}

public async Task<string> Method3Async()
{
    return await Task<string>.Run(() => nameof(Method3Async));
}
```

```
public async ValueTask Method4Async()
{
    await Task.Run(() => Console.WriteLine(nameof(Method4Async)));
}

public async ValueTask<string> Method5Async()
{
    return await Task<string>.Run(() => nameof(Method5Async));
}
```

Наличие ключевого слова `async` не означает, что метод будет выполняться во вторичном/фоновом потоке.

C# Асинхронное программирование

Типы возвращаемых значений асинхронных методов

Асинхронные методы могут иметь следующие типы возвращаемых значений :

- Тип **void** – используется только для обработчиков событий.
- Тип **Task** – для асинхронной операции, которая не возвращает значение.
- Тип **Task<TResult>** - для асинхронной операции, которая возвращает значение.
- Тип **ValueTask** – для асинхронной операции, которая не возвращает значения.
- Тип **ValueTask<TResult>** - для асинхронной операции, которая возвращает значение.

C# Асинхронное программирование

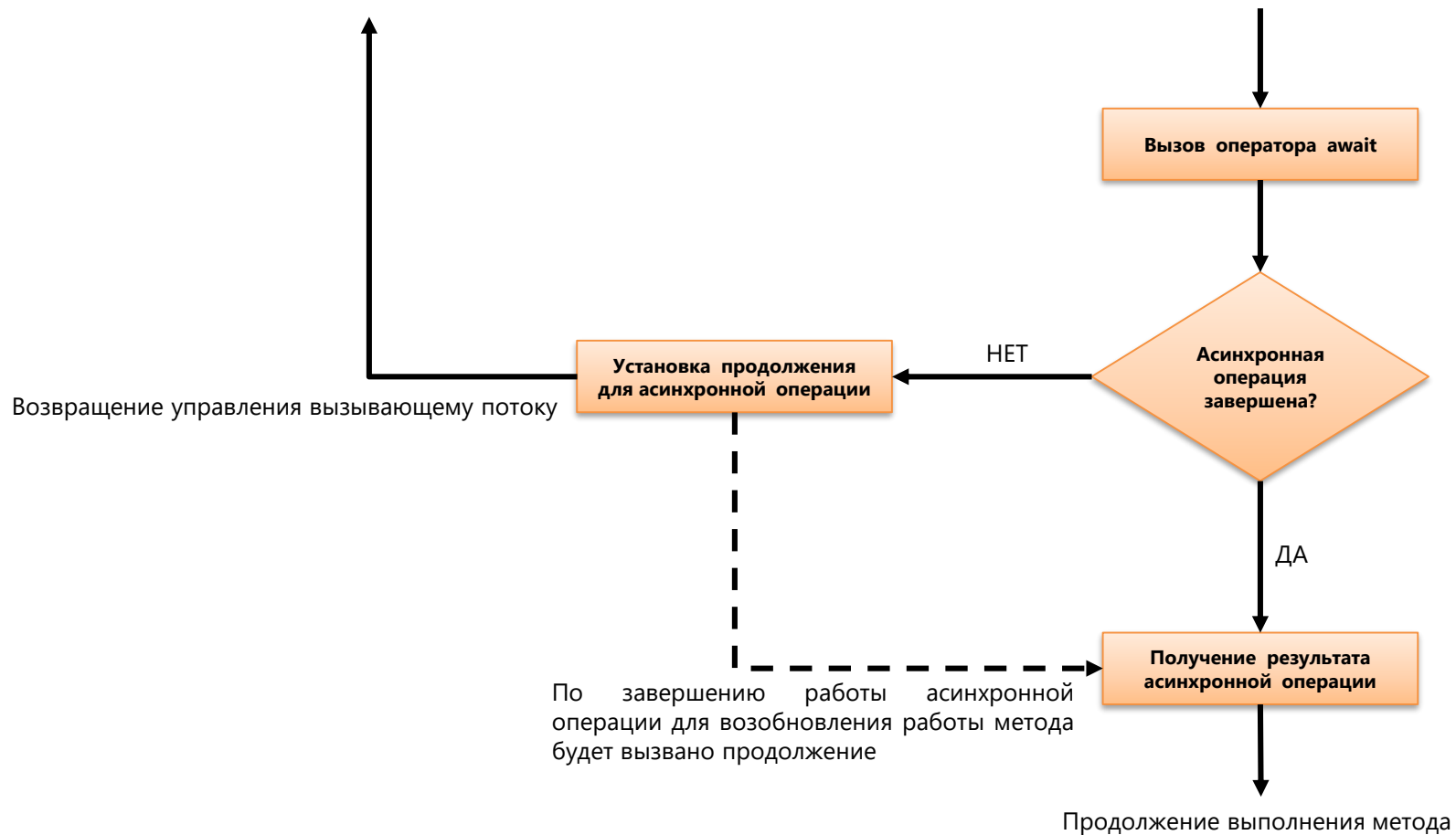
Применение оператора await

Для применения оператора `await` к типу, данный тип должен обладать следующим функционалом:

- Иметь доступный метод `GetAwaiter`, возвращаемый объект которого должен иметь:
 - ❑ Реализацию интерфейсов `ICriticalNotifyCompletion` и `INotifyCompletion`.
 - ❑ Свойство `bool IsCompleted` { get; }.
 - ❑ Метод `GetResult`. Тип возвращаемого значения метода должен зависеть от того, должна ли вернуть асинхронная операция результат. Если да, то тип должен совпадать с типом результата операции. Если нет - тип возвращаемого значения должен быть `void`.

C# Асинхронное программирование

Как работает оператор await



C# Асинхронное программирование

Ожидаемые (awaitable) методы

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**.

Другие способы ожидания:

- Свойство `Result`.
- Методы ожидания (`Wait()`, `WaitAll()`, `WaitAny()`).
- Метод `GetResult()`, вызванный на экземпляре структуры `TaskAwaiter`, которая получена через вызов метода `GetAwaiter()`.

```
await example.Method2Async();
string result1 = await example.Method3Async();

example.Method2Async().Wait();
string result2 = example.Method3Async().Result;

example.Method2Async().GetAwaiter().GetResult();
string result3 = example.Method3Async().GetAwaiter().GetResult();
```

*Не рекомендуется прибегать к вызову `GetAwaiter()`. Пользуйтесь ключевым словом **await**.
Тут `TaskAwaiter` и его члены предназначены для внутреннего использования компилятором.*

C# Асинхронное программирование

Ожидаемые (awaitable) методы

Методы сами вам подскажут, что они ожидаемые – с помощью подсветки синтаксиса. Данным методам соответствует зеленая линия подчеркивания, надпись (awaitable) и предупреждение от компилятора CS4014. **IntelliSense** также подскажет способ взаимодействия с ожидаемыми методами.

```
async.Method2Async();
```



(awaitable) Task AsyncClass.Method2Async()

Usage:

await Method2Async();

Because this call is not awaited, execution of the current method continues before the call is completed. Consider applying the 'await' operator to the result of the call.

[Show potential fixes](#) (Alt+Enter or Ctrl+.)

```
async.Method3Async();
```



(awaitable) Task<string> AsyncClass.Method3Async()

Usage:

string x = await Method3Async();

Because this call is not awaited, execution of the current method continues before the call is completed. Consider applying the 'await' operator to the result of the call.

[Show potential fixes](#) (Alt+Enter or Ctrl+.)

C# Асинхронное программирование

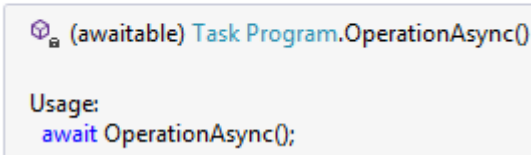
Ожидаемые (awaitable) методы

Если ваш метод будет без модификатора `async`, но с возвращаемым значением `Task/ValueTask` или их универсальными вариантами, то его вызов в обычных методах будет без предупреждения, а только с подсказкой `awaitable` при наведении.

Но вызов такого метода в асинхронном методе с модификатором `async` уже будет с предупреждением, что метод можно подождать.

```
private static Task OperationAsync()
{
    return Task.Run(() => Console.WriteLine("Operation"));
}
```

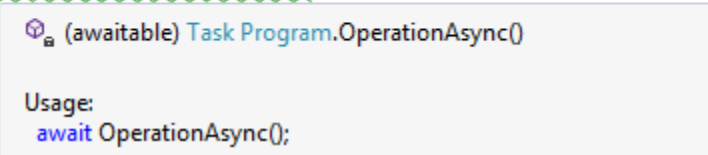
```
private static void Method()
{
    OperationAsync();
}
```



Tooltip for `OperationAsync()` in a synchronous method:

- Icon: (awaitable) Task Program.OperationAsync()
- Usage: `await OperationAsync();`

```
private static async void MethodAsync()
{
    OperationAsync();
}
```



Tooltip for `OperationAsync()` in an asynchronous method:

- Icon: (awaitable) Task Program.OperationAsync()
- Usage: `await OperationAsync();`

C# Асинхронное программирование

Ожидаемые (awaitable) методы

Если ваш метод будет с модификатором `async`, то его вызов как из обычных, так и из асинхронных методов будет происходить с предупреждением .

```
private static async Task OperationAsync()  
{  
    await Task.Run(() => Console.WriteLine("OperationAsync"));  
}
```

```
private static void Method()  
{
```

```
    OperationAsync();
```



(awaitable) Task Program.OperationAsync()

Usage:
`await OperationAsync();`

```
private static async void MethodAsync()  
{
```

```
    OperationAsync();
```



(awaitable) Task Program.OperationAsync()

Usage:
`await OperationAsync();`

C# Асинхронное программирование

Конечный автомат

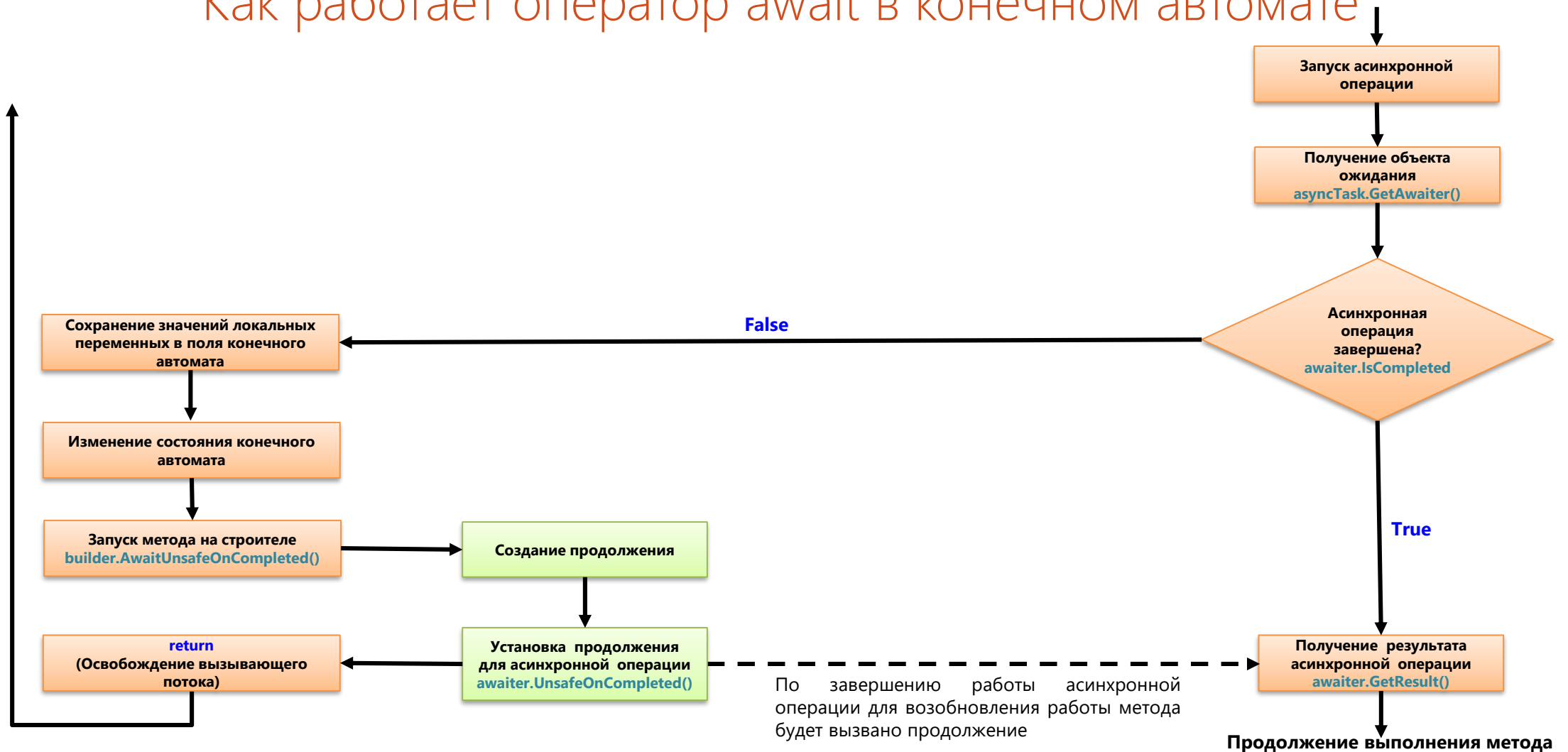
Конечный автомат (Finite-state machine) – это модель вычислений, которая позволяет объекту изменить свое поведение в зависимости от своего внутреннего состояния. Поведение объекта изменяется настолько, что создается впечатление, что изменился класс объекта.

В один момент времени может быть активно только одно состояние. По завершению выполнения действия, конечный автомат меняет свое внутреннее состояние.

Подробное рассмотрение конечных автоматов: <https://itvdn.com/ru/patterns/statemachine>

C# Асинхронное программирование

Как работает оператор await в конечном автомате



C# Асинхронное программирование

Внутренняя реализация `async await`

Работу ключевых слов `async` `await` обслуживает конечный автомат. Компилятор, с помощью интерфейса `IAsyncStateMachine` и специальных строителей, создает конечный автомат, который обслуживает асинхронный метод «под капотом».

Сам асинхронный метод превращается в метод-заглушку, который будет использовать созданный конечный автомат.

C# Асинхронное программирование

Трансформация асинхронного метода

Наш асинхронный метод превращается компилятором в метод-заглушку. Тело асинхронного метода перемещается в метод MoveNext() конечного автомата, с некоторыми дополнениями и оптимизациями.

Метод-заглушка создает конечный автомат, который обслуживает работу асинхронного метода. В нем происходит инициализация открытых полей структуры конечного автомата.

Здесь же и происходит первый запуск конечного автомата, с помощью вызова метода Start.

```
public async Task OperationAsync() // Асинхронный метод
{
    Console.WriteLine("Before await's");
    await Task.Run(() => Console.WriteLine("Task #1"));
    Console.WriteLine("After first await");
    await Task.Run(() => Console.WriteLine("Task #2"));
    Console.WriteLine("After second await");
}
```

```
[AsyncStateMachine(typeof(OperationAsyncStateMachine))]
public Task OperationAsync() // Метод-заглушка
{
    OperationAsyncStateMachine stateMachine;
    stateMachine.builder = AsyncTaskMethodBuilder.Create();
    stateMachine.state = -1;
    stateMachine.builder.Start(ref stateMachine);
    return stateMachine.builder.Task;
}
```

C# Асинхронное программирование

Задача-марионетка

Задача-марионетка – это обыкновенная задача, жизненным циклом которой управляем мы с вами. Результат выполнения задачи может быть указан позже, не в момент создания задачи-марионетки.

Результат задачи-марионетки указываем мы. Мы можем как отдать ей результат (означает успешное выполнение), так и пробросить исключение (означает провальное выполнение).



C# Асинхронное программирование

Строители асинхронных методов

Для построения задачи, которую можно сделать завершенной позже и для представления выполнения асинхронного метода используются специальные **строители асинхронных методов**. В зависимости от типа возвращаемого значения асинхронного метода используется свой строитель.

Виды строителей:

- `AsyncTaskMethodBuilder` – строитель для асинхронных методов с типом `Task`.
- `AsyncTaskMethodBuilder<TResult>` - строитель для асинхронных методов с типом `Task<TResult>`.
- `AsyncVoidMethodBuilder` – строитель для асинхронных методов с типом `void`.

C# Асинхронное программирование

Строители асинхронных методов

С появлением значимых задач (**ValueTask/ValueTask<TResult>**) в языке C# для их поддержки в виде возвращаемых значений асинхронных методов были введены новые строители асинхронных методов.

Новые строители:

AsyncValueTaskMethodBuilder – строитель для асинхронных методов с типом ValueTask.

AsyncValueTaskMethodBuilder<TResult> – строитель для асинхронных методов с типом ValueTask<TResult>.

EE

C# Асинхронное программирование

Строители асинхронных методов

Каждый строитель для повышения производительности представляет из себя структуру. Они оптимизированы под работу с async-методами.

У строителей есть весь необходимый функционал для создания задачи-марионетки. Имеются в виду методы, с помощью которых можно указать результат (**SetResult**), исключение (**SetException**) или вызвать ожидание с регистрацией продолжения для асинхронной задачи (**AwaitOnCompleted/AwaitUnsafeOnCompleted**).

Строители асинхронных методов лучше не использовать напрямую. Они созданы для использования компилятором. Для создания задач-марионеток пользователями существует открытый API в виде класса [TaskCompletionSource](#).

C# Асинхронное программирование

Конечный автомат `async await`

Конечный автомат для `async await` – 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, будет сохранено состояние и локальные переменные асинхронного метода. Это позволяет полноценно сохранить состояние асинхронного метода в любой точке, для дальнейшего возобновления его работы позже.

Конечный автомат для повышения производительности описывается структурой, ведь при синхронном завершении асинхронного метода не придется выделять память для объекта кучи.

C# Асинхронное программирование

Конечный автомат `async await`

Конечный автомат может содержать несколько разновидностей полей:

- Состояние конечного автомата;
- Строитель асинхронного метода;
- Объекты ожидания;
- Параметры асинхронного метода;
- Локальные переменные асинхронного метода;
- Временные переменные стека;
- Внешний тип.

Состояние представлено целым числом и имеет следующие варианты:

- **Значение «-1»** – начальное состояние или состояние выполнения.
- **Значение «-2»** – конечное состояние. Указывает, что работа завершена (успешно или с ошибкой).
- **Любое другое значение** – означает приостановку через оператор `await`.

C# Асинхронное программирование

Пример конечного автомата

```
[CompilerGenerated]
[StructLayout(LayoutKind.Auto)]
private struct OperationAsyncStateMachine : IAsyncStateMachine
{
    public int state;    // Поле для сохранения состояния конечного автомата.
    public AsyncTaskMethodBuilder builder;    // Поле строителя асинхронных методов.
    private TaskAwaiter awaiter;    // Поле для объекта ожидания завершения асинхронной задачи.
    // Здесь еще могут быть открытые поля для входящих параметров в асинхронный метод.
    // Здесь еще могут быть закрытые поля для сохранения локальных переменных асинхронного метода.
    // Здесь еще могут быть закрытые поля для хранения временных значений из стека.
    // Здесь еще может быть открытое поле для хранения внешнего типа.
    void IAsyncStateMachine.MoveNext()
    {
        // Здесь находится тело асинхронного метода с дополнениями и оптимизацией от компилятора.
    }

    [DebuggerHidden]
    void IAsyncStateMachine.SetStateMachine(IAsyncStateMachine stateMachine)
    {
        this.builder.SetStateMachine(stateMachine);
    }
}
```

C# Асинхронное программирование

IAsyncStateMachine

Интерфейс **IAsyncStateMachine** используется для создания конечного автомата, который обеспечивает работу асинхронного метода. Конечный автомат создается для получения объекта, способного представить состояние асинхронного метода, которое можно сохранить при достижении оператора `await` и для его дальнейшего восстановления. Таким образом происходит запоминание того, в каком месте находится приложение.

Методы:

void `MoveNext()` – выполняет тело асинхронного метода, перемещает конечный автомат в следующее состояние.

void `SetStateMachine(IAsyncStateMachine stateMachine)` – упаковывает конечный автомат из стека на кучу.

C# Асинхронное программирование

Метод MoveNext()

В основе асинхронного конечного автомата лежит метод MoveNext(). Он начинает свою работу через вызов метода Start на одном из строителей асинхронных методов. Каждый последующий запуск происходит в виде продолжения, когда ему необходимо возобновить выполнение после приостановки, вызванной оператором await.

В отличие от метода SetStateMachine, у метода MoveNext достаточно много обязанностей:

- Выполнение кода из правильного места.
- Сохранение значений локальных переменных и расположения выполнения кода в виде состояния (при возврате управления из-за инициации ожидания выполнения асинхронной задачи).
- Планирование продолжения, если было инициировано ожидание.
- Получение результатов асинхронных задач от объектов ожидания завершения асинхронных задач.
- Передача исключения.
- Передача результата или завершение выполнения асинхронного метода.

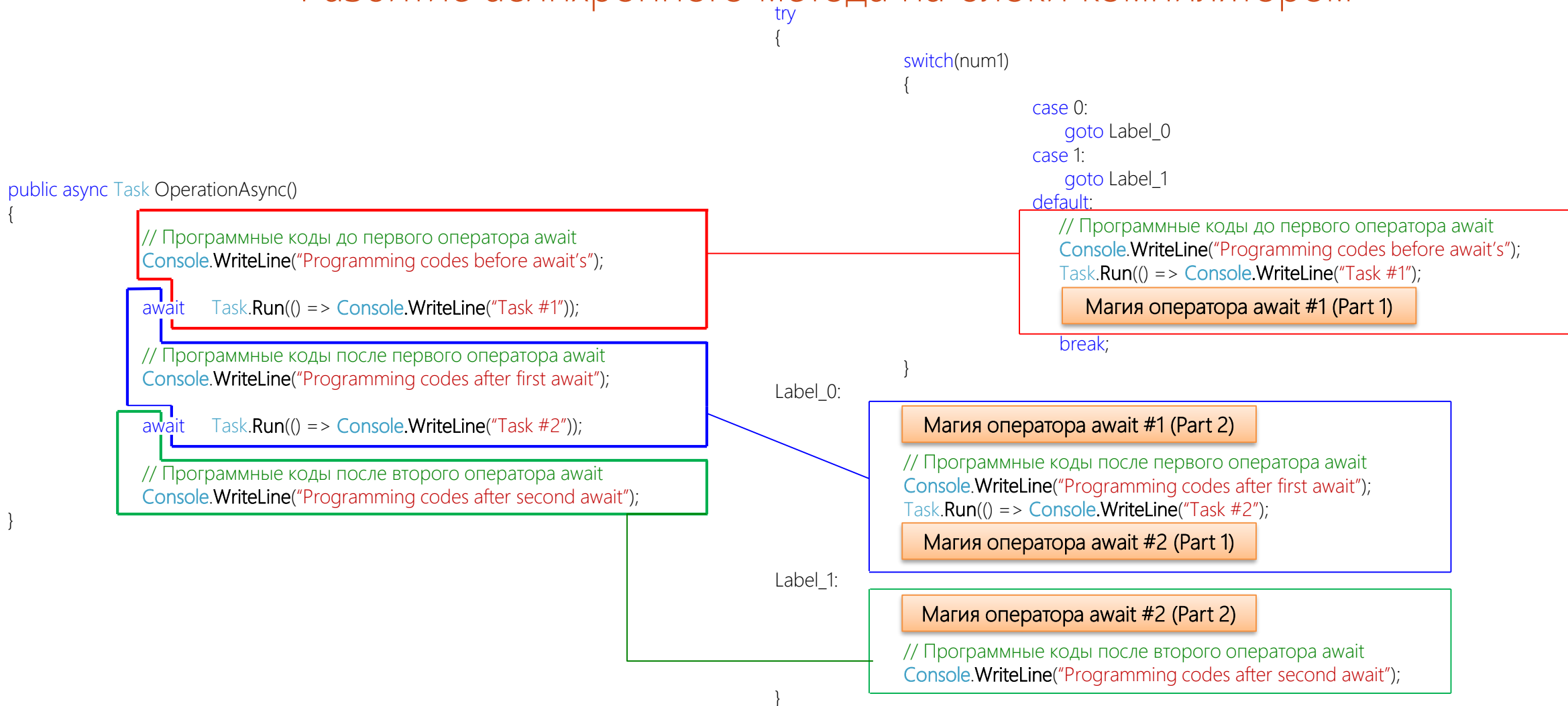
C# Асинхронное программирование

Структура метода MoveNext()

```
void IAsyncStateMachine.MoveNext()
{
    int num1 = this.state;
    try
    {
        switch(num1)
        {
            case 0: goto Label_0;
            case 1: goto Label_1;
            .... // Количество case и переходов зависит от количества вызовов операторов await.
            default:
                // Здесь находится часть кода асинхронного метода до первого оператора await.
                break;
        }
        Label_0:
            // Здесь находится часть кода, которая будет выполнена в виде продолжения после возобновления.
        Label_1:
            // Здесь находится часть кода, которая будет выполнена в виде продолжения после возобновления.
    }
    catch (Exception ex)
    {
        this.state = -2;
        this.builder.SetException(ex);
        return;
    }
    this.state = -2;
    this.builder.SetResult();
}
```

C# Асинхронное программирование

Разбитие асинхронного метода на блоки компилятором



C# Асинхронное программирование

Объект ожидания TaskAwaiter/TaskAwaiter<TResult>

TaskAwaiter/TaskAwaiter<TResult> - объект ожидания завершения асинхронной задачи. Объект ожидания поддерживает полноценный функционал для работы оператора **await**.

```
public struct TaskAwaiter : ICriticalNotifyCompletion, INotifyCompletion
{
    public bool IsCompleted { get; }

    public void GetResult();
    public void OnCompleted(Action continuation);
    public void UnsafeOnCompleted(Action continuation);
}
public struct TaskAwaiter<TResult> : ICriticalNotifyCompletion, INotifyCompletion
{
    public bool IsCompleted { get; }

    public TResult GetResult();
    public void OnCompleted(Action continuation);
    public void UnsafeOnCompleted(Action continuation);
}
```

C# Асинхронное программирование

Разбитие асинхронного метода на блоки компилятором

```
try  
{
```

```
TaskAwaiter awaiter1;  
TaskAwaiter awaiter2;  
switch(num1)  
{  
case 0:  
    goto Label_0  
case 1:  
    goto Label_1  
default:  
    // Программные коды до первого оператора await  
    Console.WriteLine("Programming codes before await's");  
    awaiter1 = Task.Run(() => Console.WriteLine("Task #1").GetAwaiter());  
    break;
```

Магия оператора await #1 (Part 1)

Label_0:

```
awaiter1.GetResult();
```

```
// Программные коды после первого оператора await  
Console.WriteLine("Programming codes after first await");  
awaiter2 = Task.Run(() => Console.WriteLine("Task #2").GetAwaiter());
```

Магия оператора await #2 (Part 1)

Label_1:

```
awaiter2.GetResult();
```

```
// Программные коды после второго оператора await  
Console.WriteLine("Programming codes after second await");
```

```
}
```

```
awaiter1 = this.awaiter;  
this.awaiter = new TaskAwaiter();  
this.state = -1;
```

```
awaiter2 = this.awaiter;  
this.awaiter = new TaskAwaiter();  
this.state = -1;
```

```
if(awaiter1.IsCompleted == false)  
{  
    this.state = 0;  
    this.awaiter = awaiter1;  
    this.builder.AwaitUnsafeOnCompleted(ref awaiter1, ref this);  
    return;  
}
```

```
if(awaiter2.IsCompleted == false)  
{  
    this.state = 1;  
    this.awaiter = awaiter2;  
    this.builder.AwaitUnsafeOnCompleted(ref awaiter1, ref this);  
    return;  
}
```

C# Асинхронное программирование

Метод AwaitUnsafeOnCompleted

Занимается планированием конечного автомата, чтобы перейти к следующему действию по завершению заданного объекта типа awaiter.

```
public void AwaitUnsafeOnCompleted<TAwaiter, TStateMachine>(ref TAwaiter awaiter, ref TStateMachine stateMachine)
    where TAwaiter : ICriticalNotifyCompletion
    where TStateMachine : IAsyncStateMachine
{
    try
    {
        AsyncMethodBuilderCore.MoveNextRunner runnerToInitialize = null;
        // Создание делегата продолжения. Повторный вызов метода MoveNext.
        Action completionAction = this.m_coreState.GetCompletionAction(AsyncCausalityTracer.LogginOn ? Task : null, ref runnerToInitialize);
        // Если машина состояний еще не была упакована, то в теле блока if произойдет упаковка. Вызов метода SetStateMachine.
        if(this.m_coreState.m_stateMachine = null)
        {
            Task<TResult> task = this.Task;
            this.m_coreState.PostBoxInitialization((IAsyncStateMachine)stateMachine, runnerToInitialize, task);
        }
        // Установка продолжения, которое должно быть выполнено по завершению работы асинхронной задачи.
        awaiter.UnsafeOnCompleted(completionAction);
    }
    catch (Exception ex)
    {
        // Регистрация и проброс исключения.
        AsyncTaskMethodBuilderCore.ThrowAsync(ex, (SynchronizationContext)null);
    }
}
```


C# Асинхронное программирование

Асинхронный метод Main

С приходом версии C# 7.1 у нас появилась возможность перегрузить метод Main дополнительными возвращаемыми типами: Task, Task<int> и модификатором async. При этом метод Main остался точкой входа в программу.

Это было сделано, чтобы упростить возможность ожидания результата работы асинхронного метода.

```
private static void Main()
{
    MethodAsync().GetAwaiter().GetResult();
}
```

```
private static async Task Main()
{
    await MethodAsync();
}
```

```
private static int Main()
{
    return DoWorkAsync().GetAwaiter().GetResult();
}
```

```
private static async Task<int> Main()
{
    return await DoWorkAsync();
}
```

C# Асинхронное программирование

Ключевые слова `async` `await`

<code>async</code>	<code>await</code>
Применяется как модификатор метода	Оператор <code>await</code> применяется к экземплярам типов, которые имеют метод <code>GetAwaiter()</code>
Разрешает методу в своем теле использовать оператор <code>await</code>	Освобождает вызывающий поток
Указывает компилятору, что нужно создать асинхронный конечный автомат	Указывает компилятору, где в конечном автомате нужно создать продолжение и сгенерировать возврат управления
Бесполезно без использования ключевого слова <code>await</code> (Может создать конечный автомат без надобности)	Бесполезно без использования ключевого слова <code>async</code> (Не работает)
Сами по себе ключевые слова не несут никакой силы. Для их работоспособности нужны либо задачи (<code>Task</code>), либо специальные методы, помеченные как (<code>awaitable</code>).	

C# Асинхронное программирование

Результат асинхронной операции

Асинхронные операции способны возвращать результат своей работы.

При работе с задачами есть несколько способов для извлечения результата из асинхронной задачи:

- Свойство `Result`, вызванное на экземпляре класса `Task<TResult>`.
- Метод `GetResult()`, вызванный на экземпляре структуры `TaskAwaiter<TResult>`.
- Оператор `await`.

```
private static async Task Main()
{
    int a = OperationAsync().Result;
    int b = OperationAsync().GetAwaiter().GetResult();
    int c = await OperationAsync();
}
```

Сейчас, для получения результата асинхронной задачи, необходимо по возможности использовать только оператор **`await`**. Он делает это максимально эффективно и безопасно.

C# Асинхронное программирование

Асинхронные методы с возвращаемыми значениями TResult

Если ваш асинхронный метод выглядит следующим образом:

```
private async Task<TResult> MethodAsync()  
{  
  
}
```

То оператор `return`, который будет содержаться в теле этого метода, должен возвращать значение типа `TResult`.

Примеры:

```
private async Task<int> Method1Async()  
{  
    ...  
    ...  
    return 1;  
}
```

```
private async Task<string> Method2Async()  
{  
    ...  
    ...  
    return "string result";  
}
```

```
private async Task<MyClass> Method3Async()  
{  
    ...  
    ...  
    return new MyClass();  
}
```

Если метод помечен модификатором `async`, то значение `TResult` будет записано в задачу-марионетку автоматически. Поэтому, вам необходимо возвращать тип `TResult` вместо `Task<TResult>`.

C# Асинхронное программирование

Ключевые слова `async` `await`

Ключевое слово `async` – является модификатором для методов. Указывает, что метод асинхронный.

Модификатор `async`:

- указывает компилятору, что необходимо создать конечный автомат для обеспечения работы асинхронного метода. Основная задача конечного автомата - приостановка и затем асинхронное возобновление работы в точках ожидания.
- позволяет использовать в теле асинхронного метода ключевое слово `await`.
- позволяет записать возвращаемое значение (если метод возвращает `Task<TResult>`) или необработанное исключение в результирующую задачу.

C# Асинхронное программирование

Ключевые слова `async` `await`

Ключевое слово `await` – является унарным оператором, операнд которого располагается справа от самого оператора. Применение оператора `await` инициирует запрос на получение «объекта ожидания». С помощью объекта ожидания происходит проверка, завершена ли уже асинхронная операция.

- Если асинхронная операция завершена:
 - Дальнейшее выполнение метода продолжается синхронно в том же вызывающем потоке.
- Если асинхронная операция не завершена:
 - Компилятор с помощью специальных типов инициирует ожидание завершения асинхронной операции.
 - Иницируется захват контекста выполнения.
 - Происходит «регистрация кода», который находился после оператора `await` в виде продолжения (Continuation), которое выполнится по завершению асинхронной операции.
 - Если есть возможность, продолжение будет отправлено на выполнение в вызывающий поток.
 - Выполняется освобождение вызывающего потока.
 - Класс-делегат, являющийся продолжением, повторно запускает асинхронный метод. Данный метод продолжит выполняться с точки, на которой оператор `await` инициировал ожидание.

Смотрите наши уроки в видео формате

ITVDN.com



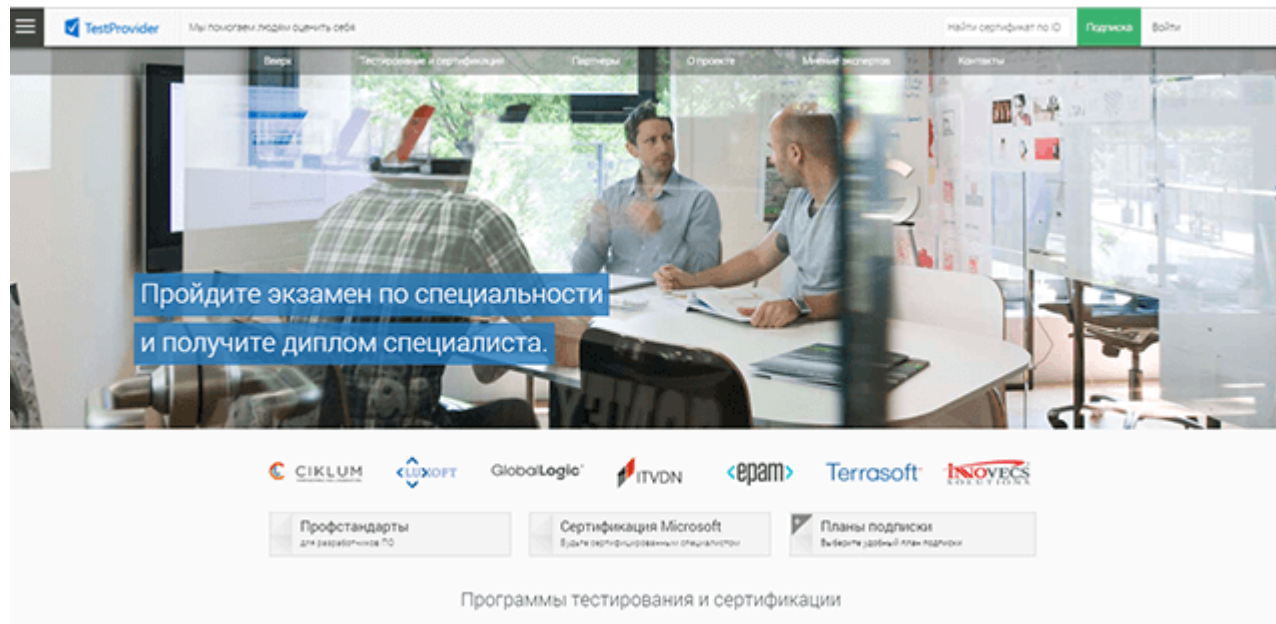
Посмотрите этот урок в видео формате на образовательном портале ITVDN.com для закрепления пройденного материала.

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



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](https://testprovider.com)

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.



Информационный видеосервис для разработчиков программного обеспечения

