

Шпаргалка по теме "Асинхронность в Windows Forms"

Асинхронность позволяет выполнять длительные операции (загрузку данных, чтение файлов) без блокировки UI.

Ключевые слова:

- `async` - Обозначает, что метод может выполняться асинхронно
- `await` - Приостанавливает выполнение метода до завершения задачи
- `Task` - Представляет асинхронную задачу
- `Task<T>` - Асинхронная задача с возвращаемым результатом

```
// Асинхронный метод
public async Task DoWorkAsync()
{
    await Task.Delay(1000); // Асинхронная задержка
    // Продолжение работы после задержки
}

// Асинхронный метод с возвращаемым значением
public async Task<string> GetDataAsync()
{
    await Task.Delay(1000);
    return "Data";
}

// Вызов асинхронных методов
private async void button1_Click(object sender, EventArgs e)
{
    await DoWorkAsync();
    string result = await GetDataAsync();
    textBox1.Text = result;
}

// Task без возвращаемого значения
Task task = Task.Run(() =>
{
    // Длительная операция
});

// Task с возвращаемым значением
Task<int> task = Task.Run(() =>
{
    // Вычисления
    return 42;
});

// Ожидание завершения задачи
await task;

// Получение результата
int result = await task; // для Task<int>
```

Пример: Асинхронная задержка:

```
public async Task DoSomethingAsync()
{
    await Task.Delay(1000); // Асинхронная задержка на 1 секунду
}
```

Асинхронные операции в Windows Forms

Пример: асинхронная загрузка данных с веб-страницы

```
private async void button1_Click(object sender, EventArgs e)
{
    using (HttpClient client = new HttpClient()) // Создаем HTTP-клиент
    {
        textBox1.Text = "Загрузка..."; // Выводим сообщение пользователю
        string data = await client.GetStringAsync("https://example.com"); // Асинхронно загружаем данные
        textBox1.Text = data; // Отображаем загруженные данные в TextBox
    }
}
```

Как это работает?

- `await` останавливает выполнение метода, пока `GetStringAsync()` загружает данные.
- UI остается отзывчивым.

Пример: Асинхронное чтение текстового файла:

```
// Асинхронный метод для чтения содержимого файла
public async Task<string> ReadFileAsync(string filePath)
{
    // Используем конструкцию using, чтобы автоматически закрыть файл после чтения
    using (StreamReader reader = new StreamReader(filePath))
    {
        // Асинхронно читаем весь файл
        string content = await reader.ReadToEndAsync();
        return content; // Возвращаем содержимое файла
    }
}

// Обработчик нажатия кнопки (асинхронный)
private async void button1_Click(object sender, EventArgs e)
{
    string filePath = "example.txt"; // Путь к файлу
    try
    {
        // Асинхронно читаем файл и ожидаем завершения операции
        string content = await ReadFileAsync(filePath);
        textBox1.Text = content; // Выводим содержимое файла в TextBox
    }
    catch (Exception ex)
    {
        // Если произошла ошибка (например, файл не найден), выводим сообщение
        MessageBox.Show("Ошибка при чтении файла: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Что делает код?

- Метод `ReadFileAsync` асинхронно читает файл, используя `StreamReader.ReadToEndAsync()`.
- В `button1_Click` вызывается `ReadFileAsync(filePath)`, и содержимое файла отображается в `textBox1`.
- В случае ошибки (например, если файл не найден) отображается `MessageBox` с сообщением об ошибке.

Пример: Асинхронная задержка с обновлением ProgressBar

```
// Асинхронный метод с задержкой и обновлением прогресса
public async Task DelayWithProgressAsync(IProgress<int> progress)
{
    for (int i = 0; i <= 100; i++) // Цикл от 0 до 100 (имитация прогресса)
    {
        await Task.Delay(50); // Асинхронная задержка на 50 мс
        progress.Report(i); // Сообщаем о текущем значении прогресса
    }
}
```

```

// Обработчик нажатия кнопки (асинхронный)
private async void button1_Click(object sender, EventArgs e)
{
    // Создаем объект Progress<int>, который обновляет ProgressBar
    var progress = new Progress<int>(value => progressBar1.Value = value);

    // Вызываем метод с передачей объекта прогресса
    await DelayWithProgressAsync(progress);

    // После завершения задержки показываем сообщение
    MessageBox.Show("Задержка завершена!");
}

```

Что делает код?

Метод DelayWithProgressAsync

- Запускает цикл от 0 до 100.
- Каждые 50 мс выполняет Task.Delay(50), чтобы не блокировать UI.
- Вызывает progress.Report(i), обновляя значение прогресса.

Метод button1_Click

- Создает объект Progress<int>, который обновляет progressBar1.Value.
- Вызывает DelayWithProgressAsync(progress), передавая ему объект прогресса.
- После завершения задержки выводит MessageBox с сообщением.

Пример: Асинхронная задача с возвратом результата

```

// Асинхронный метод, выполняющий длительную операцию в фоновом потоке
public async Task<int> CalculateAsync()
{
    return await Task.Run(() => // Запускаем задачу в фоновом потоке
    {
        int result = 0;
        // Длительный вычислительный процесс: суммируем числа от 0 до 999999
        for (int i = 0; i < 1000000; i++)
        {
            result += i;
        }
        return result; // Возвращаем результат
    });
}

// Обработчик нажатия кнопки (асинхронный)
private async void button1_Click(object sender, EventArgs e)
{
    int result = await CalculateAsync(); // Асинхронно вызываем метод CalculateAsync
    label1.Text = $"Результат: {result}"; // Отображаем результат в Label
}

```

Как работает код?

Метод CalculateAsync

- Запускает вычисление в фоновом потоке с помощью Task.Run().
- Выполняет сложение чисел от 0 до 999999.
- После завершения возвращает результат.

Метод button1_Click

- Асинхронно вызывает CalculateAsync(), не блокируя UI.
- После завершения вычисления обновляет label1 с результатом.

Отмена асинхронной операции

Пример: Использование CancellationToken

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

public partial class Form1 : Form
{
    private CancellationSource cts; // Токен для отмены операции

    public Form1()
    {
        InitializeComponent(); // Инициализация компонентов формы
    }

    // Обработчик нажатия кнопки "Старт" (асинхронный)
    private async void StartButton_Click(object sender, EventArgs e)
    {
        cts = new CancellationSource(); // Создаем новый токен отмены
        try
        {
            // Запускаем длительную операцию в фоновом потоке и передаем токен отмены
            await Task.Run(() => LongRunningOperation(cts.Token), cts.Token);
        }
        catch (OperationCanceledException) // Обрабатываем отмену операции
        {
            label1.Text = "Операция отменена"; // Сообщаем пользователю об отмене
        }
    }
}
```

```
// Длительная операция с возможностью отмены
private void LongRunningOperation(CancellationSource token)
{
    for (int i = 0; i < 100; i++)
    {
        token.ThrowIfCancellationRequested(); // Проверяем, был ли запрос на отмену
        Thread.Sleep(100); // Имитация долгого вычисления (100 * 100 мс = 10 секунд)
    }
}

// Обработчик нажатия кнопки "Отмена"
private void CancelButton_Click(object sender, EventArgs e)
{
    cts?.Cancel(); // Отправляем сигнал отмены, если токен существует
}
```

Основные асинхронные методы для работы с файлами, сетью и другими операциями.

Асинхронные методы для работы с файлами

Чтение файлов

File.ReadAllTextAsync

Читает весь текст из файла асинхронно.

```
string content = await File.ReadAllTextAsync("file.txt");
```

File.ReadAllLinesAsync

Читает все строки из файла асинхронно.

```
string[] lines = await File.ReadAllLinesAsync("file.txt");
```

File.ReadAllBytesAsync

Читает весь файл как массив байтов асинхронно.

```
byte[] bytes = await File.ReadAllBytesAsync("file.bin");
```

FileStream.ReadAsync

Читает данные из файла в буфер асинхронно (полезно для больших файлов).

```
using (FileStream fs = new FileStream("file.bin", FileMode.Open, FileAccess.Read))
{
    byte[] buffer = new byte[1024];
    int bytesRead = await fs.ReadAsync(buffer, 0, buffer.Length);
}
```

Запись в файлы

File.WriteAllTextAsync

Записывает текст в файл асинхронно.

```
await File.WriteAllTextAsync("file.txt", "Hello, World!");
```

File.WriteAllLinesAsync

Записывает массив строк в файл асинхронно.

```
string[] lines = { "Line 1", "Line 2", "Line 3" };
await File.WriteAllLinesAsync("file.txt", lines);
```

File.WriteAllBytesAsync

Записывает массив байтов в файл асинхронно.

```
byte[] data = { 0x48, 0x65, 0x6C, 0x6C, 0x6F }; // "Hello" в байтах
await File.WriteAllBytesAsync("file.bin", data);
```

FileStream.WriteAsync

Записывает данные из буфера в файл асинхронно (полезно для больших файлов).

```
using (FileStream fs = new FileStream("file.bin", FileMode.Create, FileAccess.Write))
{
    byte[] buffer = { 0x48, 0x65, 0x6C, 0x6C, 0x6F }; // "Hello" в байтах
    await fs.WriteAsync(buffer, 0, buffer.Length);
}
```

Асинхронные методы для работы с сетью

HTTP-запросы

HttpClient.GetAsync

Выполняет GET-запрос асинхронно.

```
using (HttpClient client = new HttpClient())
{
    HttpResponseMessage response = await client.GetAsync("https://example.com");
    string content = await response.Content.ReadAsStringAsync();
}
```

HttpClient.PostAsync

Выполняет POST-запрос асинхронно.

```
using (HttpClient client = new HttpClient())
{
    var data = new StringContent("{\"key\": \"value\"}", Encoding.UTF8, "application/json");
    HttpResponseMessage response = await client.PostAsync("https://example.com/api", data);
    string result = await response.Content.ReadAsStringAsync();
}
```

HttpClient.GetByteArrayAsync

Загружает данные как массив байтов асинхронно.

```
using (HttpClient client = new HttpClient())
{
    byte[] data = await client.GetByteArrayAsync("https://example.com/file.bin");
}
```

HttpClient.GetStreamAsync

Загружает данные как поток асинхронно.

```
using (HttpClient client = new HttpClient())
{
    Stream stream = await client.GetStreamAsync("https://example.com/file.bin");
}
```

Асинхронные методы для работы с MySQL

Установка соединения с базой данных

MySqlConnection.OpenAsync

Асинхронно открывает соединение с базой данных.

```
using MySqlConnector;

await using var connection = new MySqlConnection("Server=localhost;Database=test;User ID=root;Password=password");
await connection.OpenAsync();
```

Выполнение SQL-запросов

MySqlCommand.ExecuteNonQueryAsync

Асинхронно выполняет SQL-запрос, который не возвращает данные (например, INSERT, UPDATE, DELETE).

```
await using var command = connection.CreateCommand();
command.CommandText = "INSERT INTO users (name, age) VALUES ('John', 30);";
int rowsAffected = await command.ExecuteNonQueryAsync();
Console.WriteLine($"Добавлено строк: {rowsAffected}");
```

MySqlCommand.ExecuteScalarAsync

Асинхронно выполняет SQL-запрос и возвращает первое значение первой строки результата (например, для COUNT, MAX, MIN).

```
await using var command = connection.CreateCommand();
command.CommandText = "SELECT COUNT(*) FROM users;";
long count = (long)await command.ExecuteScalarAsync();
Console.WriteLine($"Всего пользователей: {count}");
```

MySqlCommand.ExecuteReaderAsync

Асинхронно выполняет SQL-запрос и возвращает MySqlDataReader для чтения данных.

```
await using var command = connection.CreateCommand();
command.CommandText = "SELECT id, name, age FROM users;";
await using var reader = await command.ExecuteReaderAsync();

while (await reader.ReadAsync())
{
    int id = reader.GetInt32("id");
    string name = reader.GetString("name");
    int age = reader.GetInt32("age");
    Console.WriteLine($"ID: {id}, Name: {name}, Age: {age}");
}
```

Чтение данных

MySqlDataReader.ReadAsync

Асинхронно читает следующую строку из результата запроса.

```
await using var reader = await command.ExecuteReaderAsync();
while (await reader.ReadAsync())
{
    int id = reader.GetInt32(0); // Чтение по индексу столбца
    string name = reader.GetString("name"); // Чтение по имени столбца
    Console.WriteLine($"ID: {id}, Name: {name}");
}
```

MySqlDataReader.GetFieldValueAsync<T>

Асинхронно читает значение столбца указанного типа.

```
int id = await reader.GetFieldValueAsync<int>(0);
string name = await reader.GetFieldValueAsync<string>("name");
```

Другие асинхронные методы

Таймеры

Task.Delay

Асинхронно ожидает указанное время.

```
await Task.Delay(1000); // Ожидание 1 секунды
```

Параллельные операции

Task.WhenAll

Ожидает завершения всех задач в массиве.

```
Task<int> task1 = Task.Run(() => 1);
Task<int> task2 = Task.Run(() => 2);
int[] results = await Task.WhenAll(task1, task2);
```

Task.WhenAny

Ожидает завершения любой задачи из массива.

```
Task<int> task1 = Task.Run(() => 1);
Task<int> task2 = Task.Run(() => 2);
Task<int> completedTask = await Task.WhenAny(task1, task2);
int result = await completedTask;
```

Асинхронные потоки (IAsyncEnumerable)

`yield return` в асинхронных методах

Возвращает элементы асинхронно.

```
public async IAsyncEnumerable<int> GetNumbersAsync()
{
    for (int i = 0; i < 10; i++)
    {
        await Task.Delay(100); // Имитация асинхронной операции
        yield return i;
    }
}

// Использование
await foreach (var number in GetNumbersAsync())
{
    Console.WriteLine(number);
}
```