Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

Asynchronous Programming

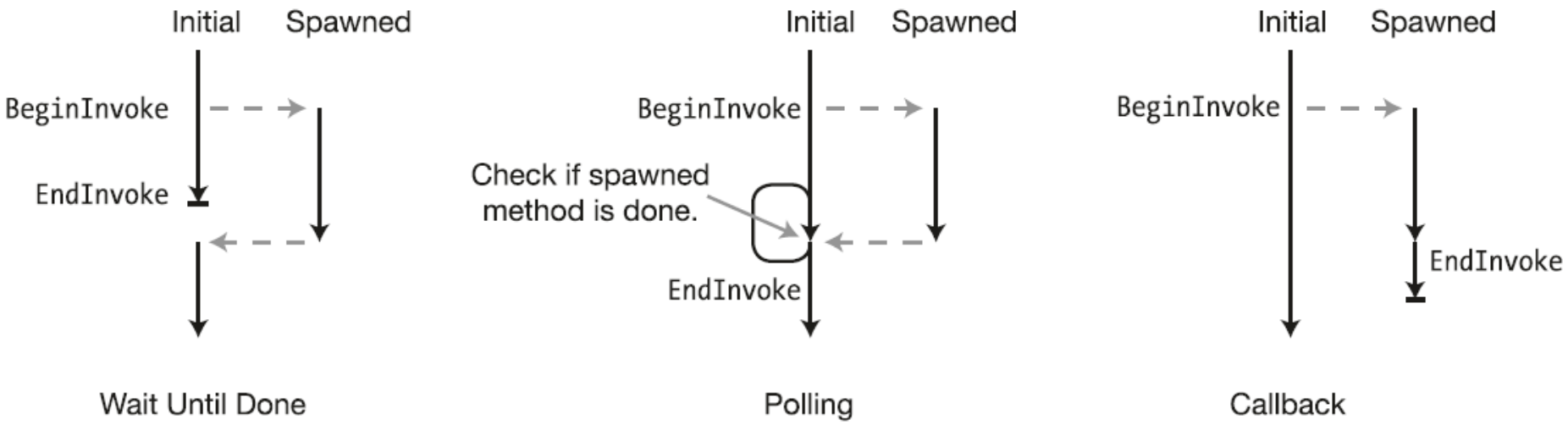
Использованы материалы пособия Daniel Solis, Illustrated C#

Параллельное программирование

Часть 2. Задачи

Паттерны асинхронного программирования

- **Ожидание выполнения (wait until done);**
- **Опрос (polling);**
- **Обратный вызов (callback).**



Методы BeginInvoke и EndInvoke

public IAsyncResult BeginInvoke(Delegate method)

public object EndInvoke(IAsyncResult asyncResult)
+ параметры с **out** и **ref**

public-свойства интерфейса **IAsyncResult**:

```
object AsyncState { get; }
WaitHandle AsyncWaitHandle { get; }
bool CompletedSynchronously { get; }
bool IsCompleted { get; }
```


DEPRECATED

https://docs.microsoft.com/ru-ru/dotnet/standard/asynchronous-programming-patterns/calling-synchronous-methods-asynchronously

Схема асинхронного обращения: BeginInvoke - начало

```
delegate long MyDel( int first, int second ); // определение делегат-типа
...
static long Sum(int x, int y){ ... } // метод, соответствующий делегату
...
MyDel del = new MyDel(Sum); // создание объекта-делегата

IAsyncResult iar = del.BeginInvoke( 3, 5, null, null );
```

↑ информация об асинхр. вызове

↑ асинхр. вызов

↑ парам.

↑ доп. параметры

Схема асинхронного обращения: EndInvoke - завершение

объект делегата

```
long result = del.EndInvoke( iar );
```

возврат значения из асинхр. метода

объект IAsyncResult

```
long result = del.EndInvoke(out someInt, iar);
```

возврат значения из асинхр. метода

out параметр

объект IAsyncResult

Паттерн Wait-Until-Done

```
delegate long MyDel(int first, int second); // делегат


static long Sum(int x, int y) { // объявление метода для асинхронного вызова
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}


public static void Main01Wait_Until_Done()
{
    MyDel del = new MyDel(Sum);
    Console.WriteLine("Before BeginInvoke");
    IAsyncResult iar = del.BeginInvoke(3, 5, null, null); // асинхронный вызов

    Console.WriteLine("After BeginInvoke");
    Console.WriteLine("Doing stuff");
    long result = del.EndInvoke(iar); // ожидание получения результата
    Console.WriteLine("After EndInvoke: {0}", result);
}
```

Результат:

Before BeginInvoke
After BeginInvoke
Doing stuff
Inside Sum
After EndInvoke: 8

DEPRECATED

Паттерн Wait-Until-Done через Task<T>

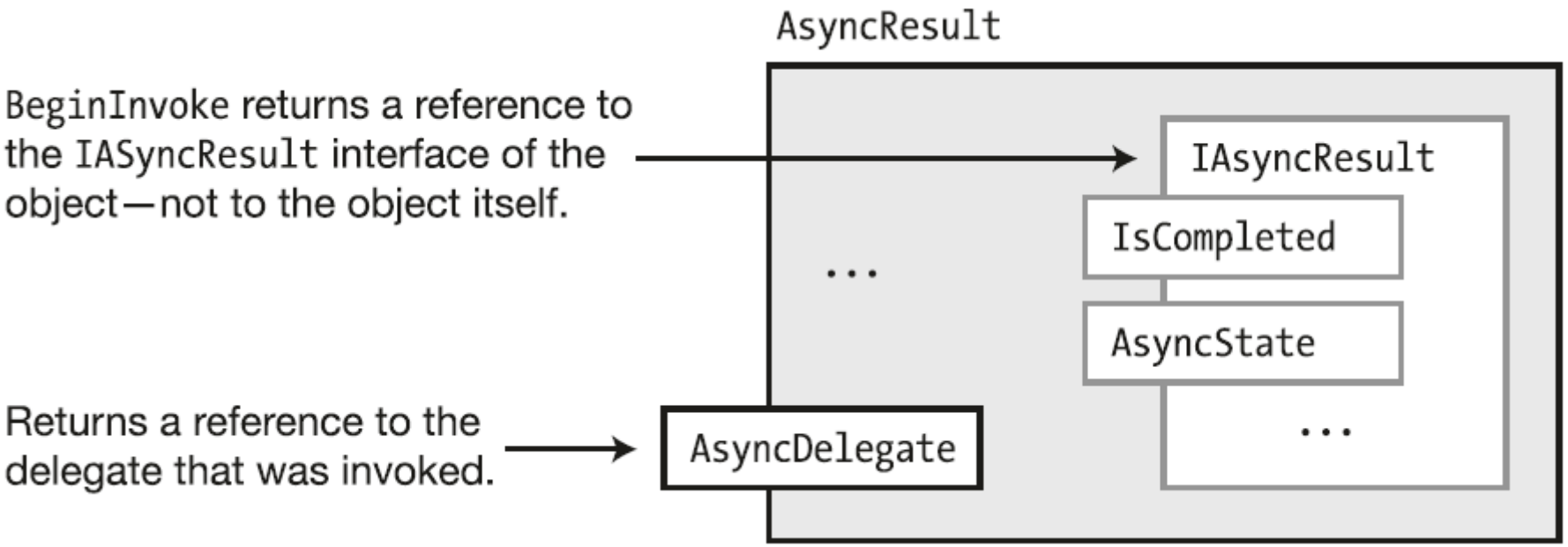
```
static long Sum(int x, int y) { // объявление метода для асинхронного вызова
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}


public static void Main01Wait_Until_Done()
{
    Task<long> task = new Task<long>(() => Sum(3, 5));
    Console.WriteLine("Before Start");
    task.Start();
    Console.WriteLine("After Start");
    Console.WriteLine("Doing stuff");
    long result = task.Result; // Wait for end and get result
    Console.WriteLine("After task.Result: {0}", result);
}
```

Результат:

Before Start
After Start
Doing stuff
		Inside Sum
After task.Result: 8

Класс AsyncResult

```
using System.Runtime.Remoting.Messaging;
```



DEPRECATED

Паттерн Polling

```
delegate long MyDel(int first, int second); // делегат-тип

static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}
public static void Main02Polling() {
    MyDel del = new MyDel(Sum); // ссылка на метод
    IAsyncResult iar = del.BeginInvoke(3, 5, null, null);
    Console.WriteLine("After BeginInvoke");
    while (!iar.IsCompleted) { // асинхронный вызов завершен?
        Console.WriteLine("Not Done");
        // продолжаем обработку, имитация работы:
        for (long i = 0; i < 10000000; i++) ; // пустой оператор
    }
    Console.WriteLine("Done");
    // вызов EndInvoke для получения результата
    long result = del.EndInvoke(iar);
    Console.WriteLine("Result: {0}", result);
}
```

DEPRECATED

Результаты выполнения программы:

After BeginInvoke	(основной поток)
Not Done	(основной поток)
Inside Sum	(другой поток)
Not Done	(основной поток)
Done	(основной поток)
Result: 8	(основной поток)

Паттерн Polling через Task<T>

```
static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}
public static void Main02Polling() {
    Task<long> task = new Task<long>(() => Sum(3, 5));
    task.Start();
    Console.WriteLine("After Start");
    // Check whether the async method is done.
    while (!task.IsCompleted)
    {
        Console.WriteLine("Not Done");
        // Continue processing, even though in this case it's just busywork.
        for (long i = 0; i < 10000000; i++) ; // Empty statement
    }
    Console.WriteLine("Done");
    // Call EndInvoke to get result and clean up.
    long result = task.Result;
    Console.WriteLine("Result: {0}", result);
}
```

Результаты выполнения программы:

```
After Start
Not Done
        Inside Sum
Not Done
Not Done
Not Done
Done
Result: 8
```

Паттерн Callback

Метод обратного вызова:

```
void CallWhenDone( IAsyncResult iar ) // Заголовок…
```

Подключение метода обратного вызова к BeginInvoke:

(1) создание делегата с методом обратного вызова

↓

```
IAsyncResult iar1 =
del.BeginInvoke(3, 5, new AsyncCallback(CallWhenDone), null);
```

(2) Используем имя метода обратного вызова.

↓

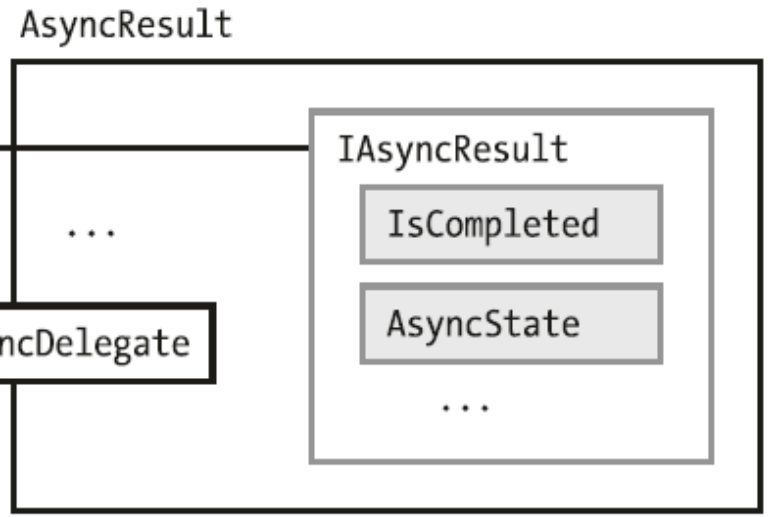
```
IAsyncResult iar2 = del.BeginInvoke(3, 5, CallWhenDone, null);
```

Вызов EndInvoke внутри метода обратного вызова

объект делегата ↓ перед. объект-дел. в кач-ве состояния ↓

```
IAsyncResult iar = del.BeginInvoke(3, 5, CallWhenDone, del);
```

```
void CallWhenDone( IAsyncResult iar )
{
   AsyncResult ar = (AsyncResult) iar;

   MyDel del = (MyDel) ar.AsyncDelegate;


   long Sum = del.EndInvoke( iar );
      ...
}
```



DEPRECATED

Паттерн Callback (исходный код)

```
delegate long MyDel(int first, int second); // делегат

static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}
static void CallWhenDone(IAsyncResult iar) {
    Console.WriteLine("\t\tInside CallWhenDone.");
    AsyncResult ar = (AsyncResult)iar;
    MyDel del = (MyDel)ar.AsyncDelegate;
    long result = del.EndInvoke(iar);
    Console.WriteLine("\t\tThe result is: {0}.", result);
}

public static void Main03Callback() {
    MyDel del = new MyDel(Sum);
    IAsyncResult iar1 = del.BeginInvoke(3, 5, new AsyncCallback(CallWhenDone), null);
    // ИЛИ МОЖНО del.BeginInvoke(3, 5, CallWhenDone, null);
    Console.ReadKey(true);  // обязательно!!!
}
```

Паттерн Callback через Task<T>

```
static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}

static void CallWhenDone(long result)
{
    Console.WriteLine("\t\tInside CallWhenDone.");
    Console.WriteLine("\t\tThe result is: {0}.", result);
}
```

TaskContinuationOptions.AttachedToParent

```
public static void Main03Callback()
{
    Task<long> task = new Task<long>(() => Sum(3, 5));
    task.ContinueWith(t => CallWhenDone(t.Result));
    task.Start();
    Console.WriteLine("After Start");
    Console.WriteLine("task.Result: {0}", task.Result);
    // Console.ReadKey(true);  // НЕобязательно
}
```

Таймеры

System.Timers.Timer **System.Windows.Forms.Timer**

System.Threading.Timer
Конструктор (один из пяти):

```
public Timer (System.Threading.TimerCallback
callback, object state, uint dueTime, uint period);
```

Прототип метода обратного вызова:

void TimerCallback(object state)

имя метода обр. вызова ↓

первый вызов через 2000 мс ↓

```
Timer myTimer=new Timer(MyCallback, someObj, 2000,1000);
```

↑ объект для передачи в метод обр. вызова

↑ вызов каждые 1000 мс

Асинхронный код через Task и Task<T>

Task (задача) — это конструкция, реализующая модель асинхронной обработки на основе обещаний (Promise).
Т.е. суть модели в том, что она "обещает", что задача будет выполнена позже, позволяя взаимодействовать с помощью "обещаний" с чистым API.

Два класса:
- **Task** представляет одну задачу, которая не возвращает значение.
- **Task<T>** представляет одну задачу, которая возвращает значение типа **T**.

Важно рассматривать задачи, как **абстракции асинхронных операций**, а не как абстракции поверх потоков. По умолчанию задачи выполняются в текущем потоке и при необходимости делегируют работу операционной системе. Для задач может также явно запрашиваться запуск в отдельном потоке через метод **Task.Run()**.

Поддержка **Task** и **Task<T>** начиная с C# 5.0 (2012) реализована через **async** + **await**.

https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/
https://docs.microsoft.com/ru-ru/dotnet/standard/async-in-depth

Пример с async + await (1)

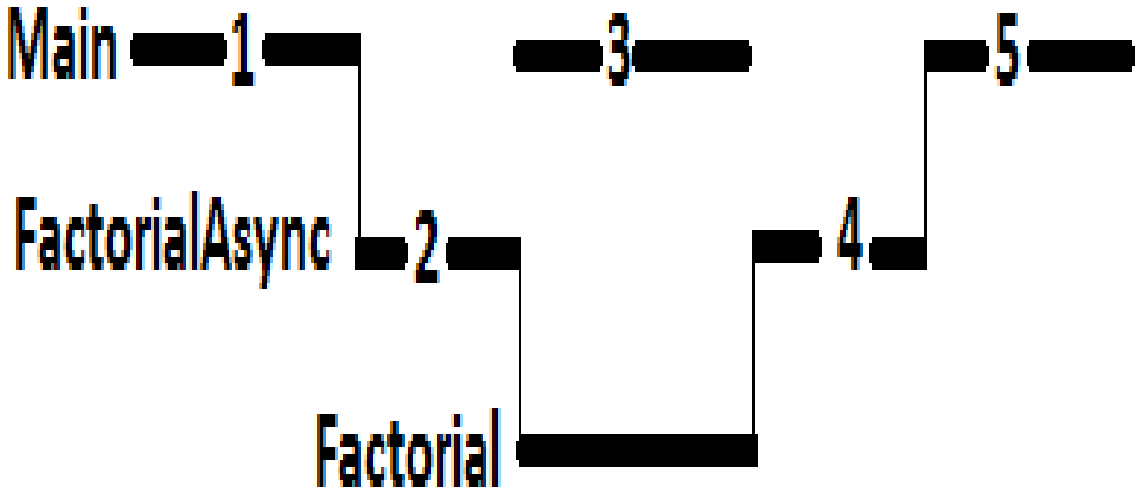
```
static long Factorial(int factor) {
    long res = 1;
    for (int k = 1; k <= factor; k++)
        res *= k;
    System.Threading.Thread.Sleep(1500);
    return res;
}
```

```
// запуск вычисления факториала в отдельном потоке
static async Task<long> FactorialAsync(int factor) {
    Console.WriteLine("2 - Запуск асинхронного потока!");
    var eee = await Task.Run(() => Factorial(factor));
    Console.WriteLine("4 - Завершен асинхронный поток!");
    return eee; // long
}
```

Пример с async + await (2)

```
public static void Main()
{
    Console.WriteLine("1 - Выполнение основного потока!");
    var result = FactorialAsync(5);
    Console.WriteLine("3 - Продолжение основного потока!");
    Console.WriteLine("5 - В основном потоке: 5! = " +
        result.Result.ToString());
}
```

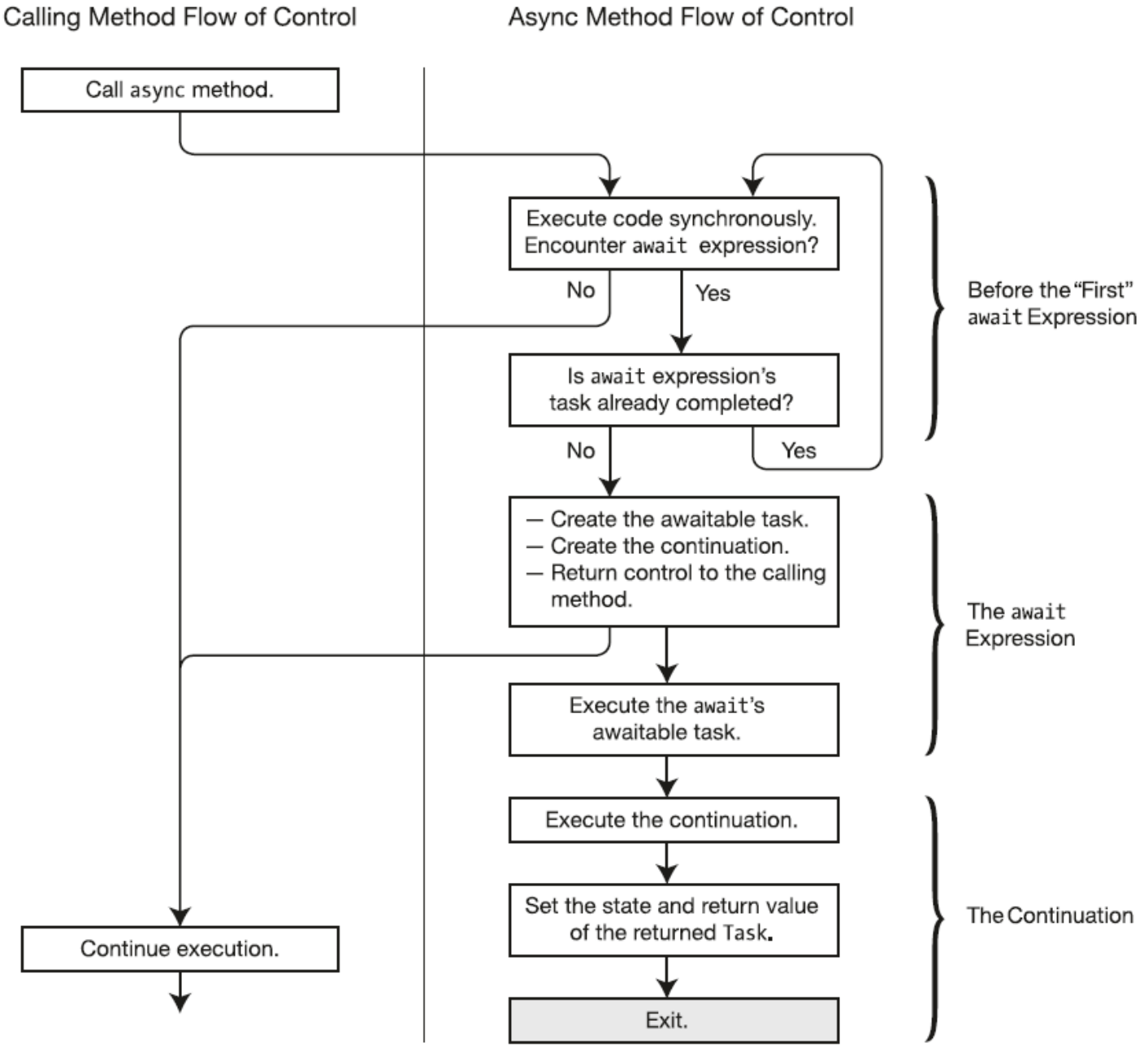
Схема двух потоков:



Результаты выполнения программы:

1 - Выполнение основного потока!
2 - Запуск асинхронного потока!
3 - Продолжение основного потока!
4 - Завершен асинхронный поток!
5 - В основном потоке: 5! = 120

Принцип работы async + await



Пример с async + await (3)

```
// Синхронный метод выполняет суммирование части массива:
static long SumAr(int[] ar, int beg, int end) {
    long res = 0;
    for (int k = beg; k < end; k++)
       res += ar[k];
    System.Threading.Thread.Sleep(1000);
    Console.Write("Beg = {0}  ", beg);
    Thread at = Thread.CurrentThread;
    Console.WriteLine("HashCode = " + at.GetHashCode());
    return res;
}

// Асинхронный метод
static async Task<long> SumAsync(int[] ar, int beg, int end) {
    var eee = await Task.Run(() => SumAr(ar, beg, end));
    return eee;     // long
}
```

async + await (4)

```
static void Main02() {
    int N = 8000;   // размер вектора
    int[] vec = new int[N];    // вектор
    for (int j = 0; j < vec.Length; j++)
        vec[j] = 1;

    long sum = 0;
    int P = 4;  // Число потоков
    object[] part = new object[P];  // Результаты потоков
    for (int j = 0; j < P; j++)
        part[j] = SumAsync(vec, j * N / P, (j + 1) * N / P);

    for (int j = 0; j < P; j++)
        sum += ((Task<long>)part[j]).Result;
    Console.WriteLine("Итоговая сумма = " + sum);
}
```

Результаты async + await

*** С задержкой:**

Beg = 4000 Beg = 2000 Beg = 0 HashCode = 3
Beg = 6000 HashCode = 5
HashCode = 4
HashCode = 6
Итоговая сумма = 8000

*****

*** Без задержки:**

Beg = 0 HashCode = 3
Beg = 6000 HashCode = 5
Beg = 4000 Beg = 2000 HashCode = 4
HashCode = 6
Итоговая сумма = 8000