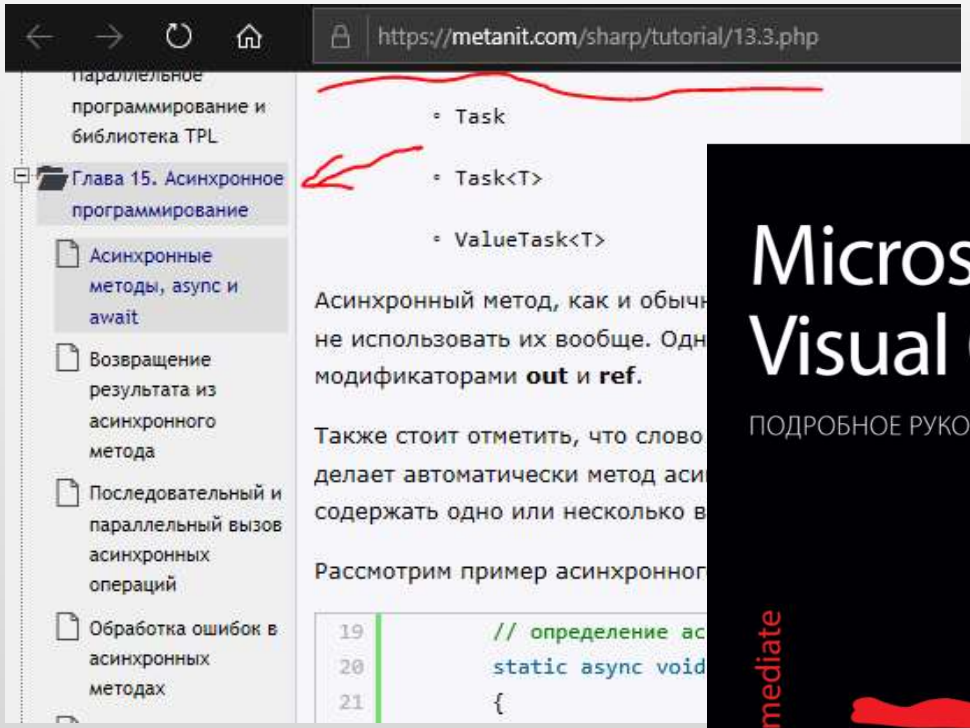
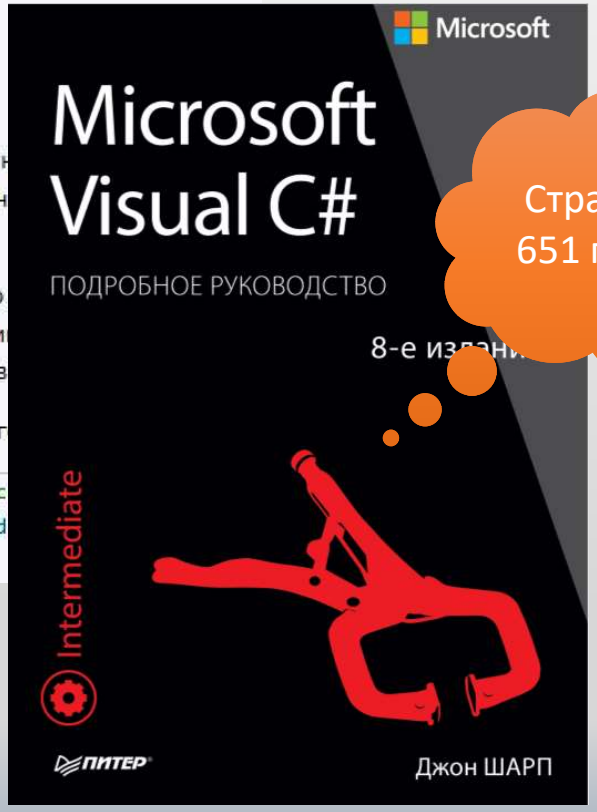
Асинхронные методы

Источники





Страницы с 651 по 664…

Асинхронные методы

- Асинхронным называется метод, не блокирующий текущий поток, в котором началось его выполнение.
  - При вызове асинхронного метода ожидается, что он быстро возвратит управление вызывающей среде и станет выполнять свою работу в отдельном потоке.
- C# предоставляет модификатор методов под названием async и оператор await, которые возлагают основные сложности этого процесса на компилятор

#1 Описание проблемы

```
private void slowMethod()
{
    doFirstLongRunningOperation();
    doSecondLongRunningOperation();
    doThirdLongRunningOperation();
    message.Text = "Processing Completed";
}

private void doFirstLongRunningOperation()
{
    ...
}

private void doSecondLongRunningOperation()
{
    ...
}

private void doThirdLongRunningOperation()
{
    ...
}
```

Если вызвать slowMethod из обработчика события Click для кнопки, интерфейс перестанет реагировать на действия пользователя вплоть до завершения выполнения метода

#2 Описание проблемы

```
private void slowMethod()
{
    Task task = new Task(doFirstLongRunningOperation);
    task.ContinueWith(doSecondLongRunningOperation);
    task.ContinueWith(doThirdLongRunningOperation);
    task.Start();
    task.Wait();
    message.Text = "Processing Completed";
}
```

Теперь вызов метода Wait блокирует поток, выполняющий метод slowMethod

#3 Описание проблемы

```
private void slowMethod()
{
    Task task = new Task(doFirstLongRunningOperation);
    task.ContinueWith(doSecondLongRunningOperation);
    task.ContinueWith(doThirdLongRunningOperation);
    task.ContinueWith((t) => message.Text = "Processing Complete");
    task.Start();
}
```

Если попытаться запустить этот код, последнее продолжение выдаст исключение:
Манипулировать элементами пользовательского интерфейса может только поток пользовательского интерфейса

async и await

- Ключевые слова async и await предназначены в C# для того, чтобы определять и вызывать методы, способные выполняться в асинхронном режиме:
  - модификатор async показывает, что метод содержит функции, которые должны выполняться в асинхронном режиме;
  - оператор await указывает места, в которых функции должны выполняться в асинхронном режиме.

Асинхронный slowMethod

```
private async void slowMethod()
{
    await doFirstLongRunningOperation();
    await doSecondLongRunningOperation();
    await doThirdLongRunningOperation();
    message.Text = "Processing Complete";
}
```

- Когда в методе, объявленном с ключевым словом async, этому компилятору встречается оператор await, происходит переформатирование операнда, который следует за этим оператором, в задачу, запускаемую в том же потоке, что и async-метод.
- Весь остальной код превращается в продолжение, запускаемое после завершения задачи, которое снова запускается в том же потоке.
- Теперь, поскольку поток, в котором был запущен async-метод, был потоком, в котором запущен пользовательский интерфейс, у него есть непосредственный доступ к элементам управления окна

Модификатор async

- Не является признаком того, что метод запускается в асинхронном режиме в отдельном потоке.
- Его роль ограничивается указанием на то, что код в методе может быть разделен для получения одного или нескольких продолжений.
- Когда запускаются эти продолжения, они выполняются в том же самом потоке, что и исходный метод.

Оператор await

- Указывает место, с которого компилятор C# может разбить код на продолжение.
- Сам оператор await ожидает в качестве своего операнда объект, допускающий ожидание (объект типа Task)

```
private Task doFirstLongRunningOperation()
{
    Task t = Task.Run(() => { /* сюда помещается код для этого метода */ });
    return t;
}
```

Параллельные операции

- Есть ли возможность разбить работу, выполняемую методом doFirstLongRunningOperation, на ряд параллельных операций?

```
private Task doFirstLongRunningOperation()
{
    Task first = Task.Run(() => { /* код для первой операции */ });
    Task second = Task.Run(() => { /* код для второй операции */ });
    return ...; // Какой из объектов возвращать, first или second?
}
```

Если метод возвращает first, оператор await в slowMethod будет дожидаться только завершения этой задачи, но не задачи second.

Решение

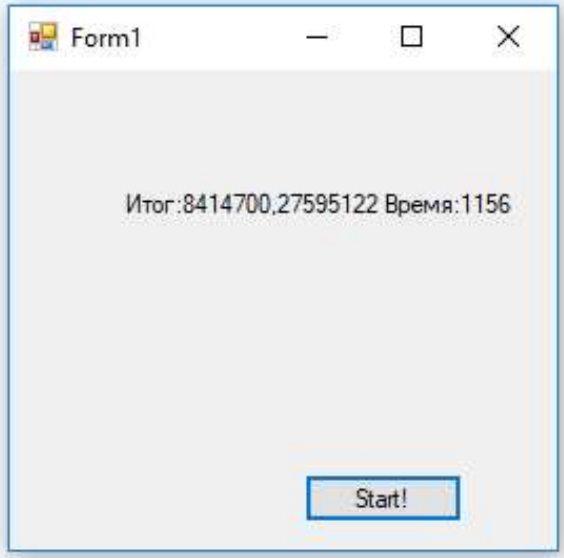
```
private async Task doFirstLongRunningOperation()
{
    Task first = Task.Run(() => { /* код для первой операции */ });
    Task second = Task.Run(() => { /* код для второй операции */ });
    await first;
    await second;
}
```

Метод **doFirstLongRunningOperation** создает и запускает в режиме параллельного выполнения задачи **first** и **second**, компилятор переформатирует операторы **await** в код, ожидающий завершения задачи first, за которой следует продолжение, ожидающее завершения задачи second,

Итерация 1

```
private void StartButton_Click(object sender, EventArgs e)
{
    Stopwatch sw = new Stopwatch();
    sw.Start();
    Task<double> t = new Task<double>(work);
    t.Start();
    t.Wait();
    sw.Stop();
    message.Text = string.Format("Итог:{0} Время:{1}", t.Result, sw.ElapsedMilliseconds);
}

static double work()
{
    double x, y, s = 0;
    for (int i = 1; i < 10000000; i++)
    {
        x = i / (i + 1.0);
        y = Math.Sin(x);
        s += y;
    }
    return s;
}
```



Итерация 2

```
private void StartButton_Click(object sender, EventArgs e)
{
    slowMethod();
}

private async void slowMethod()
{
    Stopwatch sw = new Stopwatch();
    sw.Start();
    double res = await work();
    sw.Stop();
    message.Text = string.Format("Итог:{0}  Время:{1}", res, sw.ElapsedMilliseconds);
}

static Task<double> work()
{
    double x, y, s = 0;
    return Task.Run(() =>
    {
        for (int i = 1; i < 10000000; i++)
        {
            x = i / (i + 1.0);
            y = Math.Sin(x);
            s += y;
        }
        return s;
    });
}
```